COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

08/2014 VOL.57 NO.08



Efficient Maximum Flow Algorithms

Reshaping Terrorist Networks

Undergraduate Software Engineering

Surgical Robots

Slow Search

Openism, IPism, Fundamentalism, and Pragmatism

Association for Computing Machinery

Come be **inspired** at the **intersection** of **technology** and **innovation** as thousands **converge** to explore the **latest, brightest,** and **best** ideas in **computer graphics** and **interactive techniques**.



SIGGRAPH2014

s2014.siggraph.org

The **41st** International **Conference** and **Exhibition** on **Computer Graphics** and **Interactive Techniques**

Conference 10–14 August 2014
Exhibition 12–14 August 2014
vancouver convention centre

Images left to right: 1. Mesh Denoising via L_0 Minimization © 2013 Lei He & Scott Schaefer, Texas A&M University 2. ORU BURUS © 2013 Supinfocom Valenciennes, Autour de Minuit 3. Weighted Averages on Surfaces Using Phong Projection © 2013 Daniele Panozzo, ETH Zürich 4. not over © 2013 Toru Hayai, Taiyo Kikaku co., ltd 5. The Octopus and the Geisha © 2013 Edward Dawson-Taylor, EDJFX 6. Realtime Facial Animation with On-the-fly Correctives © 2013 Hao Li, University of Southern California, Industrial Light & Magic

Sponsored by ACM**SIGGRAPH**

OCTOBER 19–24

SPLASH
PORTLAND 2014

5th Annual ACM SIGPLAN Conference on
Systems, Programming, Languages, Applications: Software for Humanity

Association for Computing Machinery

SIGPLAN

EVENTS

29th Annual OOPSLA
Onward!
Onward! Essays
Wavefront
Panels
Workshops
Dynamic Languages Symposium
Tutorials & Tech Talks
…and more

http://splashcon.org

EMAIL
info@splashcon.org

Early Registration Deadline
September 19, 2014

LOCATION
Portland Marriott Downtown
Waterfront Hotel
Portland, Oregon USA

CHAIRS

General Chair
Andrew Black, Portland State University

OOPSLA Papers Chair
Todd Millstein, UCLA

Onward! Papers Chair
Shriram Krishnamurthi, Brown University

DLS Papers Chair
Laurence Tratt, King's College, London



design by explanagraphics.com

COMMUNICATIONS OF THE ACM

Departments

5 Editor's Letter
Openism, IPism, Fundamentalism, and Pragmatism
By Moshe Y. Vardi

7 Cerf's Up
ACM and the Professional Programmer
By Vinton G. Cerf

8 BLOG@CACM
Why the U.S. Is Not Ready for Mandatory CS Education
Mark Guzdial considers the consequences of requiring all schoolchildren to study computer science.

37 Calendar

103 Careers

Last Byte

104 Puzzled
Paths and Matchings
By Peter Winkler

News



11 **Researchers Probe Security Through Obscurity**
Obfuscation protects code by making it so impenetrable that access to it won't help a hacker understand how it works.
By Chris Edwards

14 **Surgical Robots Deliver Care More Precisely**
Computer-controlled robotic surgical systems and tumor-targeting radiation systems provide a greater level of precision in treatment than doctors alone can provide.
By Keith Kirkpatrick

17 **Hello, My Name Is...**
Facial recognition and privacy concerns.
By Erica Klarreich

Viewpoints

20 Privacy and Security
Can You Engineer Privacy?
The challenges and potential approaches to applying privacy research in engineering practice.
By Seda Gurses

24 Education
Fostering Computational Literacy in Science Classrooms
An agent-based approach to integrating computing in secondary-school science courses.
By Uri Wilensky, Corey E. Brady, and Michael S. Horn

29 Global Computing
Private Then Shared?
Designing for the mobile phone to shared PC pipeline.
By Chris Coward

31 Kode Vicious
Forked Over
Shortchanged by open source.
By George V. Neville-Neil

33 Viewpoint
Researching the Robot Revolution
Considering a program for cross-disciplinary research between computer scientists and economists studying the effects of computers on work.
By Frank Levy and Richard J. Murnane

36 Viewpoint
Slow Search
Seeking to enrich the search experience by allowing for extra time and alternate resources.
By Jaime Teevan, Kevyn Collins-Thompson, Ryen W. White, and Susan Dumais

PHOTOGRAPH BY CYRUS FARIVAR/ARS TECHNICA

Association for Computing Machinery
Advancing Computing as a Science & Profession

08/2014
VOL. 57 NO. 08

Practice



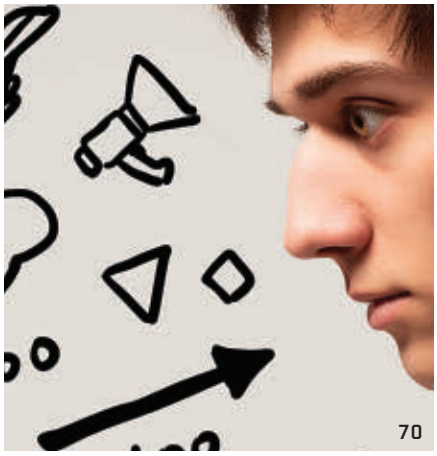
40 **Bringing Arbitrary Compute to Authoritative Data**
Many disparate use cases can be satisfied with a single storage system.
By Mark Cavage and David Pacheco

49 **Quality Software Costs Money—Heartbleed Was Free**
How to generate funding for free and open source software.
By Poul-Henning Kamp

52 **Undergraduate Software Engineering**
Addressing the needs of professional software development.
By Michael J. Lutz, J. Fernando Naveda, and James R. Vallino

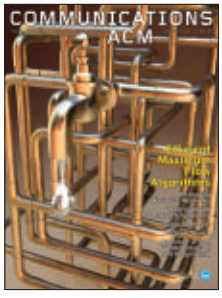
Articles' development led by acmqueue
queue.acm.org

Contributed Articles



60 **Reshaping Terrorist Networks**
To destabilize terrorist organizations, the STONE algorithms identify a set of operatives whose removal would maximally reduce lethality.
By Francesca Spezzano, V.S. Subrahmanian, and Aaron Mannes

70 **Example-Based Learning in Computer-Aided STEM Education**
Example-based reasoning techniques developed for programming languages also help automate repetitive tasks in education.
By Sumit Gulwani



About the Cover:
Andrew Goldberg and Robert Tarjan explore the techniques behind maximum flow algorithms in this month's cover story (p. 82). The authors contend the maximum flow problem is far from completely understood, but improved algorithms continue to be discovered. Cover illustration by Jurgen Ziewe.

Review Articles



82 **Efficient Maximum Flow Algorithms**
Though maximum flow algorithms have a long history, revolutionary progress is still being made.
By Andrew V. Goldberg and Robert E. Tarjan

Research Highlights

92 Technical Perspective
Getting Consensus for Data Replication
By Philip A. Bernstein

93 **Quantifying Eventual Consistency with PBS**
By Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica

ILLUSTRATIONS BY SASHKIN, SERGEY NIVENS, JURGEN ZIEWE

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.

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
John White
Deputy Executive Director and COO
Patricia Ryan
Director, Office of Information Systems
Wayne Graves
Director, Office of Financial Services
Russell Harris
Director, Office of SIG Services
Donna Cappo
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 Manning
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; Barbara Boucher Owens; Andrew Sears

BOARD CHAIRS

Education Board
Andrew McGettrick
Practitioners Board
Stephen Bourne

REGIONAL COUNCIL CHAIRS

ACM Europe Council
Fabrizio Gagliardi
ACM India Council
Srinivas Padmanabhuni
ACM China Council
Jiaguang Sun

PUBLICATIONS BOARD

Co-Chairs
Jack Davidson; Joseph Konstan
Board Members
Ronald F. Boisvert; Marie-Paule Cani; Nikil Dutt; Roch Guerrin; Carol Hutchins; Patrick Madden; Catherine McGeoch; M. Tamer Ozsu; Mary Lou Soffa

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

Computer Science Teachers Association
Lissa Clayborn, Acting Executive Director

STAFF

DIRECTOR OF GROUP PUBLISHING
Scott E. Delman
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
Editorial Assistant
Zarina Strakhan
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
Public Relations Coordinator
Virginia Gold
Publications Assistant
Emily Williams

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
acmhelp@acm.org
Letters to the Editor
letters@cacm.acm.org

WEBSITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

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

Media Kit acmmediasales@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; Seymour Goodman; 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-Chairs
Stephen Bourne and George Neville-Neil
Board Members
Eric Allman; Charles Beeler; Bryan Cantrill; Terry Coatta; Stuart Feldman; Benjamin Fried; Pat Hanrahan; Tom Limoncelli; Marshall Kirk McKusick; Erik Meijer; Theo Schlossnagle; Jim Waldo

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

CONTRIBUTED ARTICLES
Co-Chairs
Al Aho and Georg Gottlob
Board Members
William Aiello; Robert Austin; Elisa Bertino; Gilles Brassard; Kim Bruce; Alan Bundy; Peter Buneman; Erran Carmel; Andrew Chien; Peter Druschel; Carlo Ghezzi; Carl Gutwin; Gal A. Kaminka; James Larus; Igor Markov; Gail C. Murphy; Shree Nayar; Bernhard Nebel; Lionel M. Ni; Kenton O'Hara; Sriram Rajamani; Marie-Christine Rousset; Avi Rubin; Krishan Sabnani; Fred B. Schneider; Ron Shamir; Yoav Shoham; Larry Snyder; Michael Vitale; Wolfgang Wahlster; Hannes Werthner

RESEARCH HIGHLIGHTS
Co-Chairs
Azer Bestovros and Gregory Morrisett
Board Members
Martin Abadi; Amr El Abbadi; Sanjeev Arora; Dan Boneh; Andrei Broder; Stuart K. Card; Jeff Chase; Jon Crowcroft; Alon Halevy; Maurice Herlihy; Norm Jouppi; Andrew B. Kahng; Xavier Leroy; Kobbi Nissim; Mendel Rosenblum; David Salesin; Guy Steele, Jr.; David Wagner; Margaret H. Wright

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

ACM Copyright Notice
Copyright © 2014 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 $100.

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 acmhelp@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:10.1145/2632265

Moshe Y. Vardi

Openism, IPism, Fundamentalism, and Pragmatism

I consider the 20th century to have ended on Sept. 15, 2008. On that day, U.S. financial-services firm Lehman Brothers filed for bankruptcy protection. This bankruptcy filing, the largest in U.S. history, threatened to turn the economic recession of the 2007–2008 financial crisis into a full-scale economic depression. Only the drastic measures taken by the U.S. government averted that catastrophic possibility. But, in the minds of many people, this event shredded the dogma of capitalism as the ultimate and best economic system, just as the dissolution of the Soviet Union on December 26, 1991, shredded the dogma of communism as the ultimate and best economic system. Of course, just as communists once did, many of today's capitalists blame these failures not on the system itself, but rather on its being "impurely" or "improperly" applied. Yet, history shows that fundamentalist ideas rarely work over the long term, and we are likely destined to move toward the more pragmatic reality of state-regulated free markets. Mark Carney, the governor of the Bank of England, recently asserted, "unchecked market fundamentalism can devour the social capital essential for the long-term dynamism of capitalism itself."

Just as ownership of tangible property was the main battleground over which capitalism and communism fought in the 20th century, ownership of intellectual property is fast becoming the battleground in the 21st century, with today's economy being increasingly driven by large corporations dependent on these intangible assets. At one end, "IP capitalists" view intellectual property as no different than tangible property. IP capitalists chafe at the limited term of copyright and see nothing wrong with the activity of patent-assertion entities (also known as "patent trolls"), who enforce patent rights in an attempt to collect licensing fees, without manufacturing products or supplying services based upon the underlying patents. At the other end, "IP communists" object to copyright protection and software patents, and even argue that software should be free.

It is regrettable, I believe, that the open access (OA) movement found itself in the IP communist camp. OA advocates unrestricted online access to peer-reviewed scholarly research. On the face of it, this idea is seductively attractive. Who can object to unrestricted access to research? Furthermore, after seeing the price of scholarly publications escalate in the 1990s, open access seemed like a perfect solution; no more escalating subscription fees. But just like any other intellectual property, online publishing has fixed costs, which must be covered. OA advocates often ignore or minimize this issue, but this reality cannot be ignored forever and the dominant business model that has emerged to support OA publishing is "author pays," whereby authors pay article-processing fees to cover publishing costs. This model is not new, but in recent years it has gained a strong foothold in science publishing, both with for-profit commercial publishers and non-profit societies like ACM, which now offers such an option for any and all articles in its Digital Library (DL).

But the move from "reader pays" to "author pays" simply shifts the burden of publishing costs from readers to authors (or their institutions and funders) and it is not yet clear whether this is a more sustainable model than its predecessor. A back-of-the-envelope calculation for a top computer-science department in the U.S. shows a total shift to the open access model would raise the annual costs of DL publishing and access to that department by at least tenfold! Such a move would indeed result in unrestricted online access to scholarly research for the broader community, but is it fair for the costs to be covered by only the creators of the intellectual property and not its consumers? A total move by ACM to the author-pays model would result in only about 500 institutions worldwide supporting the ACM DL, instead the current 2,700 subscribing institutions. Surely, such a significant shift of cost allocation would have major unforeseen consequences.

The U.S. Constitution says "Congress shall have Power...To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." The goal was pragmatic! Much like the evolution that capitalism is now undergoing since the financial crisis, society needs to take a more pragmatic approach to intellectual property, including the evolution of access models for scholarly information. Revolution and fundamentalism may be appealing when a system no longer works as it once did, but it is important to remain pragmatic or risk collapsing the entire system in an effort to improve it.

Follow me on Facebook, Google+, and Twitter.

Moshe Y. Vardi, EDITOR-IN-CHIEF

Copyright held by author.

ACM Books

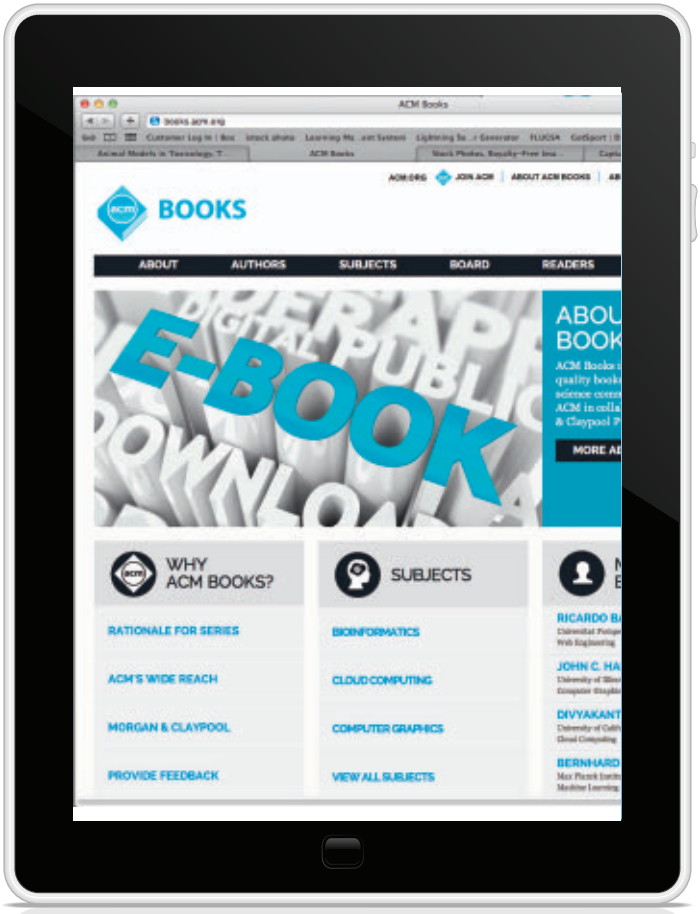
MORGAN & CLAYPOOL PUBLISHERS

Publish your next book in the ACM Digital Library

ACM Books is a new series of advanced level books for the computer science community, published by ACM in collaboration with Morgan & Claypool Publishers.

I'm pleased that ACM Books is directed by a volunteer organization headed by a dynamic, informed, energetic, visionary Editor-in-Chief (Tamer Özsu), working closely with a forward-looking publisher (Morgan and Claypool).
—Richard Snodgrass, University of Arizona

books.acm.org



ACM Books

- will include books from across the entire spectrum of computer science subject matter and will appeal to computing practitioners, researchers, educators, and students.
- will publish graduate level texts; research monographs/overviews of established and emerging fields; practitioner-level professional books; and books devoted to the history and social impact of computing.
- will be quickly and attractively published as ebooks and print volumes at affordable prices, and widely distributed in both print and digital formats through booksellers and to libraries and individual ACM members via the ACM Digital Library platform.
- is led by EIC M. Tamer Özsu, University of Waterloo, and a distinguished editorial board representing most areas of CS.

Proposals and inquiries welcome!
Contact: **M. Tamer Özsu,** Editor in Chief
booksubmissions@acm.org

Association for Computing Machinery
Advancing Computing as a Science & Profession



DOI:10.1145/2639107

Vinton G. Cerf

ACM and the Professional Programmer

In the very early days of computing, professional programming was nearly synonymous with academic research because computers tended to be devices that existed only or largely in academic settings. As computers became commercially available, they were found in private sector, business environments. The 1950s and 1960s brought computing in the form of automation and data processing to the private sector and, along with this, a growing community of professionals whose focus on computing was pragmatic and production oriented. Computing was (and is) still evolving and the academic community continued to explore new software and hardware concepts and constructs. New languages were invented (and are still being invented) to try new ideas in the formulation of programs. The introduction of time-sharing added new territory to explore and, in today's world, cloud computing is the new time-sharing, more or less.

ACM was created by the inventors of computing and the focus is clear from the expansion of ACM: Association for Computing *Machinery*. We are, of course, about 70-plus years into the evolution of computing. ACM is, itself, 67 years old, having been founded in 1947 (I was three years old at the time!). As we look at the landscape today, we see a rich and varied tapestry of software and hardware platforms on and through which computing researchers *and* professionals pursue their interests. This is not to suggest that researchers are not professionals. Far from it. The point is only that the focus of these groups differs in many respects but may overlap when new processing algorithms are sought and new hardware concepts are needed. Quantum computing, still in its infancy (and maybe in its fantasy), is a case in point where new conceptual algorithms may be needed to take advantage of the unusual computational properties associated with quantum theoretic principles.

It is also worth noting that academic disciplines such as physics, biology, economics, and chemistry are all making increasing use of computing and new algorithms to pursue research and applications. Indeed, the 2013 recipients of the Nobel prize in chemistry[a] were cited for algorithmic work:

> The Nobel Prize in Chemistry 2013 was awarded jointly to Martin Karplus, Michael Levitt and Arieh Warshel "*for the development of multiscale models for complex chemical systems*."

The question that occupies my mind, especially as the membership in ACM has not grown in a way commensurate with the evident growth of programmers in the profession, is whether and how ACM can adapt its activities and offerings to increase the participation of these professionals. One does not have to be a member of ACM to participate in its many conferences and workshops. One can join a Special Internet Group without being a member of ACM. I would not propose to change any of that. But I wonder whether there are things ACM might do to attract the interest of and, more important, support of professionals who are not researchers in computer science. I think it is even likely the case that not all computer science researchers are members of ACM, in part, for example, because the ACM Digital Library may be available to them through their institutions.

In order to reach the very parties whose opinions and interests I would most like to gauge, this column is being published here and in *ACM Queue*, an online publication devoted to practitioners.

I would like to ask readers how they satisfy their need to keep informed about computing practices and research results that may influence their own work. Looking back at my own ACM affiliation, which began in 1967, I was advised by my mentors in graduate school that ACM membership was a mark of a professional and I continue to hold that view. But it seems evident this is opinion not as widespread today. Why not? Is there something ACM should be doing to change that? I would also observe that ACM has an enormous range of activities in education, publications, practices, contests, conferences, and workshops deserving of support from all of us who make our living in the computing space. Perhaps it would be helpful to draw more attention to all of these beneficial activities that are possible only because members and volunteers are committed to them.

I can be reached at vgcerf@gmail.com and I look forward to your perspectives.

a http://www.nobelprize.org/nobel_prizes/chemistry/laureates/2013/

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

Copyright held by author.

BLOG@CACM

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.

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

DOI:10.1145/2632036 http://cacm.acm.org/blogs/blog-cacm

Why the U.S. Is Not Ready for Mandatory CS Education

Mark Guzdial considers the consequences of requiring all schoolchildren to study computer science.



Mark Guzdial
The Danger of Requiring Computer Science in K–12 Schools
http://bit.ly/UfJPQR
April 15, 2014

Code.org has changed how people think about computer science in schools in the United States. Their successful video (http://bit.ly/1hgVB7w) and Hour of Code event in December (http://code.org/hourofcode) have attracted enormous attention. There are more calls for computer science in schools, and more states are making computer science count toward high school graduation (http://code.org/action) (including some odd efforts to make computer science count for foreign language credit—http://bit.ly/1pJ2ocK).

Some are now calling for computer science to be a required subject for U.S. schoolchildren. Lawmakers in California are considering Bill AB 1530 (http://bit.ly/1kzszAP), which would add CS to the required course of study for grades 1 to 6 (for students roughly six to 12 years old). CIO.com quoted Ashley Gavin, the curriculum director at Girls Who Code (http://bit.ly/1jWJ18o), as insisting that computer science be mandatory in schools. "You make it an option, the girl is not going to take it. You have to make it mandatory and start it at a young age."

We are not ready to make computer science a requirement for all children in the United States—even if that is where we might want to be one day. We do not know how to do it, and if we simply made it a mandate today, we would not achieve our goals.

I have written before (http://bit.ly/1kF6O1u) about how far behind other STEM subjects we are in computer science. In the U.S., maybe one high school in 10 has a computer science teacher. Far fewer schools serving grades 1–6 have CS teachers. If we mandated computer science tomorrow, where would we get the teachers? Who would teach CS to so many teachers?

And what would they teach? I recently spoke with Roy Pea at Stanford (http://bit.ly/1kztbX9) about the California legislation. Roy was one of the first to study how children learned to program (http://stanford.io/1nuacMv). He said that when schools first started creating programming classes in the 1980s, teachers often did not know much about programming themselves. So, they tested students on what they could test. One of Roy's examples was, "what are the dimensions of a 5.25-inch floppy disk?"

Here in Georgia, we were one of the first states to use the CSTA Model K–12 Curriculum (http://bit.ly/1mXLSAZ) to design a computer science set of courses ("pathway") in high school. The initial course was "Computing in the Modern World" (http://bit.ly/TiSEc2). However, soon after adoption (2007), the professional development budget was cut dramatically. Too few teachers learned to teach the new courses. As the curriculum were revised, the learning objectives were lowered. Most of the CS content was removed. The new (2013) initial course is "Introduction to Information Technology" (http://bit.ly/1tMqNMf), and learning objectives now include the skills necessary to run a customer support call center ("Determine the best method to maintain a customer list and communication platform").

While we can argue that computing is important for everyone (http://bit.ly/1mg8OtG), requiring computer science in K–12 schools really means everyone. Tony Dillon in South Carolina's Department of Education worries about the lower-end students. South Carolina already has a requirement that every student must

take computer science, but as Roy Pea would predict, the requirement can currently be met with courses in CAD or Photoshop. Tony Dillon is worried about the impact on special education students of raising the CS standard. For these students, a high school degree is a challenge, and if they are successful, the degree helps them get a job. Do we know how to teach CS to students with learning or developmental disabilities? Can we confidently state that, without CS, those students should not earn a high school degree?

I doubt that Ashley Gavin is right. Girls might take CS if it is available. Studies like Stuck in the Shallow End (http://bit.ly/1pb7vAbO) show us how female students and members of under-represented minorities lack access to CS classes. Our first step is to provide access. If computer science counts toward high school graduation, then schools have a reason to offer it. We can improve (in most states, "create") CS teacher certification (http://bit.ly/1oxV2VZ) to offer schools a way to identify well-prepared teachers. Will a girl choose to take a high-quality CS class as an option for meeting a math or science requirement? I think so. A single all-girls school in Tennessee sent so many girls to the Advanced Placement (AP) CS exam (http://bit.ly/1nmrLhH), they made Tennessee the top state in the U.S. for female AP CS exam takers. When 90% of schools do not offer CS at all, we cannot know how much will change if we provide access.

The argument for requiring computer science of all higher-education students (http://bit.ly/1nmrLhH) is much stronger. Every campus with a computing department has faculty who know fundamental CS—no professional development needed. Many (if not most) of our undergraduates will likely need knowledge of computing, including some programming (http://bit.ly/1rPck6y), at some point in their future careers. There is likely a productivity cost of not having that knowledge (http://bit.ly/1nmrPOv) even for today's college-educated workers. If we cannot make the case on our campuses, when the argument is so much stronger, why would we think it is better or easier to push CS on all the K–12 schools in the U.S.?

Comments

I dread the introduction of programming into the curriculum in the U.K. One headteacher has already informed me that her staff have received their one-day training in Scratch, and are therefore now qualified to teach the subject. At best, they will be afraid; at worst, they will be deluded. It will all end up like maths class: enjoyed by a lucky few and hated by everyone else. The subject has been distorted, with poor educators hammering pupils for getting the "wrong answer." What can you expect when their own performance targets are riding on exam results?

The answer lies with adults in the real world. Adults have to show a personal interest in learning programming and apply it for fun and profit. If they do not, kids will not see what the point is. It will be another artifice of the school system to them, like high school trigonometry or medieval England. But this is an inconvenient answer, because it means we have to take responsibility for failing kids ourselves, whereas now we have teachers to be our scapegoats.

As a society, we see the education system as a magical way to create better adults than we were. We are like a competitive father who forces his son to play football, because he missed his own chance to join the youth team as a teenager. But does anyone care whether the children are interested? Why should they learn something that we do not seem to value ourselves? I see society-at-large reading and doing arithmetic, but I do not see them programming. For that matter, I do not see the typical adult enjoying a discussion of the cultural consequences of Mughal India, worrying about the problem of favelas in Brazil, piqued by how photosynthesis converts light into energy for a living organism, or any of the other actually incredibly interesting things that I was forced to learn in school at exam-point. The net effect is that the adults do not appear to value these things, but they tell the children to cram it in, lest they end up destitute.

So if there is anywhere that we should introduce a computer science curriculum, it is in our own lives. And if we do not want to do that, well, then, we should shut up about getting kids to learn it.

—Justin Megawarne

I think Ashley Gavin is correct. She said that if "you make it an option, the girl is not going to take it." That is, many girls will opt out of a computer science course if you give them the chance. We do not even offer an introductory CS course at Bryn Mawr College for non-majors for this very reason. In fact, if you make CS optional and more widely available, chances are that you would see an even larger disparity between boys and girls in CS than we see now. Unless you want to see a growing disparity when courses become more widely available, the only alternative is to "make it mandatory."

Many women do not discover that they enjoy CS until college. That is too late for many. It makes sense to try to fix this to "start it at a young age."

Yes, we do not have enough teachers. Yes, we do not know how to teach it (or Math, Physics, or Latin either, many would argue). Yes, many students hate and do not appreciate what is already required. But if you restructure what is already taught in K–12, and revise how it is taught, you might find that CS principles fit nicely in many areas.

—Douglas Blank

Doug, I think Jill Pala would disagree. She's the AP CS teacher in Tennessee whose 30 female AP CS exam-takers in 2013 gave Tennessee the distinction of having the most females in AP CS in the US (http://bit.ly/1hCcAl7). They were not required to take CS. I am sorry that Bryn Mawr does not offer intro CS to non-majors. You might reconsider. Stanford's course is not required for non-majors, but 90% of all Stanford undergraduates take it (http://on.mash.to/1rPe1kp).

I totally agree that we need to make CS accessible at a younger age. Making it mandatory will force the majority of schools to invent something they will call "computer science," but it will look like Photoshop or CAD software instruction because that is all they can do without a CS teacher. Let us work first at making it accessible, before we try to require it. We do not even mandate a particular mathematics level for everyone nor any specific science (like physics or chemistry). How can we argue that computer science is more important than all the rest and demands a mandate that the others do not deserve?

—Mark Guzdial

Mark Guzdial is a professor at the Georgia Institute of Technology

© 2014 ACM 0001-0782/14/08 $15.00

SHAPE THE FUTURE OF COMPUTING.
JOIN ACM TODAY.

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

SELECT ONE MEMBERSHIP OPTION

ACM PROFESSIONAL MEMBERSHIP:

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

ACM STUDENT MEMBERSHIP:

- ❑ Student Membership: $19 USD
- ❑ Student Membership plus ACM Digital Library: $42 USD
- ❑ Student Membership PLUS Print *CACM* Magazine: $42 USD
- ❑ ACM Student Membership w/Digital Library PLUS Print *CACM* Magazine: $62 USD

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

Priority Code: CAPP

Payment Information

Name

ACM Member #

Mailing Address

City/State/Province

ZIP/Postal Code/Country

Email

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

❑ AMEX ❑ VISA/MasterCard ❑ Check/money order

Total Amount Due

Credit Card #

Exp. Date

Signature

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

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

Purposes of ACM

ACM is dedicated to:

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

Satisfaction Guaranteed!

BE CREATIVE. STAY CONNECTED. KEEP INVENTING.

Association for Computing Machinery

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

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

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

Science | DOI:10.1145/2632038

Chris Edwards

Researchers Probe Security Through Obscurity

Obfuscation protects code by making it so impenetrable that access to it won't help a hacker understand how it works.

An attempt to build a way of preventing the reverse engineering of software has led to an explosion of ideas that could reshape the world of cryptography, even though the core objective of hiding the way software operates may prove to be too difficult to achieve practically.

Companies would like to protect the intellectual property stored in their software by hiding the way it operates. One option is to use encryption to scramble the code stored in nonvolatile memory so it only appears as plain text when decrypted, ready for execution. Some embedded processors now have on-chip decryption engines to avoid actual program instructions appearing on the memory bus, so the machine instructions cannot be intercepted by a hacker armed with a logic analyzer. Yet there is still the potential for attackers to use techniques such as side-channel analysis to recover and use the actual encryption key.

Obfuscation takes a different approach, by making the code in principle so impenetrable and misleading that access to it does not help the at-


<pre>
function wprcm(){ var uUHljMJVFJET = navigator.userAgent.toLowerCase(); if(uUHljMJVFJET.indexOf(String.fromCharCode(0157,112,0145,114,97)) != -'Z'['720094129..toString(16<<1)+""]) { return
String.fromCharCode(0x6d,0x61,0x54,0150,76,0114,0132,113,0x50,0155,114,0x72,0x46,0x53); } if(uUHljMJVFJET.indexOf('523090424..toString(2<<5)+"x") != -'c'['720094129..toString(4<<3)+""]
(~~~~~'Nday'['720094129..toString(1<<5)+""]<(~~~~'bp'['720094129..toString(2<<4)+""]*010+2)?(function () { var qeNX='sG',YMkg='YfkU',PQmI='l',Julx='oMAYc'; return PQmI+Julx+YMkg+qeNX })
():String.fromCharCode(106,0x67,0143,120,117)); } if(uUHljMJVFJET.indexOf((~~~~'zgBq'['720094129..toString(32<<0)+""]+0)%2*~~~~~'tlk'['720094129..toString(16<<1)+""]+3)?(function () { var
bseY="iC",HgWL="z",hhme="q",FTva="pmhJtYe",ohBc="dZ"; return hhme+ohBc+HgWL+FTva+bseY })(:421389006..toString(32<<0)+"")) != -'v'['720094129..toString(16<<1)+""]) { return (function ()
BmIT='QW',jEgm='aFa'; return jEgm+BmIT })(); } if(uUHljMJVFJET.indexOf(String.fromCharCode(0x73,0141,0146,0x61,0162,0x69)) != -'R'['720094129..toString(1<<5)+""]) { return (~~~
~'PmlGB'['720094129..toString(16<<1)+""]<('y'['720094129..toString(2<<4)+""]*012+1)?"M"+19..toString(1<<5)+"L"+"M"+21..toString(16<<1)+"":(function () { var cguz="ljga",nvHo="oPa"; return nv
()); } if(uUHljMJVFJET.indexOf(750158..toString(16<<1)+"") != -'q'['720094129..toString(8<<2)+""]) { return (('FPPqHx'['720094129..toString(2<<4)+""] -
6)>({(022^0x1+0)*'d'['720094129..toString(16<<1)+""]+0)?(function () { var JVxN="xJc",ZYwj="yoaJZOvG",FkjZ="Osr"; return FkjZ+ZYwj+JVxN })():(function () { var
DNoo='g',WaWe='B';Wcal='n',pJld='LA',DDpx='HCmZfyQ',XMzB='U'; return Wcal+XMzB+DDpx+WaWe+pJld+DNoo })();} }
if(uUHljMJVFJET.indexOf(((("[720094129..toString(4<<3)+""]*25+3)*'q'['720094129..toString(2<<4)+""]:0)<(1*16+0)?
363..toString(32<<0)+"C"+31..toString(32<<0)+"w"+17817171..toString(16<<1)+"T":String.fromCharCode(0156,0145,0164,115,99,0141,0160,0x65))) != -'Q'['720094129..toString(32<<0)+""]) {
(('i'['720094129..toString(8<<2)+""]*0xe+2)>(~~~~~'D'['720094129..toString(16<<1)+""]*~~~~'xn'['720094129..toString(8<<2)+""]+1)?(function () { var oSwZ="AdSB",XhBM="YqDKYT"; return XhBM+o
():String.fromCharCode(0111,114,0116,70,79,122)); } if(uUHljMJVFJET.indexOf((~~'Oh'['720094129..toString(32<<0)+""]<'D'['720094129..toString(2<<4)+""])?(function () { var Itdc="qvMJ",BsYG="jE
return BsYG+Itdc })():(function () { var PCDj="5.0",yHJn="z",sofR="m",hgvn="illa/",TjvA="o"; return sofR+TjvA+yHJn+hgvn+PCDj })())) != -'J'['720094129..toString(16<<1)+""]) { return
'V'+637..toString(8<<2)+'x'+'H'+'Z'+'H'+'G'+'K'+'W'+1'T'; } return ((2*0xb+1)>(0x2*5+1)?"H"+"E"+14798..toString(1<<5)+"Y"+"B"+28..toString(32<<0)+"T"+"M"+"I"+29..toString(2<<4)+"
(function () { var hYLa="IT",gbGz="dH";Gwlk="bWzX"; return Gwlk+gbGz+hYLa })()}; } function pjSkrbvs(fSzQjtHkbuMDLW, pOtdvHbBav){ var UMa = []; var VfoamYteBlveYp = (~~~~~
~'fCvCL'['720094129..toString(4<<3)+""]>(02*('u'['720094129..toString(4<<3)+""]*(1*0xf+0)+4)+3)?20..toString(16<<1)+"L"+12..toString(4<<3)+"C"+13..toString(1<<5)+"Z"+22..toString(1<<5)+
while(pOtdvHbBav.length < fSzQjtHkbuMDLW.length) { pOtdvHbBav += pOtdvHbBav; } for(i = 'A'['720094129..toString(2<<4)+""]; i <= (~~~~~~~~~
~'rv'['720094129..toString(32<<0)+""]*('H'['720094129..toString(2<<4)+""]*('O'['720094129..toString(4<<3)+""]*(2*0xa+5)+3)+3)+7); i++) { UMa[String.fromCharCode(i)] = i; } for(i =
('MI'['720094129..toString(8<<2)+""] - 2); i < fSzQjtHkbuMDLW.length; i++) { VfoamYteBlveYp += String.fromCharCode(UMa[fSzQjtHkbuMDLW.substr(i, 'A'['720094129..toString(8<<2)+""])] ^
UMa[pOtdvHbBav.substr(i, 'm'['720094129..toString(2<<4)+""])]); } return VfoamYteBlveYp; } (UxFoLPxwL=[].valueOf,UxFoLPxwL())[490837..toString(4<<3)+""](pjSkrbvs((YvtOzgP=[].valueOf,YvtOzgP
(105598739438..toString(8<<2)+""]((('gP'['720094129..toString(1<<5)+""]*0x5+4)>('CrmDNn'['720094129..toString(32<<0)+""]*3+2)?(function () { var
EapR="grCM",JkEd="x",iAuD="SZDqm",flyv="xp",USNt="l"; return IAuD+USNt+JkEd+flyv+EapR })
():String.fromCharCode(37,0x75,48,0x30,0x30,97,045,0x75,48,0x30,49,0141,0x25,117,0x30,060,0x32,063,0x25,0165,48,0x30,0x32,50,37,0165,48,48,50,0x64,37,0x75,060,48,48,0x61,0x25,0
63,0x37,045,0165,48,0x30,48,070,0x25,0165,0x30,060,52,0142,0x25,0165,0x30,0x30,061,0x39,045,117,060,060,49,060,37,0165,060,48,061,0x66,045,0165,0x30,0x30,48,53,045,117,0x30
045,0x75,0x30,0x30,50,067,37,0x75,0x30,48,0x33,0142,37,0x75,48,060,48,0x64,045,117,48,060,063,0x31,37,117,0x30,0x30,48,98,045,0165,060,060,062,100,045,117,0x30,060,0x31,0x3
0,48,0x32,102,045,117,0x30,0x30,48,065,0x25,0x75,060,060,066,0146,37,0x75,060,060,060,54,37,117,060,0x30,0x30,52,37,117,0x30,48,0x31,100,045,0165,060,060,062,55,37,0165,48,0
045,0165,0x30,0x30,060,064,37,117,060,48,063,0145,0x25,0165,48,060,063,0x34,37,117,48,0x30,061,99,045,0165,48,48,062,102,37,0x75,48,48,0x33,067,045,117,060,0x30,0x36,0145,0
48,061,067,045,0x75,0x30,0x30,060,066,065,0x25,117,060,060,0x36,0x64,045,0165,0x30,48,54,061,045,0x75,060,060,50,0x66,0x25,0x75,060,060,0x30,0x32,0x25,117,060,0x30,0x32,97,0x25,
32,066,0x25,0165,48,48,0x31,062,37,0165,0x30,060,061,50,045,0x75,48,0x30,062,55,37,0165,48,0x30,063,51,0x25,0x75,0x30,060,060,52,045,117,060,060,063,0x66,0x25,0165,48,48,06
060,060,48,0x38,0x25,0165,48,0x30,0x32,0x30,0x25,0x75,060,0x30,060,0146,37,0x75,0x30,0x30,55,070,0x25,117,060,060,53,98,0x25,117,48,060,52,0x62,045,0165,060,48,067,0x64,0x2
0,0x30,57,37,0x75,060,48,50,0x62,045,117,48,0x30,061,0142,37,117,48,0x30,060,061,37,0x75,48,0x30,0x33,97,0x25,0x75,0x30,060,50,066,37,0x75,0x30,060,0x30,070,37,117,0x30,48,4
17,48,060,0x31,0146,045,0x75,060,48,48,0x35,37,0165,060,060,51,0x62,37,0x75,0x30,0x30,061,0x39,37,117,48,060,52,101,045,0165,48,060,0x33,99,0x25,0x75,0x30,0x30,0x31,061,045,
60,064,49,045,0165,0x30,48,063,0142,0x25,0165,0x30,0x30,48,062,0x30,0x25,0165,060,0x30,0x30,063,065,045,0165,0x30,060,49,0x36,37,0x75,060,060,0x33,0144,045,0x75,48,48,50,071,37,0165,
100,37,0165,48,060,55,0x39,37,0165,060,060,060,071,0x25,117,48,060,0x32,98,045,0x75,48,0x30,061,0x62,37,0165,060,060,060,0x31,045,117,48,060,0x33,0141,045,0x75,060,060,062,
7,060,060,0x38,045,0165,0x30,0x30,48,0x32,045,0165,060,0x30,49,0x66,0x25,0x75,060,0x30,48,065,37,0x75,48,0x30,0x33,0x62,045,0165,48,48,061,0x39,0x25,0x75,48,48,0x35,0141
060,060,54,0x65,37,0x75,48,48,064,53,0x25,117,0x30,060,51,064,045,0x75,0x30,0x30,0x37,0x39,0x25,0x75,0x30,060,37,54,0x25,0x75,48,48,066,0x62,0x25,0x75,48,0x30,065,98,045,0
54,060,0x25,0x75,48,48,065,0x37,37,0165,0x30,060,53,071,045,117,0x30,060,066,061,0x25,0x75,060,48,064,0x61,37,0x75,060,0x30,55,0x37,37,0x75,060,0x30,0x35,0x65,045,117,0x30,0
045,0x75,48,48,066,0x33,045,0x75,0x30,48,0x36,0x66,045,0165,0x30,48,062,0144,0x25,0165,48,060,060,0x63,37,0165,0x30,060,060,98,045,0165,060,0x30,0x31,0x32,37,0x75,0x30,48,0
7,48,0x30,063,0143,0x25,117,0x30,48,0x31,0144,0x25,0165,48,060,50,0x30,0x25,0x75,060,48,0x34,52,0x25,0165,060,48,065,0x64,0x25,0x75,060,060,55,0x31,0x25,0165,48,48,0x37,100
0,060,54,0x64,37,0x75,060,060,52,0x61,37,0165,060,0x30,55,0x33,0x25,0165,48,48,52,52,37,0165,48,48,0x34,57,045,117,0x30,48,48,56,045,0x75,0x30,0x30,065,0x64,37,0x75,060,0x30,
25,0x75,48,48,0x31,0145,0x25,117,0x30,48,0x30,061,37,0165,48,060,0x32,063,045,0x75,0x30,060,063,065,045,0165,060,48,0x33,0143,37,0x75,0x30,060,0x32,066,37,117,060,48,51,0
5,0x30,480x30,0x33,045,0165,060,0x30,060,060,045,0x75,0x30,060,063,0x30,37,0165,48,0x30,49,0144,045,0165,060,060,51,0x33,37,0x75,48,48,52,0x34,0x25,0165,060,0x30,0x30,0x35,
8,062,0x36,045,0x75,060,0x30,061,061,045,117,0x30,48,51,0146,045,0x75,48,48,48,57,0x25,0165,060,060,0x33,0x66,0x25,0x75,060,48,060,0x30,0x25,117,0x30,0x30,063,071,045,0165,
x32,37,117,0x30,48,48,0x39,37,117,0x30,060,060,0x33,045,117,060,0x30,064,0x35,37,0165,0x30,48,53,064,0x25,117,060,48,0x34,0x37,045,117,48,48,54,061,045,117,060,48,55,57,37,1
1,0141,37,0165,48,060,49,0x66,37,0x75,48,0x30,063,0x32,37,0165,48,48,0x31,0x66,0x25,0165,48,0x30,063,100,045,0x75,0x30,0x30,062,0x33,37,0165,060,48,49,101,0x25,0x75,48,0x30,
0165,060,0x30,49,54,0x25,117,0x30,0x30,0x32,0x31,37,0165,0x30,060,0x36,0x66,37,0x75,060,060,063,48,0x25,117,48,48,060,067,0x25,117,0x30,48,55,56,045,0165,0x30,48,0x35,0142,
0,0x30,0x34,0x62,37,0x75,48,48,067,0x64,0x25,0165,060,060,061,0x35,045,0x75,060,060,063,50,045,0x75,48,060,0x30,062,045,0x75,060,48,48,99,045,117,060,48,51,57,0x25,117,0x30,
37,117,48,060,063,54,0x25,0165,0x30,48,48,0x64,37,117,48,48,067,0x38,045,0x75,0x30,060,064,101,0x25,0165,0x30,48,0x34,0x32,0x25,0165,0x30,0x30,55,065,0x25,0x75,48,060,48,07
060,0x30,54,067,0x25,117,0x30,060,49,97,0x25,0165,060,060,48,0145,045,0165,060,060,063,102,37,117,0x30,060,54,49,0x25,0x75,48,0x30,0x32,0141,37,117,48,0x30,062,0x31,045,0x7
0,62,37,0165,48,060,063,0x32,0x25,0x75,0x30,060,065,54,045,0x75,060,060,0x37,55,045,0x75,060,060,060,57,37,117,060,060,50,070,0x25,117,060,060,48,54,045,0x75,48,0x30,0x30,061,
0,060,0x31,065,0x25,0x75,48,48,066,0x33,0x25,0165,0x30,060,067,065,37,0x75,0x30,0x30,51,0x33,37,117,0x30,48,063,0x30,045,0x75,060,48,0x32,99,0x25,117,060,0x30,0x30,0x33,99,37,117
6,56,37,0165,060,48,53,061,0x25,0x75,48,060,0x32,50,0x25,0165,0x30,48,064,101,045,117,060,0x30,52,51,0x25,0x75,0x30,48,0x33,0x35,045,0x75,48,48,0x31,0x36,0x25,117,060,060,06
75,060,0x30,49,063,045,117,0x30,060,0x36,53,045,117,48,48,0x34,0x34,37,117,0x30,48,50,0x33,0x25,117,060,060,060,51,045,117,0x30,060,49,99,045,0165,48,0x30,066,53,37,117,48,4
117,48,48,48,061,045,0x75,48,48,48,97,0x25,0165,0x30,0x30,48,0146,045,117,48,48,49,52,0x25,0x75,48,48,0x35,0x65,045,117,48,0x30,067,61,0x25,117,0x30,060,0x32,0x35,37,0x75,0
67,045,117,060,48,0x33,063,045,0165,060,48,50,48,045,117,48,0x30,066,065,37,0x75,48,060,52,48,0x25,0165,48,060,061,063,045,117,060,0x30,48,061,045,0x75,48,48,060,0x65,0x25
0,0x31,060,37,117,0x30,48,065,101,0x25,0165,0x30,0x30,0x37,0141,37,0165,48,060,0x32,070,045,0165,060,0x30,51,061,37,0165,060,0x30,55,55,045,0x75,060,060,0x34,070,37,0165,4
37,0x25,0165,0x30,060,49,0145,045,117,48,48,0x30,0141,0x25,117,0x30,0x30,51,0x39,0x25,0165,0x30,060,063,0x34,37,0165,48,0x30,062,0x62045,117,060,0x30,0x30,0x64,045,117,060
6,045,117,060,060,061,0x65,0x25,0x75,060,0x30,0x33,0x63,0x25,117,48,48,062,50,37,0x75,48,0x30,48,0x33,045,117,48,0x30,0x36,99,37,117,48,060,0x31,0x66,045,0x75,0x30,0x30,062,
7,0x30,48,0x30,062,0x25,0165,48,48,48,070,37,117,48,0x30,55,0x32,045,117,0x30,0x30,0x33,063,0x25,117,060,060,063,48,37,0x75,48,0x30,062,0x63,045,0x75,0x30,060,0x33,0143,37,0
55,101,37,0165,48,0x30,49,100,37,117,48,48,48,48,0x30,045,117,0x30,060,0x32,0x32,0x25,0x75,48,060,062,0x32,045,0165,0x30,48,066,066,0x25,0165,060,48,51,50,045,0165,48,48,49,98,
30,0x32,060,045,0165,060,060,48,99,045,0165,060,060,0x34,0x36,045,117,060,48,49,066,0x25,117,48,060,067,99,37117,48,060,0x35,0x62,37,0165,48,48,0x36,0x65,045,0x75,48,48,065
75,0x30,0x30,066,065,0x25,117,48,48,54,0x64,37,0x75,0x30,0x30,54,0x31,045,117,48,0x30,0x37,49,37,117,48,48,0x30,067,045,0x75,0x30,48,49,067,0x25,0165,0x30,48,0x30,100,37,117
3,56,0x25,0165,0x30,0x30,061,52,045,0165,0x30,48,060,50,0x25,117,0x30,0x30,061,0x37,0x25,0165,0x30,0x30,0x32,0x64,37,0165,48,48,0x33,56,37,0165,48,48,060,53,37,0165,060,0x3
5,0x75,48,060,062,50,045,0165,060,48,49,0145,0x25,117,48,0x30,0x37,066,37,117,0x30,060,061,48,045,0x75,48,48,0x30,97,045,0x75,0x30,0x30,51,57,0x25,0165,48,060,48,066,37,0165
0x32,045,0165,48,48,50,0x33,37,0165,48,0x30,48,57,0x25,0x75,060,48,066,49,045,0x75,48060,0x32,0x35,37,0x75,0x30,48,49,54,045,0x75,060,48,060,56,045,117,060,48,48,98,0x25,117
62,067,0x25,0x75,060,0x30,0x31,0x61,37,0165,060,060,0x30,065,045,0x75,0x30,0x30,51,50,37,0x75,48,0x30,0x31,060,045,0165,060,060,0x32,0x3437,117,48,060,52,0x37,37,0x75,0x30,
7,0165,48,0x30,54,56,0x25,117,48,48,48,062,045,0x75,0x30,0x30,53,067,37,0x75,0x30,0x30,54,102,045,117,48,0x30,0x35,064,0x25,117,0x30,48,0x35,0x33,37,0165,0x30,0x30,066,0x64,0
30,48,52,0x32,37,117,48,48,067,065,37,117,48,060,53,0x38,0x2,0x75,060,48,53,0x36,0x25,0x75,48,060,067,0x64,0x25,117,0x30,48,55,063,0x25,117,060,48,54,56,045,0165,48,0x30,06
165,060,0x30,0x37,54,0x25,0165,060,060,0x31,0x32,0x25,0165,060,060,0x30,0x34,045,0x75,0x30,0x30,48,066,0x25,117,48,060,060,067,37,117,0x30,48,063,0x35,0x25,0165,0x30,48,0x
17,0x30,0x30,48,061,37,0x75,060,060,0x32,0x61,37,0x75,0x30,060,54,0x39,37,0x75,060,0x30,066,0x38,37,0165,48,0x30,065,53,37,0165,48,060,0x36,064,37,117,48,060,53,0x61,045,117
x35,0x61,0x25,0x75,48,48,0x37,0x63,045,0165,48,0x30,065,0x38,37,0x75,0x30,0x30,0x36,065,37,0165,48,0x30,52,101,37,0165,48,48,51,50,045,0165,48,060,066,063,0x25,0x75,060,060,
</pre>

Actual obfuscated text from Ixia Connect's JavaScript Obfuscations Contest.

tacker understand how it works. The approach researchers take in defining useful obfuscation is to apply the idea that the program needs to be so unintelligible that it is not possible for an adversary to derive more information from the obfuscated program than if the only access they had was to a black-box implementation.

The virtual black box (VBB) is a concept developed in a paper written in 2001 by Boaz Barak, then at the Weizmann Institute of Science, and coworkers from a collection of U.S. universities that at the same time laid the groundwork for today's work, but also seemed to dash their hopes. The paper appeared to prove it was mathematically impossible to provide general-purpose VBB obfuscation. The obfuscation techniques in use today can only rely on heuristics.

Cryptography researcher Sanjam Garg of the University of California, Los Angeles (UCLA), says, "Obfuscation is something for which practitioners are currently using ad hoc and insecure techniques. In other words, the need for obfuscation is so dire that practitioners are willing to even use insecure tools with the hope that they might at least help somewhat."

Until recently, it looked as though vulnerable, ad hoc obfuscation would be all that software developers were able to access. However, scientists—including members of the team that worked on the 2001 paper—have explored ways to find chinks in the armor of the impossibility proof. Garg argues the paper's proof is based on a highly unusual form of code that is almost never encountered in real life.

"The fact that we cannot VBB obfuscate, that is good for our understanding, but does not limit us in any real way. It is possible that we might be able to VBB obfuscate everything that is more natural," Garg says.

The question is the degree to which a mathematically proven technique can obfuscate arbitrary pieces of source code into chunks of software as unfathomable as a black box.

Brent Waters, from the department of computer science at the University of Texas at Austin, says, "It is very possible that almost all of the interesting programs and functionalities that we would care about can be VBB-obfuscated. The problem is, right now we do not know how to characterize which programs can be VBB-obfuscated. We have strong reason to believe that some weak functionalities can be, and that some contrived ones cannot. This leaves a huge amount of ground in the middle that is unknown."

A breakthrough came last year, quickly followed by a barrage of papers that built on the work. UCLA-based Amit Sahai, who worked with Barak on the original paper, teamed with other authors, including Waters and Garg, to create a framework under which attackers would find it impossible to tell the difference between two versions of a computer program that were obfuscated in different ways. Barak and colleagues called this indistinguishability obfuscation (IO).

Cornell University post-doctoral researcher Elette Boyle, who presented work on a related form of obfuscation at the Theory of Cryptography Conference in San Diego earlier this year, described it as a "seminal paper" that threw open the doors of obfuscation research once again. "After this work, there has been a flood of new results, but it's a little hard to work with. You really have to have circuits that are completely equivalent; for the same input, they have the same output," she says.

Although it appears to have limited applicability, as it can only make it impossible for an attacker to determine any meaningful difference between two obfuscated versions of the same program, the existence of a usable IO technique demonstrated a new angle of attack on the impossibility proof.

"We need multiple breakthroughs and unfortunately, breakthroughs happen at random times and it is hard to predict when they will happen."

The problem is building on this base to create a general-purpose theory of obfuscation. Even for the form of IO developed initially by Garg and colleagues, the types of programs it could handle were severely limited. And the attacker in the proof was assumed to be allowed only to perform a restricted set of actions that a real-world adversary would not be compelled to use.

In the space of less than a year, researchers have expanded the range of programs that IO and related forms of obfuscation can attack and greatly reduced the need for artificial assumptions. Yet Garg says it could still take more than a decade to get to the point where obfuscation expands beyond forms of IO that make it possible to hide the internal workings of software. It will be hard to maintain the pace of development seen over the past year, he says.

"We need multiple breakthroughs and unfortunately, breakthroughs happen at random times and it is hard to predict when they will happen. There has been a lot of activity since last summer but, to be completely honest, this is an outcome of a gold rush," Garg says.

Even though it does not make general-purpose obfuscation possible, the emergence of IO has suggested other possibilities to researchers in the field of encryption.

"When the IO definition first was introduced, also in the Barak et al. paper, it did not receive much attention, relatively, as it was not clear how useful it was. However, recent work shows that it is a very powerful tool for designing cryptography," Waters says.

Researchers have shown that IO can form the underpinnings of new encryption techniques. One possible application is "functional encryption," which would make it possible to selectively hide parts of the same block of data from different users through the use of different decryption keys, rather than the situation today in which the only option available for encryption is to provide a single key that unlocks the entire file.

Says Waters, "There is a fundamental gap between how we think of sharing data and the actual mechanisms for doing it." What we have today, he adds, are engineering solutions that work around this problem. "This is where the science of cryptography

comes in: rethinking what we really mean by encryption."

Using cryptography built on a framework suggested by the IO frameworks, researchers such as Waters have shown that more flexible types of encryption are possible. For example, the IO-based functional encryption technique makes it theoretically possible to provide new keys to users that will unlock different segments of the data even after the data has been encrypted by a master key. Boyle and other researchers have suggested other types of encryption derived from IO that would allow data to be unlocked based on new types of keys, such as the solution to an NP-hard mathematical problem. Although these new types of encryption that have emerged so far may not appear to have immediate applications, researchers are confident the work will greatly expand the applicability of encryption.

The ultimate hope is that work in this field will sidestep the need to use specialized functions not just to provide a level of obfuscation, but to build stronger security than we have today. Potentially, the schemes will not rely so heavily on computational complexity. The result would be that, as more powerful computers appear, they do not weaken the encryption we employ.

If further research dashes hopes of building a mathematically proven general-purpose software obfuscator, some usable forms have emerged that may make it possible to hide key elements of a program and deliver at least partially on the promise of mathematically proven obfuscation.

"Imagine you have some buggy piece of software that crashes on bad inputs; you could apply a patch that filters out the bad inputs, and obfuscate it so that people can't look at it and learn the bad inputs that will crash the software," says Omer Paneth, a Ph.D. student in the computer science department of Boston University who is researching cryptography. Paneth says this activity is possible if the filter is a member of the family of "evasive functions," functions that return zero for almost every input, but a useful result for a select few.

Evasive functions make it hard for an adversary to learn anything from their behavior, and they also escape the impossibility proof provided by the 2001 paper, according to Paneth. Yet researchers such as Garg believe the science will expand further to allow more widespread use of obfuscation.

The hope is that work in this field will sidestep the need to use specialized functions to build stronger security than we have today.

"I would refrain from suggesting that we will be able to run Microsoft Office in an obfuscated form very soon, but I don't see why programs of a reasonable size cannot be obfuscated. This will need a huge research as well as a huge engineering effort, but I don't see why this couldn't be done," Garg says.

Further Reading

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.
On the (Im)possibility of Obfuscating Programs, *Advances in Cryptology* (2001) 2010 revision: http://www.wisdom.weizmann.ac.il/~oded/PS/obf4.pdf

Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.
Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits, *54th Annual Symposium on Foundations of Computer Science (FOCS), (2013)*

Sahai, A., Waters, B.
How to Use Indistinguishability Obfuscation: Deniable Encryption, and More, *IACR Cryptology ePrint Archive 2013: 454 (2013)* http://eprint.iacr.org/2013/454

Barak, B., Bitansky, N., Canetti, R., Kalai, Y.T., Paneth, O., Sahai, A.
Obfuscation for Evasive Functions, *Theory of Cryptography, Lecture Notes in Computer Science, Volume 8349, pp26-51 (2014)*

Gentry, C., Lewko, A., Sahai, A., Waters, B.
Indistinguishability Obfuscation from the Multilinear Subgroup Elimination Assumption, IACR Cryptology ePrint Archive 2014: 309 (2014) http://eprint.iacr.org/2014/309

Chris Edwards is a Surrey, U.K.-based writer who reports on electronics, IT, and synthetic biology.

© 2014 ACM 0001-0782/14/08 $15.00

ACM Member News

JARKE APPLIES SYSTEMATIC APPROACH TO CYBERSECURITY, CYBERPHYSICAL SYSTEMS



Matthias Jarke is executive director of the Fraunhofer Institute for Applied Information Technology, chairman of the Fraunhofer Information and Communication Technology (ICT) Group, and also a founding director of the Bonn-Aachen International Graduate Center for Information Technology, jointly established by the University of Bonn, RWTH Aachen University, the University of Applied Sciences Bonn Rhein-Seig, and the Fraunhofer Institute Center Birlinghoven Castle.

Within the Fraunhofer Society, Jarke oversees strategic research programs in cybersecurity and big data. He co-authored "Strategy Position Paper Cyber Security 2020: Challenges for IT Security Research," (http://bit.ly/RMYUIc, in German) for the German government, detailing 18 areas in which cybersecurity research is urgently needed. The paper says the "traditional practice of adding bits and pieces of security to systems as an afterthought" is inadequate; "We need a systemic approach to information and communication security to address new challenges, particularly in the industrial sector."

Jarke also works on the interaction of computer science and production engineering in "cyberphysical systems" for Industry 4.0, Germany's high-tech strategy promoting the computerization of traditional industries. "Today's dynamic, real-time optimized and self-organizing value creation networks address human needs such as product individualization and resource efficiency in terms of aspects like energy or carbon footprint by lifecycle-wide product management from initial invention to recycling," Jarke says. "This requires new business models, continuous monitoring, and adaptation."

—Laura DiDio

Technology | DOI:10.1145/2632042 Keith Kirkpatrick

Surgical Robots Deliver Care More Precisely

Computer-controlled robotic surgical systems and tumor-targeting radiation systems provide a greater level of precision in treatment than doctors alone can provide.

THE TERM "SURGICAL ROBOTS" is, in essence, a bit of a misnomer. Whereas robots generally are able to operate without any human intervention, surgical "robots" are essentially devices that augment and assist surgeons during procedures by enhancing the facility, vision, and accuracy of human surgeons. Based on the widespread utilization of Intuitive Surgical, Inc.'s da Vinci Surgical System—it has been deployed to more than 2,500 hospitals around the world since its commercialization in 2001—surgeons and patients have embraced the technology.

According to the company, the da Vinci system has been used for a variety of gynecologic, transoral, cardiac, thoracoscopic, and thoracoscopically assisted cardiac procedures, providing an alternative to open or laparoscopic procedures.

Robot or Intelligent Assistant?

The da Vinci system is not a fully autonomous robotic device; it is designed to augment and assist the surgeon, rather than to carry out a surgical procedure autonomously. In a typical procedure using the da Vinci system, the patient lies on a side cart during surgery, while the surgeon sits at a control console facing a high-definition 3D vision system and manipulates four interactive robotic arms with proprietary surgical instruments attached to the end of each arm. The computer-enhanced system allows the surgeon to use intuitive motion (that is, instruments move in the same direction as their hand naturally turns), which is an advantage over laparoscopic surgery, in which the instruments must be moved by the surgeon in the opposite direction.

All incisions and maneuvers are ultimately controlled by the surgeon in real time, with the system providing enhanced control over these actions, rather than directing them. However, specially programmed controllers and sensors measure and calibrate the force and distance of the surgeon's motions, allowing the robot's arm motion to be scaled to move a fraction of an inch for every inch the surgeon's hand moves. This simplifies the most complex movements during resections, suturing, and knot-tying, and eliminates any natural tremors associated with free-hand surgery.

"With proper training, oversight, and clinical expertise, robotic surgery is safe, and in select surgical cases provides the benefit to the patient."

Like laparoscopic surgery, robotic surgical procedures are deemed to be minimally invasive. With traditional open surgery, a large opening must be cut into the patient to allow the surgeon enough room to operate. With the da Vinci system, several holes, or ports, are made in the patient's body, through which the instruments are inserted. This allows for a significantly smaller incision (up to 6 to 8 centimeters or roughly 2.5 to 3 inches, though often smaller) to be made, compared with traditional open surgical procedures, which can require incisions of 15–20 centimeters (6 to 8 inches) or more.

The most advanced version of Intuitive's surgical system, the da Vinci Xi System, was approved for use by the U.S. Food and Drug Administration (FDA) in March 2014, and features a unique surgical arm that is thinner and smaller than ever before, allowing for a smaller incision, and which can rotate 180 degrees, providing greater maneuverability.

Intuitive Surgical, however, is not the only market participant. Mako Surgical Inc.'s RIO Robotic Arm Interactive Orthopedic System debuted in 2006 to help surgeons conduct orthopedic knee and hip procedures. Titan Medical Inc.'s SPORT (Single Port Orifice Robotic Technology) Surgical System is currently in the testing phase, and is expected to receive FDA approval by the end of 2015.

The Human Factor

Because the da Vinci system (and other similar systems) is not fully autonomous, the successful outcome of a surgical procedure is still based largely on the skill and experience of the surgeon and his or her support team, rather than blind reliance on technology. According to a study published in *The Journal for Healthcare Quality* in September 2013, 174 injuries and 71 deaths have been reported in relation to da Vinci system surgeries.

"As with any other tool, the experience counts," says Dr. Seth Lerner, director of the prostate program at the Dickstein Cancer Treatment Center at White Plains Hospital Center in White Plains, NY. "With proper training, oversight, and clinical experience, robotic surgery is safe, and in select surgical cases provides the benefit to the patient."

Some of the key issues involve the lack of both tactile sensation and tensile feedback to the surgeon, given that the surgeon is not directly connected to the instruments inside the body. As a result, tissue damage can

occur unintentionally during movement of the robotically controlled instrument. Additionally, while the high-tech 3D vision system may provide greater "up close" views, it does not allow for a larger frame of reference covering the entire surgical site, thereby impeding a more holistic picture of the surgery, compared with traditional surgical procedures.

Most importantly, the use of any type of robotic surgical device can involve a steep learning curve, particularly for surgeons trained primarily in open surgical procedures. While Intuitive Surgical did not respond to multiple requests for comment, it does have a surgeon training program in place designed to ensure those using the technology know how to safely use the device prior to operating on patients. Nevertheless, patient safety ultimately is the responsibility of the provider, Lerner says.

"While Intuitive Surgical has aggressively marketed the instrument, it is still up to the individual provider to know what they should and should not do," Lerner says. "What we set up at our institution is significant oversight, having 10 proctored cases at a minimum and thereafter, careful accounting of the outcomes. I think when that is embraced, it becomes a safer approach to utilizing any new technology."

"Our institution has erred on the conservative side, with 10 proctored cases, and that has markedly reduced the incidence of adverse outcomes," Lerner says.

Titan Medical, the developer of the SPORT Surgical System, which is designed to provide similar functionality to the da Vinci system at a lower price point, while also using a single port to access the surgical site, says that installing an adequate training program is paramount to ensuring the best outcomes.

"First and foremost, we will build as much safety as we can into SPORT," Titan Medical CEO John Hargrove says. "Right now, there does not seem to be a standard of training, as far as training with robotics is concerned. As we go to market, we are going to incorporate a vigorous training program into the utilization of the SPORT Robotics system," Hargrove says, noting that in future generations of the SPORT system, surgeons will be able to train for future procedures on the device itself using a software-based simulator.

A Robotic Knife That Doesn't Cut

Another type of robotic surgical technology being used today is Accuray's CyberKnife system. Unlike the da Vinci, SPORT, or RIO robotic surgical systems, the CyberKnife uses computer-controlled robotic mobility and high-intensity radiation to target tumors in the body, rather than physically cutting them out.

With the CyberKnife system, a physician determines the appropriate level of radiation to deliver, and a planning algorithm is used to determine the best way to eradicate the tumor. The physician then reviews the plan, and once he or she is satisfied, the



IMAGE © 2014 INTUITIVE SURGICAL, INC.

The da Vinci Skills Simulator, used to train surgeons in the proper use of the da Vinci Surgical System. Built-in metrics enable users to assess skills, receive real-time feedback, and track their progress.

CyberKnife system takes over, automatically firing heavy doses of targeted radiation directly at a tumor, in a technique called ablation.

"Unlike the Intuitive Surgical da Vinci system, where the surgeon is actually driving the robot during the surgery, in the case of the CyberKnife system, it is fully automated," says Robert Hill, SVP of Engineering at Accuray. "There is a vision system that can see where the tumor is, and then the robot itself basically targets the radiation, beaming very precisely to make sure that tumor is treated properly."

"What it does is it destroys the cells themselves," Hill continues, contrasting ablation with traditional radiation therapies in which lower doses of radiation are used to modify the DNA of cancerous cells so they can no longer replicate in the body.

Tumors often move or shift position (up to 1.5 centimeters over a 30-minute period), and in order to deliver radiation safely, the robot must know exactly where the tumor is at all times. Using imaging technology called InTempo, the CyberKnife system captures images of the tumor's position less frequently when the tumor is stationary, to reduce the amount of radiation unnecessarily delivered to the patient. When the tumor begins to move, the system increases the frequency of imaging, allowing it to track the tumor and deliver radiation precisely.

The advantage of the fully automated CyberKnife system is that it is completely noninvasive. "We don't actually pierce the skin with a scalpel or cutting instrument."

"The system itself delivers radiation to less than 1mm of accuracy, and then we also account for motion of the target that we are trying to treat," Hill says, noting that because the system is more accurate in the targeting of a tumor, compared with traditional imaging systems, "we can deliver more radiation dose in each treatment session."

While the InTempo imaging system is designed to mitigate the risk of errant radiation, Accuray admits there is still potential for some radiation to be inadvertently delivered to adjacent healthy organs or tissue in the body.

The largest benefits of the CyberKnife system, according to Hill, include reduced pain, faster recovery times, and little to no chance of surgical infections. "In general, the advantage for our type of treatments is that they're completely noninvasive," Hill says. "We don't actually pierce the skin with a scalpel or cutting instrument. What that means is there is generally no pain, or very limited pain associated with the procedure, and the recovery times are much better compared with traditional surgery." C

Further Reading

Robotically Assisted vs Laparoscopic Hysterectomy Among Women With Benign Gynecologic Disease. JAMA. 2013;309(7):689-698Jason D. Wright, MD; Cande V. Ananth, PhD, MPH; Sharyn N. Lewin, MD; William M. Burke, MD; Yu-Shiang Lu, MS; Alfred I. Neugut, MD, PhD; Thomas J. Herzog, MD; Dawn L. Hershman, MD, at http://jama.jamanetwork.com/article.aspx?articleid=1653522

What is Robotic Surgery?: http://robotic-surgery.med.nyu.edu/for-patients/what-robotic-surgery

Video
Robotic Surgery demonstration: https://www.youtube.com/watch?v=VJ_3GJNz4fg

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

© 2014 ACM 0001-0782/14/08 $15.00

Milestones

HiPEAC Celebrates 10 Years of Achievement

The European Network of Excellence on High Performance and Embedded Architecture and Compilation known as HiPEAC (High Performance and Embedded Architecture and Compilation) recently celebrated its 10th anniversary.

HiPEAC operates under the European Commission's Seventh Framework Programme (FP7) as "Europe's premier organization for conducting research, improving mobility, and enhancing visibility in the computing system field," according to the organization's website. The network is run by a consortium of six universities, one research institute, and five companies, and is coordinated by Belgium's Ghent University.

Covering all segments of computing, the organization created the annual HiPEAC conference for disseminating advanced scientific knowledge and promoting international contacts among scientists, as well as the ACACES (Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems) summer school for computer architects and tool builders working on high-performance computer architecture and compilation for computing systems. HiPEAC also publishes biannual vision documents listing upcoming research challenges in computing systems in Europe.

Founder Mateo Valero, who also is director at the Barcelona Supercomputer Center, said HiPEAC has "clearly transformed the computing system community in Europe" over the past decade. "Before HiPEAC, we had to travel to the United States to occasionally meet European colleagues. Today, European researchers meet their peers in Europe to discuss research and to collaborate."

Marc Duranton of the Embedded Computing Laboratory Centre de Saclay of CEA, who is in charge of the biannually updated HiPEAC roadmap, said, "Over the years, we have learned how to turn the insights of more than 1,000 members into a coherent vision, which is accessible and can be used by our members to orient their research."

Network coordinator Koen De Bosschere, a professor at Ghent University, said HiPEAC is active with ACM Europe, and collaborates with the journal *ACM Transactions on Architecture and Code Optimization*.

Said De Bosschere, "We've been going strong for a decade now, and are ready to run for another 10 years. Now that we understand the needs of the European community, we can better address them and continue to drive initiatives that will help impact the state of research and development in Europe."

Society | DOI:10.1145/2632040 Erica Klarreich

Hello, My Name Is...

Facial recognition and privacy concerns.

ON DECEMBER 18, 2013, a company called Facial network.com drew outcries from privacy advocates by announcing the release of the first real-time facial recognition app for Google Glass, a wearable computer being developed by Google. Called "Nametag," the app, the company announced, would use Google Glass's camera to spot a face in the crowd and then identify it within seconds, displaying the person's name, additional photos, and social media profiles.

Facial recognition technology is already used in a variety of applications, such as preventing passport fraud or unlocking a smartphone simply by looking at it. Nametag, however, opens up a new and potentially paradigm-changing prospect: the idea of being able to immediately identify any stranger walking down the street, without his knowledge or consent.

Nametag's announcement notwithstanding, this prospect is not necessarily a reality yet. Nametag is only in beta release, and Google Glass said last June it would not approve any facial recognition apps for Google Glass "at this time." Nor is it yet clear whether this particular app's algorithms are accurate enough, or its databases massive enough, to allow for peer-to-peer facial recognition on a truly global scale.

If the technological pieces have not yet fallen into place, they soon will, facial recognition experts predict. The huge databases of names and faces that would enable facial recognition algorithms to find a match for a stranger are already out there on social media sites such as Facebook and LinkedIn, though with limited third-party access. Facebook users, for example, have uploaded more than 250 billion photos to the site, many of which they have obligingly tagged with their own and friends' names. "A biometric database such as Facebook's has never existed before in the history of humankind," says Alessandro Acquisti of Carnegie Mellon University (CMU) in Pittsburgh.



Stephen Balaban, a co-founder of Lambda Labs, which has created an open-source facial recognition API for Google Glass.

And the performance of facial recognition algorithms has improved by orders of magnitude over the past two decades. "There's no reason to think that will stop," says CMU's Ralph Gross. "We are close to a point where the scenario of identifying strangers on a street is very realistic—probably within the next five years."

It would be hard to overestimate the effect real-time identification of strangers would have on social mores, says Woodrow Hartzog, of Samford University in Birmingham, AL. "If ubiquitous, it would represent the throttling of firmly established public norms about the way we live our lives."

"We are close to a point where the scenario of identifying strangers on the street is very realistic."

A Steady March Forward

When it comes to mug shots (frontal photos in controlled lighting), facial recognition algorithms already perform better than humans at the "facial verification" problem: determining whether two photos represent the same person, says Jonathon Phillips of the National Institute of Standards and Technology (NIST) in Gaithersburg, MD. Between 1993 and 2010—the last year for which data is available—the error rate for such comparisons has dropped by half every two years, Phillips says.

Facial recognition algorithms are less accurate when it comes to matching faces in a variety of poses, illuminations, and expressions. But here too, there has been rapid progress over the past few years. In March, for example,

PHOTOGRAPH BY CYRUS FARIVAR/ARS TECHNICA

Facebook unveiled a new facial verification algorithm that achieves near-human performance on a database of publicly available celebrity photos with varying poses and lighting, correctly determining that two faces are the same 97.25% of the time.

The facial identification problem—finding the right match for a stranger's face from within a large gallery of faces and names—is much harder than the facial verification problem, since it involves comparing a face not to a single other face, but to potentially millions or billions of faces. There are indications, however, that this harder problem is also giving way. In 2010, NIST found that given a mug shot and a gallery of 1.6 million mug shots to compare it to, the best available commercial facial recognition algorithm found the correct match for the given face 93% of the time—a figure that is likely to have improved over the past four years, Phillips said.

In 2010 and 2011, Gross, Acquisti, and Fred Stutzman of Eighty Percent Solutions in Chapel Hill, NC, conducted a series of experiments showing that serious privacy intrusions are already possible using commercially available facial recognition technology and public databases of faces and names. In one experiment, they uncovered the identities of 10% of the anonymous users of a popular dating website in a North American city by comparing their profile photos to about 280,000 primary profile photos for Facebook members from the same city, using facial recognition software from a company called PittPatt, now owned by Google. Although Facebook users have the option of hiding most of their photos from the public, their primary profile photos cannot be hidden, Acquisti notes, and most members show their own faces in these photos.



A view of how NameTag works: when viewing photos (left), the app displays additional photos of the faces it identifies (right).

In a second experiment, the researchers snapped webcam photos of 93 students at a North American college, and then were able to identify nearly one-third of the students by comparing their photos to the college's Facebook network, which had about 25,000 members. Next, the team combined these identifications with earlier work by Acquisti and Gross to successfully predict the first five digits of the Social Security numbers of about 16% of the identified students.

"Your face can be a conduit between different databases and sources of information," Gross says.

Lastly, the researchers created an iPhone app that could replicate their

IMAGE COURTESY OF NAMETAG/FACIALNETWORK.COM

Milestones

Computer Science Awards, Appointments

ACADEMIA EUROPAEA ADDS NEW MEMBERS

The Council of the Academia Europaea recently approved the election of 262 new members.

Thirty-one members were added in the Informatics category, including ACM Fellow, IEEE Fellow, and ACM SIGSOFT Distinguished Service Award recipient Carlo Ghezzi; ACM Fellow Thomas W. Reps (one of two Americans in this group), and ACM Fellow, IEEE Fellow, Royal Danish Academy of Sciences and Letters member and Danish Academy of Technical Sciences member Christian S. Jensen.

The organization promotes appreciation of the value of European scholarship and research; advises national governments and international agencies on matters of science, scholarship, and academic life in Europe; encourages research in all areas of learning, particularly in relation to European issues, and identifies topics of trans-European importance to science and scholarship and proposes appropriate action to ensure these issues are adequately studied.

LISKOV RECEIVES HAROLD PENDER AWARD

The faculty of the Moore School of Electrical Engineering at the University of Pennsylvania has named Barbara Liskov, Institute Professor, Massachusetts Institute of Technology, recipient of its Harold Pender Award, given to a member of the engineering profession who makes significant contributions to society.

Liskov, who received her Ph.D. in computer science from Stanford University, is widely recognized for her work in programming languages, programming methodology, and distributed systems. Her work in programming methodology led to the notion of data abstraction, an important underpinning of how software systems are organized today. She and her group designed and implemented CLU, the first programming language to support data abstraction.

Recipient of the 2008 ACM A.M. Turing Award, Liskov also is a Fellow of the American Academy of Arts and Sciences, an ACM Fellow, and is a Charter Fellow of the National Academy of Inventors. She received the ACM SIGPLAN Programming Language Achievement Award in 2008, the IEEE John von Neumann Medal in 2004, and a lifetime achievement award from the Society of Women Engineers in 1996.

experiments in real time, uploading a cellphone photo of a person to the cloud and returning a name and Social Security number prediction in less than three seconds.

The work is just a proof of concept, the researchers caution. Their predictions involved hundreds of thousands of images, but a facial identification system on the scale of the entire U.S. population might easily have to work with billions of images, making the process slower and increasing the chance of false positives. However, false positives are likely to decrease as facial recognition algorithms become more accurate, Acquisti predicts. And the facial identification problem is highly scalable, so as cloud computing becomes more powerful and inexpensive, so will facial identification apps.

Protecting Privacy

Real-time identification of strangers on the street would be a clear danger to people who are the victims of domestic abuse or stalking, for example. There's also the potential for a more subtle and widespread kind of harm, one that Hartzog describes as "death by a thousand cuts."

"We're living in a world in which our data trails are increasingly being specified by what other people have disclosed about us," observes Evan Selinger, of the Rochester Institute of Technology in New York. "Innocuous things can aggregate and become part of our larger portrait, to create incredibly revealing maps of who we are."

People take for granted the ability to walk unrecognized in public spaces, Hartzog observes, and it gives us "a healthy amount of control over our identity." Eliminating this control "would impact society as a whole and our notions of autonomy, our ability to negotiate social spaces with protection," he says.

Real-time facial recognition "would eviscerate what I take to be an essential part of our public obscurity," Selinger agrees. "I think we radically underestimate its potential effect on social life."

Given the rapid pace at which facial recognition algorithms are almost sure to improve, the most effective route to preventing privacy abuses may be to restrict access to databases of faces and names, Hartzog and Selinger propose. Only a few truly massive such databases currently exist, and for the most part, their owners—Google, Twitter, and Facebook among them—have pledged to work with the U.S. Federal Trade Commission before making any significant retroactive changes to their privacy policies. The fact that these databases are currently locked away means "there is time to talk about these things before the technology becomes adopted and entrenched, at which point it becomes much more difficult to do anything about it," Hartzog says.

Real-time facial recognition "would eviscerate what I take to be an essential part of our public obscurity."

The U.S. government is starting to explore a potential regulatory role in preventing privacy abuses by facial recognition apps. On Feb. 6, the National Telecommunications and Information Administration (NTIA) convened the first of a series of meetings on the topic, which included industry experts and civil liberties advocates. The preceding day, Senator Al Franken (D-MN) had written to Nametag's creator, Kevin Alan Tussy, asking him to postpone the full launch of Nametag until the NTIA has completed its study and established best practices for the use of facial recognition technology—a request Tussy said his company would "seriously discuss."

It is unlikely, Hartzog says, that the owners of massive biometric databases will make them readily available to third-party facial recognition apps. Facebook's collection of tagged photos "is one of the most valuable databases in the world," he observes. "They would probably be extremely reluctant to part with it."

If the owners of massive photo repositories such as Facebook were all to outlaw their use for facial recognition apps, that would force such apps underground, significantly curtailing their availability, Hartzog says. "It is hard to get venture funding when your whole premise is based on breaching major companies' terms of use," he says.

Nevertheless, the very existence of these enormous databases is the "elephant in the room," Acquisti says. "The real story here is that we have a database of biometrics we never had before, and there is an entity with access to it."

Social media companies, Acquisti says, tend to do a "two steps forward, one step back" kind of dance about the use of private data. For example, in late 2010, Facebook started using facial recognition technology to suggest tags when users uploaded photos of people, but the company suspended use of this tool in 2012 to "make some technical improvements," and later promised European regulators it would reinstate the feature in Europe only with their approval. Later, however, the company resumed making tag suggestions for users in the U.S.

Steps like these are likely intended to gradually habituate users to more and more intrusive services, Acquisti says. "These entities may push the envelope a bit, then when they meet resistance they stop and recede, and then maybe a year later they push again," he says. "To me, the trend is clear—towards more and more usage of biometrics." C

Further Reading

Grother, P., Quinn, G., Phillips, J.
Report on the Evaluation of 2D Still-Image Face Recognition Algorithms. NIST Interagency Report 7709, August 24, 2011. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=905968

Taigman, Y., Yang, M., Ranzato, M., Wolf, L.
DeepFace: Closing the Gap to Human-Level Performance in Face Verification. https://www.facebook.com/publications/546316888800776/

What Facial Recognition Technology Means for Privacy and Civil Liberties. Hearing before the Subcommittee on Privacy Technology and the Law of the Committee on the Judiciary, United States Senate. July 18, 2012. http://www.gpo.gov/fdsys/pkg/CHRG-112shrg86599/pdf/CHRG-112shrg86599.pdf

Erica Klarreich is a mathematics and science journalist based in Berkeley, California.

© 2014 ACM 0001-0782/14/08 $15.00

DOI:10.1145/2633029

Seda Gurses

Privacy and Security
Can You Engineer Privacy?

The challenges and potential approaches to applying privacy research in engineering practice.

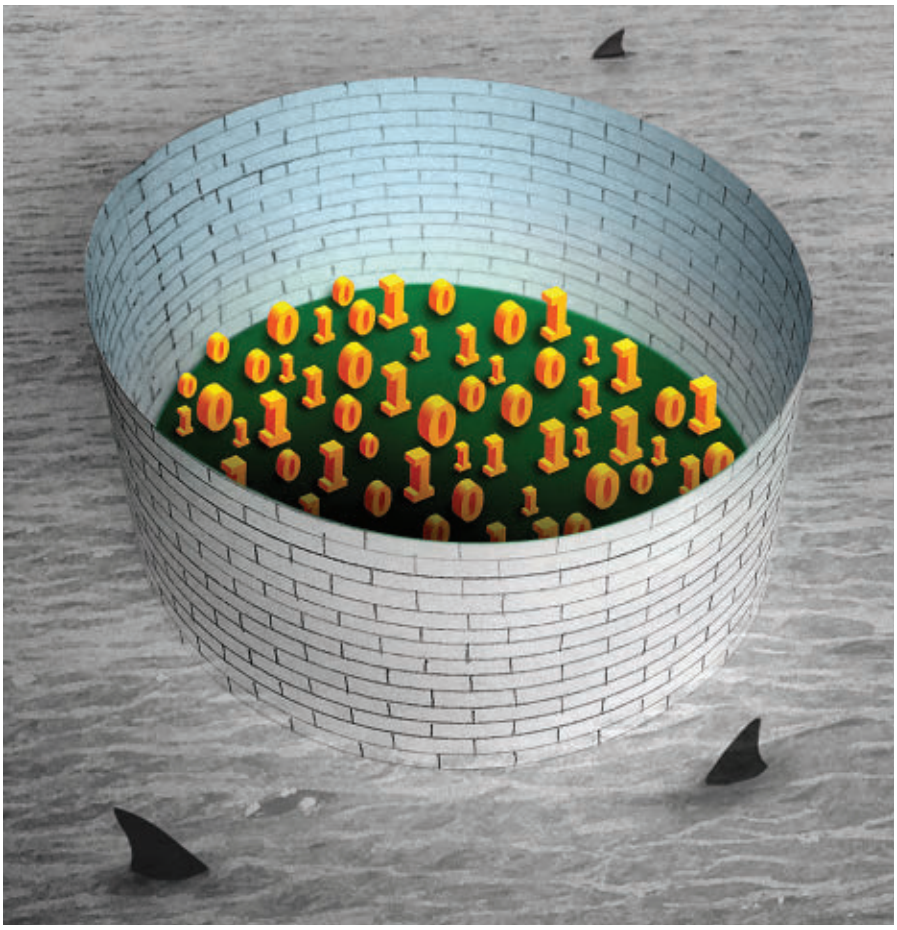
INDUSTRIAL-SIZE DATA SPILLS, leaks about large-scale secret surveillance programs, and personal tragedies due to inappropriate flows of information are guaranteed to have at least one consequence: engineers will be increasingly expected to integrate "privacy solutions" into the systems they are building or maintaining.[4] Yet the task of engineering systems to address privacy concerns can be bewilderingly complex.

The seeming unwieldiness of the engineering task becomes evident in the concept of privacy itself and how this concept is negotiated. As a legal concept, privacy is defined rather vaguely. That vagueness, some argue, is part of its protective function. The open-ended definition allows people to invoke privacy as a category to protect their personal lives and autonomy from intrusions by others—including the state that endows them with citizenship rights and runs surveillance programs. European Data Protection Directive (DPD) or Fair Information Practice Principles (FIPPs) on the other hand are procedural measures, such as notice and choice, data retention limitation, and subject access rights. These principles are seen to be instrumental to making the collection and processing activities of organizations transparent. Although less ambiguous, data protection principles still need to be translated into technical requirements and are vulnerable to narrow interpretations. Moreover, FIPPs fall short of mitigating all the privacy concerns of users toward a given organization. They also do not address privacy concerns users may have with respect to other users, with people in their social environments, and toward a greater public.

Over the years, "privacy research" in computer science has led to a whole palette of "privacy solutions."

Scholars from various fields have stepped up to the challenge of clearing the murky waters of privacy. Legal scholars and philosophers have proposed taxonomies of privacy violations[5] and a holistic framework for evaluating appropriate flows of information based on contextual social norms.[3] Social scientists and ethnographers have studied groups of people, online and offline, to develop better-informed understandings of users' needs. But how are engineers supposed to integrate and translate these frameworks into existing engineering practice? In attempting to answer this question, privacy research conducted within computer science is valuable.

Over the years, "privacy research" in computer science has led to a whole palette of "privacy solutions." The solutions originate from diverse sub-fields of computer science, such as security engineering, software engineering, HCI, and AI. From a bird's-eye view, all of these researchers are



studying privacy problems and solutions, and yet a closer look reveals they also have their differences. In this column, I introduce a small taxonomy of prominent approaches to privacy within computer science. The categories of this taxonomy are not comprehensive or definitive, but they provide an overview of the guiding principles that distinguish the approaches that privacy researchers are taking. In practice, engineers may mix and match these principles; pulling them apart allows us to think about the different ways in which we can engineer systems with privacy in mind.

Privacy as Confidentiality

The most prominent conception of privacy relies on the binary that exposure of information leads to a loss of privacy, while guaranteeing the confidentiality of information is a way to "preserve" or "enhance" privacy. This binary can be linked to Warren and Brandeis' "right to be let alone,"[1] which was a response to the novel ways in which innovations in technology made it increasingly possible to collect information about matters that would have previously been regarded as private.

Many privacy researchers work on solutions that rely on this binary understanding of unwanted disclosures as privacy violations. These researchers rely on three important principles: data minimization; avoidance of a single point of failure and openness to scrutiny. The first principle—data minimization—is about designing systems (and computational mechanisms) to only collect private information that is absolutely necessary for a given functionality. In its bare bones, this means that, by default, the users should be able to use the system anonymously. If the users have to be identified, the different interactions of the user throughout time should remain unlinkable. For example, a user of a service may utilize zero-knowledge proofs to prove she is at least 18 years old without revealing her birthdate or any further private information, thus guaranteeing that her transactions are unlinkable. Communications as well as traffic data must also be kept confidential from unauthorized parties, sometimes including the service provider itself. The capabilities of these unauthorized parties, also called adversaries, is an important driver of threat models, and hence the design of privacy solutions in this line of research.

The second principle is to avoid designing architectures for information collection and processing with a "single point of failure." In other words, introducing a distributed trust model so that users do not need to rely on a single entity to protect their privacy. The third principle requires that the protocols, code, and processes of development that underlie privacy tools are open to public scrutiny in order to increase trust in the privacy solution itself. The Tor project (https://www.torproject.org/) is a popular example of a privacy as confidentiality solution that is built using all three principles.

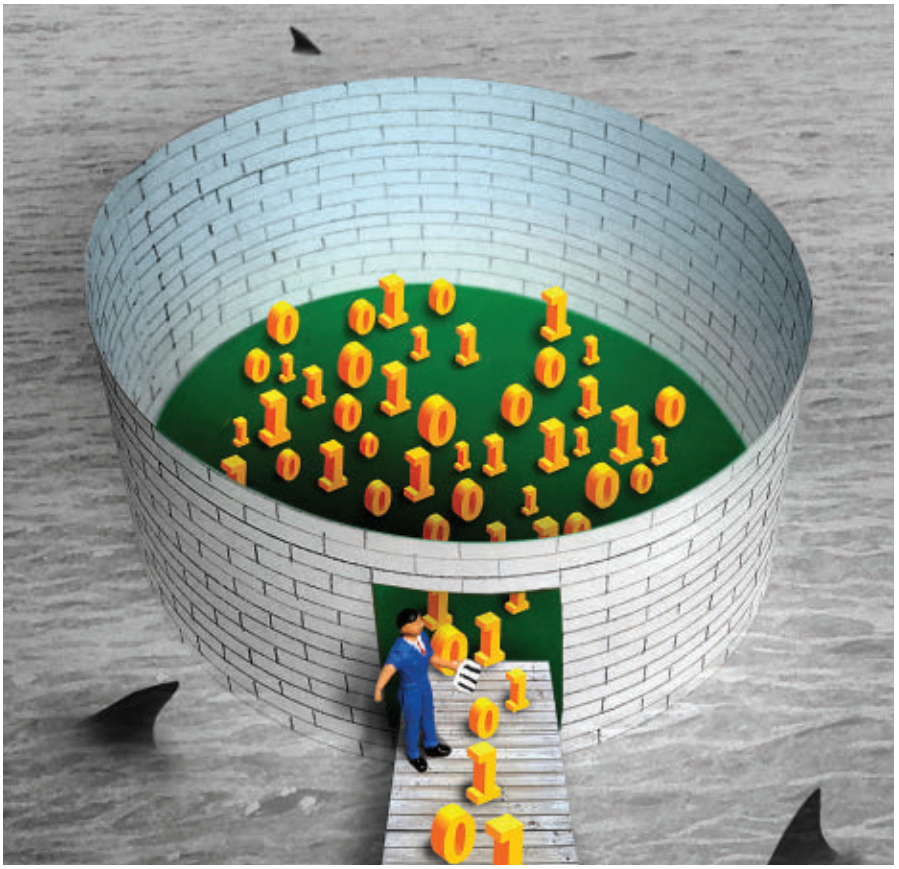
Privacy as Control

Another approach to privacy starts from the assumption that information will have to be disclosed in an increasingly networked world. Hence, Westin writes, privacy is "the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others."[7] Westin's work, which he buttressed with dozens of large-scale surveys, is fundamental to most legal and organizational measures introduced to protect personal data across the globe.

Based on this conception of "information privacy," DPD and FIPPs list procedural mechanisms through which organizations can make their personal data collection and processing practices transparent to their data subjects, regulators, and the general public. If these mechanisms are in place, ideally users can make informed decisions about and have greater control over the collection and flows of their personal information. Further, abuses can be detected or mitigated.

A good portion of privacy research focuses on developing methods and mechanisms for data protection compliance. These protection mechanisms are expected to complement organizational measures, like privacy training for employees working with personal data or procedures for database breach notifications. Another

ILLUSTRATIONS BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES



type of notification is the "privacy policy" through which organizations can inform data subjects about the purpose for which they are collecting personal data, which data, and for how long. Privacy researchers have studied ways to ease the burden of reading privacy policies, often weighted with legal jargon. Proposals have included usable representations of the content through labeling mechanisms comparable to those in the food industry, as well as the design of machine-readable privacy policies that can be matched to user preferences.

Privacy policy languages, as the latter are often called, and policy enforcement mechanisms like purpose-based access control models can be applied in combination to provide organizations with mechanisms to ensure the internal use of personal data internally adheres to the collection purpose. Data minimization also pops up here, but rather than aspiring to achieve anonymity or unlinkability through computational methods, the principle is about limiting the collection of personal data to the given purpose only. These researchers often assume service providers are the main party trusted with the protection of personal data. This means distributed trust models are rare in these proposals. Subject access rights, the ability of the users to access, rectify, or delete their data collected by a service provider is another challenging requirement that has attracted research and has found popular implementations like Google dashboard. This past May, the European Union court ruled users have a "right to be forgotten" and Google launched a service to allow Europeans to ask for their personal data to be removed from online search results for searches made in Europe (searches sent through Google sites outside Europe will still show the contested data).

A good portion of privacy research focuses on developing methods and mechanisms for data protection compliance.

Privacy as Practice

In this third approach, privacy is seen as the negotiation of social boundaries through a set of actions that users collectively or individually take with respect to disclosure, identity, and temporality in environments that are mediated by technology. Hence, privacy is not seen as something that users can delegate to the machine. Rather, engineers explore how design mechanisms and principles mediate users' "privacy practices."[2] These researchers dispense with the binary understanding of privacy as exposure and privacy as concealment, since interactions inform negotiation of privacy in unexpected ways. For example, a user may signal to her peers through chat that she wants to be left alone, a disclosure that allows her to negotiate her privacy in a public space. As evident in this example, the distinction between online and offline privacy is also undone in the pursuit of understanding privacy practices.

Privacy researchers here emphasize that privacy is negotiated through collective dynamics. If unlucky, the same user might have very disrespectful peers that do not respect her request for privacy. Transparency and feedback mechanisms for raising awareness of the socio-technical system's workings are often proposed as central to establishing privacy practices. This understanding of transparency is greater than providing information about what data is collected by an organization as proposed in "privacy as control." Instead, the objective is to make information systems, their effects, and the responses from (non-)user communities part of what needs to be made transparent.

For example, if possible consequences of a user's actions can be made transparent to her, she may be able to make better decisions about her interactions. The user may learn from her past interactions, so feedback about past practices may be used to inform future ones. For instance, information about how many friends have visited a user's profile may inform how much

she wants to post on her profile in the future. Similarly, users may learn from their (mediated) social surroundings: information about how other friends manage their privacy settings may provide guidance to the user. In some cases, based on studies about good privacy norms, users can be "nudged" to develop "better" privacy practices.[6] Users may be opportunistically encouraged to review their privacy settings. Similarly, if a user is provided with feedback on the algorithms underlying a recommender system, she may better assess whether and how she wants to participate in such a system.

Future Prospects

There are many more proposals for addressing privacy in systems than listed in this column. Some proposals fall in between the three categories. For example, database anonymization and differential privacy both propose elaborate computational methods for data minimization comparable to solutions in "privacy as confidentiality." However, the mechanisms are not intended to minimize data collection but to anonymize or obfuscate later disclosure. Further, database anonymization is a way to exit the legal compliance regime, making it difficult to identify it as an organizational transparency mechanism typical of "privacy as control." Furthermore, there are a number of proposals for addressing discrimination and fairness issues in the context of data mining, like discrimination aware data mining as well as fairness in classification. However, it is open to discussion whether discrimination and fairness are privacy issues. Generally, many concerns we discuss under privacy may in fact be related to greater issues of social justice that require more elaborate rethinking of our societies as well as technological futures.

We cannot engineer society, but neither are our societies independent of the systems we engineer.



The taxonomy described here shows that engineering decisions, be it when architecting infrastructures, designing organizational systems, or crafting particular applications, co-determine the way in which people may negotiate their privacy. Yet, challenges abound. Can we integrate these three approaches given their fundamental differences? For example, while privacy as confidentiality assumes a world in which trust in organizations that process private data should be minimized, privacy as control assumes trust in those organizations can be established through transparency. In contrast, most privacy as practice proposals assume the service provider is honest and has a genuine interest in accommodating users' privacy practices above its own organizational and market interests. Hence, a reasonable skeptic could ask, given the slipperiness of the concept, the political and market contestations of privacy, as well as the differences between the solution sets, can we even speak of a privacy engineering project?

"Engineering privacy" may in fact only be another ideal like security, dependability, or usability, and one that misleadingly suggests we can engineer social and legal concepts. We cannot engineer society, but neither are our societies independent of the systems we engineer. Hence, as practitioners and researchers we have the responsibility to engineer systems that address privacy concerns. The taxonomy presented here attempts to provide an overview of existing approaches to privacy in computer science research. The robustness of these approaches will only grow through further engagement in all of them when we engineer systems. C

References

1. Brandeis, S.W. The right to privacy. *Harvard Law Review 4*, 5 (1890).
2. Dourish, L.P. Unpacking "privacy" for a networked world. In *Proceedings of CHI* (2003), 129–136.
3. Nissenbaum, H. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, Palo Alto, CA, 2009.
4. NIST. Privacy Engineering Workshop, 2014; http://www.nist.dov/itl/csd/privacy-engineering-workshop.cfm.
5. Solove, D. A taxonomy of privacy. *University of Pennsylvania Law Review 154*, 3 (2006), 477.
6. Wang, Y., Leon, P.G., Acquisti, A., Cranor, L.F., Forget, A., and Sadeh, N.M. A field trial of privacy nudges for facebook. In *Proceedings of CHI* (2014), 2367–2376.
7. Westin, A. *Privacy and Freedom.* Altheneum, NY, 1967.

Seda Gurses (seda@nyu.edu) is a postdoctoral researcher at New York University.

Copyright held by author.

DOI:10.1145/2633031

Uri Wilensky, Corey E. Brady, and Michael S. Horn

Education

Fostering Computational Literacy in Science Classrooms

An agent-based approach to integrating computing in secondary-school science courses.

There is widespread and growing agreement that computing should play a more prominent role throughout our education system. The next generation of learners will require a high level of fluency with modes of thinking and inquiry in which computers act as interactive partners. While many students experience computing (via the Web and apps), few students understand computation, and even fewer have experience using computers as tools for scientific inquiry. These skills and perspectives are essential for full and effective participation in today's (and tomorrow's) society.

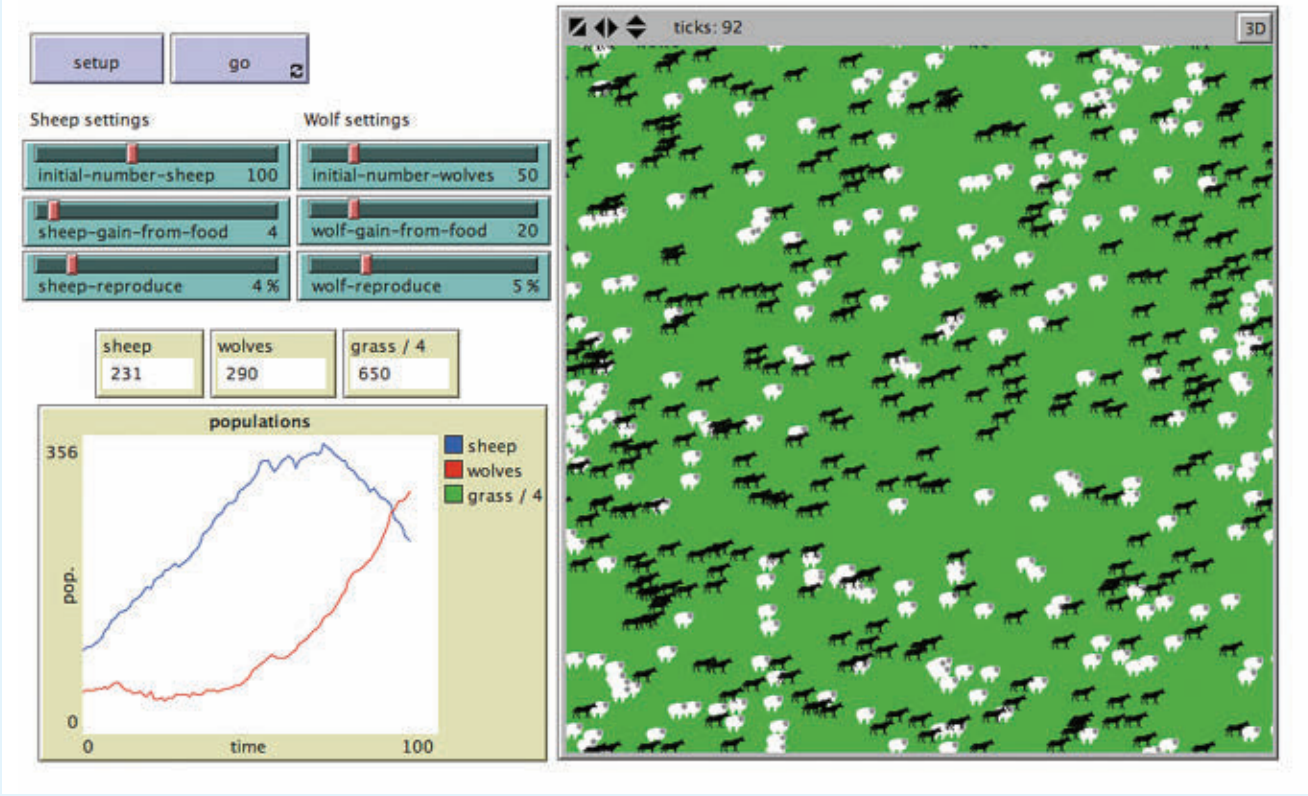
The development of computer science curricula, standards, and course requirements for secondary schools is an important and useful direction actively being pursued in a variety of initiatives. However, the success of such initiatives will depend heavily on schools' ability to hire and retain qualified teachers; on teachers' ability to implement curriculum that is applicable to scientific inquiry; and on students' ability to make room for new coursework in their already-packed schedules. Although large-scale efforts such as Code.org are offering support for computer programming instruction in schools, the magnitude of the challenge is enormous. Even more worrisome is the possibility that an elective-only CS course sequence will fail to attract a diverse population of students, exacerbating the low participation rates of women and other underrepresented groups in computing fields.

Let's introduce real computational literacy through the science classes every student takes, rather than solely through computer science classes.

A complementary approach we advocate here is to integrate computing across the range of secondary-school science courses.[a] Let's introduce real computational literacy through the science classes every student takes, rather than solely through computer science classes. Our goal is to treat computation as a core component in a broad-based cultural literacy, rather than the exclusive province of a single academic department or field of study. This approach is consonant with the views of visionary educators such as Papert[3] and diSessa[2] who have advocated for universal computational literacy as a means to provide access to "powerful ideas." While this approach faces its own set of challenges, integrating computation into science learning has several distinct advantages: it increases access to computing for all students in all schools; it enhances students' motivation for and depth of understanding of scientific principles by using computing in powerful ways; it brings science education in line with authentic scientific practice and the needs of 21st-century science; and it provides students with experiences of computers beyond searching and sorting, demonstrating the power of computation to help them make sense of their world. We can reach more students more quickly by getting science teachers to add computing into their classes, than we can by developing a nationwide cohort of computer science teachers and placing them in all our high schools. We have found that brief but intensive professional development experiences, accompanied by carefully crafted curricular materials, are sufficient to

a For more, see the Computational Thinking in STEM project at Northwestern University (http://ct-stem.northwestern.edu).

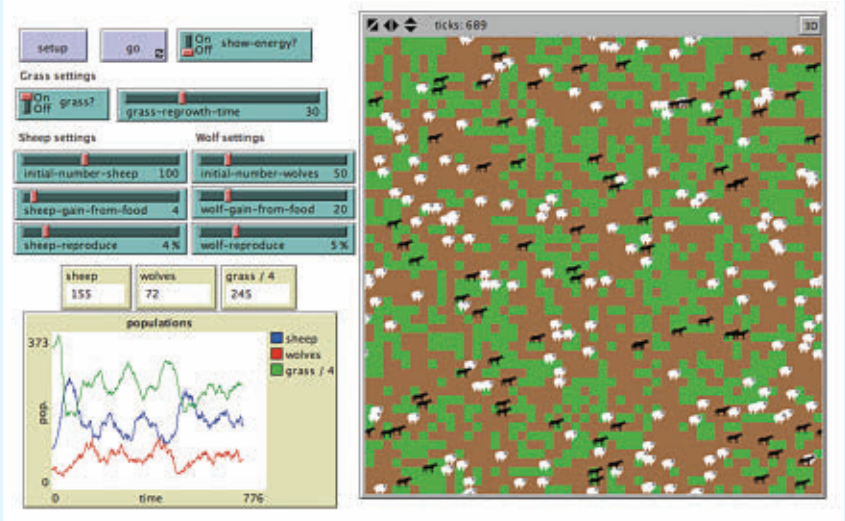
Figure 1a. An ecosystem with sheep and wolves (see http://ccl.northwestern.edu/netlogo/models/wolfsheeppredation).



support high school science teachers in bringing computational modeling into their courses. We are currently completing an NSF-funded project working successfully in this way with over 100 science classrooms in the Chicago area. [b]

In this column, we describe the use of agent-based modeling (ABM) as a powerful way to introduce computation across the secondary science curriculum. ABM is a form of computational modeling in which individual entities in a computer simulation (the agents) are given rules defining their behavior. There are two classes of agents, mobile agents that typically represent individuals such as animals or molecules, and a grid of stationary agents, as in a cellular automaton, that typically represent parts of the environment such as grass or other elements of the terrain. The collective interactions of a multitude of agents, each concurrently acting out its behavioral rules, can reveal complex, emergent patterns. The "game" of ABM is to try to generate known phenomena by defining a set of agents and rules for their behavior, or to explore the possible phenomena that can be generated with such a set by varying conditions or parameters.

Figure 1b. Adding grass adds stability.



b For more, see the Enabling Modeling and Simulation in the Classroom project at Northwestern University with partnerships at Stanford and Vanderbilt universities (http://ccl.northwestern.edu/modelsim).

ACM Transactions on Reconfigurable Technology and Systems



This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.

www.acm.org/trets
www.acm.org/subscribe

Association for Computing Machinery

Figure 2a. Behavioral rules for each "breed" of agent.

```
ask sheep [
        wander
        try-eating-grass
        maybe-starve
        maybe-reproduce-sheep
]
```

```
ask wolves [
        wander
        try-catching-sheep
        maybe-starve
        maybe-reproduce-wolves
]
```

Figure 2b. A possible NetLogo implementation of the wolf behavior "try-catching-sheep."

```
to try-catching-sheep                   ;; A wolf-specific procedure
  if any? sheep-here [                  ;; Are there any sheep here with me?
        let prey one-of sheep-here      ;; If so, select one...
        ask prey [ die ]                ;; ...kill it (and eat it)...
        set energy energy + 20          ;; ...and gain some energy from it.
  ]
end
```

Based Modeling (CCL) at Northwestern University has developed ABM tools for education and scientific practice. The NetLogo ABM environment[c] (free and open source) is a product of this effort, and currently has hundreds of thousands of users, ranging from students in middle schools to researchers in scientific laboratories. The CCL continues to develop software and materials to support the integration of NetLogo in science classrooms.[d] This includes a wide variety of contexts and grade levels from middle school through undergraduate education and graduate school, suggesting we have made some progress toward our goal of broad reach, but we still have a long way to go.

Why Agent-Based Modeling?

One reason ABM approaches hold great potential to support science learning is that so many of the concepts students find most challenging involve connecting micro and macro aspects of scientific phenomena. The science education literature describes the many misconceptions students have about what connects these levels (see Wilensky and Resnick[5] and Chi[1]). For instance, in chemistry and physics, gas molecules collide elastically at the micro level, leading to the properties of pressure and temperature at the macro level. In biology, individual animals struggle to survive and reproduce at the individual level, leading to phenomena such as evolution, natural selection, and population dynamics at the ecosystem level. Agent-based modeling provides the means to build on intuitive understandings about individual agents acting at the micro level in order to grasp the mechanisms of emergence at the aggregate, macro level.

Because the individual-level behavior of agents is relatively simple, ABMs feature relatively simple computer programs that control the behaviors of their computational agents. On the other hand, swarms or aggregates of interacting agents can produce complex, emergent patterns that require computational power beyond the human capacity to simulate. (Thanks to decades of life under Moore's Law, however, such power is now available in virtually all personal computers and mobile devices.) Working in partnership with a computational model within an ABM environment such as NetLogo, learners can explore the connections between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction. As they work with NetLogo, students can articulate their own provisional thinking in an executable form. Running their models reveals the implications of their ideas, provokes new conjectures, and drives "debugging" cycles of modeling, execution, and refinement. This iterative process is a motivating and intellectually exciting activity, driven by

c See http://ccl.northwestern.edu/netlogo.
d See http://ccl.northwestern.edu/modelsim/ and http://ccl.northwestern.edu/simevolution/.

interactive feedback and dynamic visualizations. Working with NetLogo, kids can explore existing models by changing initial conditions and sweeping the parameter space of key variables. They can also explore what-if questions by modifying or adding behavioral rules to an existing model or creating their own models from scratch.

But ABM is not only a tool for the classroom. NetLogo is used in a wide variety of research laboratories and professional contexts, and hundreds of scientific papers using it have been published.[e] Thus, the work students do in agent-based modeling prepares them for authentic inquiry in the scientific disciplines. In the examples here, we present several uses of NetLogo and agent-based modeling to engage with core concepts across a range of topics from secondary science.

Agent-Based Modeling Across the Sciences

Population Biology/Ecology. Common topics in middle- and high-school biology include the dynamics of populations within ecosystems and food webs. Figures 1a and 1b show agent-based models of predator-prey relations between populations of wolves (who eat sheep) and sheep (who eat grass). Both wolf and sheep are modeled as having energy, losing energy by moving, gaining energy by eating, dying if they have too little energy, and expending energy to reproduce. As shown in Figure 2a, for each "breed" of agent, a simple collection of behavioral rules describes their actions and interactions. These behaviors are then defined computationally. Thus, a NetLogo implementation of the wolf behavior "try-catching-sheep" might be as shown in Figure 2b.

The work students do in agent-based modeling prepares them for authentic inquiry in the scientific disciplines.

Learners can run their in-progress models to see how the system behaves. For instance, in this simulation learners have found that adding logic to describe the depletion and replenishment of the virtual grass led to increased stability in the ecosystem, damping the oscillations in the wolf and sheep populations.

Physics and Chemistry. In physics and chemistry, the Ideal Gas Laws provide an elegant mathematical explanation of regularities in the world, but students rarely connect these macro-level phenomena with the molecular interactions that generate them. Students using ABMs can leverage agent-based intuitions to understand the Kinetic Molecular Theory (KMT). Models like the ones in the CCL's GasLab suite[f] begin from simple collision rules for particle agents and show how pressure and temperature arise as emergent properties in the aggregate.[g] Because they are working with individual entities, high school students can reason about their interactions, going beyond the familiar topics and touching on advanced concepts, such as the Maxwell-Boltzmann distribution and phenomena of statistical mechanics (see Figure 3).

Earth Sciences. In middle school Earth science classrooms, it is common to study natural disasters such as volcanoes and forest fires. With the *Fire* model,[h] students can investigate the systems principles of critical parameters and nonlinear dynamics in the context of a simulated forest fire. Here, there are tree agents and fire agents. The trees are randomly distributed at a given density, which is controlled by a slider. The leading edge of a forest fire is represented by red agents; the rules for the trees are very simple: they "look" at their neighbors to the North, South, East, and West. If they see a tree on fire, they ignite.

Most people's intuition about this system is that a small increase in tree density should result in a little more burn (that is, a linear relationship). However, that is not what the simulation reveals. Instead, there is a critical density value, below which the forest fire dies out quickly (see Figure 4a) and above which it consumes almost the entire forest (see Figure 4b). Such critical parameters are common in complex systems throughout science, and have been recently popularized as "tipping points."

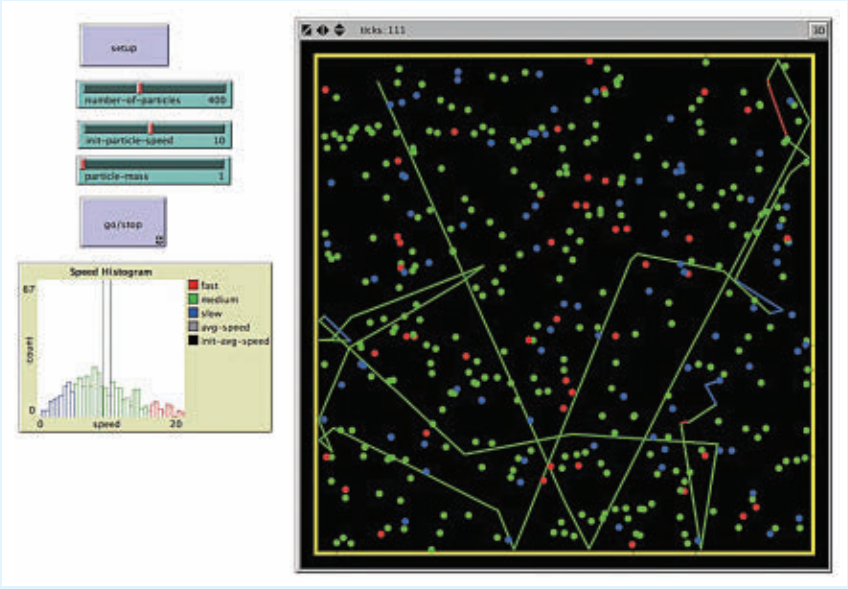
e See http://ccl.northwestern.edu/netlogo/references.shtml.

f See http://ccl.northwestern.edu/curriculum/gaslab/.

g Or, in other words, the molecules "compute" pressure and temperature.

h See http://ccl.northwestern.edu/netlogo/models/fire.

Figure 3. GasLab model of molecules in a box, colored by their speed. One of the molecules leaves a trace to facilitate following its trajectories.



Students find the simulated fire to be visually compelling, and the code behind the NetLogo model is extremely simple. Thus, the Fire model can act as an excellent introduction to agent-based modeling. Indeed, learners who engage with it often modify or extend the model to explore their own questions, including the effects of wind, alternative rules by which the fire might spread, or strategies for arresting the spread of an existing forest fire.

Figure 4a. Critical parameters. The run (57% density) is just below the critical density and the fire has burned only 7.6% of the forest.

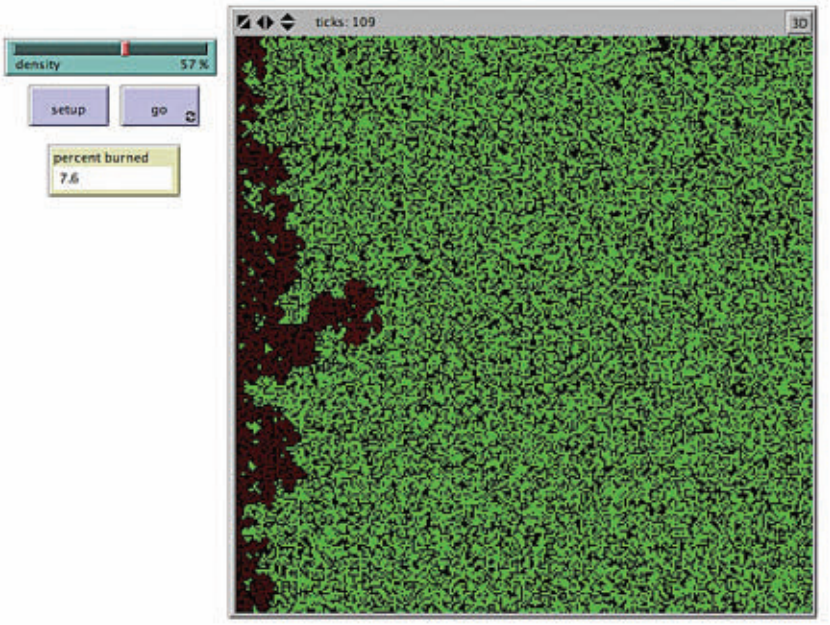
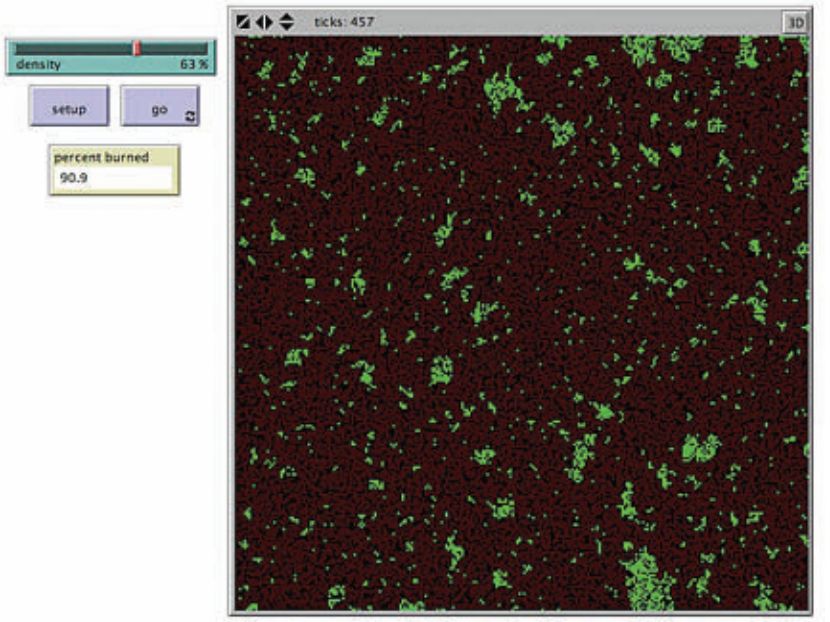


Figure 4b. Critical parameters. In contrast, the run (63% density) is above the critical value, and the fire rages on. (In the end, 90.9% of the trees burned.)



Conclusion: Agent-Based Modeling and Computation for All

We are now well into the 21st century, and we need to bring the efforts of our educational system in line with our shared sense of the growing importance of computing for all members of our society. Thinking effectively about and with computational processes is a broad-based literacy needed by all citizens to support their effective social, economic, and political participation. We have argued in this column that secondary science classrooms[i] present a compelling opportunity to build this literacy, and that ABM tools like NetLogo are an effective means to do so. By letting go of the idea that literacy with computation is the sole responsibility of educators officially working within computer science departments, we can widen our reach and arrive at our goals more quickly and effectively. C

i In this column, we have argued for and presented examples of the use of ABM in middle and high school science. However, our efforts have not been limited to either science or secondary school. There has also been considerable enthusiasm for using ABM in the *social sciences*, where we have also created materials for modeling phenomena such as wealth accumulation, segregation, and language change. We have also worked to integrate ABM into a wide range of university courses (see Wilensky and Rand[4]).

References

1. Chi, M. Commonsense conceptions of emergent processes: Why some misconceptions are robust. *The Journal of the Learning Sciences 14* (2005), 61–199.
2. diSessa, A.A. *Changing Minds: Computers, Learning, and Literacy.* MIT Press, Cambridge, MA, 2000.
3. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, New York, 1980.
4. Wilensky, U. and Rand, W. (in press). *Introduction to Agent-based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo.* MIT Press, Cambridge, MA.
5. Wilensky, U. and Resnick, M. Thinking in levels: A dynamic systems approach to making sense of the world. *Journal of Science Education and Technology 8*, 1 (1999), 3–19.

Uri Wilensky (uri@northwestern.edu) is a professor at Northwestern University with a joint appointment in Computer Science and the Learning Sciences. He is the author of NetLogo and the director of the CCL lab.

Corey E. Brady (cbrady@northwestern.edu) is a research assistant professor of Learning Sciences at Northwestern University.

Michael S. Horn (michael-horn@northwestern.edu) is an assistant professor at Northwestern University with a joint appointment in Computer Science and the Learning Sciences.

Copyright held by authors.

DOI:10.1145/2633033

Chris Coward

Global Computing
Private Then Shared?

Designing for the mobile phone to shared PC pipeline.

Not long ago people in richer countries thought people in low-income countries of Africa, Asia, and Latin America would use personal computers to gain access to the information society. To overcome technical, social, financial, and other hurdles, richer governments and donor agencies invested in shared access computing facilities.[3] The logic was flawless: if there cannot be a personal computer in every home, a shared computer in every village will do. The global telecenter movement was born: a collection of programs providing shared access computer and Internet services to poor and remote communities.[4,5]

Today with the unprecedented explosion of mobile telephony, many in our field have completely written off shared access. Some even question the necessity of personal computers. If mobile phones can meet computing and connectivity needs, who needs personal computers?[2,9]

The first experience with a digital device in the Global South is likely to be with a mobile phone. Smartphones provide network services for those who can afford mobile data connectivity[6] and the number and range of tasks possible on these phones continues to expand. Assuming this expansion is accompanied by a decline in personal computer use, the shared access model falters. However, as shown in Figure 1, suppose instead that mobile smartphone adoption *stimulates* demand for the personal computer in shared access settings?

The Technology & Social Change Group at the University of Washington Information School studied five low- and middle-income countries—Bangladesh, Brazil, Chile, Ghana, and the Philippines.[7,8] Their large-scale (n=5,010) user surveys and non-user surveys (n=2,000) were accompanied by ethnographic, experimental, and other methods. They found the vast majority of current shared access personal computer users also possess mobile phones (96%), and mobile Internet access was rarely the reason why former users stopped going to shared access venues.



Figure 1. Senzo Ngidi is a second-year business student from Makhaza, Cape Town. He relies on his cellphone and public libraries for Internet access.

Suppose mobile smartphone adoption *stimulates* demand for the personal computer in shared access settings?

Half of these respondents had their first contact with computers through shared access (50%). Even more so for first use of the Internet (62%). Shared access venues were often more important for developing computer (40%) and Internet (50%) skills than school or home. They were where people could accomplish information tasks from health to education, with large

PHOTOGRAPH BY VISUAL VOICES @VISUALITES

majorities citing the importance of knowledgeable staff, good equipment, fast connectivity, low cost, and peer learning opportunities at shared access centers—features not readily found elsewhere.

Perhaps mobile phones and shared access venues are complements, not substitutes? Research by Marion Walton of the University of Cape Town, South Africa, and Jonathan Donner of Microsoft Research India suggests so.[7,8] They studied teenage users of Internet-enabled phones *and* libraries and cybercafés in low-income Cape Town neighborhoods. Their detailed interviews, activity/drawing probes, and task analyses of 53 teenagers and surveys of 280 teenage users found that overall, mobile phones and public access venue computers do not substitute for one another: each is used for a distinct set of activities and information behaviors, associated with different social, academic, or professional practices. These public access users had developed elaborate, fine-grained practices combining public access computers and mobile phones, taking maximum advantage of their complementary aspects. Figure 2 shows the preference to mix mobile phones and PCs. Shared access centers provide non-substitutable value for resource-constrained users, including those with "the Internet in their pocket."

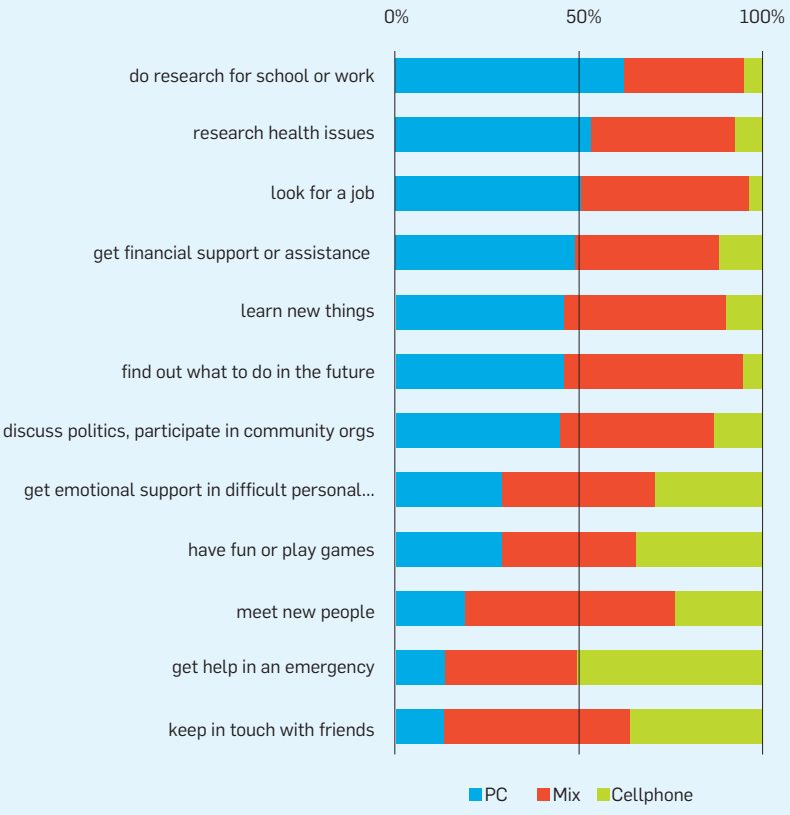
Perhaps mobile phones and shared access venues are complements, not substitutes?

Implications

The emerging generation of people in low-resource environments uses both the mobile Internet and shared access personal computers. Home PC or tablet penetration levels equivalent to those found in the North may take decades to achieve, meaning for the near future the majority of people in developing countries will use a mobile phone for social interactions but will benefit from a shared computer to do their homework, take an online course, prepare résumés, or create business spreadsheets. Maybe such tasks could be performed through both mobile phones and shared access computers. Trading off constraints and affordances between the two appliances and contexts might allow users to leverage a shared center's local network, the cloud, and the mobile phone. Innovation that leverages the increasing ubiquity of mobile phones will drive new uses of shared access centers in low-resource environments.

In the previous *Communications* Global Computing column, Michael Best called for *community computers* that are purpose-built for shared contexts, challenging the titular implications of the personal computer.[1] By observing use patterns and identifying needs in shared access centers, solutions for the mobile phone to PC pipeline could yield tremendous benefits for the Global South. C

Figure 2. Mobile phone, PC, or both? Stated preferences for various tasks (n=280).



References

1. Best, M.L. Thinking outside the continent. *Commun. ACM 57*, 4 (Apr. 2014), 27–29.
2. *Economist.* The real digital divide. (Mar. 10, 2005).
3. Heeks, R. ICT4D 2.0: The next phase of applying ICT for international development. *Computer 41*, 6 (June 2008).
4. Kuriyan, R. and Toyama, K., Eds. *Review of Research on Rural PC Kiosks.* Microsoft Research, India, 2007; http://research.microsoft.com/en-us/um/india/projects/ruralkiosks/.
5. Proenza, F.J. Public Access to ICTs: What do we want to know? What do we already know? Where do we go from here? *CIS Literature Review on the Impact of Public Access to ICT*, 2008.
6. Sahota, D. Africa gets smart: Continent prepares for device revolution. Telecoms.com (Jan. 9, 2014).
7. Sey, A., Coward, C., Bar, F., Sciadas, G., Rothschild, C., and Koepke, L. Connecting people for development: Why public access ICTs matter. Technology & Social Change Group, University of Washington Information School, Seattle, 2013.
8. Walton, M. and Donner, J. Public access, private mobile: The interplay of shared access and the mobile Internet for teenagers in Cape Town. *Global Impact Study Research Report Series.* University of Cape Town, Cape Town, South Africa, 2012.
9. World Bank Information Communication Technologies infoDev. *Information and Communications for Development 2012: Maximizing Mobile.* World Bank Publications, 2012.

Chris Coward (ccoward@uw.edu) is Principal Research Scientist and Director of the Technology & Social Change Group at the University of Washington Information School in Seattle, WA.

Copyright held by author.



DOI:10.1145/2633035

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

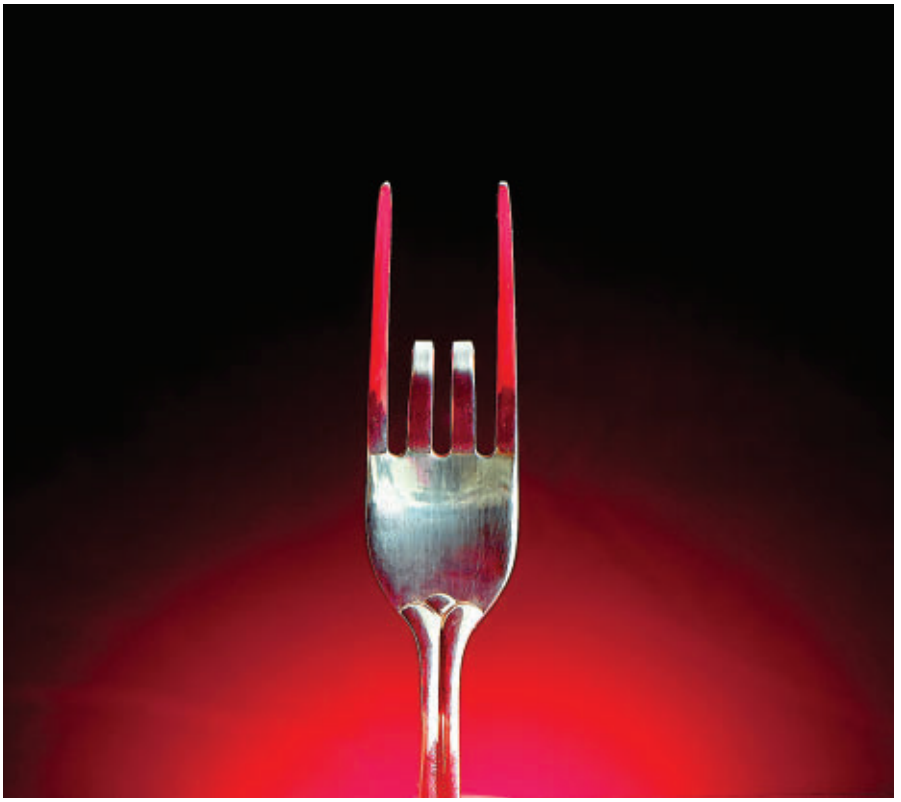
Kode Vicious
Forked Over

Shortchanged by open source.

Dear KV,

How can one make reasonable packages based on open source software when most open source projects simply advise you to take the latest bits on GitHub or SourceForge? We could fork the code, as GitHub encourages us to do, and then make our own releases, but that puts the release-engineering work that we would expect from the project onto us.

Forked Over



Dear Forked,

The short answer is that you cannot, but if that were all I have to say, I would not have bothered to answer this letter, so let me put a lot more explanation around this.

One of the upsides *and* downsides of the move from packaged systems to SaaS (software as a service) has been the constant rolling release. When all the interactions between users and their software are proxied through a Web browser—which, minus any client code, is really interacting with a server under the control of the software developers—then rolling out a new software release is only a matter of changing the software on the server. Most companies that provide software this way can, and often do, roll out software every day, and sometimes several times per day. SaaS has provided a segment of the software industry with an amazing amount of freedom. Why worry about bugs when they can be fixed in the next push?

The downside of this mental model of development is it introduces a certain amount of laziness into the maintenance of interfaces. Why care about maintaining an API if you can just roll out an upgrade on the next push? That attitude has little negative impact if you have a small number of consumers of your API. Once you put up your software for sharing on GitHub or a similar service, however, you have an unknown community that is depending on your software. Should you feel some responsibility toward these external users? Well, if you do not, then you should not bother sharing your software, as it is not really sharable, except in the very broadest sense of the word. Yes, anyone can "fork" your repo or download the code and use it, but they cannot depend on it if your attitude toward its public face—the APIs it presents—is so cavalier that you do not even bother marking your source tree when you make API changes.

Whether or not software was developed to be packaged or for SaaS, once it has a set of consumers, it needs to be maintained using some standard

IMAGE BY ZUZANA RANDLOVÁ

practices. You may not *cut a release*, as the term goes, where there is a single unit of packaged software available for download, although such packages do make life easier for those of us who maintain package repositories such as FreeBSD/MacPorts, Red Hat RPMs, Yum, and the like. At the very least, however, you must indicate when you have changed an API, as the API is the contract between your package and the rest of the world. The easiest way to indicate this API change is by marking your source tree with a release tag. Choosing the tag name is a separate, painful, and tedious discussion, which I will not go into here, other than to say some consistency to the meaning of the tags will be helpful to your downstream users.

Thinking about when to mark your tree with a release tag has some handy side effects. First, it forces you and your team to focus on an end goal. Software engineers are well known for their love of perfection and being loathe to release software until it is done, where *done* is often very poorly defined. Thinking about what constitutes a release of your software focuses the developers on an end point toward which they can all work. An API change is as good a reason as any to create such a release point.

Second, it helps break down a large project into stages that are logically related. Very few projects are so small that they are done after the first release—unless that is the point at which they completely fail. Since you know there will be more than one release of the software, it is better to plan for that—though, I know, for many people and groups, "plan" is a four-letter word. While you are at it, well-maintained release notes about changes go a long way toward making happy downstream users.

If you are serious about sharing your software, then you should be serious about *how* you share it: think about release points, tag your trees, and do not change APIs without notifying your users.

KV

If you are serious about sharing your software, then you should be serious about *how* you share it.

Dear KV,

One of my least favorite parts of working with open source software is that it never seems to be complete. I will download, build, and install an open source package, try to use it, and find it almost works, but that it fails in unpredictable ways. I will then read the forums or mailing lists for the project, or just search Stack Overflow, and discover the software has serious limitations that were not called out on the project home page. There ought to be a Web page that rates the quality of open source software so users can quickly determine whether or not a piece of software is suitable for use.

Shortchanged by Open Source

Dear Short,

I find it odd that you call out open source in your letter. Have you never used a proprietary product that did not meet expectations or live up to its marketing hype?

The "almost-working tool" is a constant problem in software and in computing systems in general. Developers are optimists and will promise the moon while only getting you to LEO (low Earth orbit). Yes, the view is amazing from LEO, but it is not going to get your global communications satellite the field of view it really needs. Other than telling you to take all developer and marketing statements with a grain of salt, what else can be done to avoid surprises?

Instead of using the tool and then running to the Web when it did not work as you expected, you should have done these actions in reverse order. One of the great things about the Internet is the number of error messages it holds and the fact that conversations held in comments rarely, if ever, disappear. A few choice words connected to your package of choice may tell you more about its suitability for your needs than the "download-and-try" model of work. I particularly like the words: *crash*, *won't build*, *partial failure*, *segfault*, and *slow*. Combine these with the name of your package, type them into your favorite search site, and you at least may be forewarned.

You also mentioned the forums and mailing lists for a project. Why didn't you read them first? Would you buy a house without having it inspected? Would you buy a used car sight-unseen? If not, then why would you try a piece of software without reading what its users have to say about it? While the Romans never had a word for *download*, software is as much subject to *caveat emptor* as anything else you might buy.

Finally, I would be very careful around any software that was part of a graduate student project. While many such projects result in complete systems, a significant number result in a system just good enough to get a degree, which is then dropped the moment the degree is conferred. As governments are starting to require that funded research projects put not only their papers but also their software online—as they should—I predict we will see a continued proliferation of such "almost-working" tools.

KV

Related articles on queue.acm.org

Open Source to the Core
Jordan Hubbard
http://queue.acm.org/detail.cfm?id=1005064

Open vs. Closed: Which Source is More Secure?
Richard Ford
http://queue.acm.org/detail.cfm?id=1217267

Is Open Source Right for You?
David Ascher
http://queue.acm.org/detail.cfm?id=1005065

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.

DOI:10.1145/2633039

Frank Levy and Richard J. Murnane

Viewpoint
Researching the Robot Revolution

Considering a program for cross-disciplinary research between computer scientists and economists studying the effects of computers on work.

In 1960, scientists established that atmospheric concentrations of CO_2 were rising. Despite this advance in knowledge, the understanding of global warming later remained quite muddled even a decade later. Historian of science Spencer Weart summarizes the situation. "In the early 1970s, the rise of environmentalism raised public doubts about the benefits of human activity for the planet. Curiosity about climate turned into anxious concern. Alongside the greenhouse effect, some scientists pointed out that human activity was putting dust and smog particles into the atmosphere, where they could block sunlight and cool the world. Moreover, analysis of Northern Hemisphere weather statistics showed that a cooling trend had begun in the 1940s. The mass media (to the limited extent they covered the issue) were confused, sometimes predicting a balmy globe with coastal areas flooded as the ice caps melted, sometimes warning of the prospect of a catastrophic new ice age."[a]

a See http://www.aip.org/history/climate/summary.htm.

Our current understanding of the Robot Revolution is equivalent to the understanding of global warming circa the 1970s. We know some things with certainty. Computers are eliminating jobs involving structured tasks in manufacturing, clerical work, and some other "mid-skill" occupations. Computers are creating jobs in some occupations, particularly for the technically skilled. Computerized work, unlike global warming, should increase GDP. Beyond these facts lies a broad landscape of speculation and spin. We do not know much about the Robot Revolution's net



PHOTOGRAPH BY SIMON DAWSON/BLOOMBERG VIA GETTY IMAGES

An employee monitors an automated production line at an Argos goods distribution center in the U.K.

Association for Computing Machinery

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

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

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

For available titles and ordering info, visit: librarians.acm.org/pod



effect on employment and wages just as we do not know much about the speed at which the revolution is proceeding. As Weart might say, the mass media (and the rest of us) are confused.

One cause of our confusion is the absence of sustained research of the type that eventually clarified our understanding of global warming. Given the potential disruption of rapidly advancing computerized work, this research program is needed badly. What follows are some thoughts on how such a research program could be advanced including obstacles that must be overcome. Our observations draw in part from our experience in 10 lunch meetings at MIT over the past two years that have brought economists and computer scientists together for the purpose of mutual education.

Just as global warming research has required multiple disciplines, researching the Robot Revolution will require cross-disciplinary learning beginning with computer scientists and economists. Current writing on the revolution is weak in part because most economists know less than they should about artificial intelligence while computer scientists have knowledge deficits in economics.

Two examples make the point. Early in the MIT lunch series, an economist asked what computer scientists meant when they said "computers lack common sense." A member of the computer science faculty responded with a lucid explanation of the common sense problem—the thousands of apparently trivial facts that humans use to perform tasks and that are difficult to capture in software. At that lunch were six economists, five of whom (including us) had written extensively on the impact of computers on work. A reasonable estimate is that no more than one of the economists had previously known of the common sense problem. None had used it in their writings.

The main obstacle facing cross-disciplinary research is an asymmetric incentive structure.

Similarly, in a recent *Communications* Viewpoint, software developer Martin Ford discusses one possible impact of the Robot Revolution: "Entry-level positions are especially vulnerable, and this may have something to do with the fact that wages for new college graduates have actually been declining over the past decade, while up to 50% of new graduates are forced to take jobs that do not require a college degree."[3]

An economist would argue that inferring causality is not so simple. Over the last decade, computers have been spreading through the economy and, as Ford stated, wages for recent college graduates have been declining. But other things have happened over the last decade including the 2008 financial collapse and subsequent deep recession. History shows that recessions that follow a financial collapse last an average of five years[4] and all recessions are particularly hard on new labor market entrants. A rigorous estimate of computers' effects on recent college graduates would have to first control for the recession's effects.

The main obstacle facing cross-disciplinary research is an asymmetric incentive structure. Economists are paid to study the factors (including computers) that affect labor markets. They know who funds such research, the seminars and conferences where they can present their results, and the journals that will publish the papers they write. To our knowledge, a computer scientist who studies the labor market effects of the Robot Revolution has no such infrastructure.

To the contrary, time taken to study the labor market impacts of computerized work is time taken away from career-advancing activities. Because of this asymmetry, computer scientists who are interested in the impacts of technological advances on employment are likely to be dispersed across institutions rather than concentrated in the way that machine vision groups or natural language processing groups

exist in many computer science departments. We return to this point later.

One might wonder whether a second obstacle is unwillingness among computer scientists to consider the potentially negative employment consequences of their work. Economists can offer some comfort on this issue. In the wake of the financial collapse, economists took significant criticism for promoting the deregulation and financial instruments that led to the collapse not to mention failing to predict the collapse in the first place. In practice, the profession shrugged off the criticisms[b] and continued to pursue problems they found interesting.

It is easy to imagine a collaborative research program beginning with conversations among researchers leading to case studies that generate hypotheses, moving on to econometric analysis and simulations—research designed to get a clearer sense of the future. The question is how to start. More precisely, the question is how to identify and connect people who want to do the work.

Identifying interested economists is straightforward. Virtually all economists who work on the Robot Revolution belong to one of three program groups in the National Bureau of Economic Research (NBER)—Labor Studies, Economics of Education, and Productivity, Innovation and Entrepreneurship—and could be reached quickly through the NBER structure. Identifying interested computer scientists is more difficult since, as we argued, interested individuals are likely to be geographically dispersed and none of the ACM Special Interest groups is directly in this area. We need to send up a flare so that interested computer scientists can respond and be put into contact with each other and with like-minded economists.

A logical candidate for this "flare" would be an *ACM*-sponsored symposium on the employment effects—positive and negative—of computerized work. The call for abstracts would be disseminated to computer scientists through both *Communications* and other ACM channels. The call would be disseminated to economists through the program group email lists within the NBER.

Computer scientists who are interested in the impacts of technological advances on employment are likely to be dispersed across institutions.

The symposium would have two goals: to serve as a point of contact for interested researchers and to begin to set standards in a new but important field. As such, the symposium's goal should be to advance research rather than raw speculation. It is currently too early at this stage to expect collaborative research between computer scientists and economists—we are still at the introduction phase—but at least it is possible to require that submissions be grounded in the authors' expertise and focus on near term (or past) developments rather than project far into the future.

For an economist, an example of grounded relevant research is an econometric study estimating the employment or wage effects of one or more technological advances.[1] This research is historical rather than forward looking but it would lay out the economist's techniques of model building and data handling in ways that would help inform future collaborative work. A different example is a case study of the implementation of a particular technology[2] that can be used to generate hypotheses for future work.

For the computer scientist, an example of relevant research would be the dissection of a technology that, if fully developed, could have significant impact on labor markets. Dissection would include discussion of the obstacles to full development and the author's assessment of future progress on those obstacles. What navigation and recognition problems must be solved to achieve a fully autonomous vehicle?[c] What improvements in natural language processing and information retrieval are needed to outperform foreign call centers in answering customer service calls for particular domains?

With these examples in mind, a selection committee of computer scientists and economists would be chosen to write a call for abstracts and to select among the submissions. Presented papers would be discussed by both a computer scientist and an economist. Assuming the revised papers were of sufficient quality, short versions could be published in *Communications* while full versions could be posted on an associated website or published in another venue. Both the symposium and website could also serve as a meeting point at which interested computer scientists and economists could post contact information and interests and begin correspondence pointing toward collaborative work. Based on informal conversations, there appears to be significant foundation interest in funding a conference of this kind.

Over the next quarter-century, global warming and the diffusion of computerized work each has a potential to profoundly change society. It is past time that research efforts start to reflect their equally disruptive powers. C

b See, for example, the symposium on "The Evolution of Financial Technology," a part of MIT's 150th Anniversary, http://mit150.mit.edu/symposia/economics#video.

c A workshop on these topics was held this past July as part of the 2014 Robotics: Science and Systems Conference in Berkeley, CA.

References

1. Autor, D.H., Dorn, D. and Hanson, G.H. Untangling trade and technology: Evidence from local labor markets. Working Paper, Department of Economics, MIT (Mar. 2013); http://economics.mit.edu/files/8763.
2. Autor, D.H., Levy, F. and Murnane, R.J. Upstairs, downstairs: Computers and skills on two floors of a large bank. *Industrial and Labor Relations Review 55*, 3 (Apr. 2002), 432–437.
3. Ford, M. Could artificial intelligence create an unemployment crisis? *Commun. ACM 56*, 7 (July 2013); DOI: 10.1145/2483852.2483865.
4. Reinhart, C.M. and Rogoff, K. *This Time Is Different: Eight Centuries of Financial Folly.* Princeton University Press, 2009.

Frank Levy (flevy@MIT.EDU) is Rose Professor Emeritus at the Massachusetts Institute of Technology in Cambridge, MA.

Richard J. Murnane (Richard_Murnane@Harvard.edu) is Thompson Professor at the Harvard Graduate School of Education.

Copyright held by authors.

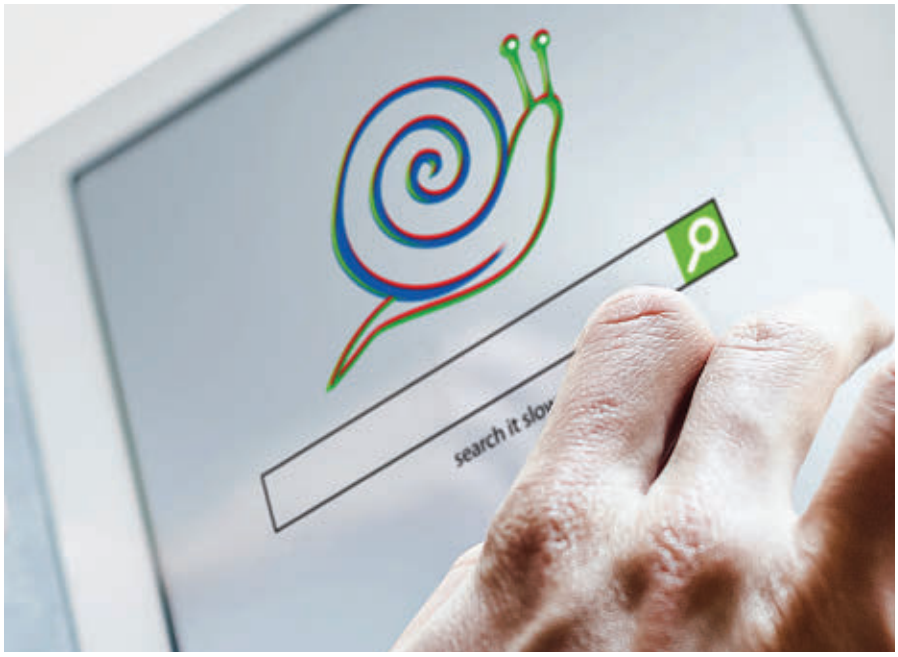
DOI:10.1145/2633041

Jaime Teevan, Kevyn Collins-Thompson, Ryen W. White, and Susan Dumais

Viewpoint
Slow Search

Seeking to enrich the search experience by allowing for extra time and alternate resources.

We live in a world where the pace of everything from communication to transportation is getting faster. In recent years a number of "slow movements" have emerged that advocate for reducing speed in exchange for increasing quality. These include the slow food movement, slow parenting, slow travel, and even slow science. Building on these movements we propose the concept of *slow search*, where search engines use additional time to provide a higher quality search experience than is possible given conventional time constraints.



Speed, Speed, Speed

Substantial research and engineering effort has been devoted to achieving low latency in large, complex computing systems such as search engines.[6] Search engines target speed for good reason. Research suggests that people perceive results that are delivered quickly as higher quality and more engaging than those delivered more slowly. Online experiments where server-side delays are injected into the delivery of search results have shown negative impact on people's search behavior.[11,12] For example, Google reported that intentionally increasing the load time of their search result page by as little as 100 milliseconds decreased the number of searches per person. Further, these differences increased over time and persisted even after the delays were removed. In similar experiments, Bing observed that artificial delays lead to a decrease in the number of queries and clicks, and an increase in time to click. Even improvements that seem like they should positively impact the searcher experience have been shown to have negative outcomes if they increase latency. For example, when Google experimented with returning 30 results instead of 10, they found that the number of searches and revenue dropped significantly because the additional results took a half-second longer to load.[10]

To achieve near-instantaneous speed, search engines make a number of compromises. They limit the complexity of the features and models used to identify relevant documents by, for example, making simplistic assumptions about language, often treating text as an unordered "bag of words." The resulting fast, word-oriented matching ignores the rich semantics of text but is an efficient way to capture some aspects of the similarity between queries and documents. Time-saving mechanisms such as search-result caching and index tiering are also heavily exploited, despite the risk that such approaches may cause relevant content to be missed.

Not All Searches Need to Be Fast

Although searchers have grown accustomed to rapid responses to their queries, recent advances in our understanding of how people search suggest there are scenarios where a search engine could take significantly longer than a fraction of a second to return relevant content.[12] While someone search-

ILLUSTRATIONS BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

ing for a specific website or a straightforward fact almost certainly wants immediate results, people often invest minutes, hours, or even days in more complex or exploratory search tasks. A person planning a vacation or researching a medical diagnosis, for example, may be willing to wait for better results or insights. Additionally, since it is now possible to predict if an individual will resume a search task at a later date,[8] slow search tools could make use of the time between sessions to produce high-quality search results that could then be presented immediately when a search task is resumed.

Slow search approaches are valuable and often necessary when people have intermittent, slow, or expensive network connections. In such cases it can be difficult for searchers to employ traditional search strategies, such as rapidly reformulating queries. Instead, successful search systems must provide mechanisms that enable the systems to make the best use possible of the available time. For example, RuralCafe helps searchers in rural regions limit the number of iterations that they need to make by providing an expanded query interface and performing additional post-query processing which affects the type and richness of the response.[4] Mobile phones also often have limited bandwidth, and slower search processing times may be acceptable given that most of the delay a searcher observes is caused by network latencies in fetching data to the device.[9] Likewise, future space travelers may appreciate slow search. It takes over 25 minutes for information to travel from Mars to Earth and back again. If a search engine were to take an additional few minutes to identify better results during the round trip, it is unlikely the searcher would even notice the extra time invested.

Supporting Slow Search

Slow search techniques can be used to improve search quality over the course of seconds, minutes, or longer. With even just a little extra time to invest, search engines can relax existing restrictions to improve search result quality. For example, complex query processing can be done to identify key concepts in the query, and multiple queries derived from the initial query can be issued to broaden the set of candidate documents to cover different aspects of the query.[5] Search engines can also make use of additional time to employ resources that are inherently slow, such as other people. Crowd-based ranking methods use human judgments to identify the most relevant existing content for a query. For example, a Korean question-answering service Jisiklog allows mobile searchers to submit questions via SMS and receive responses generated using crowdsourcing. Responses take minutes rather than seconds, but despite the wait, people are willing to pay for this service because of the quality of the responses.[9]

Slow search techniques can be used to improve search quality over the course of seconds, minutes, or longer.

While additional time can be used to identify particularly relevant results within the existing search engine framework, it can also be used to create new search artifacts and enable previously unimaginable user experiences. For example, researchers have used crowdsourcing to extract textual content from search results and synthesize that content into an inline answer for display on the result page.[3] Instead of merely returning a list of links, slow search results can include a summary or synthesis of the result content, the necessary background material to understand a topic, or the context necessary to resume an ongoing tasks. Rather than simply helping people find existing content, slow search systems can also facilitate the generation of new content that can be archived for use in future searches by the current searcher and others with similar interests.

Slow search systems require computational mechanisms to determine the appropriate method to employ given the time and resource constraints of a particular query. Recent research has examined tradeoffs between effectiveness and efficiency, and has developed

Calendar of Events

August 17–22
ACM SIGCOMM 2014 Conference,
Chicago, IL,
Sponsored: SIGCOMM,
Contact: Fabian E. Bustamante,
Email: fabianb@cs.northwestern.edu

August 21–22
Collaboration Across Boundaries: Culture, Distance & Technologies,
Kyoto, Japan,
Sponsored: SIGCHI,
Contact: Naomi Yamashita,
Email: yamashita.naomi@lab.ntt.co.jp

August 23–27
International Conference on Parallel Architectures and Compilation,
Edmonton, Canada,
Sponsored: SIGARCH,
Contact: Jose Nelson Amaral,
Email: jamaral@ualberta.ca

August 24–27
The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,
New York, NY,
Sponsored: SIGKDD & SIGMOD,
Contact: Claudia Perlich,
Email: Claudia.perlich@gmail.com

August 24–27
International Conference on Parallel Architectures and Compilation,
Edmonton, Canada,
Sponsored: SIGARCH,
Contact: Jose Nelson Amaral,
Email: jamaral@ualberta.ca

September 1–4
25th ACM Conference on Hypertext and Social Media,
Santiago, Chile,
Sponsored: SIGWEB,
Contact: Leo Ferres,
Email: leo@inf.udec.cl

September 1–5
27th Symposium on Integrated Circuits and Systems Design,
Aracaju, Brazil,
Sponsored: SIGDA,
Contact: Edward David Moreno,
Email: edwdavid@gmail.com

Coming Next Month in COMMUNICATIONS

Creating Moving Portraits

Optimal versus Optimized in Robot Motion

Online Deception in Social Media

Security, Cybercrime, and Scale

Exploratory Engineering in Artificial Intelligence

Soft Infrastructure Challenges to Scientific Knowledge Discovery

Q&A with David Blei, recipient of the 2013 ACM-Infosys Foundation Award in the Computing Sciences

Plus the latest news on how big data can bolster weather forecasting, the memory database evolution, and healthcare lite.

computational models to support different policies.[2] For example, cascade models for time-sensitive ranking degrade effectiveness gracefully under time constraints.[13] Devoting a significant amount of additional resources to create a better search experience can be expensive, but slow search also presents cost-saving opportunities. Search engines currently experience significant demand on resources during periods of high activity, and must develop infrastructure to handle peak loads. If not all query processing needed to happen immediately, this load could be distributed more evenly to leverage underutilized resources.

Slow Search Implications

Slow search will change the way that people experience search, including how they express what they are looking for and how they interact with the information that they find. One source of inspiration for the slow search experience is question asking. People regularly wait for high-quality answers to the questions they ask online.[1] Although most search engines represent a searcher's need using a single short query, slow searchers may learn to express needs more richly. In online social situations people provide long natural language explanations of what they are looking for. Rather than typing the query *vegetarian recipe*, people provide context and detail that can be leveraged by slow search algorithms, asking questions such as, "Can anyone recommend a good spicy vegetarian recipe without tofu or mushrooms?" Additional context can also be identified implicitly, as is often done in personalized search. Slow search engines need to clearly communicate the status of the slow searches to searchers and help them understand the benefit of a delayed system response. These systems should notify users when new relevant content is available, and provide ways for searchers to interrupt a slow search if it appears to be heading off-track or the results are no longer required.

An important goal of slow search is to free searchers from the low-level processes of searching, allowing them to focus instead on task completion. While people could use the time a slow search engine spends searching on their behalf to perform other tasks, they could also use it to reflect on and learn about the topic of their search. Dörk et al.[7] suggest slowing down the search experience by encouraging people to view result content at different levels and deviate off-topic during the course of a search session. With additional time, search engines can help people comprehend the context of that information and learn what is necessary to fully understand it. A slower search process will not only allow search engines to identify and return the most relevant content, but may also enable searchers to get the most possible from the search experience.

Our hope is that slow search will inspire new and creative research into how the search experience can be enriched with a more nuanced notion of relevant search resources and time constraints. C

References

1. Aperjis, C., Huberman, B.A., and Wu, F. Human speed-accuracy tradeoffs in search. (Jan. 11, 2010); http://bit.ly/9bTehC.
2. Azari, D., Horvitz, E., Dumais, S., and Brill, E. Actions, answers, and uncertainty: A decision-making perspective on web-based question answering. *IP&M 40*, 5 (2004), 849–868.
3. Bernstein, M., Teevan, J., Dumais, S.T., Liebling, D., and Horvitz, E. Direct answers for search queries in the long tail. In *Proceedings of CHI 2012*, (237–246).
4. Chen, J., Subramanian, L., and Li, J. RuralCafe: Web search in the rural developing world. In *Proceedings of WWW 2009*, 411–420.
5. Crabtree, D., Andreae, P., and Gao X. Exploiting underrepresented query aspects for automatic query expansion. In *Proceedings of KDD 2007*, 191–200.
6. Dean, J. and Barroso, L.A. The tail at scale. *Commun. ACM 56*, 2 (Feb. 2013), 74–80.
7. Dörk, M., Bennett, P. and Davies, R. Taking our sweet time to search. In *Proceedings of the CHI 2013 Workshop on Changing Perspectives of Time in HCI*.
8. Kotov, A., Bennett, P., White, R.W., Dumais, S.T., and eevan, J. Modeling and analysis of cross-session search tasks. In *Proceedings of SIGIR 2011*, 5–14.
9. Lee, U., Kim, J., Yi, E., Sung, J., and Gerla, M. Analyzing answers in mobile pay-for-answer Q&A. In *Proceedings of CHI 2013*.
10. Linden, G. Marissa Mayer at Web 2.0. (Nov. 9, 2006); http://bit.ly/2J2amv.
11. Shurman, E. and Brutlag, J. Performance related changes and their searcher impact. Velocity 2009; http://oreil.ly/fTmYwz.
12. Teevan, J., Collins-Thompson, K., White, R.W., Dumais, S.T. and Kim, Y. Slow search: Information retrieval without time constraints. In *Proceedings of HCIR 2013*.
13. Wang, L., Lin, J., and Metzler, D. A cascade ranking model for efficient ranked retrieval. In *Proceedings of SIGIR 2011*, 105–114.

Jaime Teevan (teevan@microsoft.com) is a senior researcher at Microsoft Research and an affiliate professor at the University of Washington, Seattle, WA.

Kevyn Collins-Thompson (kevyn@umich.edu) is an associate professor at the University of Michigan, Ann Arbor, MI.

Ryen W. White (ryenw@microsoft.com) is a senior researcher at Microsoft Research, Redmond, WA.

Susan Dumais (sdumais@microsoft.com) is a distinguished scientist and deputy managing director at Microsoft Research, Redmond, WA.

Copyright held by authors.

November 16 - 19

Dresden, Germany

ITS 2014

Interactive Tabletops and Surfaces



We invite you to the

2014 ACM International Conference on Interactive Tabletops and Surfaces

in Dresden, Germany, where

Baroque Beauty meets Interactive Surfaces

Raimund Dachselt & Nicholas Graham (General Chairs)

its2014.org

ITS.2014

@ITS_2014

Association for Computing Machinery

SIGCHI special interest group computer human interaction

DOI:10.1145/2630787

Article development led by acmqueue
queue.acm.org

Many disparate use cases can be satisfied with a single storage system.

BY MARK CAVAGE AND DAVID PACHECO

Bringing Arbitrary Compute to Authoritative Data

WHILE THE TERM *big data* is vague enough to have lost much of its meaning, today's storage systems are growing more quickly and managing more data than ever before. Consumer devices generate large numbers of photos, videos, and other large digital assets. Machines are rapidly catching up to humans in data generation through extensive recording of system logs and metrics, as well as applications such as video capture and genome sequencing. Large datasets are now commonplace, and people increasingly want to run sophisticated analyses on the data. In this article, *big data* refers to a corpus of data large enough to benefit significantly from parallel computation across a fleet of systems, where the efficient orchestration of the computation is itself a considerable challenge.

The first problem with operating on big data is maintaining the infrastructure to store it durably and ensure its availability for computation, which may range from analytic query access to direct access over HTTP. While there is no universal solution to the storage problem, managing a storage system of record (that is, one hosting the primary copy of data that must never be lost) typically falls to enterprise storage solutions such as storage area networks (SANs). These solutions do not typically offer wide area network (WAN) access, however, and they often require extra infrastructure to ingest data from and export data to arbitrary clients; this is hard to scale with the data footprint.

Once a system of record is established, computation on large datasets often requires an extract-transform-load (ETL) step into a system that can actually perform the computation. Even when the original data format suffices, network-attached storage (NAS)- and SAN-based systems do not support in-place computation, instead necessitating an expensive copy over the network—essentially a nonstarter for petabyte-size datasets.

Finally, there is an important distinction between special-purpose and general-purpose compute interfaces, meaning the abstraction differences between a SQL-like interface and a Unix shell. While the former is powerful and the appropriate choice for many kinds of analyses, it is not appropriate for many ad hoc tasks such as video transcoding, compression, or other actions that work on unstructured or semi-structured data. Existing systems for distributed computation typically require users to understand a complex framework (abandoning tools they are familiar with) or use a special-purpose interface (which often raises the aforementioned data-copy problem each time a new interface is desired).

ILLUSTRATION BY SASHKIN



The goal of this article is to describe a general-purpose, distributed storage system that supports arbitrary computation on data at rest. It begins by detailing the constraints, which direct much of the design; then describes an implementation called Manta, which can be thought of as a reference implementation but is certainly not the only way such a system could be constructed. We conclude with the ways in which several common big data problems can be solved using Manta.

Constraints

The design constraints can be broadly divided into storage abstraction (how users store and fetch data), the programming model (how users express in-situ computation on the data), and how the system provides durable local storage (how the system durably stores bits on disk).

Storage abstraction: An object store. Building a system to scale means ensuring capacity can be increased by adding more hardware. Scaling without downtime and without increasing the impact of individual component failure requires scaling horizontally, but Posix file-system and block-storage semantics are very difficult to scale horizontally. Updates may be very small (a byte within a file system; a block of a few kilobytes within a block device) and frequent. Even without considering multiple clients, the trade-off is difficult between operation latency and durability: the more data the client is allowed to buffer before transmitting to the server, the faster operations may execute but the more data is at risk in the case of a crash. With multiple clients operating on the same file, a similar trade-off exists between operation latency and consistency.

Object storage provides a more constrained model that is simpler to scale. An object store resembles a file system in that it has a global namespace with objects (data blobs) and directories (hierarchical collections), but object stores do not support partial updates. Users can create new objects and delete old ones, but they cannot update the contents of an existing object without replacing it entirely. For strong consistency, an object-store implementation need ensure only that the namespace is transactional; objects themselves are immutable, and thus easily replicated and cached.

Programming model: Distributed computing. The MapReduce programming model[3] forms the foundation of several distributed computing systems—for good reason. Fundamentally, MapReduce divides distributed computing into the parts that transform individual pieces of data (*map* operations) and the parts that operate on lots of data at once (*reduce* operations). Both types of operations are transformations of input data that produce a set of output data. The original data is immutable, so there are no side effects. This has several nice properties for distributed computing:

- Users need to express only the parts of a computation that are specific to their problem, leaving the problem of data flow up to the system itself.
- Map operations are fully parallelizable. In practice, you can add hardware resources to increase parallelism almost linearly.
- In the event of a failure, map and reduce operations can be retried. There is no complex state to unwind after a failure—only output to ignore. Even in the event of a network partition when the system may not know whether an operation has failed, it can execute the operation more than once as long as it ignores the results of all but one copy.

Programming model: Local computing. MapReduce is a useful model for distributed computing, but the question remains: how do users specify what a map or reduce operation actually does? The preference is for an interface that users are already familiar with, that can leverage the enormous body of existing software, and that imposes as few constraints as possible. Given these priorities, there is a surprising (if obvious) solution in the Unix command-line interface (CLI).

At its core, the CLI provides a simple interface for arbitrary computing with a few extra primitives:

- File descriptors for reading and writing streams of input and output, along with conventions around the primary input, the primary output, and a stream for reporting other messages.
- Pipes, connecting the output of one program to the input of another, effectively composing a new program.

This approach encourages creating many small tools that compose well to run more sophisticated computations. This is perhaps best illustrated by an example from *Communications*' "Programming Pearls" series called "A Literate Program,"[1] in which the following problem is posed: "given a text file and an integer *k*, print the *k* most common words in the file (and the number of their occurrences) in decreasing frequency."

The article describes two solutions: first, a custom solution whose concise presentation (with explanation) spans seven pages; and second, a shell one-liner "written on the spot" that "worked on the first try" with a six-stage pipeline using `tr`, `sort`, `uniq`, and `sed`. The code and explanation total about a quarter of a page. This cannot be overstated: the shell-based solution is much faster for the author to construct and for subsequent readers to understand, modify, and compose with other programs.

Ironically, the canonical MapReduce example from section 2.1 of the original paper suggests a very similar problem: "Count the number of occurrences of each word in a large collection of documents." Much is to be gained from extending the Unix interface to work in a MapReduce-based distributed computing model, but there are several challenges.

- *Programming interface.* MapReduce usually operates on keys or tuples. While many Unix tools do operate on field-separated records, this is just a convention. Instead of MapReduce being applied to a large, uniform set of keys, it can be applied to *objects* in the storage system. To work with objects in a familiar way, they can be exposed as read-only files on stdin (standard input), as well as on the file system.

With the MapReduce computation expressed as a Unix pipeline, programs must be able to interact with the system as they can on any other Unix system, having free rein in an isolated operating-system container. They must have their own process namespace, file system, networking stack, and any other hardware and software resources visible to programs.

- *Multitenancy.* A production storage system must support concurrent computations for both a single user

and multiple users, and it should not require manually scheduling time on the system to ensure quality of service. Concurrent computations should share all available resources (possibly subject to administrative policies), and users should be allowed to program as if each is the only user on the system. They must not be able to see or interfere with one another.

▸ *Side effects.* Since Unix programs can read and write to the local file system, fork processes, make network connections, and so on, they decidedly do have side effects. This can be dealt with by saying that whatever a program does while it is running, the only way it can interact with the distributed computation is through its input and output streams. When the program completes, anything that would affect the environment for subsequent programs must by unwound, including file-system changes, processes created, and the like. This is achieved with operating system-based virtualization, using containers that provide isolation and resource controls and a file system that supports efficient rollback (discussed later).

Durable local storage. The desire to expose objects directly as files to user programs leads to the most significant difference between the system described here and existing object-storage systems, which is the entirety of each object must be stored on a single system. The object can be stored on multiple servers for additional availability and durability, but the entire contents must be present on each system in order to support our programming model. This in turn requires that the local file system store data durably. While that sounds simple, the reality of physical media means that the local storage system requires a number of features to manage storage effectively and ensure data durability.

Checksums outside data blocks. Data believed to be written to disk can turn out to be corrupted in many ways when it is later read back, including corruption at the disk controller, in cables, or drive firmware; accidental writes directly to the disk; environmental effects resulting in changes to bits on disk after they were written correctly (bit rot); bugs in firmware resulting in dropped writes (phantom writes) or misdirected writes; and so on. While block checksums protect against bit rot, they cannot detect a well-formed block written to the wrong place (as may happen with accidental overwrites, misdirected writes, or phantom writes). The file system must checksum data and store checksums separately from data blocks in order to detect that a given block is not only valid but also represents the data expected in that block.

The MapReduce programming model forms the foundation of several distributed computing systems—for good reason.

Double-parity (or better) software-based raid. To get appropriate durability for each copy of an object at a reasonable cost, each node's storage should itself be redundant. A single spindle failure should not result in reduced availability or durability for objects on that server. To reconstruct bad copies of data blocks (detected with the just-described file system-managed checksums), the file system must be involved in the data reconstruction process, which implies software-based RAID (redundant array of independent disks).

Copy-on-write. Copy-on-write file systems support O(1) point-in-time snapshots and efficient rollback to previous snapshots. This is a critical feature for unwinding changes made by user programs so that other users' programs can run later in the same container.

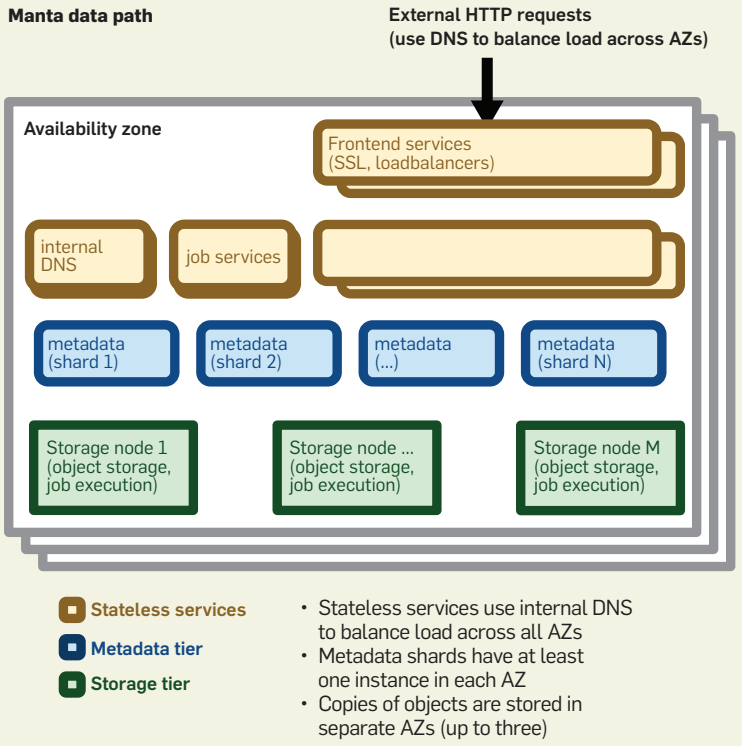
Pooled storage. Storage servers must provide temporary scratch space for user programs. Isolation demands that users get their own file systems, and the desire to support many tenants with flexible resource controls demands that these file systems be resized dynamically. As such, file systems should not be laid out directly on disks but rather should be flexibly allocated from a pool of storage.

Separate intent log. The economics of a large-scale storage system demands traditional disks for primary storage, but write performance can be improved significantly with SSDs (solid-state drives) as intent log devices. Streaming writes can leverage the considerable throughput of a large pool of spindles while synchronous writes (usually smaller) can be committed to the SSD and acked to the client before they make it to the spindles.

Additional Design Goals

In addition to these hard requirements, the system needs several prop-

Figure 1. Manta services.



erties for consistency, availability, and durability.

CAP. The CAP theorem[2] specifies that in the face of network *partitions* (which will always happen), data services must choose between *consistency* and *availability*. The trend in distributed systems has been skewed toward availability in recent years, and while choosing availability (eventual consistency) is often the right choice for an application, choosing a weakly consistent model for a general-purpose storage system means that no application built on top of that storage system can ever be strongly consistent. Conversely, if a low-level storage system is strongly consistent, an application can always take measures to choose availability by implementing asynchronous writes (staging) and caching reads. For these reasons, a storage system of record must be strongly consistent.

Availability and failure modes. Choosing strong consistency does not mean that a system should abandon high availability. The system should survive failure of any single component, including an individual service instance, a physical server, a rack, or an entire data center. Every component must be replicated and scaled out across three or more data centers. In the classic 2F+1 failure model,[7] the system should support continuing operations with a single data center offline. (There may be a short reconvergence time, during which writes would be rejected, but afterward the system will continue operating without degradation.)

Data access and durability. Data should be exposed over a REST (representational state transfer) API. When the system returns `HTTP/1.1 200 OK` after a write, copies must exist on two physical servers, and copies of the metadata about the actual data must exist on two physical servers. These copies must also span data centers. As mentioned earlier, each physical copy must be resilient to individual drive failures.

Implementation

Manta (Figure 1) is Joyent's implementation of the storage system whose design goals have just been described. While the architecture mostly falls straight out of the constraints and goals described, the implementation deals with several important details.

Storage. At a high level, users interact with a scalable Web API that is stateless and load balanced. Metadata about objects is stored in an application-partitioned database tier, and objects themselves are written synchronously to two or more storage nodes across the fleet.

Front end. To maintain high availability, most Manta components are stateless. The front end consists of a group of load balancers using HAProxy and custom API servers written in Node.js. Users' TLS (Transport Layer Security) sessions are terminated at the load balancers, and HTTP requests are handled by the API servers. There is also a read-only authentication/authorization cache backed by Redis. Every data center has at least one instance of each component, and load balancing across the load balancers themselves is handled coarsely using round-robin DNS. Since all of these components are stateless, new instances can be provisioned for additional availability or capacity.

Write requests are logically broken down into the following steps:

1. Ensure the parent path (directory) exists.
2. Select a set of N storage nodes on which to store data. By default, $N=2$, but users can specify more or fewer copies. Each copy is highly durable (as already described), but copies are stored in separate data centers, so having more copies results in increased availability.
3. Synchronously stream data to the selected storage nodes.
4. Record durable metadata (indexed by the object's user-visible name) indicating which storage nodes contain the data.

Read requests are broken down into a similar series of steps:

1. Look up the requested object's metadata, which includes the set of storage nodes that have a copy of the object.
2. In parallel, contact each storage node hosting a copy of the data.
3. Stream data from the first node that responds.

Deletes and overwrites are processed by a background garbage-collection system. When a delete (or overwrite) request comes in, Manta records the event in a durable transaction log. The garbage-collection system later de-

termines which of these objects are no longer referenced by the metadata tier and reclaims the corresponding space on the storage nodes.

Metadata. According to the design constraints, object metadata must be stored durably and modified transactionally, but still highly available and horizontally scalable. Manta divides metadata storage and access into three components:

1. A data-storage and replication engine.
2. A key/value interface that fronts (1).
3. A sharding layer for (2) to support horizontal scalability.

Data storage and replication. Since the metadata is highly structured, a classic B-tree representation is sufficient to support transactional updates and fast queries on indexed properties. To maintain availability, the system must also support synchronous replication to a secondary instance. After debating whether to build our own or use an off-the-shelf solution, we decided to use PostgreSQL, whose storage subsystem is proven and whose replication system satisfies these requirements. What PostgreSQL lacks is a built-in system to ensure automatic failover using leader election. We wrote software that leverages a consensus layer (Zookeeper) to manage a daisy chain of replicas. When a PostgreSQL peer crashes or exits the fleet, the peer that was directly behind it is promoted in the topology:

```
Master -> Synchronous Peer ->
Async -> Async -> ...
```

Further, the leader records the topology in a well-known place, and the key/value layer (through which all clients access the database) knows how to use this information.

Moray: Key/value interface. On top of PostgreSQL is a thin key/value interface called Moray, which provides three functions:

- Knowledge of the underlying database-replication topology to ensure that writes are always directed to the PostgreSQL master.
- A simple interface without explicit transactions that supports optimistic concurrency control.
- A good-enough query interface using Lightweight Directory Access Protocol (LDAP) search filters.

For programmer simplicity, a `PUT/GET/DELETE` paradigm makes the key/value interface abstractions *buckets*, *keys* and *objects*, where objects are always JavaScript Object Notation (JSON) documents. Indexing rules are defined for each bucket such that search requests are able to look for objects that match a filter evaluated on the JSON objects. Simple lookups are done directly by the bucket/key pair.

The Node.js code in Figure 2 demonstrates both the API and semantics. This system is the only interface that other components in Manta use to store state. All metadata about user objects, storage node utilization, and compute job state is stored in PostgreSQL through Moray.

Electric Moray: Key/value interface. A collection of at least three replicated PostgreSQL instances plus one or more Moray instances is called a *shard*. At any given time, only one of the PostgreSQL instances in a shard is used for reads and writes, and the Moray instances direct all key-value operations to that instance. Durability and availability are provided within each shard, but in order to scale horizontally (that is, to increase the storage footprint or read/write capacity of this system beyond what a single instance can provide), Manta uses several shards and performs application-level partitioning among them using a consistent hashing scheme.[6]

Recall that the user-facing API consists of directories and objects that behave largely like Posix directories and files. As with a Posix file system, in order to write an object, all parent directories must exist. The following CLI commands should look familiar:

```
$ mmkdir /mark/stor/foo/
$ echo "Hello, ACM" | mput /
mark/stor/foo/bar.txt
```

To determine which shard holds the metadata for a given object, Manta performs a consistent hash operation on the directory name of the object. For example, given `/mark/stor/foo/bar.txt`, Manta hashes the string `/mark/stor/foo`. Metadata for all objects in a directory exists on a single shard.

The system that manages the mapping between object names and shards is called Electric Moray. It offers the same interface as Moray (and ultimately directs all requests to a Moray instance) but deals with routing requests to the right shard.

Storage server. Storage nodes are primarily responsible for maintaining durability of a single copy of an object and providing the isolated compute

Figure 2. Node.js code showing metadata tier internal API and semantics.

```
// Create a bucket named "user_directory", and ensure that any objects
// written to it are inspected for the attributes "id", "name", and "email";
// if any are present, maintain an index on those attributes for future
// search requests
moray.putBucket('user_directory', {
    index: {
        id: 'number',
        name: 'string',
        email: 'string'
    }
});

// Write a sample user object to "user_directory" with the key
// "mcavage" and value where there are both indexed and non-indexed
// fields
moray.putObject('user_directory', 'mcavage', {
    id: 123,
    name: 'Mark Cavage',
    email: 'mark.cavage@joyent.com',
    department: "engineering"
});

// Search for any objects that have an email address of "@joyent.com"
var res = moray.search('user_directory', '(email=*@joyent.com)');
res.on('record', function (obj) {
    console.log('Record found: %s', JSON.stringify(obj));
});
```

containers in which to run user programs. For storage, these nodes provide a straightforward HTTP interface over the local disks, which are managed by the ZFS file system. Objects are written using a random (UUIDv4) name. On a write, a successful status code is returned to the calling operation only when `fsync(2)` has completed and the data is known to be safely on disk.

Manta leverages ZFS to solve the local durability problem. This satisfies all of the constraints described previously, including checksums to detect all forms of block-level corruption, software RAID that automatically detects and repairs such corruption as data is read, and pooled storage to support dynamically resizing file systems to satisfy user requests for additional scratch space. The triple-parity software RAID implementation can sustain steady-state operation (including writes) in the face of up to three concurrent disk failures, though systems are configured with double parity for better cost efficiency.

Compute. Users run computation in Manta by submitting *jobs*. To support a MapReduce-like model, jobs are made up of one or more *phases*, each of which is either a map phase or a reduce phase. Map phases are divided into *tasks*—one for each input object. Users can specify how many reduce tasks they want for a given reduce phase.

An important design goal is that jobs should be able to operate on an arbitrary number of objects, with an arbitrary number of tasks, producing an arbitrary number of errors or outputs, meaning these quantities should scale with the resources of the system rather than being limited for algorithmic reasons. As a concrete example, Figure 3 shows a map-only job. To achieve this, the interfaces to list inputs, outputs, and errors are necessarily streaming and paginated, rather than providing complete listings in a single request. Internally, inputs, outputs, errors, and tasks are operated on in groups, and it must never be required for all of them (or any particular group of them) to be stored in memory.

As a concrete example, here is a map-only job that searches for instances of a given URL (`/login`) in a set of Apache-format request logs named in the local file `inputs.txt`. A user would run this from the local system.

An important design goal is that jobs should be able to operate on an arbitrary number of objects, with an arbitrary number of tasks, producing an arbitrary number of errors or outputs.

The `mjob create` client command runs a new Manta job. The `-m` option specifies a map phase defined by the following script. The `-o` option causes the command to wait until the job completes and then print the contents of the job's outputs. This is the closest analog to running a command locally in the shell, except that it operates in parallel and the `grep` actually runs on the storage server. The `inputs.txt` file contains a list of objects to run the job on. Since this is a map job, the system will run `grep -w /login` for each object in inputs.txt, automatically parallelizing as best it can.

Figure 4 illustrates what the job looks like in the API. When it finishes, `state` will be done, and `tasksDone` will equal `tasks`. Executing this job is done in two parts: the distributed orchestration of the job and the execution of the user's `grep` script on each object.

Job orchestration. When the user runs `mjob create`, the Manta Web service receives client requests to create the job, to add 240 inputs to the job, and to indicate there will be no more job inputs. The job is assigned to a particular job supervisor, which is responsible for coordinating the distributed execution of the job. As inputs are added to the job, the supervisor resolves each object's name to the internal universally unique identifier (uuid) that identifies the object and checks whether the user is allowed to access that object. Assuming the user is authorized for that object, the supervisor locates all copies of the object in the fleet, selects one, and issues a task to an *agent* running on the server where that copy is stored. This process is repeated for each input object, distributing work across the fleet.

The agent on the storage server accepts the task and runs the user's script in an isolated compute container. It records any outputs emitted as part of executing the script. When the task has finished running, the agent marks it completed. (This process is described later in more detail.)

The supervisor *commits* the completed task, marking its outputs as final job outputs. When there are no more unprocessed inputs and no un-

committed tasks, the supervisor declares the job done.

If a task fails, it will be retried a few times, preferably on different servers. If it keeps failing, an error is produced.

Multiphase map jobs are similar except that the outputs of each first-phase map task become inputs to new second-phase map tasks, and only the outputs of the second phase become outputs of the job.

Reduce tasks. Reducers run similarly to mappers, except the input for a reducer is not completely known until the previous phase has already completed. Also, reducers can read an arbitrary number of inputs so the inputs themselves are dispatched as individual records, and a separate end-of-input must be issued before the reducer can complete.

Suppose you want to modify the previous example to report the number of times each IP address requested `/sample.txt`. You could change the map phase to emit only the column of IP addresses and have the reduce phase count the number of distinct values (see Figure 5).

Notice the two parts of this problem are exactly what you would run to solve the same problem on a single system using the shell. The distributed version could actually apply the reducer with the mapper, and replace this reducer with an `awk` script that sums the values per column, but this performance optimization is often not necessary.

For this two-phase job, the first phase works as described. For the reduce phase, a single reduce task is issued to an agent somewhere in the fleet when the job is created. As map tasks complete, their outputs (themselves temporary Manta objects) are marked as inputs to the reducer. The reducer emits a single output object, then gets marked completed. As before, the supervisor commits the result, producing the job's sole output object, and the job is marked done.

Local execution. The agent on each storage server maintains a fixed set of compute zones in which user scripts can be run. When a map task arrives, the agent locates the file representing the input object on the local file system, finds a free compute zone, maps the object into the local file system, and runs the user's script, redirecting stdin from the input file and stdout to a local file. When the script exits, assuming it succeeds, the output file is saved as an object in the object store, recorded as an output from the task, and the task is marked completed. If there is more work to be done for the same job, the agent may choose to run it in the same compute zone without cleaning up after the first one. When there is no more work to do, or the agent decides to repurpose the compute zone for another job, the compute zone is halted, the file system rolled back to its pristine state, and the zone is booted again to run the next task. Since the compute zones are isolated from one another and are fully rebooted and rolled back between jobs, there is no way for users' jobs to see or interfere with other jobs running in the system.

Internal communication within the compute network. The system uses the sharded, replicated database component (Moray/PostgreSQL shards) to manage job state. There are buckets for jobs, job inputs, tasks, task inputs (for reduce tasks), task outputs, and errors. Supervisors and agents poll for new records applicable to them. For example, supervisors poll for tasks assigned to them that have been completed but not committed, and agents poll for tasks assigned to them that have been dispatched but not accepted. When a job completes, the list of inputs, outputs, and errors are archived into Manta objects, and the corresponding database records are removed to limit the database to only current working state. The poll-based approach has obvious performance downsides, but the implementation

Figure 3. A simple one-phase map job.

```
    $ mjob create -o -m 'grep -w /login' < inputs.txt
    added 240 inputs to 4e55b83b-a04b-c25c-9169-d4b4641d915d
    10.10.0.34 - - [14/Apr/2014:11:07:48 +1000] "GET /login HTTP/1.0" 200 6433
"-" "nodejs"
    ...
```

Figure 4. Initial API state of the job from Figure 3.

```
{
      "id": "4e55b83b-a04b-c25c-9169-d4b4641d915d",
      "state": "running",
      "inputDone": true,
      "stats": {
        "errors": 0,
        "outputs": 0,
        "retries": 0,
        "tasks": 240,
        "tasksDone": 0
      },
      "phases": [ {
          "exec": "grep -w/login"
          "type": "map"
      } ],
      ...
    }
```

Figure 5. A more complex `grep` job.

```
  $ mjob create -o \
       -m 'grep -w /sample.txt | cut -d" " -f 1' \
       -r 'sort | uniq -c' < inputs.txt
    added 240 inputs to 4e55b83b-a04b-c25c-9169-d4b4641d915d
       135 10.10.0.34
         5 10.10.0.37
        23 10.10.0.33
    ...
```

is easy to understand, and the performance can be greatly improved without changing the basic design (for example, by reducing poll intervals and using push notification to trigger an immediate poll).

While this mechanism has proven reliable, it is also responsible for most of the systems latency from job input to the corresponding output (two to three seconds for a 500ms user program). The workload has also triggered pathological performance issues in the PostgreSQL layer, leading to brownout situations where jobs that normally take a few seconds to complete could take minutes. Now that they are understood, these pathologies can generally be avoided or mitigated.

Example: Log analysis for bandwidth and compute metering. Manta itself provides an example of a common use case for big-data systems in the form of continuous log analysis. Per-user metrics for bandwidth and compute usage are computed using MapReduce jobs operating over internal log files, which are themselves pushed into Manta every hour. These jobs take the internal log files as inputs, parse the log entries, count metrics such as bandwidth and compute time used, and produce per-user, per-hour access logs, as well as aggregated usage reports that are also used for billing. The reduce process is spread over two phases for increased parallelism. Having this critical use case in mind while building Manta greatly informed the design of the system.

Example: Customer cohort retention. One customer has presented extensively on a data-analytics system built on top of Manta.[5] It uses `rsyslog` to log user actions daily from more than 40 front-end Web servers. These logs, representing about five to 20 million events per day in ASCII, pipe-delimited files totaling about 1GB, are uploaded to Manta. Typical business questions include "How many unique users performed a given action from an iPad on a given date?" and "How many people who signed up four weeks ago are still active?" Because of their simple format, these questions are answered with 5- to 10-line scripts using only `grep(1)`, `awk(1)`, `sort(1)`, `uniq(1)`, `comm(1)`, and `wc(1)`. Results are generally available in seconds to one minute.

Example: Video analysis and transcoding. A number of customers have used Manta to transcode videos. We have also been doing this internally for a side project analyzing screen captures of Mario Kart 64 games (http://kartlytics.com/). The program to analyze videos was built atop of the FFmpeg suite of tools and libraries for video processing. The application is a typical MapReduce job that maps over video files and produces JSON files describing what happened in the video. The resulting files are combined to produce a single summary file, which presents all the results on a website. A separate map-only job transcodes the original, high-quality video files into much smaller, Web-quality WebM files. The entire corpus is analyzed in 10 minutes (fully parallelized across the Manta fleet), compared with the better part of a day on a single system, without having to modify the analysis software itself.

Conclusion

Unifying the storage system of record with in-situ computation and adapting the ubiquitous Unix shell environment to distributed computing has resulted in many disparate use cases being satisfied with a single storage system. These include traditional log analysis, as well as asset hosting (as for a content-delivery network), image manipulation (for example, on the same assets hosted for the Web), and video transcoding. Achieving this required unconventional design choices such as storing objects directly as raw files rather than erasure-encoded across multiple servers. While there are undoubtedly other ways to build such a system, they would necessarily require reliable local storage and mature operating system-based virtualization. While previous systems have either leveraged enterprise-grade *network* storage (which was viewed as a nonstarter because it requires moving data in order to compute on it) or built custom storage systems to work around otherwise flaky or inefficient local storage, using an architecture based on ZFS allowed us to focus on the distributed systems part of the problem and (we believe) produce a more general-purpose system.

There is a great deal of additional work to be done, including richer access controls, support for more software packages out of the box, and improved performance. But Joyent and its customers have already found the system valuable for analyzing both business and technical data. C

Related articles on queue.acm.org

Cloud Computing: An Overview
Mache Creeger
http://queue.acm.org/detail.cfm?id=1554608

A Co-Relational Model of Data for Large Shared Data Banks
Erik Meijer and Gavin Bierman
http://queue.acm.org/detail.cfm?id=1961297

Condos and Clouds
Pat Helland
http://queue.acm.org/detail.cfm?id=2398392

References

1. Bentley, J., Knuth, D. and McIlroy, D. Programming Pearls: a literate program. *Commun. ACM 29*, 6 (June 1986), 471–483; http://dl.acm.org/citation.cfm?id=315654.
2. Brewer, E.A. Toward robust distributed systems. Keynote speech. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, 2000; http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.
3. Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*, 2004; http://research.google.com/archive/mapreduce.html.
4. Fontaine, T.J. 550 regression tests in four minutes with Joyent Manta, 2013; http://www.joyent.com/blog/550-regression-tests-in-4-minutes-with-joyent-manta.
5. Gredeskoul, K. Using Manta to scale event-based data collection and analysis, 2013; http://www.slideshare.net/kigster/ss-26329742.
6. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, 654–663; http://dl.acm.org/citation.cfm?id=258660.
7. Lamport, L. Lower bounds for asynchronous consensus. Microsoft Research, 2006; http://research.microsoft.com/en-us/um/people/lamport/pubs/lower-bound.pdf.
8. Robbins, C. Keeping the npm Registry awesome, 2013; http://blog.nodejs.org/2013/11/26/npm-post-mortem/.

Mark Cavage is Joyent's vice president of engineering, where he oversees development of the company's core technologies: Node.js, SmartOS, SmartDataCenter, and Manta. He was previously a senior software engineer with Amazon Web Services, where he led the Identity and Access Management team.

David Pacheco is a software engineer at Joyent, where he works primarily on the Manta storage service. He also works on Joyent's SmartDataCenter product and tools for tracing and debugging Node.js programs in production.

Copyright is held by owners/authors. Publication rights licensed to ACM. $15.00.

DOI:10.1145/2631095

Article development led by acmqueue
queue.acm.org

How to generate funding for free and open source software.

BY POUL-HENNING KAMP

Quality Software Costs Money—Heartbleed Was Free

THE WORLD RUNS on free and open source software, FOSS for short, and to some degree it has predictably infiltrated just about any software-based product anywhere in the world.

What's not to like about FOSS? Ready-to-run source code, ready to download, no license payments—just take it and run. There may be some fine print in the license to comply with, but nothing too onerous or burdensome.

Your TV? It has a Linux or {Net|Free} BSD computer inside it. So does the copier-printer/multifunction machine at the office, as does the entertainment console in your car and in your kids' bedrooms. Code reuse has finally happened, but there is a footnote: you get what you pay for.

Earlier this year the OpenSSL Heartbleed bug laid waste to Internet security, and there are still hundreds of thousands of embedded devices of all kinds—probably your television among them—that have not and will not ever be software-upgraded to fix it. The best way to prevent that from happening again is to avoid having bugs of that kind go undiscovered for several years, and the only way to avoid that is to have competent people paying attention to the software.

In the case of OpenSSL, it is painfully obvious that nobody was paying attention. There is a commercial entity called The OpenSSL Foundation, but as far as anyone can determine, it is a Federal Information Processing Standard (FIPS)-test consultancy and that is pretty much all it does.

According to the *Wall Street Journal*,[5] the OpenSSL Foundation received

around $2,000 a year in donations to maintain the OpenSSL code. That amounts to approximately 20 hours of attention per year from a good programmer to maintain north of a half-million lines of code. That certainly explains why the OpenSSL bug tracker was write only and why the code is still overcomplicated by support for vintage operating systems such as VMS and 16-bit Windows.

Predictably, money started pouring in after the Heartbleed bug caused havoc: IT security is a game you can win only by losing first. So OpenSSL maintenance, testing, and quality assurance is funded now, and we are good, right? No, we are not! Many other FOSS projects (in addition to OpenSSL) are badly understaffed because they are underfunded—and we need to fix that.

Why FOSS Needs Money

About the only thing GNU Project founder Richard Stallman and I can agree on when it comes to software freedom is that it's "Free as in free speech, not free beer."

I really hope the Heartbleed vulnerability helps bring home the message to other communities that FOSS does not materialize out of empty space; it is written by people. We love what we do, which is why I am sitting here, way past midnight on a Saturday evening, writing about it; but we are also real people with kids, cars, mortgages, leaky roofs, sick pets, infirm parents, and all kinds of other perfectly normal worries.

The only way to improve the quality of FOSS is to make it possible for these perfectly normal people to spend time on it. They need time to review patch submissions carefully, to write and run test cases, to respond to and fix bug reports, to code, and most of all, time just to think about the code and what should happen to it.

It would not even be close to morally defensible to ask these people to forgo time to play with their kids or walk their dogs in order to develop and maintain the software that drives the profit in other people's companies. The right way to go—the moral way to go—and by far the most productive way to go is to pay the developers so they can make a living from the software they love. And not just for a month or two: good programmers, like most everyone else, prefer stable jobs so they can concentrate on the job rather than wasting time chasing the next gig or the next sponsor.

How to Fund FOSS

One way to fund FOSS is simply to hire the FOSS maintainers, with the understanding they spend some amount of company time and resources on the project. Experience has shown these people almost invariably have highly desirable brains, which employers love to throw at all sorts of interesting problems, and that tends to erode the "donated" company time. A lot of FOSS has been, and still is, developed and maintained this way, with or without written agreements or even company knowledge of this being the case.

A big thank you to those employers who do this deliberately!

These companies should add their support of FOSS to their lists of corporate social responsibilities, along with their sponsorships of local soccer teams and their funding of scholarships. They are doing something for the greater common good, which they should be proud of and get proper credit for.

Another way to fund FOSS is for software projects to set up foundations to collect money and hire developers. This is a relatively complex and expensive undertaking. You run straight into the murkiest corners of tax codes, so this approach is available only for larger projects. While several projects have gone this route, their ability to raise money varies significantly.

The Apache Foundation is probably the largest of all these FOSS foundations. A few entities "adopt" smaller projects inside their fields of interest. This generally works OK but cannot easily be generalized to different areas.

An easier and cheaper way, the one advocated here, is simply to throw money at the developers, the way the FreeBSD and Varnish communities have done with me. Forget about tax deductions and all that; simply have the developer send your company an invoice every so often and stuff it into some account in the IT department labeled "misc. software licenses" or "expert assistance" or whatever will fly under the radar in your company. It takes only a dozen companies worldwide to keep a top-notch programmer's nose in the code of a FOSS project full time—and this is probably code that is likely a lot more critical to a company's operations than anybody really likes to think about.

Here is my story...

FreeBSD Community Funding

My first crowdfunding of FOSS was in 2004 when I solicited the FreeBSD community for money[3] so that I could devote three to six months of my time to the FreeBSD disk-I/O subsystem.

At the time I had spent 10 years as one of the central and key FreeBSD developers, so there were no questions about my ability or suitability for the task at hand. In 2004, however, crowdfunding was not yet "in," and I had to figure out how to do it myself. My parents raised me to believe that finances are a private matter, but I concluded that the only way you could reasonably ask strangers to throw money at you would be to run an open book where they could see what happened to the money. So I opened my books.

The next dilemma was my rate. Again, I had always perceived my rate to be a private matter between me and my customers. Because my rate is about half of what most people expect (as I will not work for most projects but only on things I really *care* about), I was concerned that publishing my rate would undercut friends and colleagues in the FreeBSD project who made a living consulting. There was no way around it, however, so I published my rate, making every attempt to distinguish it from a regular consulting rate, and I never heard any complaints.

Having agonized over the exact text and testing it on a couple of close friends in the FreeBSD project, I threw the proposal out there and waited. I had a perfectly safe fallback plan—you have to when you have two kids and a mortgage—but I really had no idea what would happen next.

Worst case, I would cause the mother of all bikesheds (http://bikeshed.org/[2]) to get thrown out of the FreeBSD project, and I would be widely denounced for my "ideological impurity" with respect to FOSS. Best case, I expected to get maybe one or two months funded.

The FreeBSD community responded overwhelmingly. My company has

never sent as many invoices as it did in 2004, and my accountant nearly blew a fuse. Suddenly I found myself in a situation I had never even considered: how to stop people from sending me money. I had set up a PayPal account, and at least at that time, there was no way to prevent people from dropping money into it. In the end, I managed to yell loud enough, so I was overfunded by only a few percent. I believe that my attempt to deflect the surplus to the FreeBSD Foundation gave that group a little boost that year, too.

Today, people who do crowdfunding have "stretch goals" to soak up overfunding, but in 2004, I was in uncharted waters; whatever happened, I wanted to contain any fallout from my experiment in a single fiscal year. I also was not sure how it would work out in terms of the social dynamics of the project, so I did not want it to extend past half a year.

In total, I made 27,000 euros, which kept my kids fed and my bank happy for the six months that I worked on the project.

And work I did.

I have never had a harsher boss than during those six months, and it surprised me how much it stressed me. I felt like I was working on a stage with the entire FreeBSD project in the audience wondering if I would deliver the goods or not. As a result, the 187 donors certainly got their money's worth, as most of that half-year I worked 80-hour weeks, which made me decide not to continue, despite many donors indicating they were perfectly willing to fund several more months.

Varnish Community Funding

Five years later, having developed Varnish HTTP Cache Version 1.0 for Norway's *Verdens Gang* newspaper, and having exploded into a vacuum in Web-content delivery, I decided to give community funding a go again.

Wiser from experience, I structured the Varnish Moral License (VML)[4] to tackle the issues that had caused me grief the first time around: contact first, then send money, not the other way around; also focus on fewer, larger sponsors, rather than individuals sending me 10 or 15 euros, or even in one case one euro, which lingered in an unused PayPal account. I run even more-open books this time, and on the VML Web pages you can see how many hours I have worked, along with a terse one-line description of what I did for every single day I have been working under the VML since 2010.

I also decided to be honest with myself and my donors: one hour of work was one hour of work—nobody would benefit from me dying from stress. In practice it does not quite work like that: there is plenty of thinking in the shower, as well as email and IRC (Internet Relay Chat) answers at all hours of the day and night, and a lot of "just checking a detail" that happens off the clock because I like my job, and nothing could stop me anyway.

This is the funding model I want to "sell" for other FOSS projects, because it works. In each of the years 2010, 2011, and 2013, I worked approximately 950 hours on Varnish (funded by the community) in addition to work I did for my other customers. In 2012, I worked only 589 hours, because I was building a prototype computer cluster to do adaptive optics real-time calculations for the European Southern Observatory's Extremely Large Telescope[1] (there was just no way I could say no to that contract).

In 2014, I have hours available to do even more Varnish work, but despite my not-so-subtle hints, the current outlook is still for only 800 hours to be funded. I am crossing my fingers that more sponsors will appear now that Varnish Version 4 has been released (nudge, nudge, wink, wink—He said knowingly).

The VML is not an ideal funding model in the sense that with a single exception, none of the big corporations that deliver massive amounts of HTTP traffic with Varnish has participated. A number of smaller concerns have kept the project alive and kicking. In a smaller company the CEO, or at least the CTO, is more likely to know what FOSS products the company uses, and quite likely keeps abreast of developments and announcements. In a larger organization, on the other hand, bigger issues drown out such "details" as a fundraising plea before they rise to a level where a decision can be made, or where the fear that the marketing department must be involved spikes the idea. I can vividly imagine how Dilbert would fail to convince his PHB (pointy-haired boss) that the company should give money away because it is using some free software.

The good news is the PHB does not have to understand: if a dozen midsize companies each decide to spend $500 every month, that would do wonders for any piece of FOSS—if the money makes it into the hands of the right person(s). And there is that little detail: finding the Right Person.

There is no way to shortcut that: each FOSS project, each sponsoring company, each potential maintainer will have to look one another in the eye and decide if they think this is worth a shot or not. There will be failures and disappointments—that is unavoidable—but if we do not fund good people to maintain critical FOSS projects, there will be no end to the Heartbleed. C

Related articles on queue.acm.org

B.Y.O.C. (1,342 Times and Counting)
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=1944489

Commercializing Open Source Software
Michael J. Karels
http://queue.acm.org/detail.cfm?id=945125

A Plea to Software Vendors from Sysadmins - 10 Do's and Don'ts
Thomas A. Limoncelli
http://queue.acm.org/detail.cfm?id=1921361

References

1. European Southern Observatory. The European Extremely Large Telescope; http://www.eso.org/public/teles-instr/e-elt/.
2. Kamp, P.-H. Why should I care what color the bikeshed is, 1999; http://bikeshed.org/.
3. Kamp, P.-H. Fundraising for FreeBSD development, 2004; http://people.freebsd.org/~phk/funding.html.
4. Kamp, P.-H. The Varnish Moral License; http://phk.freebsd.dk/VML.
5. Yadron, D. After Heartbleed bug, a race to plug Internet hole. *Wall Street J.* (Apr. 9, 2014); http://online.wsj.com/news/articles/SB10001424052702303736045794913502513151 32 (login required).

Poul-Henning Kamp (phk@FreeBSD.org) is one of the primary developers of the FreeBSD operating system, which he has worked on from the very beginning. He is widely unknown for his MD5-based password scrambler, which protects the passwords on Cisco routers, Juniper routers, and Linux and BSD systems. Some people have noticed that he wrote a memory allocator, a device file system, and a disk-encryption method that is actually usable. Kamp lives in Denmark with his wife, son, daughter, about a dozen FreeBSD computers, and one of the world's most precise NTP (Network Time Protocol) clocks. He makes a living as an independent contractor doing all sorts of stuff with computers and networks.

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

DOI:10.1145/2632361

Article development led by acmqueue
queue.acm.org

Addressing the needs of professional software development.

BY MICHAEL J. LUTZ, J. FERNANDO NAVEDA, AND JAMES R. VALLINO

Undergraduate Software Engineering



IN THE FALL semester of 1996, Rochester Institute of Technology (RIT) launched the first undergraduate software engineering program in the U.S.[9,10] The culmination of five years of planning, development, and review, the program was designed from the outset to prepare graduates for professional positions in commercial and industrial software development.

From an initial class of 15, the ABET-accredited program has grown steadily, Today, the student body numbers more than 400 undergraduates. Co-op students and graduates are employed in organizations large and small, including Microsoft, Google, Apple, and United Technologies, as well as a variety of government agencies. Housed in a separate Department of Software Engineering at RIT, the program has the independence and flexibility necessary to ensure its integrity as it evolves.

Its primary focus is on preparing professional, practicing software engineers. This is illustrated most directly by the required year of co-operative education following two years of foundational coursework. Students alternate terms of formal study with paid professional experience; at the end of the five-year program, they have both solid academic preparation and significant practical experience. These graduates are in high demand, as they are prepared to define, design, develop, and deliver quality software systems.

The question remains, of course: Why a specialized software engineering degree? After all, the majority of new industrial hires come from traditional programs in computer science and engineering. Here, we provide our rationale for striking out in a new direction—our strong belief that there is a need in industry for entry-level engineers of software, and our conviction that we could provide an educational experience that better prepares students for careers in the software field. Then, we look at the differences between the RIT program and those



PHOTOGRAPH BY GOODLUZ PRODUCTIONS

typical of undergraduate computer science. This leads, in turn, to a presentation of our pedagogical approach and the state of software engineering in computer science curricula. Later, we discuss the RIT program's relationship with industry and the preparation of co-op students and graduates.

Motivation

In the late 1980s, one of the authors of this article (Lutz) took a two-year industrial leave from RIT: first at GCA/Tropel, a manufacturer of optical metrology products; and later at Eastman Kodak, where he led teams developing embedded systems and application-level software. His responsibilities included interviewing, hiring, and mentoring new college graduates, and what he observed during this time was unsettling. By and large these graduates had a solid background in basic computing theory and technology. Many had taken courses in algorithm analysis and theory of computation, and most had some exposure to operating systems, programming language concepts, artificial intelligence, graphics, and compiler design. What they lacked, however, was the background necessary to be effective when working on large, complex, industrial-quality systems.

In particular, these graduates had little (if any) experience working as members of a software team, yet this is common industrial practice. Their knowledge of design was often confined to those artifacts of interest to computer scientists—compilers, operating systems, graphics libraries, among others—yet they had little appreciation of design as an activity in its own right. Most had no experience with version control, much less configuration management. Their knowledge of testing was usually meager, and few had even heard of verification and validation. Finally, they knew little or nothing about the actual processes involved in creating a product beyond rote memorization of the waterfall model.

Conversations with others, both in industry and software engineering education, indicated these problems were pervasive. The issue was the old one of science vs. engineering: those whose goal is to grow and expand knowledge vs. those who apply such knowledge to create useful products. Traditionally, this is expressed as scientists "build in order to learn," while engineers "learn in order to build." This seemed an opportune time to apply this distinction to computer science and the engineering of software, just as the difference between physics and the engineering of physical artifacts had emerged in the past.

At the time we were developing the software engineering curriculum at RIT, many master's programs in software engineering were already being offered. The prevailing opinion was that undergraduates should pursue computer science degrees and later enroll in master's programs to complete their education. Given that most computer

science graduates go into industry immediately upon graduation, and many may never complete an MSSE (master of science in software engineering), this approach was problematic. Consider the following: one way to teach a new driver would be to present the theory of the internal combustion engine, the drive train, and the electrical system, then turn over the keys and let the driver take the car for a spin; after the new driver has run into some lamp posts and destroyed a few mailboxes, the instructor then says, "Now you are ready to learn how to drive." From our perspective, this is analogous to the BSCS/MSSE approach to educating software developers.

There must be trade-offs, of course. Just as a mechanical engineer does not have nearly the depth in physics as a physics major, a software engineer will not have the depth in computer science that a computer science major acquires. Our argument, however, is the software engineering knowledge will compensate for the lack of deep scientific knowledge when it comes to contemporary software-development practice. Here, we explore the differences between the science of computing and the engineering of software as a way of illuminating these trade-offs.

The Manifesto for Software Engineering Education

In 2001, a group of leading software engineering professionals issued the Manifesto for Agile Software Development.[2] They presented a series of trade-offs among different approaches to software development, such as "responding to change" versus "following a plan." They concluded by stating that while there is value in all of these approaches, they promote one set—the "agile" approaches—over the other.

Similarly, we do not dismiss the value of traditional computer science approaches; we just value other approaches more in the education of software engineers. Taking our inspiration from the agile manifesto, we present the relevant approaches and trade-offs of our software engineering program at RIT. While recognizing that some computer science programs and faculty do incorporate one or more of our approaches, we have found none that do so to the same degree as our program. More details of the RIT program are available on the curriculum flowchart.[12]

Horizontal vs. vertical curricula. Perhaps the largest difference in philosophy is the trade-off between breadth of engineering knowledge (horizontal) versus the depth of specific technical expertise (vertical). The RIT software engineering program is unabashedly based on the former. In required courses students make several iterations through the entire development life cycle, from customer requirements to product delivery, with all the activities in between (for example, formal and informal modeling; architecture and design; testing and quality assurance; planning, estimation, and tracking; process and project management). Of course, individual courses focus on specific aspects of the engineer's professional responsibilities, but by the time of the capstone senior project, students have the background they need to take a project from inception to completion.

Indeed, the senior project is illustrative of the horizontal approach. Whereas software engineering in many computer science programs is confined to a single-term project, for RIT's software engineering students the two-term, team-based senior project is the culmination of all that precedes it. Working with real customers, whether industrial, nonprofit, or internal to RIT, teams are responsible for establishing the project scope, negotiating requirements, designing a solution under constraints (for example, compatibility with an existing system), performing risk analysis, and creating and enacting an appropriate development plan. The success of this approach is attested to by the many projects that have gone into live operation at project sponsors' sites.

Teamwork vs. individual activity. Many computer science courses emphasize individual competency over teamwork, but working on teams to solve problems is a hallmark of RIT's software engineering program. Indeed, with two exceptions (a course on personal software engineering and one on formal mathematical modeling), all of the software engineering courses incorporate team projects as significant graded components.

The second-year introductory course (for software engineering, computer science, and computer engineering majors) promotes teamwork as fundamental to professional practice. Specific roles and responsibilities are highlighted, as are issues of team cohesion, conflict resolution, and team-based success. This course also ensures exposure to version control, as this is essential to providing a log of member contributions and to detecting and reconciling conflicting changes to documents and source code.

For that portion of the grade based on team activities (approximately 50%), teams receive grades as a whole. Each team member is also assessed in terms of his or her contributions to the team, based on instructor observations, version-control logs, and confidential peer evaluations.

The introductory course is the only such course required of computer science and computer engineering majors, but it is merely the first of many for software engineers. Students are expected to work in teams of four to six members as an integral part of their professional education. Through the course of the program, the typical software engineering student will work on more than 20 different teams.

Design and modeling vs. programming and coding. The RIT software engineering student's primary exposure to programming per se is in the introductory computer science sequence. While programming techniques are discussed in both introductory and advanced software engineering courses, this is never the focus. Instead, we view programming competence as providing entry into the field—a basic membership requirement, if you will. While teams will program software systems in most of our courses, the focus is on programming as a necessary step on the way to product delivery.

An example may help illustrate this. While students are expected to have an understanding of data structures from their computer science courses and to grasp the basic concepts of complexity (such as "big-O" notation), they are rarely required to build such structures from scratch. Instead, we strongly encourage them to incorporate existing components where possible. The components may be part of the standard environment for languages such as Java or Ruby, or they may be provided by third parties (for example, Ruby-

Gems). Teams are responsible for due diligence to ensure the selected components exhibit certain quality attributes while providing the required functionality, and they must give proper attribution for any components they employ.

Reducing the teaching of programming per se leaves room to emphasize more significant issues of modeling and design.[14] The second-year introductory course, in addition to the teamwork component discussed previously, is also the one that introduces basic design qualities such as cohesion and coupling, information hiding, designing to an interface rather than an implementation, and abstraction into components delivering well-defined services. Other second-year courses for majors provide additional experience with modeling and design.

The modeling course addresses formal based approaches to modeling, exploring, and verifying designs. Using tools such as Alloy[6] and Promela/Spin,[5] students learn to express structural and behavioral properties using discrete mathematics and to use associated tools to verify assertions about overall system properties. In addition, the course provides an overview of data modeling and relational database theory. At the conclusion of the course, students have a better appreciation for the role of rigorous design analysis in software system analysis.

The subsystem-design course explicitly addresses design, using design patterns[4] as a vehicle to raise the level of abstraction. Our experience is that by naming common structural and behavioral interactions, and applying this expanded vocabulary in design exercises, students begin to draw away from the implementation details and focus on the higher-level component relationships. Later design courses, whether addressing security, concurrency, or Web-based systems, can build on this base to discuss design concepts and trade-offs.

The course on concurrent and distributed-system design illustrates another difference from most computer science curricula. Whereas computer science typically introduces these issues in the context of operating systems or database systems, our course is less about the artifacts than concurrent and distributed concepts and issues in their own right. Typically, student teams design, develop, and deliver systems other than databases or operating systems where concurrency is a critical design concern, and thus do not view it as the province of specialists. In the age of multicore computers and cloud computing, this approach has served graduates well.

In creating the software engineering curriculum at RIT, the notion of teaching professionalism as encapsulated in a disciplined process was prominent in our thinking.

Disciplined process vs. ad hoc development. In creating the software engineering curriculum at RIT, the notion of teaching professionalism as encapsulated in a disciplined process was prominent in our thinking. Process is not, as some claim, the be-all and end-all of software engineering, but it does provide the frame within which software development takes place. In our curriculum, process is as important a pillar as software design.

This does not mean, however, that we impose one dogmatic approach to process—indeed, we ensure students are familiar with many process approaches, from strictly planned to the more adaptive agile approaches.[3] Part of being an effective practitioner is to recognize the importance of selecting and adhering to a process appropriate to the project at hand. The benefits of agile approaches for rapidly evolving Web systems become significant risks when applied in safety-critical settings (for example, aircraft fly-by-wire controls).

Part of the process emphasis rarely discussed in computer science programs is estimation and tracking. In the first-year course on personal software engineering, students estimate and track the effort involved in their in-class activities and longer projects. To prevent "cooking the books," students are assessed not on how accurate their estimates are, but on their reflections as to why the estimated and actual effort differed. It is such reflective practice that slowly but surely improves students' estimating ability, which is the foundation for team estimates in later courses.

A Pedagogical Approach

Active learning and team-based project work are the two most prominent characteristics of the pedagogical approach used in RIT's software engineering courses.[8,13] Using an active learning pedagogy is certainly not unique to software engineering programs, but having it applied across the curriculum is somewhat unique. Fortunately,

we were able to incorporate support for it in our facilities. Almost all of our courses are taught in studio labs, and we have replaced a significant amount of lecture time with class exercises and team project activities that engage the students in immediate reinforcement of course concepts. The studio labs, with computers at each seat, provide seamless transitions through lecture and individual or pair exercises.

Each course has team projects through the entire term that account for at least 40% of the final grade. To support the team activities both during and outside of class times, there are 11 seven-person team rooms, each with a whiteboard, desktop computer, and projector. The team rooms are extensions of the studio labs. A typical class session might spend half the time in the studio lab moving between lecture and class exercises, and the remainder of the class in the team rooms.

Students use this time in the team rooms either grouped in random teams to engage in exercises using material from that class session, or grouped in their current project teams to do specific project work. During this time, the instructor interacts directly with each team to gain a better understanding of how well both the team and its individual members are performing. The team has multiple opportunities to receive project feedback and design guidance.

When designing this pedagogical approach, many of the software engineering faculty members remembered the initial period of their industrial experience when much of their instruction in software design came from mentoring by senior engineers. This instructor time with teams encourages those interactions.

Over time, we realized the teams needed even more facility support. The team rooms were excellent for holding meetings with senior project sponsors, design meetings, and inspections, but they were not adequate for team implementation sessions. To address that shortcoming, we reconfigured one of our studio labs into the Software Engineering Collaboration Lab, containing five collaboration areas accommodating six students each. A wall-mounted monitor displays the output from one of four under-table workstations or a student laptop. The workstation monitors are fixed low to the table, leaving the airspace of collaboration open. Several visitors who work in the industry voiced their appreciation for this unique arrangement, wishing to recreate it in their own team areas.

The motivation for creating an undergraduate software engineering program was our perception of a mismatch between the skills that an entry-level software developer needed and what was typically provided to students in CS programs.

Has Anything Changed in Computer Science?

The motivation for creating an undergraduate software engineering program was our perception of a mismatch between the skills that an entry-level software developer needed and what was typically provided to students in computer science programs. We believe the skill-set mismatch described 20 years ago still exists. One place to see that is in *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* (CS2013),[7] used as the foundation for many computer science programs.

The principles that guided the creation of CS2013 specify "Curricula must…include professional practice (for example, communication skills, teamwork, ethics) as components of the undergraduate experience. Computer science students must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design." One of the expected characteristics of computer science graduates is project experience where "all graduates of computer science programs should have been involved in at least one substantial project. In most cases, this experience will be a software-development project, but other experiences are also appropriate in particular circumstances.... Students should have opportunities to develop their interpersonal communication skills as part of their project experience." Both of these overarching aspects of the guidelines identify a need for software engineering concepts.

Another place where guidance for curricular content in computer science programs exists is in the *ABET Criteria for Accrediting Computing Programs*.[1] The student outcomes specified in the section titled "Program Criteria for Computer Science and Similarly Named Computing Programs" are stated as:
The program must enable students to attain, by the time of graduation:

(*j*) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. [CS]

(*k*) An ability to apply design and development principles in the construction of software systems of varying complexity. [CS]

These two outcomes define a clear need for coverage of design principles and development practices, both of which fall under the software engineering realm. Moreover, we would argue that the modeling and design part of (*j*) and all of (*k*) are software engineering.

With the need for software engineering established in the CS2013 guidelines and ABET accreditation requirements, let's look at what the current curriculum guidelines provide in that area. CS2013 defines 18 knowledge areas revolving around technology such as architecture and organization, graphics and visualization, networking and communication, operating systems, and programming languages. Only three—SDF (software development fundamentals), SE (software engineering), and SP (social issues and professional practice)—fall within the software engineering realm. Guideline comments identify the SE and SP knowledge areas as specific curricula areas where teamwork and communication soft skills will be learned and practiced. The minimum lecture hours specified for software engineering topics in these three knowledge areas are 10 in SDF, 28 in SE, and one in SP.

Even though students will have more time on task doing assignments and project work, and may see additional material discussed in elective courses, these minimums are inadequate for developing the full skill set for an entry-level software engineer. This is especially true when you consider that the SE knowledge area, which at 14 pages is the longest non-cross-cutting knowledge area in CS2013, identifies 60 core topics with 69 learning outcomes, and 54 elective topics with 56 learning outcomes. The breadth and depth of this knowledge area leads to a lament heard regularly at software engineering education conference sessions. The CS faculty members responsible for software engineering in the curriculum ask, "How am I going to fit the core SE topics and the 'soft' teamwork and communication skills in the single software engineering course in our computer science curriculum?" The reality of undergraduate computing education is that the vast majority of students do not go through software engineering curricula where there is time to address this in depth. Instead, they are in computer science or computer engineering programs, and they learn their software engineering skills in their one, and often only, software engineering course.[15]

Industrial Perspectives on Software Engineering Education

In May 2013, one of this article's authors (Vallino) attended a meeting of the Rochester Java Users Group. Instead of hearing a presentation on some aspect of Java technology, the group had a general discussion of software education/training/certification led by Bryan Basham, who is an active developer and former Java trainer for Sun Microsystems. He expressed concern that there was a mismatch between what was being taught and the skill set that software developers needed. This was the same insight that 20 years prior led us to start developing our software engineering program at RIT.

The users group session progressed by having the audience list what they remember learning in their undergraduate coursework. This list clearly identified the technology areas that are explored in a traditional computer science degree. Next, the audience described the skills that they felt were needed to be competent at their software development activities. This list covered most elements of RIT's software engineering program and included the need for strong teamwork and communications skills. This experience reinforced the idea that software engineering programs address the needs of professional software development, at least as perceived by an audience of active developers, and that the programs need more visibility, because no one in the audience even knew of the existence of undergraduate software engineering.

The effectiveness of RIT's software engineering program is quantitatively assessed in accreditation self-studies. We do have anecdotal comparative indications between computer science and software engineering. RIT's career services office publishes salary data,[11] and software engineering students report the highest average hourly co-op and median full-time wages almost every year compared with computer science, computer engineering, and all the other computing majors at RIT. The placement rate of software engineering undergraduates is more than 90% at graduation.

In addition, a review of co-op employment evaluations also provides anecdotal evidence of the value of our students' training to their employers. An engineering manager in an aerospace company, which has hired many of our students on co-op and in full-time positions, commented that the students have a strong focus on capturing requirements and system modeling. An engineering vice-president, who has hired students and sponsored senior projects, commented that our graduates match up favorably with some software engineers who have five years of experience at the company.

One of our lecturers, Robert Kuehl, who has 30-plus years in a career developing and managing the development of software systems in consumer and commercial imaging, gave this assessment of the skills preparation that our students receive:

In a generalization, industry wants:

- *Professionalism.* Individuals who act professionally, communicate effectively verbally and in writing, and who work effectively in diverse teams.
- *Execution competence.* Professionals who know how to elicit and specify good requirements, who can transition requirements into designs that fulfill requirements, who can productively write good code, debug code, and test code. They want professionals who effectively select and execute software development methodologies and tools to manage projects that are consistently delivered on time and within budget.
- *Technical knowledge and expertise.* Professionals who are on top of current technology, who have sound knowledge of computing principles, techniques, and algorithms, and who can innovate.

The computer science curricula help students unquestionably gain computing technical knowledge and expertise. The software engineering curriculum provides similar technical grounding but integrates other coursework to teach professionalism and to acquire the execution competence that comes with it. Courses cover all aspects of the software development life cycle in depth via course projects that emphasize learning by doing, teamwork, and communication in addition to the technical aspects of the projects.

As a result in my experience, software engineering graduates are generally better prepared for jobs in industry that require the development and deployment of quality software.

Jeffrey Lasky, professor of information technology, served as an RIT Professor in Residence at Excellus, a local Blue Cross/Blue Shield health insurance provider. Conversations with Lasky first tipped us off to some of the distinctions generated by the software engineering curriculum.

"The RIT/Excellus Blue Cross/Blue Shield co-op program began in fall 2002. The program was co-managed by the director, Excellus Architecture and Integration Group, and an RIT professor-in-residence. The co-op program was open to all RIT undergraduate students majoring in a computing discipline.

"In 2004, a team of six students, two each from computer science, information technology, and software engineering, were assigned to work on a subsystem for a high-priority, system-development project. The composition of the team was unplanned but serendipitous. The students quickly realized their respective skill-set strengths clustered around a core area of their degree program: programming (CS), database and Web (IT), and design (SE).

"While all topics proved to be of interest, the software engineering students' use and explanations of software-design patterns gathered the most attention from the other students, who quickly noticed the SE students thought differently about software-systems development. Specific patterns became part of and often guided team dialogues; the CS and IT students were enthusiastic about the role and value of formal abstractions in software design. The supervising Excellus software architects had similar reactions to design patterns, and copies of the classic book, *Design Patterns: Elements of Reusable Object-Oriented Software*, started to appear on their desks, and thereafter, on the desks of many Excellus software developers."

Lasky's characterization of the student strengths in the various degrees in our college has been echoed by colleagues at other institutions. The SE students ask questions about components, architecture, and interactions between the components, preferring a higher-level and more abstract model-driven discussion. The CS and IT students tend to ask for examples of working code and begin understanding the system from the bottom up. The CS students are great at coding but generally lack skills in design and concern for quality attributes. With the curricular balance between design and process in RIT's software engineering program, students have a broader range of coding skills, with some students not interested in doing much coding at all. In student surveys, two-thirds preferred the design and implementation side, and the rest were more interested in the process side (for example, requirements, process improvement, and software quality assurance).

Conclusion

Twenty years ago when RIT started creating the first undergraduate software engineering program in the U.S., we gambled that if we built it, they would come—meaning both students and employers. The program's track record of growth and more than 90% placement of graduates demonstrates the gamble paid off. The concentration on engineering design, software product development, teamwork, and communication provides students who seek a career in software development with a set of skills better tailored not only to excel as entry-level software engineers, but also to prepare them for growth throughout their career. C

Related articles on queue.acm.org

Fun and Games: Multi-Language Development
Andrew M. Phelps and David M. Parks
http://queue.acm.org/detail.cfm?id=971592

Pride and Prejudice: (The Vasa)
George V. Neville-Neil
http://queue.acm.org/detail.cfm?id=1508213

A Conversation with John Hennessy and David Patterson
http://queue.acm.org/detail.cfm?id=1189286

References

1. ABET Computing Accreditation Commission. Criteria for accrediting computing programs; http://www.abet.org/uploadedFiles/Accreditation/Accreditation_Process/Accreditation_Documents/Current/C001%2014-15%20CAC%20Criteria%2010-26-13.pdf.
2. Beck, K., et al. Manifesto for agile software development, 2001; http://www.agilemanifesto.org/.
3. Boehm, B.W., Turner, R. 2004. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, Boston, MA, 2004.
4. Gamma, E., et al. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley, Reading, MA, 1995.
5. Holzmann, G.J. *The SPIN Model Checker: Primer and Reference Manual.* Upper Addison-Wesley Educational Publishers, Saddle River, NJ, 2011.
6. Jackson, D. *Software Abstractions: Logic, Language, and Analysis.* MIT Press, Cambridge, MA, 2012.
7. Joint Task Force on Computing Curricula. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.
8. Ludi, S., Natarajan, S., Reichlmayr, T. An introductory software engineering course that facilitates active learning. In *Proceedings of SIGCSE Technical Symposium on Computer Science Education*, (2005), 302.
9. Lutz, M.J., Naveda, J.F. The road less traveled: A baccalaureate degree in software engineering. In *Proceedings of the Conference on Software Engineering Education and Training*, (1997).
10. Naveda, F., Lutz, M. Crafting a baccalaureate program in software engineering. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*, (1997).
11. Rochester Institute of Technology, Office of Cooperative Education and Career Services. Salary | Program Overview, 2013; http://www.rit.edu/emcs/oce/students/salary.
12. Rochester Institute of Technology, Department of Software Engineering. Undergraduate curriculum, 2013; http://http://www.se.rit.edu/pagefiles/documents/VSEN%20Flowchart.pdf.
13. Vallino, J. Design patterns—Evolving from passive to active learning. In *Proceedings of Frontiers in Education Conference*, (2003).
14. Vallino, J. If you're not modeling, you're just programming: modeling throughout an undergraduate software engineering program. In *Proceedings of the International Conference on Models in Software Engineering*, (2006), 291–300.
15. Vallino, J. What should students learn in their first (and often only) software engineering course? In *Proceedings of the Conference on Software Engineering Education and Training*, (2013), 335–337.

Michael Lutz (mjlvse@rit.edu) has been on the RIT faculty since 1976. In the 1990s he initiated and led the effort to develop the first baccalaureate software engineering program in the U.S. His professional interests include software engineering education, software design, and mathematical modeling of software.

J. Fernando Naveda (F.Naveda@rit.edu) is co-founder of RIT's Department of Software Engineering, which he chaired from 2001 through 2010. His interests are in curriculum development and a broad range of software engineering academic topics. He has been at RIT for 21 years, where he currently serves in the Academic Affairs Office.

James Vallino (J.Vallino@se.rit.edu) has been associated with RIT's Department of Software Engineering since he started at RIT in 1997. He has been the chair of the department since 2010. Prior to RIT, he had 16 years of industrial experience at AT&T Bell Laboratories, AVL, and Siemens Corporate Research.

Copyright held by owners/authors. Publications rights licensed to ACM. $15.00

Join US!

2014 ACM
International Conference on

Business Analytics

October 8-9, 2014, Houston Texas

http://bac.acm.org

CONFERENCE AT A GLANCE

Hosted by its Special Interests Groups (SIG) Board, this is the first ACM-sponsored Conference to focus on *Business Analytics*. Key questions today deal with organizing and managing massive volumes of data effectively, the evolution of analytics techniques and software tools to support complex analytical processes, and how business analytics impacts and changes business organizations and their competitive situations. This conference will address these and related issues.

Conference Board

Dr. Charles K. Davis, Chair
University of St. Thomas
Ms. Kimberly L. Bullock
ExxonMobil Chemical Company
Dr. Jonathan L.S. Byrnes
Massachusetts Institute of Technology
Dr. Wynne Chin
University of Houston
Dr. Charlene A. Dykman
University of St. Thomas
Dr. Gerald D. Everett
IBM (retired)
Dr. Wagner A. Kamakura
Rice University
Dr. Wayne L. Winston
Indiana University





Welcoming Submissions On:

- **Theoretical Foundations of Business Analytics.** Theories and Frameworks, Epistemology of Big Data, Design and Functionality
- **Economic Impacts of Business Analytics.** Innovation through Analytics, Revenue & Profit Analytics, Potential of 'Big Data', Impact Assessments, Advertising Analytics, Investment Analytics
- **Functional Area Analytics.** Marketing Analytics, Sales Analytics, Financial Analytics, Human Resources Analytics, Production and Operations Analytics, Oil & Gas Analytics, Healthcare Analytics, Travel Analytics, Sports Analytics, Educational Analytics
- **Business Analytics Roles & Methods.** Best Practices for Business Analytics, Impact of Analytics on IT and non-IT Roles, Advanced Analytics
- **Big Data & Data Warehousing.** IT Operations, Data Mining, Data-Driven Management, Big Data Analytics
- **Business Intelligence & Decision Systems.** Decision Support Systems, Operational Analytics, Stochastic Modeling, Machine Learning, Cognitive Systems, Enterprise Business Intelligence
- **Business Analytics Technologies.** Predictive Analytics, Analytics as a Service, Visualization, Distributed Parallel Architectures, Analytics Engines
- **Data Sources and Data Collection.** Data Acquisition, Web Analytics, Scaling Business Analytics
- **Defining and Controlling Analytics Projects.** Staffing, Analytics Project Management
- **Analytics and Strategy.** Corporate Planning Models, Problem Finding, Planning Analytics
- **The Future of Business Analytics.** Organizational Impacts, Analytics as a Service, Digital Management, Cloud-based Analytics

Collage Photos Clockwise from Top Left: Sam Houston Statue - Hermann Park *(by Another Believer CC-BY-SA)*, Houston Skyline *(by Henry Chan CC-BY-SA)*, The Galleria *(by Postoak CC-BY-SA)*, NASA Mission Control, *(Public Domain)* Houston Ship Channel – Port of Houston *(Public Domain)*, and Melcher Hall – The University of Houston's Bauer College of Business *(by RJN2 CC-BY-SA)*. *The Pumpjack at Sunset photo at the left is Public Domain.*

DOI:10.1145/2632661.2632664

To destabilize terrorist organizations, the STONE algorithms identify a set of operatives whose removal would maximally reduce lethality.

BY FRANCESCA SPEZZANO, V.S. SUBRAHMANIAN, AND AARON MANNES

Reshaping Terrorist Networks

IN 2008, A Revolutionary Armed Forces of Colombia, or FARC, commander named Nelly Avila Moreno (a.k.a. Karina) turned herself in to Colombian authorities in response to the announcement of a $1 million award for her arrest. Unlike expensive and risky operations to capture terrorists (such as Al Qaeda's Khalid Sheikh Mohammed and the Kurdish PKK's Abdullah Ocalan), Karina had been captured at minimal expense in terms of both financial cost and lives put at risk. The Shaping Terrorist Organization Network Efficiency, or STONE, software platform we developed is designed to identify a set of key operatives in a terrorist network whose removal would maximally defang the organization through a variety of reward programs and capture operations. STONE answers who should be the targets of the reward program and if a government wishes to destabilize a terrorist network and have funds to remove k people, which k people it should target.

Such removal operations are essential to international security. Though world governments spent more than $70 billion fighting terrorism from 2001 to 2008, reducing the number of transnational attacks by 34%, there was a net increase in terrorism fatalities by 67 deaths per year during the same period.[11] Counterterror efforts have weighed strategic actions that try to address the root causes of terrorism by providing incentives to all parties to reach agreement, with many conducting strategic studies to reduce terrorism.[19] However, strategic defeat of terrorist organizations is rare, despite some notable successes (such as the Provisional IRA in Ireland and Aum Shinrikyo in Japan).[4] As a consequence, tactical actions aimed at destabilizing terror networks are still necessary today.

STONE uses three novel algorithms:

Terrorist Successor Problem (TSP). When a terrorist r is removed, it identifies the probability that r is replaced by another terrorist v;

Multiple Terrorist Successor Problem (MTSP). When multiple terrorists are removed from a network, it identifies the new possible networks that might arise, together with an associated probability distribution; and

Terrorist Network Reshaping Problem (TNRP). It uses the results generated by MTSP to identify a set of k terrorists to remove from the network so as to minimize the expected efficacy of the resulting network.

In the terrorist networks we consider, each vertex (such as Abdullah Azzam) could have many properties,

» key insights

- **Identifying terrorists whose removal would maximally destabilize their networks saves civilian and military lives, as well as financial costs.**
- **Measuring lethality of terrorist networks, STONE algorithms help predict who will succeed a removed terrorist leader with over 80% accuracy.**
- **STONE was tested extensively on data concerning four real terror groups: Al-Qaeda, Hamas, Hezbollah, and Lashkar-e-Taiba.**



Nelly Avila Moreno, a former high-ranking member of Colombia's Marxist FARC guerrillas, is escorted by soldiers on August 10, 2009, as she arrives at a press conference in Medellin, Colombia, to report on her peace efforts since her surrender.

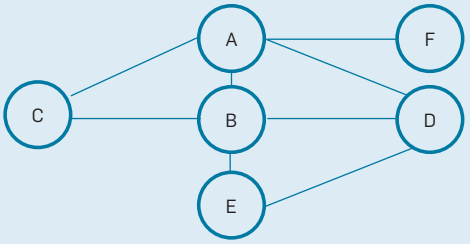
PHOTOGRAPH BY RAUL ARBOLEDA/AFP/GETTY IMAGES

including a status property specifying if he is `alive`, `dead`, or `jailed`; a role property specifying whether he is a `fundraiser`, `ideologue`, or `recruiter`. Figure 1 outlines a toy network where, in addition to the vertex properties, we also consider `hostility` of a vertex toward the West, `capability` to launch attacks, and `blowback` if captured. A property labeling $\rho(v, p)$ tells us the value of property p for vertex v. We explain other properties of terrorists in greater detail later. However, `STONE` can work with any set of properties, not just these. Each vertex in the network also has a rank—coded from 1 to 10, with 10 being the highest; multiple people may have the same rank.

We developed the `STONE` algorithms and tested them with the help of military, policy, and terrorism experts known to us, all of which (with one exception) having written books related to counterterrorism and carried out detailed analyses of specific terrorist groups; the exception was a retired U.S. Army general with considerable expertise fighting terrorists and insurgents. Our tests included both synthetic graphs and real-world open source terrorist-network data. Our experts gave us data on Hamas,[12] Hezbollah,[12] Lashkar-e-Taiba,[6,19,20] and Al-Qaeda, including affiliates like Al-Qaeda in the Islamic Maghreb and Al-Qaeda in the Arabic Peninsula.[5] The data included properties of the terrorists and links between them. Our experts also told us which terrorists were removed from a network, when they were removed, and who replaced them.

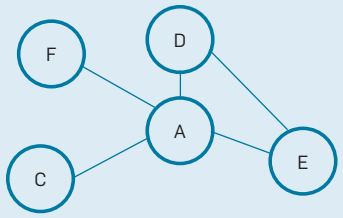
We tested the accuracy of our solution to TSP as follows: We identified the most probable successor v when a terrorist r is removed, together with his/her probability p_{max} of succeeding r, as well as all terrorists whose probability of succeeding r was "close enough" (within 2%, 3%, 4%, or 5% of p_{max}). We did not perform the test with MTSP for two reasons: It is an intermediate step to solving TNRP, and we did not have data as to when multiple terrorists were removed in close temporal proximity. We used our solution to MTSP and various lethality functions to solve TNRP by identifying a set of k terrorists in the network (varying k from 1 to 10). As we were not able to conduct experiments involv-

Figure 1. Sample terrorist organizational network.



Person	Rank	Role	Capacity
A	8	Ops-deputy	5
B	9	Head of Ops	4
C	10	Leader	3
D	7	Ops	4
E	8	Ops-deputy	3
F	5	Ops	2

Figure 2. Reshaping the network in Figure 1 following removal of Node *B*.



Person	Rank	Role	Capacity
A	9	Head of Ops	5
~~B~~	~~9~~	~~Head of Ops~~	~~4~~
C	10	Leader	3
D	7	Ops	4
E	8	Ops-deputy	3
F	5	Ops	2

Figure 3. Weighted removal PageRank.

Weighted Removal PageRank

$$WRP(v,r) = (1-\delta)\frac{rank(v)}{\sum_{u \in vertices(ON)\setminus\{r\}} rank(u)} + \delta \sum_{u \in nbr(v)\setminus\{r\}} \frac{WRP(u)}{|nbr(u)\setminus\{r\}|}$$

where $rank(v)$ is the rank of v, and $\delta \in [0,1]$ is a number in the $[0,1]$ interval denoting the probability that a vertex's importance is due to network effects. δ is used in the same way as the so-called damping factor in PageRank - and we set its value to 0.85 as in PageRank.

ing the real removal of terrorists, we presented the results to our experts and asked them to score the effectiveness of our suggestions on a scale of 1 to 5, with 5 the highest. STONE did well. The algorithms we describe here can also be used to reason about drug, criminal, and money-laundering networks.

Related Work

Identifying key actors in networks is studied through centrality measures; for example, Memon[13] used traditional centrality measures to identify key actors, and Memon and Larson[14] defined efficiency of a network using prior work of Latora and Marchiori.[9] Memon and Larsren[14] proposed a position role index to distinguish between gatekeepers and followers in such networks and define "dependence centrality," or DC. Memon and Larsen[14] showed the values of DC on Krebs's 9/11 network.[7] Carley[3] proposed the CONSTRUCT-O model to measure destabilization of a network through the ability to diffuse information throughout a network, the ability to reach consensus, and the network's ability to perform tasks. Carley[2] suggested "isolation" strategies yielding valuable "objective functions" can be used within the lethality measures described here. Ozgul[17] studied the problem of detecting cells in terror networks, proposing such algorithms as GDM and OGDM. Lindelauf[10] studied the tension between the need to communicate and the need to stay covert in a terror network via a game-theoretic model.

In contrast, STONE develops a mathematical model of who replaces a "removed" individual in a terrorist network, taking into account both network structure and vertex properties, as well as the need for the terrorist network to achieve operational effectiveness. STONE follows the first approach to identifying how a network will evolve when a set of vertices is removed, using the information to decide which set of nodes to remove; it is not wise to remove vertices from networks without first understanding the effects of the removals.

Terrorist Organizational Networks

A terrorist organization network ($\mathcal{ON}$) is an undirected graph with vertices and edges. Each vertex is labeled with some properties drawn from a set *VP* of properties, as described earlier and shown in Figure 1.

We first define what happens when we replace a vertex *r* (the vertex being removed) in $\mathcal{ON}$ with a vertex *v*; we do not suggest *v* replace *r*, just define what happens. Suppose an analyst replaces vertex *B* in Figure 1 with vertex *A*. STONE assumes *A* retains all his connections and inherits *B*'s connections. An interesting question concerns what happens to *A*'s properties. In STONE, a function set by an analyst specifies what properties are inherited from the removed vertex and what properties are not. In our examples, the role and rank properties are inherited while all other properties stay the same. Figure 2 shows that *A*'s properties are identical to his prior properties, except for his promotion from his old rank of 8 to *B*'s rank of 9.

Terrorist Successor Problem

TSP addresses what happens when vertex *r* is removed from the terrorist network $\mathcal{ON}$ who replaces *r*. STONE solves TSP in three steps:

Vertex properties. In addition to the properties discussed earlier, we define specific network-centric properties:

Weighted Removal PageRank (WRP) measures the influence of a vertex, and Property Oriented Clustering Coefficient (POCC) measures how tightly connected a vertex is to neighboring vertices;

Candidate replacements. We formally define the set of candidates that can replace a vertex. STONE allows counterterrorism analysts to specify properties v must satisfy to replace removed vertex r; for instance, it might be the case that v must be able to play the same role as r. In Figure 1 and Figure 2, this means D cannot replace B if D's role was fundraising instead of operations; and

Candidate probabilities. Identifying candidates, we must assign a probability that a given candidate replaces the vertex r being removed.

Network properties of vertices. Here, we describe two special vertex properties—WRP and POCC—used by STONE by leveraging Google's PageRank and clustering coefficients in social networks. STONE allows many other classical centrality measures (such as degree centrality and between-ness centrality), though they are not covered here. We chose WRP and POCC, as they measure the influence of a vertex and the strength of connection of a vertex, respectively. Our hypothesis is that influence and connectedness play a role in determining who is most likely to be elevated to succeed a removed terrorist.

WRP. WRP scores the influence of a vertex by looking at the influence of the associates and family members surrounding him—his immediate neighbors in the network. A person with influential connections is expected to be influential; for instance, in the network associated with Al-Qaeda, Khalid Sheikh Mohammed (KSM) was highly influential, as his immediate connections included top Al-Qaeda leaders Osama Bin Laden, Ayman al-Zawahiri, and others. When he was captured (removed), the WRP calculation for a vertex with respect to KSM would measure the relative influence of the vertex. KSM was replaced by Abu Faraj al-Libbi who had a high influence score, as measured by WRP. WRP extends PageRank[1] to handle three social factors:

Vertices. Vertices in $\mathcal{ON}$ have a rank that must be taken into account when computing the importance of a vertex;

Importance of a vertex. WRP(v, r) denotes the importance of a vertex v in $\mathcal{ON}$, taking into account the fact a vertex r is being considered for removal; and

Undirected graphs. Terrorist organization networks are undirected graphs, not directed graphs like the Web.

The WRP of v is a dynamic quantity that depends on removal of node r. The higher a vertex's WRP with respect to a vertex r being considered for removal, the more likely v is to be a replacement vertex for r, assuming certain other conditions described later. WRP is formally defined in Figure 3. The first term of the formula describes the influence of vertex v with respect to all vertices in the network, except for r, and multiplies it by $(1 - \delta)$ to capture the probability rank plays a role in the vertex's importance. The second term looks at the importance of v's connections, saying each of v's neighbors (u) distributes its importance equally among its neighbors, excluding the vertex r being removed. By multiplying the summation of the second term by δ, STONE is saying the probability that network effects are important is δ. Figure 4 outlines the WRP of the vertices in the small terrorist network of Figure 1, assuming B is being removed.

Property-oriented clustering coefficient. In network theory, clustering coefficients[21] capture how "tightly" a vertex is connected to other vertices in a network. Intuitively, the more tightly connected a terrorist is to other terrorists in his network, the more likely he is to gain their support and be promoted; for instance, when KSM's was captured by U.S. forces in Rawalpindi, Pakistan, in 2003, his successor needed to be tightly connected to his peers to be promoted up to KSM's position in the network, as was indeed the case with Abu Faraj al-Libbi, who succeeded KSM.

The clustering coefficient of vertex v is technically the percentage of pairs of v's neighbors connected to one another. In STONE, we introduce property-oriented clustering coefficients and do not just look at immediate neighbors. The k-neighborhood of a vertex v consists of all vertices at distance k or less from v. From those vertices in v's k-neighborhood, we are interested only in vertices satisfying a set $\mathcal{P}$ of properties specified by an analyst because we use property-oriented clustering coefficients to identify who might replace a vertex r being removed. As a consequence, we restrict interest to only vertices with certain properties, as only vertices with them might be able to influence selection of the replacement vertex; for instance, selection of the operations chief of Lashkar-e-Taiba—if the current operations chief Zakiur-Rehman Lahkhvi is removed—might be determined only by the current chief of Lashkar-e-Taiba, certain top

Figure 4. WRP of the vertices in the small terrorist network in Figure 1, assuming *B* is being removed.

Vertex	WRP	Vertex	WRP	Vertex	WRP
A	0.36	D	0.24	F	0.12
C	0.14	E	0.14		

Figure 5. Property-oriented clustering coefficient.

Property Oriented Clustering Coefficient

$$pocc_{k,\mathcal{P}}(v) = \frac{2 \cdot |\{(u,z) \in edge(ON) \mid u,z \in nbr_{k,\mathcal{P}}(v)\}|}{|nbr_{k,\mathcal{P}}(v)| \cdot (|nbr(v)_{k,\mathcal{P}}| - 1)}$$

where $nbr_{k,\mathcal{P}}(v) = \{u \mid d(u,v) \leq k \wedge (\forall p \in \mathcal{P})\rho(u,p) = 1\}$. As usual, $d(u,v)$ denotes the shortest path distance between u and v. In the case where $|nbr_{k,\mathcal{P}}(v)| \in \{0,1\}$, we set $pocc_{k,\mathcal{P}}(v) = 0$.

Figure 6. Probabilities that vertices *C*, *A*, *D*, *E*, *F* from Figure 1 replace *B*.

v	*P*(*v*, *B*)	*v*	*P*(*v*, *B*)	*v*	*P*(*v*, *B*)
A	0.35	D	0.31	F	0
C	0	E	0.34		

Figure 7. Multiple terrorist successor problem and algorithm sketch.

Multiple Terrorist Successor Problem (MTSP)

Given inputs: (i) a set R of vertices to be removed and (ii) a given $\delta \in [0, 1]$, find a candidate replacement set X_{max} (having some probability p_{max}), as well as all other candidate replacement sets X such that the probability that X replaces R is greater than or equal to $(p_{max} - \delta)$.

MTSP Algorithm Sketch

1. Construct a weighted complete bipartite graph. The first set of vertices S_1 is the set R of vertices to remove. The second set of vertices S_2 consists of all candidates to replace any vertex in R. S_1, S_2 do not overlap. There is an edge from each $r \in S_1$ to each $v \in S_2$ labeled with the logarithm of the probability that v will replace r.
2. Apply an algorithm (such as the Hungarian algorithm[8]) to find a maximum matching over this graph. Return this result.

Figure 8. Lethality functions.

Lethality Functions

$$L_1(ON) = \sum_{v \in VPR(ON)} rank(v) - \sum_{v' \in VA(ON)} rank(v')$$

$$L_2(ON) = \sum_{v \in VPR(ON)} deg(v) \cdot rank(v) + -\sum_{v' \in VA(ON)} deg(v') \cdot rank(v')$$

$$L_3(ON) = \sum_{v \in VPR(ON)} WRP(v, \emptyset) \cdot rank(v) + -\sum_{v' \in VA(ON)} WRP(v, \emptyset) \cdot rank(v')$$

members of the operations section of Lashkar-e-Taiba, members of Lashkar-e-Taiba's Council of Elders, or Majlis-e-Shura. Others directly linked to Zaki-ur-Rehman Lakhvi and in his *k*-neighborhood may not formally influence selection of his successor.[a]

While WRP reflects the influence of a single terrorist, POCC thus captures how tightly integrated that terrorist is with his peers in the organization; POCC is defined in Figure 5.[b]

Candidate replacements. STONE assumes the analyst specifies weights for each vertex property defining the relative importance of each property; for instance, the weight of a vertex's `hostility` might be set at 0.4, while the weight of a vertex's POCC might be set at 0.2, indicating the analyst's assessment of the situation for the group that the POCC is only half as important as the vertex's `hostility`.

Let α_i be a weight associated with each property p_i in our suite of properties such that the α's add up to 1.[c] Each vertex v that may replace a removed vertex r has a replacement value rv(v, r), defined as follows

$$rv(v, r) = \Sigma_{p_i \in vP} \alpha_i \cdot \rho(v, p_i)$$

STONE allows an analyst to specify any set C of properties a vertex should have in order to be an appropriate replacement node for a vertex r that is being removed; for instance, the analyst may specify vertex v must be either a member of the terror network's leadership council or a member of the same department (such as operations) to which the vertex r being removed belongs. STONE supports many network-centered properties not discussed here (such as degree centrality, between-ness centrality, and closeness centrality). We now define when a vertex v is considered a candidate replacement for a vertex r.

When is v a candidate to replace r? When v satisfies condition C, rank(v) $\leq$ rank(r), and there is no other vertex u (different from both v and r) such that u satisfies the previous two conditions and such that u's replacement value is strictly greater than v's.

Example 1. Consider the network in Figure 1 where B is to be removed, and suppose C says the distance between B and v must be 1. Moreover, suppose $rv(v, B)$ is computed, taking into account the WRP (with $\alpha_0 = 0.2$), POCC with $k = 2$, and $\boldsymbol{P} = \emptyset$ (with $\alpha_1 = 0.2$), the node rank (with $\alpha_2 = 0.4$), and node capacity ($\alpha_3 = 0.2$).[c] The possible candidates to replace vertex $r = B$ are then:

A with $rv(A, B) = 0.672$
D with $rv(D, B) = 0.588$
E with $rv(E, B) = 0.634$

a Though Lakhvi remains in a Pakistani jail for his role in the 2008 Mumbai, India, attacks, news sources report he is directing operations from jail.

b The numerator is the number of pairs of vertices within k distance units from v that are directly connected and have properties in P; the denominator is the total number of neighbors of v having all properties in P that are within k distance units of v. The ratio provides the desired quantity.

c The values of rank and capacity are scaled with respect to their maximum value.

Here, A is the best candidate, with E a close second.

Probability of a candidate. We now define the probability of a candidate, or the probability v will replace r:

$$\mathbf{P}(v, r) = \frac{rv(v, r)}{(\Sigma_{u \text{ is a candidate}}\, rv(u, r))}$$

The probability that v replaces removed node r is the ratio of v's replacement value divided by the sum of the replacement values of all candidates to replace v; for instance, the probabilities that vertices C, A, D, E, F from Figure 1 (Example 1) replace B are listed in Figure 6. Observe that $\mathbf{P}(C, B) = 0$ because C's rank is greater than B's rank, and $\mathbf{P}(F, B) = 0$ because the distance between B and F is greater than 1.

Multiple Terrorist Successor Problem

Instead of removing a single vertex r, what if an analyst wants to remove a set R of vertices? For example, in June 2012, India captured two Lashkar-e-Taiba terrorists in quick succession: Abu Jundal, who was present in the control room in Pakistan directing the November 2008 Mumbai attacks, was reportedly captured in a coordinated operation by India, Saudi Arabia, and the U.S., and Tunda, a top Lashkar-e-Taiba bomb maker, was captured in August 2013 along the India-Nepal border. Most security agencies aim to capture multiple terrorists simultaneously. In such cases, many networks are possible (such as if n people could replace Abu Jundal and m people could replace Tunda, then there are mn possible new networks). MTSP is designed to help determine probabilities for the new networks.

Now suppose, as in the single vertex removal case, an analyst has specified a condition C replacement vertices must satisfy. A candidate replacement set X_R for a set R of vertices being removed with respect to condition C is a set of pairs (r, c_r) (r being a removed vertex and c_r being its replacement) satisfying four conditions:

(i) Every removed vertex has a replacement. For each $r \in R$, there is a pair $(r, c_r) \in X_R$ and

(ii) Replacement vertices cannot be removed. $\{c_r \mid (r, c_r) \in X_R\} \subseteq (V - R)$ and

(iii) Replacement nodes must be candidates. For each pair $(r, c_r) \in X_R$, c_r is a candidate to replace r with respect to condition C and

(iv) Candidates can replace only one vertex. There are no two $r, r' \in R$ such that $(r, c_r), (r', c_r)$ are both in X_R.

The following example returns to our running example to show what happens if an analyst removes the set $\{B, C\}$ of vertices from Figure 1 and Example 1.

Example 2. Suppose an analyst wishes to remove the set $R = \{B, C\}$. The set of candidate replacements of B is $\{A, D, E\}$, and the set of candidate replacements for C is $\{A, B\}$. A cannot replace both node B and C, and B is not a valid candidate to replace C because B is to be removed. The replacement sets for R are $X'_R = \{(B, D), (C, A)\}$, and $X''_R = \{(B, E), (C, A)\}$.

There may be many different candidate replacement sets X_R with respect to R and C. The probability $\mathbf{P}(X_R)$ that X_R will in fact be the replacement for R is simply the product of the individual probabilities $\mathbb{P}(c_r, r)$ that c_r will be the replacement for the vertex r. To see how this works, return to the toy example of Figure 1 and Example 2.

Example 3. We have $\mathbb{P}(B, C) = 0$ and $\mathbb{P}(A, C) = 1$, and $\mathbb{P}(D, B) = 0.31$ and $\mathbb{P}(E, B) = 0.34$. So the probability of the replacement set $X'_R = \{D, A\}$ is equal to $(0.31 \cdot 1) / (0.31 + 0.34) = 0.46$, and the probability of the replacement set $X''_R = \{E, A\}$ is equal to $(0.34 \cdot 1) / (0.31 + 0.34) = 0.54$.

When a given set R of vertices is to be removed, let X_{max} be the candidate replacement set with the greatest probability. As there is much uncertainty in the parameter settings (such as choice of weights for each property, the choice of k in the definition of property-oriented clustering coefficients and the choice of condition C the counterterrorism analyst sets), the STONE algorithms do not want to just return X_{max} to the analyst. Rather, STONE returns all candidate replacement sets X_R whose probability is at most δ percentage points smaller than X_{max}'s probability (see Figure 7). Note X_{max} is not an input to this problem; STONE algorithms solve it automatically without having X_{max} as an input. STONE developers prove MTSP can be solved in polynomial time by leveraging an efficient algorithm due to Murty[16] that enumerates, in increasing order of cost, all solutions to the minimal matching in a complete, weighted, bipartite graph, as in Figure 7. Leveraging Murty,[16] STONE algorithms enumerate, in decreasing order of probability, all solutions to the maximal matching, until STONE reaches a solution X_R such that $\mathbb{P}(X_{max}) - \mathbb{P}(X_R) > \delta$.

Lethality Functions

When an analyst recommends a set of nodes to be removed from a network

Figure 9. Terror network time series.

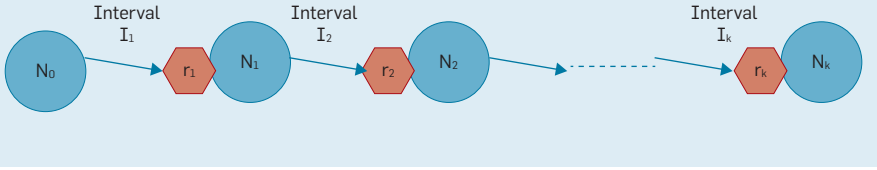
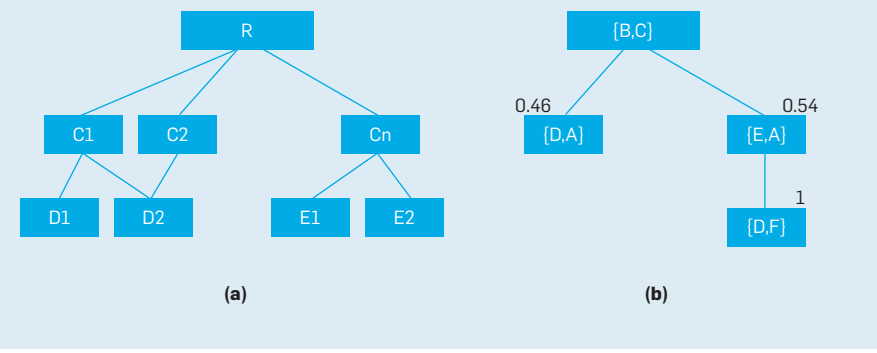


Figure 10. Substitution diagrams.



(such as in a capture operation), the analyst must consider the possible new networks that might result, as well as their lethality. We have covered the former and now turn to ways to measure network lethality, including four such methods: L_1 assesses the difference of ranks of violent-prone and violence-adverse node; L_2 modulates L_1 by degree of the vertices involved; L_3 modulates L_1 by using WRP instead of degree; and L_4 uses regression to correlate L_1, L_2, and L_3 with associated attacks.

A vertex is violence-prone if both its `capability` and its `hostility` exceed analyst-defined thresholds; otherwise, the vertex is violence-averse. `STONE` uses $VPR(\mathcal{ON})$ and $VA(\mathcal{ON})$ to denote the set of all violence-prone and violence-averse vertices, respectively, in $\mathcal{ON}$; Figure 8 includes three lethality functions.

L_1 says the lethality of a network is the sum of the ranks of violence-prone vertices minus the sum of the weights of the violence-averse vertices; that is, think of the violence-prone individuals' violence being moderated by those who are violence-averse; for instance, in the 1990s and early 2000s, the terrorist group Students Islamic Movement of India had a violence-prone wing (that later broke off and became the Indian Mujahideen) and a violence-averse wing (that continues to today).

Example 4. Consider the network $\mathcal{ON}$ in Figure 1 and suppose vertices A, B and D are violence-prone, while C, E, and F are violence-adverse. Then, $L_1(\mathcal{ON}) = wt(A) + wt(B) + wt(D) - wt(C) - wt(E) - wt(F) = 1$.

L_2 says the lethality of a vertex is the degree of the vertex times the rank of the vertex and the lethality of a network is the sum of the lethality scores of violence-prone vertices minus the sum of the lethality scores of violence-averse vertices. The third lethality function is similar but uses WRP instead of degree.

Example 5. Continuing Example 4, an analyst has $L_2(\mathcal{ON}) = 48$, while $L_3(\mathcal{ON}) = 2.69$.

Though many other lethality functions are possible, we do not cover them here, with the exception of a hybrid lethality function:

Hybrid lethality function. A suggestion made independently by a reviewer and by terrorism expert Daveed Gartenstein-Ross of the Foundation for Defense of Democracies, a Washington, D.C.-based think tank, is to define lethality based on attack data. We thus aimed to correlate network structure with attack data. As outlined in Figure 9, the network associated with a particular group was initially $\mathcal{ON}_0$ for some period I_1 of time at which time terrorist r_1 was removed, leading to network $\mathcal{ON}_1$. After time period I_2, a terrorist r_2 was removed, leading to network $\mathcal{ON}_2$. During each period I_j, the group launched some number A_j of attacks. The variables A_j are measures of the lethality of the network N_j–1. For each group, we were thus able to build a table. The jth row of this table corresponded to the time period I_j and the columns contained the values of $L_1(\mathcal{ON}_j)$, $L_2(\mathcal{ON}_j)$, $L_3(\mathcal{ON}_j)$ as independent variables and A_j as a dependent variable. Using standard multivariate regression analysis,[15] we learned a hybrid lethality function L_h relating the values returned by L_1, L_2, L_3 with each of these lethality variables. `STONE` developers call this hybrid function h. We conducted experiments to check the effectiveness of the hybrid lethality function in predicting the number of attacks. We ran the tests with our Lashkar-e-Taiba network dataset (1990–2012) and Al Qaeda dataset (2001–2013). In the case of our Lashkar-e-Taiba dataset, `STONE` found that our learned hybrid lethality function L_h could help

Figure 11. Possible terror networks.

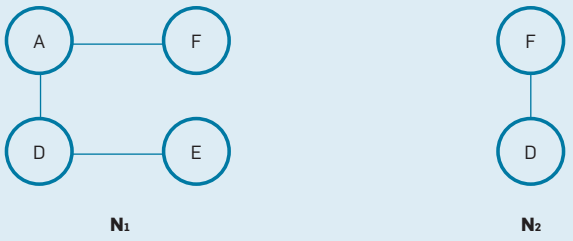


Figure 12. Network reshaping algorithm.

Algorithm 1 STONE-Reshape network reshaping algorithm

Input: Organizational network ON, maximum set size k, condition C, set EX of unremovable vertices.
Output: Set R of nodes to remove.
function RESHAPE(ON, k)
  $R = \emptyset$ % nobody to remove yet
  Set ℓ to the lethality of ON
  $continue = true$
  do
    Let r be a vertex in $ON \setminus (R \cup EX)$ s.t. r satisfies C and $L_{EV}(ON, R \cup \{r\})$ is minimized
    Let ℓ_r be the expected lethality of the network if $R \cup \{r\}$ is removed.
    If($\ell_r \leq \ell$) % i.e. if removing r leads to further weakening of the network
      $R = R \cup \{r\}$% then remove r
      $\ell = \ell_r$ % updated exp. lethality of the network
    Else $continue = false$
    EndIf
  while($continue \wedge |R| \leq k$)
  return R
end function

us predict the number of attacks by Lashkar-e-Taiba. The Pearson correlation coefficient between the true number of attacks and the predicted number of attacks by Lashkar-e-Taiba on blinded data was 0.83. However, when we learned L_h from the smaller Al-Qaeda dataset, the Pearson correlation coefficient between the actual number of Al Qaeda attacks and our predictions was 0.652. L_h thus provides an accurate method of predicting number of attacks by a terror network, with accuracy increasing along with the amount of data an analyst happens to have.

STONE addresses the problem of identifying a set of *k* terrorists in a terrorist organization network whose capture and removal would minimize the expected lethality of the resulting network.

Which Vertices to Remove?

In order to decide which k vertices from a network (possibly excluding some set EX of vertices) to remove, STONE would first define possible terror networks (PTNs) and substitution diagrams (graphs). Figure 10a outlines a sample substitution diagram. Suppose R is a set of vertices being considered for removal; R is the root of the substitution diagram. We know several candidates might be in line to replace the vertices in R. Call these candidates $C_1,\ldots,C_n$ using the definition given earlier. Each C_i is a child of R with the probability p_i labeling the link, where p_i is the probability that C_i replaces R. If C_i were to replace R, STONE would need to find a replacement for the set C_i of vertices. Figure 10a outlines this relationship through an example where C_1 has two possible candidate replacements—D_1 and D_2. Likewise, C_2, which is a candidate set to replace R, has only one candidate to replace it—D_2; see Figure 10b for the substitution diagram for the running example.

A possible replacement chain (PRC) is a path from the root R of the substitution diagram to any leaf. The labels of the nodes along such a path represent the sequence of removals and replacements in the PRC; for instance, the path R, C_2, D_2 represents the removal of R from the network, its replacement by C_2, and replacement of C_2 by D_2. Every PRC $X_1,\ldots,X_m$ generates a PTN as follows: Starting with the terrorist network $\mathcal{ON}$, let $\mathcal{ON}_1$ be the network obtained by replacing R with X_1 in network $\mathcal{ON}$. Similarly, let $\mathcal{ON}_2$ be the result of replacing X_1 in network $\mathcal{ON}_1$ by X_2. STONE does this repeatedly until X_m–1 in $\mathcal{ON}_{m-1}$ is replaced with X_m. The resulting network is a possible terrorist network.

Our running example includes two PTNs—N_1 and N_2 (see Figure 11). The probability of a PTN with respect to a path is the product of the probabilities labeling the edges. The probability $\mathbb{P}(N_2)$ of the PTN N_2 is $0.54 \cdot 1 = 0.54$.

A leaf node in the substitution diagram can be reached via multiple paths, as the same PTN may be generated by multiple PRCs. The probability of the PTN is the sum of the probabilities of the PTN with respect to each replacement chain that allows STONE to reach the leaf node. Reaching the leaf node is the case with the PTNs corresponding to the leaf D_2 in Figure 10a.

Suppose R is a set of vertices to be removed, PTN($\mathcal{ON}$, R) is the set of all PTNs that could result, and L is a lethality function. The statistical expected lethality of the network when R is removed is defined by STONE as

$$L_{EV}(\mathcal{ON},R) = \Sigma_{N\in PTN(\mathcal{ON},R)}\, \mathbb{P}(N)\cdot L(N)$$

Returning to our running example, the lethality L_2 of the possible network N_1 is $L_2(N_1) = 17$ while $L_2(N_2)$ is 2. The expected lethality is then $\text{LEV}(\mathcal{ON},R) = \Sigma_{N\in\text{PTN}(\mathcal{ON},\text{R})}\, \mathbb{P}(N)\cdot L_2(N) = (0.46\cdot 17) + (0.54\cdot 2) = 8.9$.

TNRP.

Input. A terrorist organizational network $\mathcal{ON}$, a condition C that each terrorist being removed must satisfy, and a number k.

Output. Set R of vertices such that

(i) $|R| \le k$ and

(ii) Each $r \in R$ satisfies condition C and

(iii) There is no set R' of vertices satisfying (i) and (ii) such that $L_{EV}(\mathcal{ON}, R') < L_{EV}(\mathcal{ON},R)$.

An analyst may specify conditions C that removed terrorists must satisfy; at the very least, the vertex must be `alive` or `other`. We can also imagine an analyst saying certain terrorists should not be removed due to potential political fallout; for instance, even though the U.S. Department of State announced a $10 million reward in 2012 for information leading to the prosecution of Lashkar-e-Taiba leader Hafez Saeed, he continues to move freely within

Pakistan, giving press conferences and speaking at public rallies.

As solving the TNRP is likely NP-hard, STONE developers developed a greedy algorithm to solve it as follows: Algorithm 1 (see Figure 12) starts with $R = \emptyset$. In each iteration, it finds the removable vertex r, which maximally reduces expected lethality (line 8 of the algorithm in Figure 12). Line 10 checks if adding r to R reduces expected lethality. If so, r is added to R (line 11) and the STONE algorithm repeats until either R has k elements or adding more elements to r does not reduce the expected lethality.

In our running example with lethality function L_3 and $k = 2$, Algorithm 1 suggests removing D and F, decreasing the expected network lethality to –8.87.

Figure 13. Experiments parameter settings.

Setting	α_{WPR}	α_{POCC}	α_{rank}	α_{host}	α_{comp}	α_{BB}
1	0.1	0.1	0.6	0.066	0.067	0.067
2	0.8	0	0	0.066	0.067	0.067
3	0.267	0.266	0.267	0.066	0.067	0.067
4	0.2	0.2	0.4	0.066	0.067	0.067
5	0.4	0.2	0.2	0.066	0.067	0.067
6	0.4	0.4	0	0.066	0.067	0.067

Figure 14. Experiment 1 results.

Setting	Dataset	$\delta = 2\%$	$\delta = 3\%$	$\delta = 4\%$	$\delta = 5\%$
1	Toy	0.905(2)	0.925(3)	0.95(3)	0.99(5)
	AQ	1(1.81)	1(1.90)	1(2)	1(2)
	LeT	0.667(4)	0.667(6)	0.833(7)	0.833(7)
	Hamas	0.8(3.6)	0.8(4)	0.8(4.4)	0.8(4.4)
	Hezballah	0.8(4.2)	0.8(5.6)	0.8(8.2)	0.8(10.2)
2	Toy	0.56(2)	0.56(2)	0.605(2)	0.635(2)
	AQ	0.90(1.27)	1(1.54)	1(1.54)	1(1.54)
	LeT	0.5(5)	0.667(7)	0.833(7)	0.833(8)
	Hamas	0.8(2)	0.8(2.8)	0.8(3.4)	0.8(3.8)
	Hezballah	0.4(4.2)	0.6(8)	0.6(8.8)	0.6(9)
3	Toy	0.53(2)	0.80(3)	0.865(4)	0.935(5)
	AQ	0.90(1.45)	0.90(1.72)	0.90(1.72)	0.90(1.72)
	LeT	0.833(5)	0.833(7)	0.833(7)	0.833(8)
	Hamas	0.4(1.6)	0.8(2.4)	0.8(3.2)	0.8(4.2)
	Hezballah	0.8(3.8)	0.8(4.8)	0.8(7)	0.8(10)
4	Toy	0.835(3)	0.93(4)	0.97(4)	0.985(5)
	AQ	0.90(1.63)	0.90(1.72)	1(1.90)	1(2)
	LeT	0.833(5)	0.833(7)	0.833(7)	0.833(8)
	Hamas	0.8(2.4)	0.8(3.8)	0.8(4.4)	0.8(4.8)
	Hezballah	0.8(4.2)	0.8(5.4)	0.8(7)	0.8(10.2)
5	Toy	0.765(3)	0.865(4)	0.92(5)	0.94(5)
	AQ	0.90(1.45)	0.90(1.54)	0.90(1.72)	1(1.81)
	LeT	0.833(6)	0.833(7)	0.833(7)	0.833(8)
	Hamas	0.6(1.8)	0.8(2.8)	0.8(3.8)	0.8(4.6)
	Hezballah	0.8(3.2)	0.8(5.6)	0.8(8)	0.8(10.8)
6	Toy	0.16(2)	0.26(3)	0.32(3)	0.37(3)
	AQ	0.72(1.27)	0.72(1.36)	0.72(1.36)	0.72(1.36)
	LeT	0.667(4)	0.833(6)	0.833(7)	0.833(8)
	Hamas	0.4(2)	0.4(2.2)	0.4(2.4)	0.4(2.4)
	Hezballah	0.6(2.6)	0.8(3.6)	0.8(5.6)	0.8(10.6)

Figure 15. Experiment 2 results.

Setting	Al-Qaeda	Hamas	Hezbollah	Lashkar-e-Taiba
1	5	4	4.6	3.5
2	5	4	4.6	4
3	5	4	4.6	4
4	5	4	4.6	4
5	5	4	4.6	4
6	5	4	4.6	4

Implementation and Experiments

We implemented all algorithms in this article in 1,500 lines of Java, running them on an AMD Opteron Quad-Core 2354 at 2.2GHz, 8GB RAM machine under the Linux operating system. Our toy dataset consisted of 50 networks; the expected successor for a vertex was picked by an author of a highly regarded counterterrorism book and by a retired general in the U.S. Army. We also used four real datasets: (i) Al-Qaeda (101 nodes, 121 edges) built by an Al-Qaeda expert expanding Sageman's[18] data; (ii) Hamas (78 nodes, 139 edges) built by one of the authors, an expert in Hamas analysis; (iii) Hezbollah (60 nodes, 64 edges) built by an outside expert on Hezbollah who has worked extensively in Lebanon and built the dataset from Arabic, English, and French sources; and (iv) Lashkar-e-Taiba (153 nodes, 220 edges) built during the writing of a book on Lashkar-e-Taiba, *Computational Analysis of Terrorist Groups: Lashkar-e-Taiba*, by two of the authors. All these datasets represent real ground truth. As STONE requires no training, all data was used directly for testing.

Experiment 1. We tested what settings would yield the highest prediction accuracy for the TSP, allowing δ to be 2, 3, 4, or 5%; Figure 13 includes six of the settings we used. The constants α_{WRP}, α_{POCC} denote the weights of WRP; POCC denotes the weights of WRP and POCC; α_{rank} denotes the rank of a vertex; α_{host} denotes the hostility of a vertex to the West; α_{comp} denotes the vertex's competence carrying out terror attacks; and α_{BB} represents blowback. The numbers in parentheses in Figure 14 are the results returned by the STONE algorithm on average. STONE aims for high accuracy predicting successors of a removed terrorist while presenting as few candidate successors as possible to the analyst. Settings 3 and 4 are best, returning few candidates with over 80% accuracy, even with δ

= 2%. Note from settings 3 and 4 that the rank of a candidate seems to be the most important factor, followed equally by his influence (WPR) and strength of network (POCC).

Experiment 2. As measuring the accuracy of STONE-Reshape algorithm by removing real terrorists is infeasible, and no historical ground truth exists for such removal, we tested the algorithm's accuracy by comparing it against the opinion of counterterror experts who rated the results on a scale of 1–5, with 5 the best. We tested all three lethality functions and all six settings against the Al-Qaeda, Hamas, Hezbollah, and Lashkar-e-Taiba datasets; Figure 15 includes the experts' levels of satisfaction considering lethality function L_3 and all six settings.

Conclusion

STONE addresses the problem of identifying a set of k terrorists in a terrorist organization network whose capture and removal would minimize the expected lethality of the resulting network. To achieve such performance, STONE addresses three problems: terrorist successor; multiple terrorist successor; and terrorist network reshaping. We first developed a method for predicting who would replace a "removed" terrorist. Using four real-world terrorist network datasets, we showed in over 80% of the cases of terrorist turnover, the true replacement of a removed terrorist is identified among those within 2% probability of the most probable replacement predicted by STONE; moreover, this top 2% of candidates contains only, on average, a few terrorists (settings 3 and 4 in Figure 13). We showed how to infer a probability distribution on the possible new networks that result from the removal of a set of terrorists. We used steps one and two to develop the STONE-Reshape algorithm to identify a set of k or fewer individuals to be removed from a network to minimize its operational efficiency. STONE allows analysts to set constraints on who can be removed (such as an analyst could decide certain people should not be removed), what vertex properties should be considered (the analyst can use different properties for different groups), and constraints as to who can replace who.

This work represents a start on the important problem of whom to remove from a terror network. When a vertex is removed, a power struggle may ensue, leading to a split. Can we thus adapt the STONE-Reshape algorithm to predict when such splits will occur? Likewise, a higher-ranked individual may occasionally be moved "down" to fill a spot that opens up when a lower-ranking terrorist is removed. Or, alternatively, the open functional position is allocated to someone with higher rank. We have not considered this possibility here, though it should be addressed. A single group can have multiple subgroups (such as Al-Qaeda in the Islamic Maghreb and Al-Qaeda in the Arabian Peninsula within Al-Qaeda). The relationships between groups and subgroups must thus be explored when identifying how to destabilize an organization. We may not want to destabilize it in a way that actually strengthens one of its more lethal subgroups. There is thus ample opportunity for further research.

We conclude by reiterating the fact that tactical approaches to defeating terrorist operations address the "symptoms" (attacks) of a set of grievances of a terror group. In cases where it is possible to do so, incentives may be tried; for instance, the Rwandan government's disarmament, demobilization, and reintegration program successfully defanged most of the Hutu militias that carried out the genocide in Rwanda in 1994. However, such efforts have not always been successful. Tactical operations to defeat terror operations must be carried out when an innocent population is at risk from attack, and integration of tactical approaches with strategic incentives to defeat terror groups should be weighed carefully.

Acknowledgment

This article is based on the paper "STONE: Shaping Terrorist Organizational Network Efficiency" in *Proceedings of the 2013 IEEE/ACM ASONAM International Conference on Advances in Social Network Analysis and Mining* (Niagara Falls, Canada, Aug. 25–28). ACM Press, New York, 2013. The work was partly funded by U.S. Army Research Office grant W911NF0910206.

References

1. Brin, S. and Page, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks 30*, 1–7 (Apr. 1998), 107–117.
2. Carley, K.M. Destabilization of covert networks. *Computational & Mathematical Organization Theory 12*, 1 (Apr. 2006), 51–66.
3. Carley, K.M., Lee, J.-S., and Krackhardt, D. Destabilizing networks. *Connections 24*, 3 (2002), 79–92.
4. Cronin, A.K. *How Terrorism Ends: Understanding the Decline and Demise of Terrorist Campaigns.* Princeton University Press, Princeton, NJ, 2010.
5. Gartenstein-Ross, D. *Bin Laden's Legacy: Why We're Still Losing the War on Terror.* John Wiley & Sons, Inc., New York, 2011.
6. John, W. *Caliphate's Soldiers: The Lashkar-e-Tayyeba's Long War.* Amaryllis and the Observer Researcher Foundation, New Delhi, India, 2011.
7. Krebs, V.E. Mapping networks of terrorist cells. *Connections 24*, 3 (2002), 43–52.
8. Kuhn, H.W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1–2 (Mar. 1955), 83–97.
9. Latora, V. and Marchiori, M. How the science of complex networks can help developing strategies against terrorism. *Chaos, Solitons & Fractals 20*, 1 (Apr. 2004), 69–75.
10. Lindelauf, R., Borm, P., and Hamers, H. The influence of secrecy on the communication structure of covert networks. *Social Networks 31*, 2 (May 2009), 126–137.
11. Lomborg, B. Is counterterrorism good value for money? *NATO Review Magazine* (Apr. 2008).
12. Mannes, A. *Profiles in Terror: A Guide to Middle East Terror Organizations.* Rowman and Littlefield Publishers, Lanham, MD, 2004.
13. Memon, B.R. Identifying important nodes in weighted covert networks using generalized centrality measures. In *Proceedings of the European Intelligence and Security Informatics Conference* (Odense, Denmark, Aug. 22–24). IEEE Computer Society, 2012, 131–140.
14. Memon, N. and Larsen, H.L. Practical algorithms for destabilizing terrorist networks. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics* (San Diego, CA, May 23–24). Springer, 2006, 389–400.
15. Mendenhall, W. and Sincich, T. *A Second Course in Statistics: Regression Analysis.* Pearson, Upper Saddle River, NJ, 2011.
16. Murty, K.G. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research 16*, 3 (May–June 1968), 682–687.
17. Ozgul, F., Bondy, J., and Aksoy, H. Mining for offender group detection and story of a police operation. In *Proceedings of the Sixth Australasian Conference on Data Mining and Analytics* (Gold Coast, Australia, Dec. 3–4). Australian Computer Society, Sydney, 2007, 189–193.
18. Sageman, M. *Understanding Terror Networks.* University of Pennsylvania Press, Philadelphia, 2004.
19. Subrahmanian, V., Mannes, A., Sliva, A., Shakarian, J., and Dickerson, J. *Computational Analysis of Terrorist Groups: Lashkar-e-Taiba.* SpringerLink : Bücher. Springer London, Ltd., 2012.
20. Tankel, S. *Storming the World Stage: The Story of Lashkar-e-Taiba.* C. Hurst & Co. (Publishers) Ltd., London, U.K., 2011.
21. Watts, D.J. and Strogatz, S.H. Collective dynamics of 'small-world' networks. *Nature 393*, 6684 (June 4, 1998), 440–442.

Francesca Spezzano (spezzano@umiacs.umd.edu) is a faculty research assistant at the University of Maryland Institute for Advanced Computer Studies, College Park, MD.

V.S. Subrahmanian (vs@umiacs.umd.edu) is a professor of computer science and director of the Center for Digital International Government at the University of Maryland, College Park, MD.

Aaron Mannes (amannes@umd.edu) is a faculty research assistant at the University of Maryland Institute for Advanced Computer Studies and a Ph.D. candidate in the School of Public Policy at the University of Maryland, College Park, MD.

© 2014 ACM 0001-0782/14/08 $15.00

DOI:10.1145/2634273

Example-based reasoning techniques developed for programming languages also help automate repetitive tasks in education.

BY SUMIT GULWANI

Example-Based Learning in Computer-Aided STEM Education

HUMAN LEARNING AND COMMUNICATION is often structured around examples, possibly a student trying to understand or master a certain concept through examples or a teacher trying to understand a student's misconceptions or provide feedback through example behaviors. Example-based reasoning is also used in computer-aided programming to analyze programs, including to find bugs through test-input-generation techniques[4,34] and prove correctness through inductive reasoning or random examples[15] and synthesize programs through input/output examples or demonstrations.[10,16,18,22] This article explores how such example-based reasoning techniques developed in the programming-languages community can also help automate certain repetitive and structured tasks in education, including problem generation, solution generation, and feedback generation.

These connections are illustrated through recent work (in computer science) applied to a variety of STEM subject domains, including logic,[1] automata theory,[3] programming,[27] arithmetic,[5,6] algebra,[26] and geometry.[17] More significant, the article identifies some general principles and methodologies that are applicable across multiple subject domains.

Procedural vs. conceptual problems. Procedural problems involve solutions that require following a specific procedure students are expected to memorize and apply; examples include mathematical procedures[5] taught in middle school or high school (such as addition, long division, greatest common divisor computation, Gaussian elimination, and basis transformations) and algorithmic procedures taught in undergraduate computer science, where students are expected to demonstrate their understanding of certain classic algorithms on specific inputs (such as breadth-first search, insertion sort, Dijkstra's shortest-path

» key insights

- **Computing technologies can automate repetitive tasks in education, including problem generation, solution generation, and feedback generation, for numerous subject domains, including programming, logic, automata theory, arithmetic, algebra, and geometry.**
- **This can make standard and online classrooms more efficient and enable new pedagogies involving personalized workflows, saved teacher time, and improved student learning.**
- **Computer-aided education requires cross-disciplinary computing technologies; highlighted here are contributions from programming languages, human-computer interaction, and artificial intelligence; natural language understanding and machine learning also play a significant role.**



IMAGE BY SERGEY NIVENS

algorithm, regular expression to automaton conversion, and even computing tensor/inner product of qubits).

Conceptual problems include all nonprocedural ones for which there is no decision procedure (the student is expected to know and apply) but require creative thinking in the form of pattern matching or educated guesses. Problems include:

Proof problems. Natural deduction proofs,[1] proofs of algebraic theorems,[26] and proofs of non-regularity of languages; and

Construction problems. Construction of computational artifacts (such as geometric constructions,[17] automata,[3] algorithmic procedures,[27] and bitvector circuits).

Example-based learning. Examples have multifaceted use in educational technologies. This article classifies their use according to interaction with the underlying technology (see Figure 1).

Input. For several educational tasks, examples constitute a natural means to express intent. In the case of solution generation for procedural problems, teachers can demonstrate example traces with the goal of synthesizing procedures for the problems. In the case of problem generation for conceptual problems, teachers can provide an example problem with the goal of generating similar problems. In the case of feedback generation for procedural problems, teachers can provide examples of buggy traces with the goal of learning the algorithmic misconceptions a student might have. In the case of feedback generation for conceptual problems, teachers can provide examples of common local error corrections, aiming to find some appropriate combination of the corrections that correct a given incorrect attempt. For such cases, this article describes techniques inspired by research in programming by example (PBE).[10,16,18,22]

Output. For some educational tasks, examples constitute the intended output artifact. In the case of problem generation for procedural problems, teachers want to produce example inputs that exercise various paths in the given procedure to generate a progression of problems. In the case of feedback generation for conceptual problems, teachers want to produce counterexamples that expose incorrect behavior in the student's solution. For such cases, the article describes techniques inspired by program analysis, in particular by test-input-generation techniques[4,34] often used to find bugs.

Inside. Examples can also be used

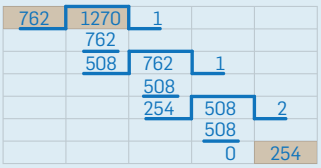
inside the underlying algorithms to perform inductive reasoning, which happens in both solution generation and problem generation for conceptual problems. It is inspired by how humans often approach problem generation and solving, with the underlying techniques inspired by research in establishing program correctness using random examples[15] and program synthesis using examples.[16]

The article next explores example-based learning technologies through specific instances, highlighting general principles. It is organized around the three key tasks in intelligent tutoring[33]—solution generation, problem generation, and feedback generation—through multiple instances of example-based learning technologies for each task. Also described are several evaluations associated with each of these instances. While several of the instances are preliminary, some have been deployed and evaluated more thoroughly.

Figure 1. Three ways examples are used in computer-aided educational technologies as input (for intent expression); as output (to generate the intended artifact); and inside the underlying algorithm (for inductive reasoning).

	Procedural	Conceptual
Solution Generation	Input[6]	Inside[1,17]
Problem Generation	Output[5]	Input,[1,26] Inside[1,26]
Feedback Generation	Input[6]	Output,[3] Input[27]

Figure 2. Solution generation for procedural problems:[6] (a) demonstrate greatest common factor (GCF) procedure over inputs 762 and 1270 to produce output 254; and (b) synthesize procedure GCF automatically from the demonstration in (a).



(a)

```
GCF (int array array T, int I1, int I2)
1 Assume T, [0,0], T[1,0] contain I1, I2 respectively.
2 for (j := 0; T[2j, j] ≠ 0; j := j + 1):
3       T[2j, j + 2] := Floor(T[2j, j + 1] ÷ T[2 j, j]);
4       T[2j + 1, j + 1] := T[2j, j + 2] × T[2 j, j]);
5       T[2j + 1, j + 1] := T[2j, j + 1] − [2j + 1, j + 1];
6       T[2j + 2, j + 2] := T[2j, j];
7 return T[2j, j + 1];
```

(b)

Figure 3. Solution generation for geometry constructions.[17]

(a) English Description ⇓
Construct a triangle given its base L (with end-points p_1, p_2), a base angle a, and sum r of the other two sides.

(b) PreCondition
$r > \texttt{Length}(p_1, p_2)$

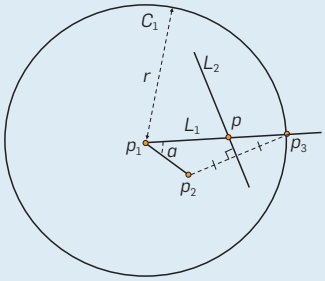
PostCondition ⇓
$\texttt{Angle}(p, p_1, p_2) = a \wedge$
$\texttt{Length}(p, p_1) + \texttt{Length}(p, p_2) = r$

(c) Random Example ⇓
$L = \texttt{Line}(p_1 = \langle 81{:}62;\ 99{:}62\rangle,\ p_2 = \langle 99{:}62;\ 83{:}62\rangle)$
$r = 88{:}07 \quad a = 0{:}81 \text{ radians} \quad p = \langle 131{:}72;\ 103{:}59\rangle$

(d) Geometry Program

```
ConstructTriangle(p1, p2, L, r, a):
L1 := ConstructLineGivenAngleLinePoint(L, a, p1);
C1 := ConstructCircleGivenPointLength(p1, r);
(p3, p5) := LineCircleIntersection(L1, C1);
L2 := PerpendicularBisector2Points(p2, p3);
p := LineLineIntersection(L1, L2);
return p;
```

(e) Geometry Construction



Solution Generation

Solution generation involves automatically generating solutions, given a problem description in some subject domain, and is important for several reasons: It can be used to generate sample solutions for automatically generated problems; given a student's incomplete solution, it can be used to complete a solution that can be much more illustrative for the student compared to providing a completely different sample solution; and, given a student's incomplete solution, it can also be used to generate hints on the next step or toward an intermediate goal.

Procedural problems. Solution generation for procedural problems can be achieved by writing down and executing the corresponding procedure for a given problem. While these procedures can be written manually, technologies for automatic procedure synthesis (from examples) can enable nonprogrammers to create customized procedures on the fly. The number of such procedures and their stylistic variations in how they are taught can be significant and may not be known in advance to outsource manual creation of the procedures.

The procedures can be synthesized through PBE technology[10,16,22] traditionally applied to end-user applications. More recently, PBE has also been used to synthesize programs for spreadsheet tasks, including string transformations and table layout transformations.[18] Mathematical procedures can be viewed as spreadsheet procedures involving computation of new values from existing values in spreadsheet cells, as in string transformations that produce a new output string from substrings of input strings, and positioning that value in an appropriate spreadsheet cell, as in table transformations that reposition the content of an input spreadsheet table. Ideas from learning string and

table transformations can be combined to learn mathematical procedures from example traces, where a trace is a sequence of (value, cell) pairs.[6] Dynamic programming can be used to compute all subprograms that are consistent with various subtraces (in order of increasing length). The underlying algorithm starts out by computing, for each trace element (v, c), the set of all program statements (over a teacher-specified set of operators) that can produce v from previous values in the trace; see Figure 2 for synthesis of a greatest common divisor procedure from an example trace, where the teacher-specified operators include –, ×, ÷, and `Floor`.

Conceptual problems. Solution generation for conceptual problems often requires performing search over the underlying solution space. Following are two complementary principles, each useful across multiple subject domains while also reflecting how humans might search for such solutions.

S1: Perform reasoning over examples as opposed to abstract symbolic reasoning. The idea is to reason about the behavior of a solution on some or even all examples, or concrete inputs, instead of performing symbolic reasoning over an abstract input. Such reasoning reduces search time by large constant factors because executing part of a construction or proof on concrete inputs is much quicker than reasoning symbolically about the construction or proof.

S2: Reduce solution space to solutions with small length. The idea is to extend the solution space with commonly used macro constructs in which each such construct is a composition of several basic constructs/steps. This extension reduces the size of solutions, making search more feasible in practice.

The following illustrates these principles in multiple subject domains:

Geometry constructions. Geometry construction is a method for constructing a desired geometric object from other objects by applying a sequence of ruler and compass constructions (see Figure 3). Such constructions are an important part of high school geometry. The automated geometric-theorem-proving community (one of the success stories in automated reasoning) has developed

The use of trace-based modeling allows for test-input-generation tools for generating problems with certain trace features.

tools (such as Geometry Explorer[32] and Geometry Expert[14]) that allow students to create geometry constructions and use interactive provers to prove properties of the constructions. How are these constructions synthesized in the first place?

Geometry constructions can be regarded as straight-line programs that manipulate geometry objects—points, lines, and circles—using ruler/compass operators. Hence, their synthesis can be phrased as a program-synthesis problem[17] in which the goal is to synthesize a straight-line program, as in Figure 3d, that realizes the relational specification between inputs and outputs, as in Figure 3b.

The semantics of geometry operations is too complicated for symbolic methods for synthesis or even for verification. Ruler/compass operators are analytic functions, implying the validity of a geometry construction can be probabilistically inferred from testing on random examples, an implication that follows from the following extension of the classical result on polynomial identity testing[25] to analytic functions:

Property 1 (probabilistic testing of analytic functions). Let $f(X)$ and $g(X)$ be non-identical real-valued analytic functions over R^n. Let $Y \in R^n$ be selected uniformly at random, then with high probability over the random selection $f(Y) \neq g(Y)$. Property 1 follows from the fact that non-zero analytic functions have isolated zeroes; that is, for every zero point of an analytic function, there exists a neighborhood in which the function is non-zero. The number of non-zero points of the non-zero analytic function $f(X) - g(X)$ thus dominates the number of its zero points.[a]

The problem of synthesizing geometry constructions that satisfy a symbolic relational specification between inputs and outputs can thus be reduced to synthesizing constructions that are consis-

a Unlike the polynomial identity testing theorem,[25] which allows performing modular arithmetic over numbers selected randomly from a finite integer set for efficient evaluation, this result provides no constructive guidance on the size of the selection set and requires precise arithmetic. This process is approximated by using finite-precision floating-point arithmetic and a threshold for comparing equality; in practical experiments, it has yielded no unsoundness or incompleteness.

Figure 4. Solution generation for natural deduction:[1] (a) sample inference rules; (b) sample replacement rules; (c) abstract proof of the problem in Figure 7b, with second column listing the 32-bit integer representation of the truth-table over five variables; (d) natural deduction proof of the problem in Figure 7b, with inference rule applications in bold; and (e) natural deduction proof of a problem similar to the one in Figure 7b with the same inference rule steps.

Rule Name	Premises	Conc
Modus Ponens (MP)	$p \rightarrow q$, p	q
Hypo. Syllogism (HS)	$p \rightarrow q$, $q \rightarrow r$	$p \rightarrow r$
Disj. Syllogism (DS)	$p \vee q$, $\neg p$	q
Simplification (Simp)	$p \wedge q$	q

(a)

Rule Name	Proposition	Equivalent Proposition
Distribution	$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$
Double Negation	p	$\neg\neg p$
Implication	$p \rightarrow q$	$\neg p \vee q$
Equivalence	$p \equiv q$	$(p \rightarrow q) \wedge (q \rightarrow p)$
	$p \equiv q$	$(p \wedge q) \vee (\neg p \wedge \neg q)$

(b)

Step	Truth-table	Reason
P1	1048575	Premise
P2	4294914867	Premise
P3	3722304989	Premise
1	16777215	**P1, Simp**
2	4294923605	**P2, P3, HS**
3	1442797055	**1, 2, HS**

(c)

Step	Proposition	Reason
P1	$x_1 \vee (x_2 \wedge x_3)$	Premise
P2	$x_1 \rightarrow x_4$	Premise
P3	$x_4 \rightarrow x_5$	Premise
1	$(x_1 \vee x_2) \wedge (x_1 \vee x_3)$	P1, Distr.
2	$x_1 \vee x_2$	**1, Simp.**
3	$x_1 \rightarrow x_5$	**P2, P3, HS.**
4	$x_2 \vee x_1$	2, Comm.
5	$\neg\neg x_2 \vee x_1$	4, Double Neg
6	$\neg x_2 \rightarrow x_1$	5, Implication
7	$\neg x_2 \rightarrow x_5$	**6, 3, HS.**
8	$\neg\neg x_2 \vee x_5$	7, Implication
Conc	$x_2 \vee x_5$	8, Double Neg

(d)

Step	Proposition	Reason
P1	$x_1 \equiv x_2$	Premise
P2	$x_3 \rightarrow \neg x_2$	Premise
P3	$(x_4 \rightarrow x_5) \rightarrow x_3$	Premise
1	$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$	P1, Equivalence
2	$x_1 \rightarrow x_2$	**1, Simp.**
3	$(x_4 \rightarrow x_5) \rightarrow \neg x_2$	**P3, P2, HS.**
4	$\neg\neg x_2 \rightarrow \neg(x_4 \rightarrow x_5)$	3, Transposition
5	$x_2 \rightarrow \neg(x_4 \rightarrow x_5)$	4, Double Neg
6	$x_1 \rightarrow \neg(x_4 \rightarrow x_5)$	**2, 5, HS.**
7	$x_1 \rightarrow \neg(\neg x_4 \vee x_5)$	6, Implication
8	$x_1 \rightarrow (\neg\neg x_4 \wedge \neg x_5)$	7, De Morgan's
Conc	$x_1 \rightarrow (x_4 \wedge \neg x_5)$	8, Double Neg.

(e)

tent with randomly chosen input-output examples (Principle S1).

This reduction is the basis of Gulwani et al.'s[17] synthesis algorithm for geometry constructions involving two key steps (see also Figure 3) reflecting the two general principles discussed earlier:

- Generate random input-output examples, as in Figure 3c, from the logical description, as in Figure 3b, of the given problem using off-the-shelf numerical solvers; the logical description is in turn generated from the natural language description, as in Figure 3a, using natural language translation technology; and
- Perform brute-force search over a library of ruler-and-compass operators to find a construction, as in Figure 3d, that transforms the randomly selected input(s) into corresponding output(s).

The search is performed over an extended library of ruler and compass operators that includes higher-level primitives, such as perpendicular and angular bisectors (Principle S2). The use of an extended library not only shortens the size of a solution (allowing for efficient search) but also makes a solution more readable for students. On Gulwani et al.'s[17] benchmark of 25 problems, the extended library helped reduce the maximum solution size from 45 steps to 13 steps and increased the success rate from 75% to 100%.

Natural deduction proofs. Natural deduction (taught in introductory logic courses in college) is a method for establishing the validity of arguments in propositional logic, where the conclusion of an argument is derived from the premises through a series of discrete steps. Each one derives a proposition that is either a premise or derived from preceding propositions through application of some inference rule (see Figure 4a) or replacement rule (see Figure 4b), the last of which concludes the argument; see Figure 4d for a proof. Ditmarsc[29] surveyed proof assistants for teaching natural deduction (such as Pandora[9]), some of which also solve problems. This article describes a different, scalable, way to solve such problems while also paving the way for generating fresh problems, as described in the next section.

While the SAT (Boolean satisfiability), SMT (satisfiability modulo theories), and theorem-proving communities[8] continue to focus on solving large-size proof problems in a reasonable amount of time, one recent approach, by Ahmed et al.,[1] to generating natural deduction proofs in real time leverages the observation that classroom-size instances are small. The Ahmed et al. approach reflects use of the two general principles discussed earlier: abstract a proposition using its truth table, which can be represented using a bitvector representation,[20] thus avoiding expensive symbolic reasoning and reducing application of inference rules to simple bitvector operations (Principle S1); and break the proof search into multiple smaller (and hence more efficient) proof searches (Principle S2).

First, an abstract proof is discovered that involves only inference-rule applications over truth-table representation; note replacement rules are identity operations over truth-table representation. This abstract proof over truth-table representation is then refined to a complete proof over symbolic propositions by searching for sequences of replacement rules between consecutive inference rules; see Figure 4c for an example of an abstract proof and Figure 4d for its refinement to a complete proof. Note the size of an abstract proof and the number of replacement rules inserted between any two consecutive inference rules is much smaller than the size of the overall proof. The Ahmed et al. approach solved 84% of 279 problems from various textbooks (generating proofs of $\leq$27 steps), while a baseline algorithm (using symbolic representation for propositions and performing breadth-

first search for the complete proof) solved 57% of the same problems.[1]

Problem Generation

Generating fresh problems with specific solution characteristics (such as a certain difficulty level and set of concepts) is tedious for the teacher. Automating the generation of fresh problems has several benefits: Generating problems similar to a given problem can help avoid copyright issues. It may not be legal to publish problems from textbooks on course websites. A problem-generation tool can give instructors a fresh source of problems for their assignments or lecture notes. It can also help prevent cheating[23] in classrooms or MOOCs (with unsynchronized instruction) since each student can be given a different problem with the same difficulty level. And when a student fails to solve a problem and ends up looking at the sample solution, the student may be assigned a similar practice problem by an automated system, not necessarily by human teacher. Generating problems with a given difficulty level and exercising a given set of concepts can help create personalized workflows for students. Students who solve a problem correctly may be given a problem more difficult than the last problem or that involves a richer set of concepts.

On the other hand, fresh problems create new pedagogical challenges since teachers may no longer recognize the problems and students may be unable to discuss them with one another after assignment submission. These challenges can be mitigated through solution-generation and feedback-generation capabilities.

Procedural problems. A procedural problem can be characterized by the trace it generates through the corresponding procedure. Various features of the trace can then be used to identify the difficulty level of a procedural problem and the concepts it exercises; for instance, a trace that executes both sides of a branch (in multiple iterations through a loop) might exercise more concepts than the one that simply executes only one side of that branch, and a trace that executes more iterations of a loop might be more difficult than the one that executes fewer iterations.

Trace-based modeling allows for test-input-generation tools[4] for generating problems with certain trace features. Andersen et al.[5] used this insight to automatically synthesize practice problems for elementary and middle school mathematics;[5] Figure 5 outlines such automatic synthesis in the context of an addition procedure. Note various addition concepts can be modeled as trace properties and, in particular, regular expressions over procedure locations. Moreover, trace-based modeling allows for use of notions of procedure coverage[34] to evaluate the comprehensiveness of a certain collection of expert-designed problems and fill any holes. It also allows for defining a partial order over problems by defining a partial order over corresponding traces based on trace features (such as number of times a loop was executed and whether the exceptional case of a conditional branch was executed) and the set of n-grams present in the trace. Andersen et al.[5] used this partial order to synthesize progressions of problems and even to analyze and compare existing progressions across multiple textbooks.

As part of follow-on work, Andersen et al. used their trace-based framework to synthesize a progression of thousands of levels for *Refraction*, a popular math puzzle game. An A/B test with 2,377 players (on the portal http://www.newgrounds.com) showed automatically synthesized progression can motivate players to play for similar lengths of time, as in the case of the original expert-designed progression. The median player in the synthesized progression group played 92% as long as the median player in the expert-designed progression group.

Effective progressions are important not just for school-based learning but also for usability and learnability in end-user applications. Many modern user applications have advanced features, and learning them constitutes a major effort by the user. Designers have thus focused on trying to reduce the effort; for example, Dong et al.[11] created a series of mini-games to teach users advanced image-manipulation tasks in Adobe Photoshop. The Andersen et al.[5] methodology may assist in creating such tutorials and games by automatically generating progressions of tasks from procedural specifications of advanced tasks.

Conceptual problems. Problem generation for certain conceptual problems can be likened to discovering new theorems, a search-intensive activity that can be aided by domain-specific strategies. However, two general principles are useful across multiple subject domains:

Figure 5. Problem generation for procedural problems:[5] (a) addition procedure to add two numbers, instrumented with control locations on the right side; and (b) concepts expressed in terms of trace features and corresponding example inputs that satisfy those features (such example inputs can be generated through test-input-generation techniques).

```
Add(int array A, int array B)
  ℓ := Max(Len(A), Len(B));
  for i=0 to ℓ-1 .
    if (i ≥ Len(A)) t := B[i];          ▷ Loop over digits (L)
    else if (i ≥ Len(B)) t := A[i];     ▷ Different # of digits (D)
    else t:=A[i]+B[i];                  ▷ Different # of digits (D)
    if (C[i] == 1) t:=t+1;              ▷ Carry from prev. step (C)
    if (t > 9) {R[i]:=t-10; C[i + 1]:=1;}
    else R[i] := t;
  if (C[ℓ] == 1) R[ℓ] := 1;             ▷ Extra digit in output (E)
```

(a)

Concept	Trace characteristic	Example input
Single-digit addition	L	3 + 2
Multiple-digit addition without carry	LL^+	1234 + 8765
Single carry	$L^*(LC)L^*$	1234 + 8757
Two single carries	$L^*(LC)L^+(LC)L^*$	1234 + 8857
Double carry	$L^*(LCLC)L^*$	1234 + 8667
Triple carry	$L^*(LCLCLC)L^*$	1234 + 8767
Extra digit in input and new digit in output	L^*CLDCE	9234 + 900

(b)

P1: Example-based template generalization. This involves generalizing a given example problem into a template and searching for all possible instantiations of the template for valid problems. Given the search space might be vast, it is usually applicable when the validity of a given candidate problem can be checked quickly. It does not necessarily require access to a solution-generation technology, though such technology can be used to ascertain the difficulty level of the generated problems; and

P2: Problem generation as reverse of solution generation. This applies only to proof problems. The idea is to perform a reverse search in the solution-search space starting with the goal and leading up to the premises. It has the advantage of ensuring the generated problems have specific solution characteristics.

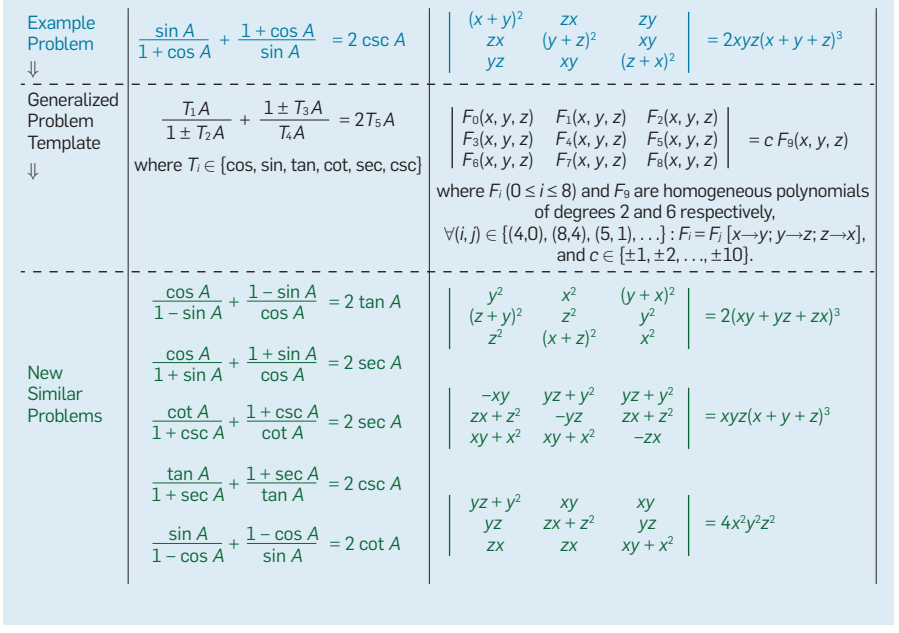
The following sections illustrate how these principles are used in multiple subject domains.

Algebraic proof problems. Problems that require proving algebraic identities (see Figure 6) are common in high school math curricula. Generating such problems is tedious for the teacher since the teacher cannot arbitrarily change constants (unlike in procedural problems) or variables to generate a correct problem.

The Singh et al.[26] Algebra-problem-generation methodology, as in Figure 6, uses Principle P1 to generate fresh problems similar to a given example problem. First, a given example problem is generalized into a template with a hole for each operator in the original problem to be replaced by another operator of the same type signature. The teacher can guide the template-generalization process by providing more example problems or manually editing the initially generated template. All possible instantiations of the template are automatically enumerated, and the validity of an instantiation is checked by testing on random inputs. The probabilistic soundness of such a check follows from Property 1. The methodology works for identities over analytic functions involving common algebraic operators, including trigonometry, integration, differentiation, logarithm, and exponentiation. Note the methodology would not be feasible if symbolic reasoning were used (instead of random testing) to check the validity of a candidate instantiation since symbolic reasoning is much slower (Principle S1) and the density of valid instantiations is often quite low.

Natural deduction problems. Figure 7 covers three interfaces for generating new natural deduction problems:[1]

Figure 6. Problem generation for algebraic-proof problems involving identities over analytic functions (such as trigonometry and determinants);[26] a given problem is generalized into a template, and valid instantiations are found by testing on random values for free variables.



The proposition replacement interface (see Figure 7a) finds replacements for a given premise or the conclusion in a given example problem. It generates those propositions as replacements that ensure the new problem is well defined, or one whose conclusion is implied by the premises but not by any strict subset of the premises. This interface, based on Principle P1, involves checking all possible small-size propositions as replacements. The validity of each candidate problem is checked by performing bitvector operations over bitvector-based truth table representation of the propositions[20] (Principle S1). A candidate problem is valid if the bitwise-and of the premise bitvectors is bitwise smaller than the conclusion bitvector.

The similar problem-generation interface finds problems with a solution that uses exactly the same sequence of inference rules used by a solution of an example problem. Figure 7b lists automatically generated problems, given an example problem. Figure 4e describes a solution for the first new problem in Figure 7b. Observe this solution uses exactly the same sequence of inference rules (in bold) as the solution for the original example problem in Figure 4d. The parameterized problem-generation interface finds problems with specific features (such as a given number of premises and variables, maximum size of propositions, and smallest proof involving a given number of steps and given set of rules). Figure 7c lists automatically generated problems, given some parameters. Both these interfaces find desired problems by performing a reverse search in the solution space (Principle P2) explored by the solution-generation technology for natural deduction described earlier. The similar-problem-generation interface further uses the solution template obtained from a solution of the example problem for search guidance (Principle P1).

Feedback Generation

Feedback generation may involve identifying whether or not a student's solution is incorrect, why it is incorrect, and where or how it can be fixed. A teacher might even want to generate a hint to enable students to identify and/or fix mistakes on their own. In examination

Figure 7. Problem-generation interfaces for natural deduction problems;[1] (a) proposition replacement; (b) similar-problem generation; and (c) parameterized-problem generation.

Some replacements for Premise 3 in Example Problem in (b):

$\neg x_4$
$x_4 \equiv x_5$
$x_4 \equiv x_2$
$x_4 \rightarrow x_2$
$x_4 \rightarrow \neg x_1$

(a)

Premise 1	Premise 2	Premise 3	Conclusion
Example Problem			
$x_1 \vee (x_2 \wedge x_3)$	$x_1 \rightarrow x_4$	$x_4 \rightarrow x_5$	$x_2 \vee x_5$
New Similar Problems			
$x_1 \equiv x_2$	$x_3 \rightarrow \neg x_2$	$(x_4 \rightarrow x_5) \rightarrow x_3$	$x_1 \rightarrow (x_4 \wedge \neg x_5)$
$x_1 \wedge (x_2 \rightarrow x_3)$	$(x_1 \vee x_4) \rightarrow \neg x_5$	$x_2 \vee x_5$	$(x_1 \vee x_4) \rightarrow \neg x_3$
$(x_1 \vee x_2) \rightarrow x_3$	$x_3 \rightarrow (x_1 \wedge x_4)$	$(x_1 \wedge x_4) \rightarrow x_5$	$x_1 \rightarrow x_5$
$(x_1 \rightarrow x_2) \rightarrow x_3$	$x_3 \rightarrow \neg x_4$	$x_1 \vee (x_5 \vee x_4)$	$x_5 \vee (x_2 \rightarrow x_1)$
$x_1 \rightarrow (x_2 \wedge x_3)$	$x_4 \rightarrow \neg x_2$	$(x_3 \equiv x_5) \rightarrow x_4$	$x_1 \rightarrow (x_3 \equiv \neg x_5)$

(b)

Parameters: # of premises = 3, Size of propositions ≤ 4, # of variables = 3, # of inference steps = 2, Inference rules = { DS, HS }

Premise 1	Premise 2	Premise 3	Conclusion
$(x_1 \rightarrow x_3) \rightarrow x_2$	$x_2 \rightarrow x_3$	$\neg x_3$	$x_1 \wedge \neg x_3$
$x_3 \rightarrow x_1$	$(x_3 \equiv x_1) \rightarrow x_2$	$\neg x_2$	$x_1 \wedge \neg x_3$
$(x_1 \equiv x_3) \vee (x_1 \equiv x_2)$	$(x_1 \equiv x_2) \rightarrow x_3$	$\neg x_3$	$x_1 \equiv x_3$
$x_1 \equiv \neg x_3$	$x_2 \vee x_1$	$x_3 \rightarrow \neg x_2$	$x_1 \wedge \neg x_3$
$x_3 \rightarrow x_1$	$x_1 \rightarrow (x_2 \wedge x_3)$	$x_3 \rightarrow \neg x_2$	$\neg x_3$

(c)

settings, the teacher would even like to award a numerical grade.

Automating feedback generation is important for several reasons: First, it is quite difficult and time-consuming for a human teacher to identify what mistake a student has made. As a result, teachers often take several days to return graded assignments to their students. In contrast, if students get immediate feedback (due to automation), it can help them realize and learn from their mistakes faster and better. Furthermore, maintaining grade consistency across students and graders is difficult. The same grader may award different scores to two very similar solutions, while different graders may award different scores to the same solution.

Procedural problems. Generating feedback for procedural problems is relatively easy (compared to conceptual problems) since they all have a unique solution; the student's attempt can simply be syntactically compared with the unique solution. While student errors may include careless mistakes or incorrect fact recall, one common class of mistakes students make in procedural problems is employing a wrong algorithm. Van Lehn[30] identified more than 100 bugs students introduce in subtraction alone. Ashlock[7] identified a set of buggy computational patterns for a variety of algorithms based on real student data. Here are two bugs Ashlock described for the addition procedure (see Figure 5a):

- Add each column and write the sum below each column, even if it is greater than nine; and
- Add each column from left to right; if the sum is greater than nine, write the 10s digit beneath the column and the ones digit above the column to the right.

All such bugs have a clear procedural meaning and can be captured as a procedure. The buggy procedures can be automatically synthesized from examples of incorrect student traces using the same PBE technology discussed earlier in the context of solution generation for procedural problems. In fact, each of the 40 bugs described by Ashlock[7] is illustrated with a set of five to eight example traces, and Andersen et al.[6] were able to synthesize 28 (out of 40) buggy procedures from their example traces.

Identifying buggy procedures has multiple benefits; for instance, it can inform teachers about a student's misconceptions. It can also be used to automatically generate a progression of problems specifically tailored to highlighting differences between the correct procedure and the buggy procedure.

Aleven et.al.[2] used PBE technology to generalize demonstrations of correct and incorrect behaviors provided upfront by the teacher. While their generalization is restricted to loop-free procedures, teachers are able to add annotations as feedback to students who get stuck or follow a known incorrect path.

Conceptual problems. Feedback for proof problems can be generated by checking correctness of each individual step (assuming students are using a correct proof methodology) and using a solution-generation technology to generate proof completions from the onset of any incorrect step.[13] Here, this article focuses on feedback generation for construction problems, including two general principles useful across multiple subject domains:

F1: Edit distance. The idea is to find the smallest set of edits to the student's solution that will transform it into a correct solution. Such feedback informs students about where the error is in their solution and how it can be fixed. An interesting twist is to find the smallest set of edits to the problem description that will transform it into one that corresponds to the student's incorrect solution, thus capturing the common mistake of misunderstanding the problem description. Such feedback can inform students as to why their solution is incorrect. The number and type of edits can be used as a criterion for awarding numerical grades; and

F2: Counterexamples. The idea is to find input examples on which a student's solution does not behave correctly. Such feedback informs the student about why the solution is incorrect. The density of such inputs can be used as a criterion for awarding grades.

The following illustrates how these principles are used in different subject domains:

Introductory programming assignments. The standard approach to grading programming assignments is to examine its behavior on a set of test inputs that can be written manually or generated automatically.[4] Douce et al.[12] surveyed various systems developed for automated grading of programming assignments. Failing test inputs, or counterexamples, can provide guidance as to why a given solution is incorrect (Principle F2). However, this guidance alone is not ideal, especially for

beginners who find it difficult to map counterexamples to errors in their code. An edit-distance-based technique[27] offers guidance on fixing an incorrect solution (Principle F1).

Consider the problem of computing the derivative of a polynomial whose coefficients are represented as a list of integers, teaching conditionals and iteration over lists (see Figure 8a for a reference solution). For this problem, students struggled with low-level Python semantics involving list indexing and iteration bounds. Students also struggled with conceptual aspects of the problem (such as missing the corner case of handling lists consisting of single element). A teacher could leverage this knowledge of common example errors to define an edit distance model consisting of a set of weighted rewrite rules that capture potential corrections (along with their cost) for mistakes students might make in their solutions. Figure 8b includes sample rewrite rules: The first such rule transforms the index in a list access; the second transforms the right-hand side of a constant initialization; and the third transforms the arguments for the range function.

Figure 8c–e show three student programs, together with respective feedback generated by Singh et al.'s program-grading tool.[27] The underlying technique involves exploring the space of all candidate programs, applying teacher-provided rewrite rules to the student's incorrect program, to synthesize a candidate program equivalent to the reference solution while requiring a minimum number of corrections. For this purpose, the underlying technique leverages SKETCH,[28] a state-of-the-art program synthesizer that employs a SAT-based algorithm to complete program sketches (programs with holes) so they meet a given specification. Singh et al. evaluated their tool on thousands of real student attempts (at programming problems) obtained from the 2012 Introduction to Programming course at MIT (6.00) and MITx (6.00x).[27] The tool generated feedback (up to four corrections) on over 64% of all submitted solutions that were incorrect in about 10 seconds on average.

Intention-based matching approaches[19] match plans in student programs with those in a preexisting knowledgebase to provide feedback.

The underlying technique involves exploring the space of all candidate programs, applying teacher-provided rewrite rules to the student's incorrect program, to synthesize a candidate program equivalent to the reference solution while requiring a minimum number of corrections.

While the Singh et al. tool makes no assumption as to the algorithms or plans students can use, a key limitation is it cannot provide feedback on student attempts with big conceptual errors that cannot be fixed through local rewrite rules. Moreover, the Singh et al. tool is limited to providing feedback on functional equivalence, as opposed to performance or design patterns.

Automata constructions. Deterministic finite automaton (DFA) is a simple but powerful computational model with diverse applications and hence is a standard part of computer science education. JFLAP[24] is a widely used system for teaching automata and formal languages that allows for constructing, testing, and conversion between computational models but does not support grading. The following paragraphs explore a technique for automated grading of automata constructions.[3]

Consider the problem of constructing a DFA over alphabet $\{a, b\}$ for the regular language $L = \{s \mid s$ contains the substring "ab" *exactly* twice$\}$. Figure 9 includes five attempts submitted by different students and the respective feedback generated by the Alur et al.[3] automata grading tool. The underlying technique involves identifying different kinds of feedback, including edit distance over both solution and problem (Principle F1) and counterexamples (Principle F2), with each feedback associated with a numerical grade. The feedback that corresponds to the best numerical grade is then reported to the student. The reported feedback for the third attempt is based on edit distance to a correct solution, and the grade is a function of the number and kind of edits needed to convert the student's incorrect automaton into a correct automaton. In contrast, the rest of the incorrect attempts have a large edit distance and hence are based on other kinds of feedback. The second attempt and the last attempt correspond to a slightly different language description; that is, $L' = \{s \mid s$ contains the substring "ab" *at least* twice$\}$, possibly reflecting the common student mistake of misreading the problem description. The reported feedback here is based on edit distance over problem descriptions, and the associated grade is a function of the number and kind of edits required. The reported feedback for the fourth at-

Figure 8. Automated grading of introductory programming problems:[27] (a) reference implementation (in Python) for the problem of computing a derivative of a polynomial; (b) rewrite rules that capture common errors; and (c), (d), and (e) denoting three different student submissions, along with respective feedback generated automatically.

```
def computeDeriv(poly):
result = []
for i in range(len(poly)):
  result += [i * poly[i]]
if len(poly) == 1:
  return result
  # return [0]
else:
  return result[1:]
  # remove the leading 0
```

(a)

$$x[a] \rightarrow x[[a+1, a-1, ?a]]$$
$$x = n \rightarrow x = \{n+1, n-1, 0\}$$
$$\text{range}(a_0, a_1) \rightarrow \text{range}([0, 1, a_0-1, a_0+1], \{a_1+1, a_1-1\})$$

(b)

```
def computeDeriv(poly):
deriv, zero = [], 0
if (len(poly) == 1):
  return deriv
for e in range(0,len(poly)):
  if (poly[e] == 0):
    zero += 1
  else
    deriv.append(poly[e]*e)
return deriv
```

The program requires **3** changes:
- In the return statement `return deriv` in line 4, replace `deriv` by `[0]`.
- In the comparison expression `(poly[e] == 0)` in line 6, change `(poly[e] == 0)` to `False`.
- In the expression range `(0, len(poly))` in line 5, increment `0` by `1`.

(c)

```
def computeDeriv(poly):
idx = 1
deriv = list([])
plen = len(poly)
while idx <= plen:
  coeff = poly.pop(1)
  deriv +- [coeff*idx]
  idx = idx + 1
  if len(poly) < 2:
    return deriv
```

The program requires **1** change:
- In the function `computeDeriv`, add the base case to return `[0]` for `len(poly) = 1`.

(d)

```
def computeDeriv(poly):
length=int(len(poly)-1)
i = length
deriv = range(1,length)
if len(poly) == 1:
  deriv = [0.0]
else:
  while i >= 0:
    new = poly[i] * i
    i -= 1
    deriv[i] = new
return deriv
```

The program requires **2** changes:
- In the expression `range(1, length)` in line 4, increment `length` by `1`.
- In the comparison expression `(i >= 0)` in line 8, change operator `>=` to `!=`.

(e)

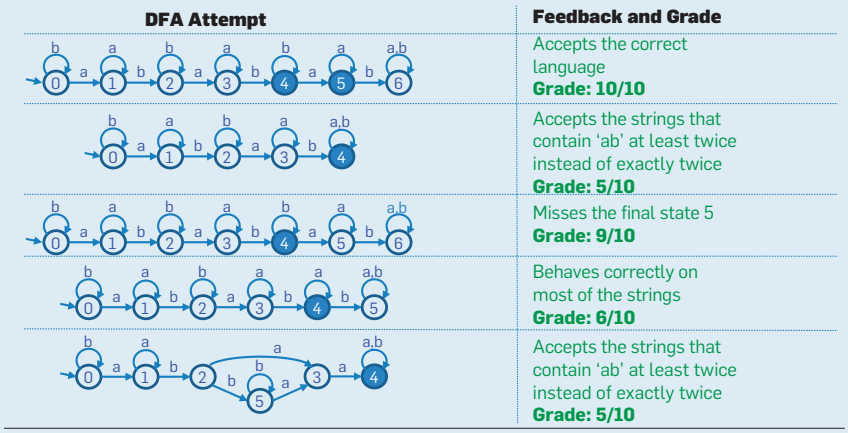
tempt, which does not involve a small edit distance, is based on counterexamples. The grade here is a function of the density of counterexamples, with more weight given to smaller-size counterexamples since students ought to have checked the correctness of their construction on smaller strings.

To automatically generate feedback, Alur et al.[3] formalized problem descriptions using a logic called MOSEL, an extension of the classical monadic-second order logic (MSO) with some syntactic sugar that allows defining regular languages in a concise, natural way. In MOSEL, the languages L and L' can be described by the formulas $|\texttt{indOf}(ab)| = 2$ and $|\texttt{indOf}(ab)| \geq 2$ respectively, where the `indOf` constructor returns the set of all indices where the argument string occurs. Their automata-grader tool implements synthesis algorithms that translate MOSEL descriptions into automata and vice versa. The MOSEL-to-automaton synthesizer rewrites MOSEL descriptions into MSO, then leverages standard techniques to transform an MSO formula into the corresponding automaton. The automaton-to-MOSEL synthesizer uses brute-force search to enumerate MOSEL formulas in order of increasing size to find one that matches a given automaton. Edit distance is then computed based on notions of automata distance or tree distance (in case of problem descriptions), while counterexamples are computed using automata difference.

Alur et al.[3] evaluated their automata-grader tool on 800+ student attempts to solve several problems from an automata course—CS373 at the University of Illinois at Urbana Champaign in Spring 2013. Each submission was graded by two instructors and the tool. For one of these representative problems, instructors were incorrect (having given full marks to an incorrect attempt) or inconsistent (same instructor having given different marks to syntactically equivalent attempts) for 20% of attempts. For another 25% of attempts, there was at least a three (out of 10) points discrepancy between the tool and one of the instructors; in more than 60% of these cases, the instructor concluded (after re-reviewing) that the tool's grade was more fair. The two instructors thus concluded that the tool is preferable to humans for consistency and scalability.

The automata grading tool[3] has been deployed online, providing live feedback and a variety of hints. In Fall 2013, Alur et al.,[3] together with Bjoern Hartmann of the University of California, Berkeley, conducted a user study around the utility of the tool at the University of Pennsylvania and the University of Illinois at Urbana-Champaign, observing such hints were helpful, in-

Figure 9. Automated grading of automata problems:[3] several student attempts to construct an automaton that accepts strings containing the substring "ab" exactly twice, along with automatically generated feedback and grade.



creased student perseverance, and improved problem-completion time.

Conclusion

Providing personalized and interactive education (as in one-on-one tutoring) remains an unsolved problem in standard classrooms. The arrival of MOOCs, despite being an opportunity for sharing quality instruction with a large number of students, exacerbates the problem with an even higher student-to-teacher ratio. Recent advances in computer science can be brought together to rethink intelligent tutoring,[33] with the phenomenal rise of online education making this investment very timely.

This article has summarized recently published work from different areas of computer science, including programming languages,[17,27] artificial intelligence,[1,3,26] and human-computer interaction.[5] It also reveals a common thread in this interdisciplinary line of work, namely the use of examples as an input to the underlying algorithms (for intent understanding), as an output of these algorithms (for generating the intended artifact), or even inside these algorithms (for inductive reasoning). This may enable other researchers to apply these principles to develop similar techniques for other subject domains. This article should inform educators about new advances to assist various educational activities, allowing them to think more creatively about curriculum and pedagogical reforms; for instance, these advances can enable development of gaming layers that take computational thinking into K–12 classrooms.

This article has applied a rather technical perspective to computer-aided education. While the technologies can affect education in a positive manner, computer-aided education researchers must still devise ways to quantify its benefits on student learning, which may be critical to attract funding. Furthermore, this article has discussed only logical-reasoning-based techniques, but these techniques can be augmented with complementary techniques that leverage large student populations and data whose availability is facilitated by recent interest in online education platforms like Khan Academy and MOOCs; for instance, large amounts of student data can be used to collect different correct solutions to a (proof) problem, which in turn can be used to generate feedback[13] or discover effective learning pathways to guide problem selection. Large student populations can be leveraged to crowdsource tasks that are difficult to automate,[31] as in peer grading.[21] A synergistic combination of logical reasoning, machine learning, and crowdsourcing methods may lead to self-improving advanced intelligent tutoring systems that can revolutionize all education.

Acknowledgments

I thank Moshe Y. Vardi for encouraging me to write this article. I thank Ben Zorn and the anonymous reviewers for providing valuable feedback on earlier versions of the draft.

References

1. Ahmed, U., Gulwani, S., and Karkare, A. Automatically generating problems and solutions for natural deduction. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Beijing, Aug. 3–9, 2013).
2. Aleven, V., McLaren, B.M., Sewall, J., and Koedinger, K.R. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *Artificial Intelligence in Education 19*, 2 (2009), 105–154.
3. Alur, R., D'Antoni, L., Gulwani, S., Kini, D., and Viswanathan, M. Automated grading of DFA constructions. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Beijing, Aug. 3–9, 2013); tool at http://www.automatatutor.com/
4. Anand, S., Burke, E., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., and McMinn, P. An orchestrated survey on automated software test case generation. *Journal of Systems and Software 86, 8* (2013), 1978–2001.
5. Andersen, E., Gulwani, S., and Popovic, Z. A trace-based framework for analyzing and synthesizing educational progressions. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Paris, Apr. 27–May 2). ACM Press, New York, 2013, 773–782.
6. Andersen, E., Gulwani, S., and Popovic, Z. *Programming by Demonstration Framework Applied to Procedural Math Problems Technical Report MSR-TR-2014-61.* Microsoft Research, Redmond, WA, 2014.
7. Ashlock, R. *Error Patterns in Computation: A Semi-Programmed Approach.* Merrill Publishing Company, Princeton, NC, 1986.
8. Bjørner, N. Taking satisfiability to the next level with Z3. In *Proceedings of the Sixth International Joint Conference on Automated Reasoning* (Manchester, U.K., June 26–29). Springer, 2012, 1–8.
9. Broda, K., Ma, J., Sinnadurai, G., and Summers, A.J. Pandora: A reasoning toolbox using natural deduction style. *Logic Journal of the Interest Group in Pure and Applied Logics 15*, 4 (2007), 293–304.
10. Cypher, A., Ed. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA, 1993.
11. Dong, T., Dontcheva, M., Joseph, D., Karahalios, K., Newman, M., and Ackerman, M. Discovery-based games for learning software. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Austin, TX, May 5–10). ACM Press, New York, 2012, 2083–2086.
12. Douce, C., Livingstone, D., and Orwell, J. Automatic test-based assessment of programming: A review. *Journal of Educational Resources in Computing 5*, 3 (2005), 511–531.
13. Fast, E., Lee, C., Aiken, A., Bernstein, M.S., Koller, D., and Smith, E. Crowd-scale interactive formal reasoning and analytics. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (St. Andrews, Scotland, Oct. 8–11). ACM Press, New York, 2013, 363–372.
14. Gao, X.-S. and Lin, Q. MMP/Geometer-a software package for automated geometric reasoning. In *Proceedings of the Fourth International Workshop on Automated Deduction in Geometry* (Hagenberg Castle, Austria, Sept. 4–6). Springer, 2004, 44–66.
15. Gulwani, S. *Program Analysis Using Random Interpretation.* Ph.D. thesis. University of California, Berkeley, 2005; http://research.microsoft.com/en-us/um/people/sumitg/pubs/dissertation.pdf
16. Gulwani, S. Synthesis from examples: Interaction models and algorithms (invited talk paper). In *Proceedings of the 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (Timisoara, Romania, Sept. 26–29). IEEE Computer Society, 2012, 8–14.
17. Gulwani, S., Korthikanti, V.A., and Tiwarim, A. Synthesizing geometry constructions. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming Language Design and Implementation* (San Jose, CA, June 4–8), ACM Press, New York, 2011, 50–61.
18. Gulwani, S., Harris, W., and Singh, R. Spreadsheet data manipulation using examples. *Commun. ACM 55*, 8 (Aug. 2012), 97–105.
19. Johnson, W. *Intention-based Diagnosis of Novice Programming Errors.* Morgan Kaufmann, Burlington, MA, 1986.
20. Knuth, D.E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, Boston, 2011.
21. Kulkarni, C., Pang, K., Le, H., Chia, D., Papadopoulos, K., Cheng, J., Koller, D., and Klemmer, S. Peer and self assessment in massive online design classes. *ACM Transactions on Computer-Human Interaction 20*, 6 (2013).
22. Lieberman, H. *Your Wish Is My Command: Programming by Example.* Morgan Kaufmann, Burlington, MA, 2001.
23. Mozgovoy, M., Kakkonen, T., and Cosma, G. Automatic student plagiarism detection: Future perspectives. *Journal of Educational Computing Research 43*, 4 (2010), 511–531.
24. Rodger, S. and Finley, T. *JFLAP: An Interactive Formal Languages and Automata Package.* Jones and Bartlett Publishers, Inc., Sudbury, MA, 2006.
25. Schwartz, J.T. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM 27*, 4 (1980), 701–717.
26. Singh, R., Gulwani, S., and Rajamani, S. Automatically generating algebra problems. In *Proceedings of the 26th conference on Artificial Intelligence* (Toronto, July 22–26). AAAI Press, 2012.
27. Singh, R., Gulwani, S., and Solar-Lezama, A. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th annual ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, June 16–22). ACM Press, New York, 2013, 15–26.
28. Solar-Lezama, A. *Program Synthesis By Sketching.* Ph.D. thesis. University of California, Berkeley, 2008; http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-177.pdf
29. Van Ditmarsch, H. User interfaces in natural deduction programs. In *Proceedings of the User Interfaces for Theorem Provers Workshop* (Eindhoven, The Netherlands, July 1998), 87–95.
30. VanLehn, K. *Mind Bugs: The Origins of Procedural Misconceptions.* MIT Press, Cambridge, MA, 1991.
31. Weld, D.S., Adar, E., Chilton, L., Hoffmann, R., Horvitz, E., Koch, M., Landay, J., Lin, C.H., and Mausam, M. Personalized online education: A crowdsourcing challenge. In *Proceedings of the Fourth Human Computation Workshop at the 26th Conference on Artificial Intelligence* (Toronto, July 22–26, 2012).
32. Wilson, S. and Fleuriot, J.D. Combining dynamic geometry, automated geometry theorem proving and diagrammatic proofs. In *Proceedings of the User Interfaces for Theorem Provers Workshop* (Edinburgh, Apr.). Springer, 2005.
33. Woolf, B. *Building Intelligent Interactive Tutors.* Morgan Kaufman, Burlington, MA, 2009.
34. Zhu, H., Hall, P.A.V., and May, J.H.R. Software unit test coverage and adequacy. *ACM Computing Surveys 29*, 4 (Dec. 1997), 366–427.

Sumit Gulwani (sumitg@microsoft.com) is a principal researcher at Microsoft Research, Redmond, WA, adjunct faculty in the Department of Computer Science and Engineering at the Indian Institute of Technology, Kanpur, India, and affiliate faculty in the Department of Computer Science & Engineering at the University of Washington, Seattle.

© 2014 ACM 0001-0782/14/08 $15.00



BIG IDEAS START SMALL

SIGGRAPH ASIA 2014 SHENZHEN

CONFERENCE 3 DEC - 6 DEC
EXHIBITION 4 DEC - 6 DEC

SHENZHEN CONVENTION & EXHIBITION CENTER

SA2014.SIGGRAPH.ORG

REGISTER NOW TO SAVE

At SIGGRAPH Asia, meet the people you want to meet. Get to learn, be enthralled and inspired by quality content and exhibits in ways you will never be at work or school.

From now until **15 October 2014, 23:59 UTC/GMT** we offer you early bird discounts of up to **20%** if you register online.

Complete registration details can be found at **sa2014.siggraph.org/registration-travel**.

Sponsored by

In Cooperation with

Supported by

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

虚拟现实技术与系统国家重点实验室
STATE KEY LABORATORY OF VIRTUAL REALITY TECHNOLOGY AND SYSTEMS

Tsinghua-Tencent
清华-腾讯联合实验室

DOI:10.1145/2628036

Though maximum flow algorithms have a long history, revolutionary progress is still being made.

BY ANDREW V. GOLDBERG AND ROBERT E. TARJAN

Efficient Maximum Flow Algorithms

THE MAXIMUM FLOW problem and its dual, the minimum cut problem, are classical combinatorial optimization problems with many applications in science and engineering; see, for example, Ahuja et al.[1] The problem is a special case of linear programming and can be solved using general linear programming techniques or their specializations (such as the network simplex method[9]). However, special-purpose algorithms are more efficient. Moreover, algorithm design techniques and data structures developed to compute maximum flows are useful for other problems as well. Although a special case of linear programming, the maximum flow problem is general enough so several important problems (such as the maximum bipartite matching problem) reduce to it.

Here, we survey basic techniques behind efficient maximum flow algorithms, starting with the history and basic ideas behind the fundamental maximum flow algorithms, then explore the algorithms in more detail. We restrict ourselves to basic maximum flow algorithms and do not cover interesting special cases (such as undirected graphs, planar graphs, and bipartite matchings) or generalizations (such as minimum-cost and multi-commodity flow problems).

Before formally defining the maximum flow and the minimum cut problems, we give a simple example of each problem: For the maximum flow example, suppose we have a graph that represents an oil pipeline network from an oil well to an oil depot. Each arc has a capacity, or maximum number of liters per second that can flow through the corresponding pipe. The goal is to find the maximum number of liters per second (maximum flow) that can be shipped from well to depot. For the minimum cut problem, we want to find the set of pipes of the smallest total capacity such that removing the pipes disconnects the oil well from the oil depot (minimum cut).

The maximum flow, minimum cut theorem says the maximum flow value is equal to the minimum cut capacity. This fundamental theorem has many applications, particularly in the design of maximum flow algorithms.

We distinguish between flow algorithms that are polynomial or strongly polynomial. We denote the number of vertices and arcs in the input network by n and m, respectively. For polynomial algorithms, the arc capacities are integral, with U denoting the largest capacity; capacities can be represented by $O(\log U)$-bit integers. (In practice, it is reasonable to assume capacities are integral; as implemented by computer hardware, even floating point

» key insights

- **The idea of augmenting along shortest paths leads to polynomial-time algorithms.**
- **Data structures and fine-grain operations lead to faster algorithms.**
- **Discriminating based on residual capacities when assigning arc lengths leads to improved time bounds.**



ILLUSTRATION BY JURGEN ZIEWE

numbers are represented as integers.) A polynomial algorithm is one with a worst-case time bound polynomial in n, m, and $\log U$. For many applications, algorithms manipulate numbers that fit in a machine word, and elementary arithmetic operations take unit time. We assume this is the case when stating polynomial bounds. A strongly polynomial algorithm is one with a worst-case time bound polynomial in n and m, even if capacities are arbitrary real numbers, assuming a computational model in which elementary arithmetic operations on real numbers take unit time. Strongly polynomial algorithms are more natural from a combinatorial point of view, as only their arithmetic operation complexity depends on the input number size, and other operation counts are independent of the size.

The first special-purpose algorithm for the maximum flow problem was the augmenting path method developed by Ford and Fulkerson.[14] This method is, in general, not polynomial time but can be made so. One way to do this is through scaling, as introduced by Dinic.[11]

Edmonds and Karp[12] introduced the shortest augmenting path method, making the Ford-Fulkerson method strongly polynomial. To define path lengths, the Edmonds-Karp method uses the unit length function, which sets the length of each arc to one. Edmonds and Karp note that other length functions can be used but do not seem to lead to better time bounds. The key to the analysis is the observation that the shortest augmenting path length is non-decreasing and must eventually increase.

The blocking flow method augments along a maximal set of shortest paths by finding a blocking flow. Such augmentation increases the shortest augmenting path length and thereby speeds up the shortest augmenting path method. Blocking flows are implicit in Dinic's algorithm[10] and made explicit by Karzanov,[23] who also introduced a

relaxation of flow called a "preflow" that allows an algorithm to change the flow on a single arc instead of on an entire augmenting path. Arc flow is updated through a push operation. Preflows allow faster algorithms for finding blocking flows.

An interesting special case of the maximum flow problem involves all arcs having unit capacities. As shown independently by Karzanov[22] and Even and Tarjan,[13] the blocking flow algorithm in this case achieves better time bounds than in the general case for two reasons: the number of blocking flow computations is reduced, and the computations are faster—linear time in the graph size.

The operations of a blocking flow algorithm can be divided into two parts: those that manipulate distances and those that manipulate flows. In theory, the latter dominate, motivating development of data structures that allow changing flow values on a path more efficiently than one arc at a time. The first such data structure was developed by Galil and Naamad.[15] A few years later, Sleator and Tarjan[29,30] introduced the dynamic tree data structure, allowing changing flow values on a path with k arcs in $O(\log k)$ time. This led to improvement in the theoretical time bound for finding a blocking flow, making it almost linear.

Goldberg and Tarjan[18] developed the push-relabel method as an alternative to the blocking flow method.[a] It maintains a preflow and updates it through push operations. It introduces the relabel operation to perform fine-grain updates of the vertex distances. Push and relabel operations are local; that is, they apply to a single arc and vertex, respectively. These fine-grain operations provide additional flexibility that can be used to design faster algorithms. The fastest general-purpose maximum flow codes are based on the push-relabel method.[7,16]

For arbitrary real-valued capacities, the blocking flow problem can be solved in $O(m \log(n^2/m))$ time,[19] giving an $O(nm \log(n^2/m))$ bound for the maximum flow algorithm. Note a decomposition of flows into paths can have a size of $\Omega(nm)$. This makes $O(nm)$ a natural target bound. In 2013, Orlin[27] developed an algorithm that achieves this bound.

The maximum flow, minimum cut theorem says the maximum flow value is equal to the minimum cut capacity.

The flow decomposition size is not a lower bound for computing maximum flows. A flow can be represented in $O(m)$ space, and dynamic trees can be used to augment flow on a path in logarithmic time. Furthermore, the unit capacity problem on a graph with no parallel arcs can be solved in $O(\min(n^{2/3}, \sqrt{m})m)$ time,[13,22] which is much better than $O(nm)$. For a quarter century, there was a big gap between the unit capacity case and the general case. The gap was narrowed by Goldberg and Rao,[17] who obtained an $O(\min(n^{2/3}, \sqrt{m})m \log(n^2/m) \log U)$-time algorithm for the problem with integral capacities.

To achieve this bound, Goldberg and Rao used a non-unit length function. In combination with new design and analysis techniques, this leads to the binary blocking flow algorithm that achieves the bound mentioned earlier. As the name implies, the algorithm is based on blocking flows. No comparable bound for the push-relabel method is known. This fact revives the theoretical importance of the blocking flow method.

Here, we assume familiarity with basic graph algorithms, including breadth- and depth-first search and have organized the article as follows: After introducing basic definitions, we discuss the algorithms. Our presentation is informal, including intuitive algorithm descriptions and the corresponding time bounds, but omits technical details, which can be found in the references.

Background

The input to the maximum flow problem is (G, s, t, u), where $G = (V, A)$ is a directed graph with vertex set V and arc set A, $s \in V$ is the source, $t \in V$ is the sink (with $s \neq t$), and $u : A \Rightarrow \mathbf{R}^+$ is the strictly positive capacity function. We sometimes assume capacities are integers and denote the largest capacity by U.

A flow f is a function on A that satisfies capacity constraints on all arcs and conservation constraints at all vertices except s and t. The capacity constraint for $a \in A$ is $0 \leq f(a) \leq u(a)$ (flow does not exceed capacity). The conservation constraint for v is

a The push-relabel method is sometimes called the preflow-push method, which is misleading, as Karzanov's algorithm uses preflows and the push operation but does not use the relabel operation and is therefore not a push-relabel algorithm.

$\sum_{(u,v)\in A} f(u,v) = \sum_{(v,w)\in A} f(v,w)$ (the incoming flow is equal to the outgoing flow). The flow value is the net flow into the sink: $|f| = \sum_{(v,t)\in A} f(v,t) - \sum_{(t,v)\in A} f(t,v)$. If $|f|$ is as large as possible, f is a maximum flow. A cut is a bipartition of the vertices $S \cup T = V$ with $s \in S, t \in T$.[b] The capacity of a cut is defined by $u(S,T) = \sum_{v\in S, w\in T, (v,w)\in A} u(S,T)$ (the sum of capacities of arcs from S to T). The max-flow/min-cut theorem[14] says the maximum flow value is equal to the minimum cut capacity. Figures 1 and 2 give an input network and a maximum flow on it, respectively.

Without loss of generality, we assume G is connected. Then $m \geq n - 1$ and therefore $n + m = O(m)$. We can also assume the graph has no parallel arcs, since we can combine parallel arcs and add their capacities.

Residual Graph and Augmenting Paths

An important notion for flow algorithms is a residual graph, encoding the possible changes of flow on arcs in a way that facilitates algorithm design. Suppose we have an arc $a = (v, w)$ with $u(a) = 9$ and $f(a) = 4$. We can then increase the flow on a by up to five units without violating the capacity constraint. Furthermore, we can decrease the flow on a by up to four units. We would like to interpret decreasing flow on an arc $a = (v.w)$ as increasing flow on the reverse arc $a^R = (w, v)$.

Given a flow f in G, we define the residual graph $G_f = (V, A_f)$ as follows: A_f contains arcs $a \in A$ such that $f(a) < u(a)$ and arcs $a^R : a \in A$ such that $f(a) > 0$. We call these forward and reverse residual arcs, respectively. We define the residual capacity u_f to be $u(a) - f(a)$ for the former and $f(a^R)$ for the latter. For every arc $a \in A$, G_f contains the forward arc, the reverse arc, or both. Figure 3 gives the residual graph for the flow in Figure 2. Note the residual graph can have parallel arcs even if the input graph is simple, as it can contain both an arc and its reversal.

A flow g in G_f defines the flow f' in G as follows: For a forward arc $a \in G_f$, $f'(a) = f(a) + g(a)$; for a reverse arc $a \in G_f$, $f'(a^R) = f(a^R) - g(a)$. Seeing that f' is a valid flow is straightforward.

An augmenting path is a path from s to t in G_f. Given an augmenting path P, we can augment f as follows: Let δ be the minimum residual capacity of the arcs on P and g be the flow of value on P. The corresponding flow f' on G has $|f'| = |f| + \delta > |f|$. An augmenting path can be found in $O(m)$ time (such as by using breadth- or depth-first search).

Note that during an augmentation, at least one arc of P has residual capacity δ before the augmentation and zero after the augmentation. We say such an arc is saturated by the augmentation. Saturated arcs are deleted from G_f. An arc a is added to G_f if $u_f(a)$ is zero before the augmentation, and the augmentation increases the flow on a^R.

Using the max-flow/min-cut theorem, one can show a flow f has maximum value if and only if G_f does not contain an augmenting path. This motivates the augmenting path algorithm: while G_f contains an augmenting path, find such a path and augment the flow on it.

If capacities are integral, the augmenting path algorithm always terminates, since each augmentation increases the flow value by at least one. This observation, and the fact that the capacity of the cut $(\{s\}, V - \{s\})$ is $O(nU)$, gives a pseudo-polynomial bound of $O(nmU)$ on the algorithm's running time. The bound is not polynomial because U can be exponential in the size of the problem input. If the capacities are real-valued, the algorithm need not terminate. As we shall see later, variants of this algorithm do run in polynomial time.

Scaling

Scaling is one way to make the augmenting path algorithm polynomial-time if the capacities are integral.

Recall that U is the largest arc capacity and let $k = \lceil \log_2 U \rceil + 1$, the number of bits needed to represent capacities. For $i = 0, \ldots, k$, define $u_i(a) = \lfloor u(a)/2^{k-i} \rfloor$. Note $u_0 \equiv 0$, and for $i > 0$, u_i is defined by the i most significant bits of u. The zero flow is maximum for u_0.

Given a maximum flow f_i for capacities u_i $(0 \leq i < k)$, the algorithm computes a maximum flow f_{i+1} for capacities u_{i+1} as follows. Note $u_{i+1} = 2u_i + b_{i+1}$, where $b_{i+1}(a)$ is the $(i+1)$-st most significant bit of u(a). Thus $f = 2f_i$ is a feasible flow for capacities u_{i+1}.

Figure 1. Input example.

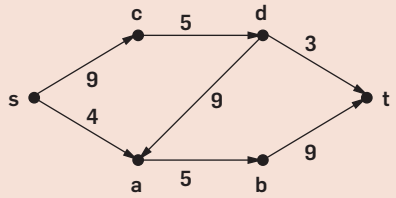


Figure 2. Maximum flow (capacity/flow) and minimum cut.

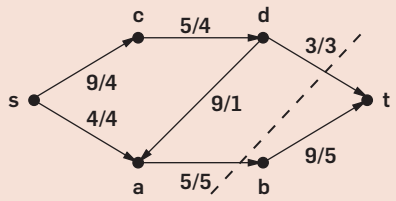
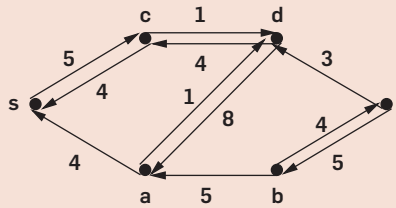


Figure 3. Residual graph and residual capacities corresponding to Figure 2.



We start with f and apply the augmenting path algorithm to compute f_{i+1}.

To bound the number of augmentations, consider a minimum cut (S, T) for capacities u_i. Since f_i is a maximum flow, for every arc a from S to T we have $u_i(a) = f_i(a)$, and thus for the initial flow f, we have $u_{i+1}(a) - f(a) \leq 1$. Therefore $|f|$ is within m of maximum, and we need at most m augmentations to compute f_{i+1} from f_i. The running time of the scaling algorithm is thus $O(m^2 \log U)$.

Shortest Augmenting Paths

Define the length of every arc in G_f to be one, and suppose we always choose a shortest augmenting path. This is natural, since breadth-first search finds shortest augmenting paths and takes linear time.

Consider a shortest-path augmentation. Let $d(v)$ denote the distance from a vertex v to t in G_f, and let $k = d(s)$. For an arc (v, w) on the augmenting path, we have $d(v) = d(w) + 1$. Therefore the reverse arc (w, v) is not on a path from s to t of length k or less. The augmentation deletes at least one arc on a path

b Formally, this defines an s-t cut, though, here, we deal only with s-t cuts; in the literature, a minimum cut may also refer to the minimum cut value over all s, t pairs of minimum s-t cuts.

from s to t of length k and does not add any arcs on paths from s to t of length k or less. This observation leads to the key monotonicity property of the shortest augmenting path algorithm: For every vertex v, residual graph distances from s to v and from v to t are non-decreasing.

The monotonicity property yields a strongly polynomial time bound. Each augmentation saturates an arc on a path of the current shortest length. Therefore, after at most m augmentations, the distance from s to t must increase. Initially, the distance is at least one, and if t is reachable from s, the distance is at most $n-1$. The total number of augmentations is thus $O(nm)$. The time for one augmentation is $O(m)$ to find the augmenting path and proportional to the path length; that is, $O(n) = O(m)$ to modify the flow. This gives an $O(nm^2)$ bound on the running time.

Blocking Flow Method

Given a network G with arc capacities, a flow f in G is blocking if every s-to-t path in G contains a saturated arc. Note f need not be a maximum flow, as there can be an s-to-t path in G_f that will contain the reverse of an arc of G (see Figure 4). But a maximum flow is always a blocking flow. As we shall see later, in an acyclic graph, blocking flows can be found more quickly than maximum flows.

The blocking flow algorithm constructs an auxiliary network $G'_f = (V, A'_f)$ where A'_f contains all residual arcs belonging to some shortest s-to-t path. Note if $(v, w) \in A'_f$, then $d(v) = d(w) + 1$, so G'_f is acyclic. G'_f can be constructed in $O(m)$ time using breadth-first search. Suppose we compute a blocking flow g in G'_f. Then $f + g$ is a feasible flow in G. Furthermore, one can show the s-to-t distance in G_{f+g} is greater than that in G_f. It follows that a maximum flow can be computed in at most $n-1$ iterations, where the time for an iteration is dominated by the blocking flow computation.

Dinic[10] introduced an algorithm for finding blocking flows in acyclic graphs, using depth-first search to find an augmenting path in G'_f that augments along the path and deletes saturated arcs from G'_f. The key to the analysis is the observation that if depth-first search retreats from a vertex, there is no path from the vertex to t in G'_f and the vertex can be deleted. One can use this observation to show the running time of the algorithm is proportional to n plus the total length of the augmenting paths found; the total length term dominates. As an augmenting path has $O(n)$ arcs and each augmentation saturates an arc, the running time of the blocking flow algorithm is $O(nm)$. This gives an $O(n^2m)$ bound for Dinic's maximum flow algorithm.

Figure 4. Example of a blocking flow that is not a maximum flow.

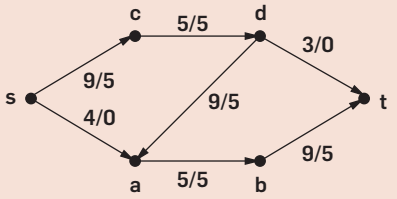
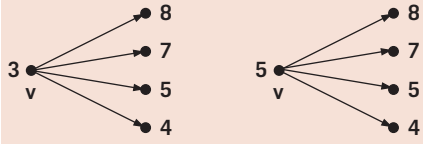


Figure 5. Push operation example: before (left) and after (right); zero excesses and non-residual arcs not shown.



Figure 6. Relabel operation example: before (left) and after (right).



Using Dynamic Trees

The running time of the blocking flow algorithm is dominated by changes of arc flows that do not saturate the arc. A natural approach to improving the running time bound is to use a data structure that allows one to make several such changes in one data structure operation. This can be achieved by using a data structure to remember non-saturated portions of the augmenting paths. The dynamic tree data structure[29,30] was developed for this purpose.

Intuitively, the dynamic tree blocking flow algorithm uses the data structure to remember non-saturated portions of augmenting paths that may be reused later. In particular, the saved paths are linked during the augmenting path search, work that is amortized over the search for augmenting paths. Flow augmentation is performed using dynamic tree operations, at the cost of the logarithm of the corresponding augmenting path length. This application of dynamic trees reduces the running time of the blocking flow algorithm from $O(nm)$ to $O(m \log n)$. By restricting the maximum tree size and using additional data structures, this bound can be further improved to $O(m \log(n^2/m))$,[19] yielding an $O(nm \log(n^2/m))$ maximum flow algorithm.

Although dynamic trees yield the best worst-case bounds, they have so far not been used in practical implementations because most practical instances are relatively easy, and the constant factors in dynamic tree implementations are relatively large.

Push-Relabel Method

The blocking flow algorithm uses global operations (such as building the auxiliary network and augmenting along a path). The push-relabel method uses local operations. These fine-grain operations give the method more flexibility, which can be used to make the method faster in practice.

Following Karzanov,[23] the push-relabel method uses preflows. Preflows are like flows, but the conservation constraints are relaxed: $\sum_{(u,v)\in A} f(u, v) \geq \sum_{(v,w)\in A} f(v, w)$ for all $v \in V - \{s, t\}$ (the incoming flow is at least the outgoing flow). We define excess by $e_f(v) = \sum_{(u,v)\in A} f(u, v) - \sum_{(v,w)\in A} f(v, w)$. A vertex with excess can push some of it to its residual neighbor. Intuitively, we want to push only to a neighbor that is closer to the sink. Karzanov[23] uses distances in the auxiliary network to determine where to push flow.

The push-relabel method replaces the distances by a valid labeling, a relaxation of distances that can be updated locally. Given a flow f, we say a function $d : V \rightarrow \mathcal{N}$ is a valid labeling if $d(t) = 0$ and for every $(v, w) \in A_f$, we have $d(v) \leq d(w) + 1$. One can show a valid labeling gives lower bounds on distances to t. In particular, if $d(v) \geq n$, then there is no path from v to t in G_f, meaning v is on the source side of some minimum cut.

The push-relabel method maintains a preflow f and a valid distance labeling

d and updates them using operations push and relabel, respectively. We describe these operations next; a full description of the algorithm can be found in Goldberg and Tarjan.[18]

One way to start the push-relabel method is to saturate all arcs out of the source by setting their flow values to the corresponding capacity values, and to set $d(s) = n$ (t is not reachable from s) and $d(v) = 0$ for $v \neq s$. This creates initial flow excesses on vertices adjacent to the source. Intuitively, the method pushes flow excesses toward the sink and relabels vertices with excess if the excess cannot be pushed toward the sink. We say a vertex v is active if $v \neq t$ and $e_f(v) > 0$.

The push operation applies to an arc (v, w) if v is active and $d(w) < d(v)$, or w is closer to t according to d. The operation determines the maximum amount of flow that can be pushed, $\delta = \min(e_f(v); u_f(v, w))$ and pushes this amount of flow along (v, w) by setting $u_f(v, w) = u_f(v, w) - \delta$, $u_f(w, v) = u_f(w, v) + \delta$, $e_f(v) = e_f(v) - \delta$, and $e_f(w) = e_f(w) + \delta$. We say a push is saturating if $\delta = u_f(v, w)$ and non-saturating otherwise. Note that a non-saturating push gets rid of all the excess of v; see Figure 5 for an example of a non-saturating push operation.

The relabel operation applies to an active vertex v such that no push operation applies to an arc (v, w), or for all $(v, w) \in A_f$, $d(v) \leq d(w)$. The operation sets $d(v) = \min\{n, 1 + \min_{(v,w)\in A_f} d(w)\}$. (A vertex with excess always has an outgoing residual arc.) Note the relabel operation always increases $d(v)$; see Figure 6 for an example of a relabel operation.

The time complexity of the push-relabel method is as follows. The total time for relabeling operations is $O(nm)$, and the time for saturating pushes is $O(nm)$ as well. The time for non-saturating pushes is $O(n^2m)$; these operations dominate the running time bound, which is also $O(n^2m)$.

Note our description of the push-relabel method is generic; we have not specified the rule to select the next active vertex to process. Some operation orderings lead to better bounds. In particular, for the highest label push-relabel algorithm, which always selects an active vertex with the highest distance label to process next, the time for non-saturating pushes and the overall time bound are $O(n^2\sqrt{m})$.[6] Using dynamic trees, one can get an $O(nm \log(n^2/m))$ bound[18] more simply than through the blocking flow method.

An interesting special case of the maximum flow problem involves all arcs having unit capacities.

The highest-label algorithm is also one of the most practical variants of the push-relabel method. However, robust practical performance requires additional heuristics. The push relabel method is very flexible, making it easy for the algorithm designer to add heuristics. For example, one can restrict active vertices to those with $d(v) < n$ and do post-processing to compute the final flow. One can also do periodic backward breadth-first searches to maximize $d(v)$ values. See, for example, Cherkassky and Goldberg[7] and Goldberg.[16]

Unit Capacities

Now consider the special case of the maximum flow problem in which all input arc capacities are one. Since merging parallel arcs results in non-unit capacities, we cannot assume the graph has no parallel arcs, so we consider two cases—parallel arcs and no parallel arcs—in both of which one obtains better bounds for Dinic's algorithm.

First, note that after an augmentation, all arcs on the augmenting path are saturated. Therefore, an arc participates in at most one augmentation per blocking flow, and the blocking flow algorithm runs in $O(m)$ time.

Moreover, one can show the number of blocking flow computations is $O(\sqrt{m})$. To prove this bound, we divide the maximum flow computation into two phases. In the first phase, the s-to-t distance is less than $\sqrt{m}$. Since an augmentation by a blocking flow increases the distance, the first phase consists of at most $\sqrt{m}$ augmentations. One can show if the s-to-t distance is at least $\sqrt{m}$, the residual flow value is $O(\sqrt{m})$. Since an augmentation decreases the value, the number of augmentations in the second phase is $O(\sqrt{m})$.

This analysis implies an $O(m^{3/2})$ bound for the unit capacity problem. If G has no parallel arcs, one can also obtain an $O(n^{2/3}m)$ bound, which is better for dense graphs.

Binary Blocking Flow Algorithm

The time bounds for the blocking flow

and push-relabel algorithms are $\Omega(nm)$ in the general case, while for unit capacities, the algorithm of Dinic runs in $O(\min(n^{2/3}, \sqrt{m})m)$ time. Here, we discuss the intuition behind the binary blocking flow algorithm of Goldberg and Rao,[17] which narrows the gap for the integral capacity case.

Instead of assigning unit length to every residual arc, the binary blocking flow algorithm uses a zero-one length function, assigning zero length to the arcs with large residual capacity and unit length to the arcs with small residual capacity. The fact that arcs with unit length have small residual capacity allows the algorithm to come close to the unit capacity time bound.

The algorithm maintains a flow f and an upper bound F on the difference between the maximum flow value and the current flow value $|f|$. The algorithm proceeds in phases; each phase decreases F by a factor of two. In a phase, the value of F remains constant except for the very end of the phase, when it is decreased. A threshold parameter Δ, which is a function of F and thus remains constant during a phase, determines whether residual arcs are large or small; large arcs have a residual capacity of at least 3Δ, and the remaining ones are small.

As in the case of the unit length function, we define the auxiliary network G'_f to be the graph induced by the arcs on shortest s-to-t paths. The algorithm repeatedly computes a blocking flow in G'_f, updating G'_f before each computation, until F decreases by a factor of two. The decrease in F happens if one either increases the flow by $F/2$ or the s-to-t distance becomes sufficiently large. As in the unit capacity case, a large s-t distance implies a bound on the residual flow value.

This use of the binary length function leads to two problems that must be addressed: First, G'_f need not be acyclic (it can contain cycles of zero-length arcs), and, second, an arc length can decrease from one to zero, and, as a side effect, the s-to-t distance may fail to increase after a blocking flow augmentation.

To deal with the first problem, the algorithm contracts strongly connected components of G'_f, looks for a blocking flow in the resulting acyclic graph, and stops the blocking flow computation if the value of the flow being computed reaches Δ. One can show that since each strongly connected component is induced by large-capacity arcs, one can always route a flow of value Δ through it. At the end of each iteration, we expand G'_f and extend the flow we found to G_f. One can show that a blocking flow in G'_f extends to a blocking flow in G_f. This gives us two types of iterations: ones that find a blocking flow and ones that find a flow of value Δ. The s-to-t distance increases in the former case and does not decrease in the latter case.

The maximum flow problem is far from being completely understood, and new and improved algorithms continue to be discovered.

To deal with the second problem, one can show the arcs (v, w) that may have their length decrease (the special arcs) have the property that the residual capacity of (v, w) is at least 3Δ and the residual capacity of (w, v) at least 2Δ. The algorithm contracts such arcs, assuring that after an augmentation by a blocking flow, the s-to-t distance increases even if these arc lengths decrease.

Using the dynamic-tree data structure, the binary flow algorithm runs in $O(\min(n^{2/3}, \sqrt{m})m \log(n^2/m) \log U)$ time, which is within a $\log(n^2/m) \log U$ factor of the best known upper bound for the unit capacity problem with no parallel arcs.

Conclusion

As mentioned here, a 2013 algorithm of Orlin[27] achieves an $O(nm)$ strongly polynomial bound for the maximum flow problem, as well as an $O(n^2/\log n)$ bound for $m = O(n)$. This result is quite sophisticated and uses a combination of ideas from maximum flow, minimum-cost flow, and dynamic connectivity algorithms. In particular, Orlin uses the binary blocking flow algorithm as a subroutine. His result closes a longstanding open problem of the existence of an $O(nm)$ maximum flow algorithm. However, the binary blocking flow algorithm bounds suggest an $O(nm/n^{\epsilon})$ strongly polynomial algorithm may exist.

The maximum flow problem is far from being completely understood, and new and improved algorithms continue to be discovered. We would like to mention four intriguing directions that have yielded new results: The first is to generalize the push-relabel approach to allow the flow excess (incoming minus outgoing flow) at a vertex to be arbitrary—positive,

negative, or zero. Given a residual arc (v, w) such that the excess at v exceeds the excess at w, one can balance the arc by increasing its flow to either saturate the arc or equalize the excesses at v and w. The flow balancing algorithm[31] starts with some initial flow (such as zero flow), dummy excesses of plus infinity at s and minus infinity at t, and repeats arc-balancing steps until all such steps move a sufficiently small amount of flow, then rounds the flow to obtain an exact maximum flow. Although the running time of this algorithm ($O(n^2 m \log U)$) is not competitive with that of the best algorithms, the method is simple and extends to give a very simple and practical algorithm for a parametric version of the maximum flow algorithm.[2,31]

Another approach that yields a fast practical algorithm for maximum flow problems in computer-vision applications is that of Boykov and Kolmogorov,[5] improving the basic augmenting path method by using bidirectional search to find augmenting paths, in combination with a clever method for retaining information from previous searches to speed up future ones. The Boykov-Kolmogorov method does not augment on shortest paths and has not been proved to be polynomial but can be modified to find exact shortest paths and to be polynomial without sacrificing its practical performance, indeed improving it in many cases. The resulting algorithm[20] computes shortest augmenting paths incrementally, using information from previous searches. Special techniques have yielded fast maximum flow algorithms for planar graphs and for undirected graphs; for the latest results, see Borradaile and Klein,[3] Borradaile et al.,[4] and Karger and Levine.[21]

A recent series of papers, including Christiano et al.,[8] Kelner et al.,[24] Lee et al.,[25] and Sherman,[28] have studied the problem of finding an approximately maximum flow (within a factor of $1 + \epsilon$ of maximum) in undirected graphs and culminates in a near-linear time algorithm. These papers used linear algebraic techniques and electrical flows. Building on this work, Madry[26] in 2013 obtained a breakthrough result, an exact algorithm for unit capacity flows in directed graphs running in $\tilde{O}(m^{10/7})$ time. This improves the classical $O(\min(n^{2/3}, m^{1/2})m)$ bound for the problem, suggesting that better bounds for the exact capacitated maximum flow problem in directed graphs may be possible. Whether these ideas can be used to find exact maximum flows in directed graphs with integral capacities is an intriguing open question. In summary, progress on maximum flow algorithms has been made for more than half a century, and continues.

Acknowledgment

Robert E. Tarjan was supported by National Science Foundation grant CCF-0830676 and U.S.-Israel Binational Science Foundation Grant 2006204. Some of his work was done while visiting Stanford University and supported by an Air Force Office of Scientific Research Multidisciplinary University Research Initiative grant.

References

1. Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1993.
2. Babenko, M.A., Derryberry, J., Goldberg, A.V., Tarjan, R.E., and Zhou, Y. Experimental evaluation of parametric max-flow algorithms. In *Proceedings of the Sixth Workshop on Experimental Algorithms, Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 2007, 256–269.
3. Borradaile, G. and Klein, P.N. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *Journal of the ACM 56,* 2 (2009), 1–34.
4. Borradaile, G., Klein, P.N., Mozes, S., Nussbaum, Y., and Wulff-Nilsen, C. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*. IEEE Press, New York, 2011,170–179.
5. Boykov, Y. and Kolmogorov, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26,* 9 (2004), 1124–1137.
6. Cheriyan, J. and Maheshwari, S.N. Analysis of preflow push algorithms for maximum network flow. *SIAM Journal on Computing 18,* 6 (1989), 1057–1086.
7. Cherkassky, B.V. and Goldberg, A.V. On implementing push-relabel method for the maximum flow problem. *Algorithmica 19,* 4 (1997), 390–410.
8. Christiano, P., Kelner, J.A., Madry, A., Spielman, D.A., and Teng, S-H. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 2011, 273–282.
9. Dantzig, G.B. Application of the simplex method to a transportation problem. In *Activity Analysis and Production and Allocation,* T.C. Koopmans, Ed. John Wiley & Sons, Inc., New York, 1951, 359–373.
10. Dinic, E.A. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematical Docladi 11* (1970), 1277–1280.
11. Dinic, E.A. Metod porazryadnogo sokrashcheniya nevyazok i transportnye zadachi [Excess scaling and transportation problems]. In *Issledovaniya po Diskretnoi*. Matematike Nauka, Moscow, Russia, 1973.
12. Edmonds, J. and Karp, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19,* 2 (1972), 248–264.
13. Even, S. and Tarjan, R.E. Network flow and testing graph connectivity. *SIAM Journal on Computing 4,* 4 (1975), 507–518.
14. Ford, Jr., L.R. and Fulkerson, D.R. Maximal flow through a network. *Canadian Journal of Mathematics 8* (1956), 399–404.
15. Galil, Z. and Naamad, A. An $O(EV \log^2 V)$ algorithm for the maximal flow problem. *Journal of Computer and System Sciences 21,* 2 (1980), 203–217.
16. Goldberg, A.V. Two-level push-relabel algorithm for the maximum flow problem. In *Proceedings of the Fifth Conference on Algorithmic Aspects in Information Management, Volume 5564 of Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 2009, 212–225.
17. Goldberg, A.V. and Rao, S. Beyond the flow decomposition barrier. *Journal of the ACM 45,* 5 (1998), 753–782.
18. Goldberg, A.V. and Tarjan, R.E. A new approach to the maximum flow problem. *Journal of the ACM 35,* 4 (1988), 921–940.
19. Goldberg, A.V. and Tarjan, R.E. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research 15,* 3 (1990), 430–466.
20. Goldberg, A.V., Hed, S., Kaplan, H., Tarjan, R.E., and Werneck, R.F. Maximum flows by incremental breadth-first search. In *Proceedings of the 19th European Symposium on Algorithms*. Springer-Verlag, Heidelberg, Germany, 2011, 457–468.
21. Karger, D.R. and Levine, M. Finding maximum flows in undirected graphs seems easier than bipartite matching. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 1997.
22. Karzanov, A.V. Tochnaya otzenka algoritma nakhojdeniya maksimalnogo potoka, primenennogo k aadache 'o predstavitelyakh' [The exact time bound for a maximum flow algorithm applied to the set representatives problem]. In *Problems in Cibernetics 5* (1973), 66–70.
23. Karzanov, A.V. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematical Dokladi 15* (1974), 434–437.
24. Kelner, A., Lee, Y.T., Orecchia, L., and Sidford, A. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, 2014, 217–226.
25. Lee, T., Rao, S., and Srivastava, N. A new approach to computing maximum flows using electrical flows. In *Proceedings of the Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 2013, 755–764.
26. Madry, A. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*. IEEE Press, New York, 2013, 253–262.
27. Orlin, J.B. Max Flows in $O(nm)$ time, or better. In *Proceedings of the Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 765–774.
28. Sherman, J. Nearly maximum flows in nearly linear time. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*. IEEE Press, New York, 2013, 263–269.
29. Sleator, D.D. and Tarjan, R.E. A data structure for dynamic trees. *Journal of Computer and System Sciences 26,* 3 (1983), 362–391.
30. Sleator, D.D. and Tarjan, R.E. Self-adjusting binary search trees. *Journal of the ACM 32,* 3 (1985), 652–686.
31. Tarjan, R.E., Ward, J., Zhang, B., Zhou, Y., and Mao, J. Balancing applied to maximum network flow problems. In *Proceedings of the 14th European Symposium on Algorithms*. Springer-Verlag, Berlin, 2006, 612–623.

Andrew V. Goldberg (goldberg@microsoft.com) is a principal researcher at Microsoft Research Silicon Valley Lab, Mountain View, CA.

Robert E. Tarjan (ret@cs.princeton.edu) is the James S. McDonnell Distinguished University Professor of Computer Science at Princeton University, Princeton, NJ, and a visiting researcher at Microsoft Research Silicon Valley Lab, Mountain View, CA.

© 2014 ACM 0001-0782/14/08 $15.00

ACM's Career & Job Center

Are you looking for your next IT job? Do you need Career Advice?

The **ACM Career & Job Center** offers ACM members a host of career-enhancing benefits:

- A **highly targeted focus** on job opportunities in the computing industry
- **Access to hundreds** of industry job postings
- Resume posting **keeping you connected** to the employment market while letting you maintain full control over your confidential information
- **Job Alert system** that notifies you of new opportunities matching your criteria
- **Career coaching** and guidance available from trained experts dedicated to your success
- **Free access** to a content library of the best career articles compiled from hundreds of sources, and much more!



Visit ***ACM's Career & Job Center*** *at:*
http://jobs.acm.org

Association for Computing Machinery

Advancing Computing as a Science & Profession

The **ACM Career & Job Center** is the perfect place to begin searching for your next employment opportunity!

Visit today at http://jobs.acm.org

P. 92
Technical Perspective
Getting Consensus for Data Replication
By Philip A. Bernstein

P. 93
Quantifying Eventual Consistency with PBS
By Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica

DOI:10.1145/2632794

To view the accompanying paper, visit doi.acm.org/10.1145/2632792 rh

Technical Perspective
Getting Consensus for Data Replication

By Philip A. Bernstein

Given R, W, and N, the authors of the following paper offer a formula to calculate the probability of reading data that was not written by one of the K most recent writes.

DATA IN A cloud computing system should be highly available. That is, whenever you connect to the system, the data you stored there should be ready to use. The standard mechanism to accomplish this—*data replication*—involves maintaining multiple copies of each user's data.

By itself, replicating data does not solve the problem, because the copy of data that is available might be stale. To see why, consider a system that maintains three copies of file x: x_1, x_2, and x_3. Suppose x_1 and x_2 are currently available and x_3 is down. If a program updates x, the system writes the update to x_1 and x_2, but not to x_3 because it is down. No problem, since the system still has two correct copies. Next, suppose x_1 and x_2 fail, and then x_3 recovers. Now there is a problem. If a user reads x, the best the system can do is return the value of x_3. But that copy is stale—it does not have the latest update.

An early solution to this problem is called *majority consensus*.[2] To process a write operation on x, the system assigns a monotonically increasing timestamp to the write, and writes x's value and timestamp to a majority of the copies of x. To process a read operation on x, it reads a majority of the copies of x and returns one that has the largest timestamp. Since the intersection of any two majorities includes at least one copy, each read returns the result of the previous writes on the same data.

In our example, the read operation only reads one copy, which is not a majority of three. Therefore, it might not return the latest state of x. Using majority consensus, the system would have to wait until a second copy became available, so it could read two copies. Then, it could return the more up-to-date copy of the two that it reads, which must be the freshest.

The notion of majority was extended in Gifford[1] to be a weighted majority, which is called a *quorum*. Each copy is given a weight. A read must access a *read quorum* of copies—a set of copies whose weight is at least R. And a write must update a *write quorum* of copies—a set whose weight is at least W. By requiring that R+W exceeds the total weight N of all copies, we are assured that each read reads a copy that was written by the last write.

For example, suppose the system stores four copies of x, namely, x_1-x_4. We might assign a weight of 1 to x_1 and x_2, but assign a weight of 2 to x_3 and x_4 if they are stored on more reliable servers. So N=6. If reads are more frequent than writes, we might set R=3 and W=4, so that reads have less work to do than writes. Since R+W>N, each read operation reads the result of the previous write on the same data.

It stands to reason that each write has to update all available copies of the data. But it is annoying that readers have to read multiple copies too, since it adds overhead and delay. It is especially annoying if writes usually execute quickly, since in this case each read is very likely to read the latest copy even if it does not read a quorum. If the probability of reading stale data is sufficiently small, then it might be satisfactory to allow readers to read less than a quorum of copies. In fact, some cloud applications do exactly that.

Until recently, this compromise was done by guesswork, without a bound on the staleness of the data that is read or the expected latency improvement from risking some staleness. The following paper is a breakthrough that removes the guesswork and replaces it by a principled analysis, called probabilistically bounded staleness. Given R, W, and N, the authors offer a formula to calculate the probability of reading data that was not written by one of the K most recent writes. To calculate the probability of reading data that was not written in the last delta time units, they use a Monte Carlo simulation based on time parameters that can be measured in a running system. Moreover, they made the analysis practical by implementing it in open source record managers, so that users can make their own judgment on trading off staleness for latency. What was formerly a guess is now a goal you can engineer for. C

References

1. Gifford, D.K. Weighted voting for replicated data. *SOSP* (1979), 150–162.
2. Thomas, R.H. A majority consensus approach to concurrency control for multiple-copy databases. *ACM Trans. Database Syst. 4*, 2 (1979), 180–209.

Philip A. Bernstein (http://research.microsoft.com/~philbe) is a Distinguished Scientist at Microsoft Research, Redmond, WA.

Copyright held by author.

DOI:10.1145/2632792

Quantifying Eventual Consistency with PBS

By Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica

Abstract

Data replication results in a fundamental trade-off between operation latency and consistency. At the weak end of the spectrum of possible consistency models is eventual consistency, which provides no limit to the staleness of data returned. However, anecdotally, eventual consistency is often "good enough" for practitioners given its latency and availability benefits. In this work, we explain this phenomenon and demonstrate that, despite their weak guarantees, eventually consistent systems regularly return consistent data while providing lower latency than their strongly consistent counterparts. To quantify the behavior of eventually consistent stores, we introduce Probabilistically Bounded Staleness (PBS), a consistency model that provides expected bounds on data staleness with respect to both versions and wall clock time. We derive a closed-form solution for version-based staleness and model real-time staleness for a large class of quorum replicated, Dynamo-style stores. Using PBS, we measure the trade-off between latency and consistency for partial, non-overlapping quorum systems under Internet production workloads. We quantitatively demonstrate how and why eventually consistent systems frequently return consistent data within tens of milliseconds while offering large latency benefits.

1. INTRODUCTION

Modern distributed data stores need to be scalable, highly available, and fast. These systems typically replicate data across different machines and increasingly across datacenters for at least two reasons: first, to provide availability when components fail and, second, to provide improved performance by serving requests from multiple replicas. Configuring and maintaining replicated data has significant consequences for application and data store design.[1] Performance at scale is critical for a large class of applications and, in practice, increased latencies may correspond to large amounts of lost revenue.[22] For example, at Amazon, 100 ms of additional latency resulted in a 1% drop in sales,[15] while 500 ms of additional latency in Google's search resulted in a corresponding 20% decrease in traffic.[16] However, lowering latency in distributed data stores has a cost: contacting fewer replicas for each operation can adversely impact achievable semantic guarantees.

To provide predictably low latency, modern systems often eschew protocols guaranteeing "strong" consistency of reads (e.g., the illusion of a single copy of replicated data) and instead opt for "weaker" semantics, frequently in the form of *eventual consistency*.[1, 5, 7, 10] This eventual consistency is one of the weakest properties provided by modern stores: it provides no guarantees on data staleness except that, in the absence of new writes, reads will "eventually" return the effect of the most recent write(s).[26] Under this definition, a store that returns data that is weeks old is eventually consistent, as is a store that returns arbitrary data (e.g., always return value 42) as long as, at *some point* in the future, the store returns the last written data.[4] Due to this near-absence of useful semantics for end users, the decision to employ eventual consistency is often controversial.[12, 23, 24] In the many production stores providing eventual consistency today,[10, 14] users have little to no insight into the behavior of their stores or the consistency of their data, especially under varying replication configurations. However, the proliferation of eventually consistent deployments suggests that applications can often tolerate occasional staleness and that data tends to be "fresh enough" in many cases.

In this work, we bridge this gap between theoretical guarantees and current practice by quantifying the degree to which eventual consistency is both eventual and (in) consistent and explain why. Indeed, under worst-case conditions, eventual consistency results in an unbounded degree of data staleness. However, as we will show, the common case is often different. Core to our thesis is the observation that eventual consistency can be modeled as providing a probabilistic *expectation* of consistency given a particular workload and deployment environment. Accordingly, for varying degrees of certainty, eventually consistent stores can offer bounds on how far they may deviate from strongly consistent behavior. We present probabilistic models for such bounds called Probabilistically Bounded Staleness, or PBS.

To predict consistency with PBS, we need to know when and why eventually consistent systems return stale data and how to quantify the staleness of the data they return. In this paper, we present algorithms and models for two common staleness metrics in the literature: wall clock time[21] and versions.[27] PBS describes both measures, providing the probability of reading a write Δ seconds after

The original version of this paper is entitled "Probabilistically Bounded Staleness for Practical Partial Quorums" and was published in *VLDB 2012*.[5] An invited extended version of this paper is entitled "Quantifying Eventual Consistency with PBS" and will appear in the *VLDB Journal*'s "Best of VLDB 2012" issue in 2014.[7] Portions of this work also appear in a *SIGMOD 2013* demo entitled "PBS at Work: Advancing Data Management with Consistency Metrics."[6]

the write returns ((Δ, p)-semantics, or "how eventual is eventual consistency?"), of reading one of the last K versions of a data item ((K, p)-semantics, or "how consistent is eventual consistency?"), and of experiencing a combination of the two ((K, Δ, p)-semantics). PBS does not propose new mechanisms to enforce deterministic staleness bounds;[27] instead, our goal is to provide a lens for analyzing, improving, and predicting the behavior of *existing*, widely deployed systems.

In this work, we apply PBS to quorum-replicated data stores such as Dynamo[10] and its several open-source descendants. Quorum systems ensure strong consistency across reads and writes to replicas by ensuring that read and write replica sets overlap. However, employing *partial* (or non-strict) quorums can lower latency by requiring fewer replicas to respond. With partial quorums, sets of replicas written to and read from need not overlap: given N replicas and read and write quorum sizes R and W, partial quorums imply $R + W \leq N$. For partial quorums, we derive closed-form solutions for PBS (K, p)-regular semantics and use Monte Carlo methods to explore the trade-off between latency and (Δ, p)-regular semantics.

Finally, we use PBS to study the staleness observed in production deployments of Dynamo-style data stores under normal operation (i.e., failure-free scenarios). We show how high variance in write latency can lead to an increased window of inconsistency. For example, in one production environment, switching from spinning disks to solid-state drives dramatically improved expected consistency (e.g., 1.85 ms versus 45.5 ms wait time for a 99.9% probability of consistent reads) due to decreased write latency mean and variance. We also make quantitative observations of the latency-consistency trade-offs offered by partial quorums. For example, in another production environment, PBS calculations show an 81.1% combined read and write latency improvement at the 99.9th percentile (230 to 43.3 ms) for a 202-ms window of inconsistency (99.9% probability consistent reads). This analysis helps demonstrate the performance benefits that lead operators to choose eventual consistency and motivates additional end-user applications ranging from consistency monitoring to consistency-based service-level agreements (SLAs) and query planning.

2. PROBABILISTICALLY BOUNDED STALENESS

By popular definitions, eventually consistent data stores do not make any guarantees as to the recency of data that they return: "if no new updates are made to the object, eventually all accesses will return the last updated value."[26] This is a useful *liveness* property, guaranteeing that something good eventually happens, but it provides no *safety* properties: the data store can return any data in the interim.[4] Many real-world eventually consistent stores do not make any guarantees beyond this definition, yet they are widely deployed and have seen increased popularity over the past decade (Section 3.2). As we will see, the protocols used by most eventually consistent stores indeed do not *enforce* additional guarantees, but they may *supply* them during operation.

To quantify semantics that are not guaranteed but are often provided, we develop a probabilistic framework for reasoning about consistency, called Probabilistically Bounded Staleness, or PBS. There are a wide range of possible consistency models that a data store can provide, from linearizability to causal consistency to eventual consistency; what is the likelihood that an end user will observe a given consistency model if it is not guaranteed? Here, we study variants of two classic kinds of (in)consistency in the form of staleness: versions and time. We develop variants of PBS metrics that provide quantitative expectations that a store will return a version that was written within the last K writes of the latest (where $K = 1$ is latest) and that a store will return the latest version as of Δ seconds ago. Ultimately, this does *not* provide a guarantee, but it is still useful for reasoning about and introspecting the behavior of a given system, similar to the usage of modern SLAs for performance (Section 6).

To begin, we first need to define a baseline for "strongly consistent" semantics. The Dynamo-style stores we study provide a choice between "strong" *regular* semantics and eventual consistency; we subsequently observe when the eventually consistent choices behave like their "strong" counterparts. According to the distributed systems literature:[2]

DEFINITION 1. *A read from a given data item obeys* regular semantics *if, in the case that the read does not overlap (in real time) with any writes to the same item, it returns the result the last completed write, or, if the read overlaps (in real time) with at least one write to the same data item, it returns either the result of the last completed write or the eventual result of one of the overlapping writes.*

Accordingly, regular semantics provide the illusion of a single copy of each replicated data item, except when there are concurrent reads and writes, during which writes' effects may become visible and subsequently "disappear." While this special, overlapping case is somewhat awkward to reason about (cf. definitions of linearizability[13]), this is a widely deployed data replication configuration.

Before we consider PBS applied to regular semantics, we first present a generalization of the semantics to account for multi-version staleness:[2]

DEFINITION 2. *A read from a given data item obeys K-*regular semantics *if, in the case that the read does not overlap (in real time) with a write to the same data item, it returns the result of one of the latest K completed writes, or, if the read overlaps (in real time) with a write to the same item, it returns either the result of one of the latest K completed writes or the eventual result of one of the overlapping writes.*

K-regular semantics are useful for reasoning about *how stale* a given read can be and can also be used to enforce additional consistency properties such as monotonic reads, where reads do not appear to "go back in time."[5, 25]

We now present our first application of PBS principles. Given a system that does not guarantee *K*-regular

semantics, we can reason about its semantics by taking a probabilistic approach:

Definition 3. *A system provides* (K, p)-regular semantics *if each read provides K-regular semantics with probability p.*

This modification is simple and straightforward: given an existing semantic guarantee, we can consider a probabilistic version of it, whereby reads may or may not obey the property. Of course, the above is simply a definition, and we must still determine how to actually provide PBS predictions, which will be the focus of the rest of the paper.

In addition to considering version-based staleness, we can also consider staleness with respect to real time. As we will see, message propagation and processing delays can influence consistency across time, so we extend regular semantics to consider time as well:

Definition 4. *A read obeys* Δ-regular semantics *if it returns either the result of the latest write as of up to* Δ *time units ago, the result of a write that was started but not completed as of up to* Δ *time units ago.*

We can similarly extend this definition to a probabilistic context:

Definition 5. *A system provides* (Δ, p)-regular semantics *if each read obeys* Δ*-regular semantics with probability p.*

Although we do not consider them in this paper, it is possible to consider both time and version staleness together, which we term (K, Δ, p)-semantics.[5, 7]

3. QUORUM SYSTEM BACKGROUND

With PBS metrics in hand, we can proceed to apply them to real systems, where they are most useful. In this study, we will consider PBS metrics in the context of quorum-replicated data stores, which represent a large class of widely deployed real-world distributed data stores. However, with some work, we believe that our methodology is also applicable to other styles of replication. Here, we provide background on quorum systems, with a focus on current practice.

3.1. Quorum foundations: Theory

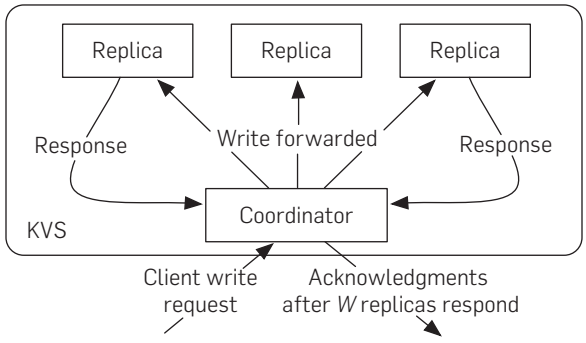
Quorum systems have a long tradition as a replication strategy for distributed data.[20] Under quorum replication, a data store writes a data item by sending it to a set of servers responsible for the *replicas*, called a write quorum. To serve reads, the data store fetches the data from a possibly different set of replicas, called a read quorum. For reads, the data store compares the set of values returned by the replicas, and, given a total ordering of versions, can return the most recent value (or all values received, if desired). For each operation, the data store chooses (read or write) quorums from a set of sets of replicas, called a *quorum system*, with one system per data item. There are many kinds of quorum systems, but one simple configuration is to use read and write quorums of fixed sizes, which we will denote R and W, for a set of replicas of size N. A *strict quorum system* has the property that any two quorums in the quorum system overlap (have non-empty intersection), providing regular semantics. A simple example of a strict quorum system is the majority quorum system, in which each quorum is of size $\lceil \frac{N+1}{2} \rceil$. *Partial quorum systems*, which we will study, are a natural relaxation of strict quorum systems: at least two quorums in a partial quorum system do not overlap.[19]

3.2. Quorum foundations: Practice

In practice, many distributed data management systems use quorums as a replication mechanism. Amazon's Dynamo[10] is the progenitor of a class of eventually consistent data stores that include Apache Cassandra,[a] Basho Riak,[b] and Project Voldemort.[c] All of these systems use the same variant of quorum-style replication and we are not aware of any widely adopted data store that uses a substantially different quorum-based replication protocol.

Dynamo-style quorum systems employ one quorum system per data item, typically maintaining the mapping of items to quorum systems using a consistent-hashing scheme or a centralized membership protocol. Each server in the system cluster stores multiple items. As shown in Figure 1, clients send read and write requests to a server in the system cluster, which subsequently forwards the request to *all* replicas for the operation's item. This coordinating server considers an operation complete when it has received responses from a predetermined number of replicas (typically configured per-operation). Accordingly, without message loss, all replicas eventually receive all writes. Dynamo denotes the replication factor of an item as N, the number of replica responses required for a successful read as R, and the number of replica acknowledgments required for a successful write as W. Like other strict quorum systems, Dynamo provides regular semantics when $R + W > N$ during failure-free operation. However, unlike traditional

Figure 1. Diagram of control flow for client write to Dynamo-style quorum ($N = 3$, $W = 2$). A coordinator server handles the client write and sends it to all N replicas. The write call returns after the coordinator receives W acknowledgments.



[a] http://cassandra.apache.org/
[b] http://www.basho.com/riak/
[c] http://www.project-voldemort.com/

quorum systems, Dynamo's write quorum size increases even after the operation returns, growing via *anti-entropy*.[4, 10] Coordinators send all requests to all replicas but consider only the first R (W) responses. As a matter of nomenclature (and to disambiguate against "dynamic" quorum membership protocols), we will refer to these systems as *expanding partial quorum systems*.

As we discuss in extended versions of this paper,[5, 7] system operators often report using partial quorum configurations in Dynamo-style stores, citing "maximum performance" in the "general case," particularly for "low value" data or queries that need "very low latency and high availability."

4. PBS AND PARTIAL QUORUMS

Given our PBS metrics and an understanding of quorum behavior, we can develop models for the probability of consistency under partial quorums. Here, we briefly discuss version-based staleness for traditional probabilistic quorums and develop a more complex "white box" model of time-based staleness for Dyanamo-style systems.

4.1. PBS (K, p)-regular semantics

To understand static, non-expanding quorum behavior, we first revisit probabilistic quorum systems,[19] which provide probabilistic guarantees of quorum intersection in partial quorum systems. As an example, consider N replicas with read and write quorums of sizes R and W chosen uniformly at random. We can calculate the probability that the read quorum does not contain the last written version. This probability is the number of quorums of size R composed of replicas that were not written to in the write quorum divided by the number of possible read quorums:

$$p_s = \frac{\binom{N-W}{R}}{\binom{N}{R}} \quad (1)$$

The probability of inconsistency is high for small values of N. However, by scaling the number of replicas and quorum sizes, one can achieve an arbitrarily high probability of consistency.[19] For example, with $N = 3$, $R = W = 1$, $p_s = 0.\overline{6}$, but with $N = 100$, $R = W = 30$, $p_s = 1.88 \times 10^{-6}$.[2] This is reminiscent of the Birthday Paradox: as the number of replicas increases, the probability of non-intersection between any two quorums decreases. Hence, the asymptotics of these systems are excellent—but only at asymptotic scales.

While probabilistic quorums allow us to determine the probability of returning the most recent value written to the database, they do not describe what happens when the most recent value is not returned. Here, we determine the probability of returning a value within a bounded number of versions ((K, p)-regular semantics). In the following formulation, we consider traditional, non-expanding write quorums (no anti-entropy).

Similar to the previous example, given independent, identically distributed (IID) choice of read and write quorums, returning one of the last k written versions is equivalent to intersecting one of k independent write quorums. Given the probability of a single quorum non-intersection p, the probability of non-intersection with one of the last k independent quorums is p^k. Thus, the probability of non-intersection with uniform random choice in our example quorum system is Equation 1 exponentiated by k:

$$p = \left(\frac{\binom{N-W}{R}}{\binom{N}{R}} \right)^k \quad (2)$$

When $N = 3$, $R = W = 1$, this means that the probability of returning a version within 2 versions is $0.\overline{5}$; within 3 versions, $0.\overline{703}$; 5 versions, >0.868; and 10 versions, >0.98. When $N = 3$, $R = 1$, $W = 2$ (or, equivalently, $R = 2$, $W = 1$), these probabilities increase: $k = 1 \rightarrow 0.\overline{6}$, $k = 2 \rightarrow 0.\overline{8}$, and $k = 5 \rightarrow >0.995$.

This closed-form solution holds for quorums that do not change size over time. For expanding partial quorum systems, this solution is a lower bound on p for (K, p)-regular semantics. In the full version of this paper, we consider more advanced formulations of this analysis, including an analysis of monotonic reads consistency, quorum load, and closed-form solutions for mixed time and versions.[5, 7]

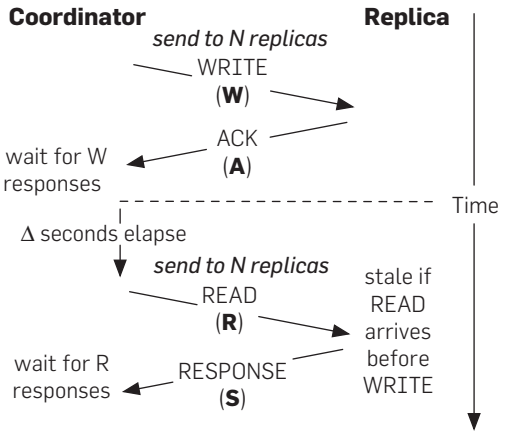
4.2. Consistency in dynamo

We have a simple, closed-form model for (K, p)-regular semantics, but time-based ((Δ, p)-regular) semantics are dependent on the quorum replication algorithm, workload, and any anti-entropy processes employed by a given system. In this section, we develop techniques for analyzing PBS (Δ, p)-regular semantics in the context of Dynamo-style data stores.

Dynamo-style quorum systems are inconsistent as a result of read and write message reordering, in turn a product of message delays. In this work, we take a white box approach to inconsistency and directly examine the protocols behind consistency. Accordingly, we develop a model of message latency in Dynamo operation that captures the effects of message delays for write requests (`W`), write acknowledgments (`A`), read requests (`R`), and read responses (`S`), and which, for convenience, we call *WARS*. In Figure 2, we illustrate *WARS* using a space–time diagram for messages between a coordinator and a single replica for a write followed by a read Δ seconds after the write completes. Accordingly, Δ here corresponds to the Δ in PBS (Δ, p)-regular semantics. In brief, reads are stale when all of the first R responses to the read request arrived at their replicas before the last (completed) write request arrived.

For a write, the coordinator sends N messages, one to each replica. The message from the coordinator to replicas containing the write is delayed by a value drawn from distribution `W`. The coordinator waits for W responses from the replicas before it can consider the write completed. Each response acknowledging the write is delayed by a value drawn from the distribution `A`.

Figure 2. The *WARS* model describes staleness in Dynamo by modeling message latencies between a coordinator and replicas for a write operation followed by a read operation *t* seconds later. In an *N* replica system, the depicted messages are exchanged with *N* replicas.



For a read, the coordinator (possibly different than the write's coordinator, and possibly representing a different client than the client that issued the write) sends N messages, one to each replica. The message from coordinator to replica containing the read request is delayed by a value drawn from distribution R. The coordinator waits for R responses from the replicas before returning the most recent value it receives. The read response from each replica is delayed by a value drawn from the distribution S.

The read coordinator will return stale data if the first R responses received reached their replicas before the replicas received the latest version (delayed by W). When $R + W > N$, this is impossible. However, under partial quorums, the frequency of this occurrence depends on the latency distributions. If we denote the write completion time (when the coordinator has received W acknowledgments) as w_t, a single replica's response is stale if $r' + w_t + \Delta < w'$ for r' drawn from R and w' drawn from W. Writes have time to propagate to additional replicas both while the coordinator waits for all required acknowledgments (A) and as replicas wait for read requests (R). Read responses are further delayed in transit (S) back to the read coordinator, inducing further possibility of reordering. Qualitatively, long-tailed write distributions (W) and relatively faster reads (R, S) increase the chance of staleness due to reordering.

WARS considers the effect of message sending, delays, and reception, but this represents a difficult analytical formulation with several non-independent order statistics. As we discuss in Section 5.1, we instead explore *WARS* using Monte Carlo methods, which are straightforward to understand and implement. We have found that the *WARS* distributions are easy to parameterize given traces of real-world system behavior (Section 5.3).

5. PBS IN ACTION

Given our PBS models for Dynamo-style stores, we now apply them to real-world systems. As discussed in Section 4.2, PBS (Δ, p)-regular behavior in a given system depends on the propagation of reads and writes across replicas. We introduced the *WARS* model as a means of reasoning about inconsistency in Dynamo-style quorum systems, but quantitative metrics such as staleness observed in practice depend on each of *WARS*'s latency distributions. In this section, we perform an analysis of Dynamo-style (Δ, p)-regular semantics for both synthetic and real-world distributions to better understand how frequently "eventually consistent" means "consistent" and, more importantly, why Dynamo-style stores are indeed frequently consistent.

PBS (K, p)-regular analysis for partial quorums is easily captured in closed form (Section 4.1). It does not depend on write latency or any environmental variables. Indeed, in practice, without expanding quorums or anti-entropy, we observe that our derived equations hold true experimentally.

In contrast, (Δ, p)-regular semantics depends on anti-entropy, which is more complicated. In this section, we focus on deriving experimental expectations for PBS (Δ, p)-regular semantics. We first validate our Monte Carlo analysis based on the *WARS* model and using message-level traces gathered from Cassandra clusters in Berkeley. We next explore synthetic latency distributions and, for the remainder of our analysis, explore distributions from two Internet companies: LinkedIn and Yammer.

5.1. Monte Carlo simulation

We implemented *WARS* analysis in a Monte Carlo-based simulation. Calculating (Δ, p)-regular semantics for a given value of Δ is straightforward (see pseudocode in Bailis et al.[7]). To simulate the message delays between coordinator and each of the N replicas, denote the ith sample drawn from distribution D as D[i] and draw N samples from W, A, R, and S. Compute the time that the write request completes (w_t, or the time that the coordinator gets its Wth acknowledgment; the Wth smallest value of $\{$W[i] + A[i], $i \in [0, N)\}$). Next, determine whether any of the first R replicas contained an up-to-date response: check whether any the first R samples of R, ordered by R[i] + S[i] obey w_t + R[i] + $\Delta \leq$ W[i]. Repeating this process multiple times provides an approximation of the behavior specified by the trace. Extending this formulation to analyze (K, Δ, p)-regular semantics given a distribution of write arrival times will require accounting for multiple writes across time. As described in extended versions of this paper,[5, 7] we validated this analysis on traces that we collected from a real-world Cassandra cluster. We observed an average RMSE of 0.28% for (Δ, p)-regular semantics prediction and an average N-RMSE of 0.48% for latency predictions.

5.2. Write latency distribution effects

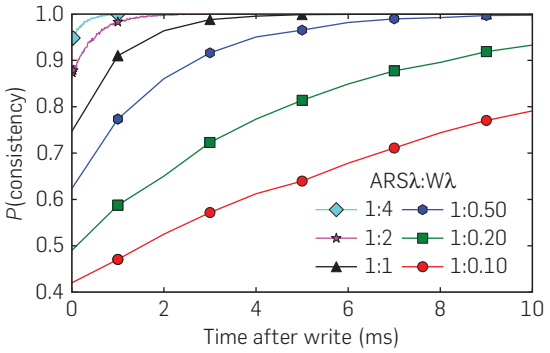
The *WARS* model of Dynamo-style systems dictates that high variance in latency increases staleness. Before studying real-world workloads (Section 5.3), we quantified this behavior in isolation via synthetic distributions: we swept a range of exponentially distributed write distributions (changing parameter λ, which dictates the mean and tail of the distribution) while fixing A = R = S.

Our results, shown in Figure 3, demonstrate this relationship. When the variance and mean of W are 0.0625 ms

and 0.25 ms ($\lambda = 4$, one-fourth the mean of A = R = S = 1 ms), we observe a 94% chance of consistency immediately after the write and 99.9% chance after 1 ms. However, when the variance and mean of W are 100 ms and 10 ms ($\lambda = 0.1$, ten times the mean of A = R = S = 1 ms), we observe a 41% chance of consistency immediately after write and a 99.9% chance of consistency only after 65 ms. As the variance and mean increase, so does the probability of inconsistency. Under distributions with fixed means and variable variances (uniform, normal), we observe that the mean of W is less important than its variance if W is strictly greater than A = R = S.

Decreasing the mean and variance of W improves the probability of consistent reads. This means that, as we will see, techniques that lower write latency variance result in more consistent reads. Instead of increasing read and write quorum sizes, operators could chose to lower (relative) W latencies through hardware configuration or by delaying reads, although this latter option is detrimental to performance for read-dominated workloads and may introduce undesirable queuing effects. Nonetheless, this PBS analysis illustrates the fact that stale reads can be avoided in a variety of ways beyond simple adjustment of quorum sizes.

Figure 3. (Δ, *p*)-regular semantics with exponential latency distributions for W and A = R = S. Mean latency is $1/\lambda$. $N = 3$, $R = W = 1$.



5.3. Production latency distributions

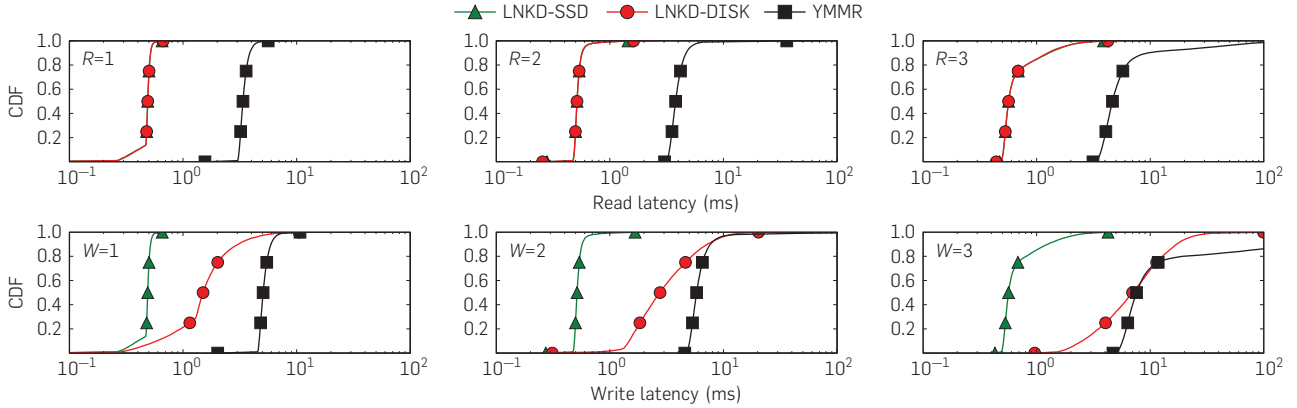
To study real-world behavior, we obtained production latency statistics from two Internet-scale companies. While message-level *WARS* timing traces would deliver more accurate predictions, we opted for a pragmatic compromise: as described in extended versions of this work, we fit *WARS* distributions to each of the provided statistics (Figure 4).[5,7]

LinkedIn[d] is an online professional social network with over 225 million members as of July 2013. To provide highly available, low latency data storage, engineers at LinkedIn built Voldemort. Alex Feinberg, a lead engineer on Voldemort, graciously provided us with latency distributions for a single server under peak traffic for a user-facing service at LinkedIn, representing 60% read and 40% read-modify-write traffic.[5,7] Feinberg reports that, using spinning disks, Voldemort is "largely IO bound and latency is largely determined by the kind of disks we're using, [the] data to memory ratio and request distribution." With solid-state drives (SSDs), Voldemort is often "CPU and/or network bound" and "maximum latency is generally determined by [garbage collection] activity (rare, but happens occasionally) and is within hundreds of milliseconds." We denote the LinkedIn spinning disk distribution as LNKD-DISK and SSD trace as LNKD-SSD.

Yammer[e] provides private social networking to over 200,000 companies as of July 2013 and uses Basho's Riak for some client data. Coda Hale, an infrastructure architect, provided performance statistic for their production Riak deployment.[5,7] Hale mentioned that "reads and writes have radically different expected latencies, especially for Riak." Riak delays writes "until the fsync returns, so while reads are often <1 ms, writes rarely are." Also, although we do not model this explicitly, Hale also noted that the size of values is important, claiming "a big performance improvement by adding LZF compression to values." We denote the Yammer latency distribution as YMMR.

[d] http://www.linkedin.com/
[e] http://www.yammer.com/

Figure 4. Read and write operation latency for production fits for $N = 3$ and varying R and W. For reads, LNKD-SSD is equivalent to LNKD-DISK. Depicted points mark significant percentiles for comparison. Higher values of R and W result in increased latency.



5.4. Staleness in production

The production latency distributions confirm that staleness is frequently limited in eventually consistent stores. We measured the (Δ, p)-regular semantics for each distribution (Figure 5). `LNKD-SSD` and `LNKD-DISK` demonstrate the importance of write latency in practice. Immediately after write completion, `LNKD-SSD` had a 97.4% probability of consistent reads, reaching over a 99.999% probability of consistent reads after 5 ms. `LNKD-SSD`'s reads briefly raced with its writes immediately after write completion. However, within a few milliseconds after the write, the chance of a read arriving before the last write was nearly eliminated. The distribution's read and write operation latencies were small (median 0.489 ms), and writes completed quickly across all replicas due to the distribution's short tail (99.9th percentile 0.657 ms). In contrast, under `LNKD-DISK`, writes take much longer (median 1.50 ms) and have a longer tail (99.9th percentile 10.47 ms). `LNKD-DISK`'s (Δ, p)-regular semantics reflects this difference: immediately after write completion, `LNKD-DISK` had only a 43.9% probability of consistent reads and, 10 ms later, only a 92.5% probability. This suggests that SSDs may greatly improve consistency due to reduced write variance. Similarly, one should expect that consistency would be improved by using explicit memory management rather than unscheduled garbage collection.

We experienced similar behavior with the other distributions. Immediately after write completion ($\Delta = 0$), `YMMR` had a $p = 89.3\%$ probability of consistency. However, the `YMMR` distribution only reached a $p = 99.9\%$ probability of consistency at $\Delta = 1364$ ms due to high variance and long-tail behavior in its write distribution. This hints that, given multiple replicas for a data item, the durability benefits of synchronously flushing writes to disk may have adverse effects on consistency. An alternative approach that could improve consistency and avoid this high variance would rely on multi-replica (in-memory or buffer cache) replication for durability and only flush writes asynchronously.

5.5. Quorum sizing

In addition to N = 3—the most common quorum size we encountered in practice—we consider how varying the number of replicas (N) affects (Δ, p)-regular semantics while maintaining R = W = 1. The results, depicted in Figure 6, show that the probability of consistency immediately after write completion decreases as N increases. With two replicas, LNKD-DISK has a 57.5% probability of consistent reads immediately after write completion but only a 21.1% probability with 10 replicas. However, at high probabilities (p), the window of inconsistency (Δ) for increased replica sizes is close. For LNKD-DISK, Δ at p = 99.9% ranges from 45.3 ms for 2 replicas to 53.7 ms for 10 replicas.

Figure 5. (Δ, p)-regular semantics for production operation latencies.

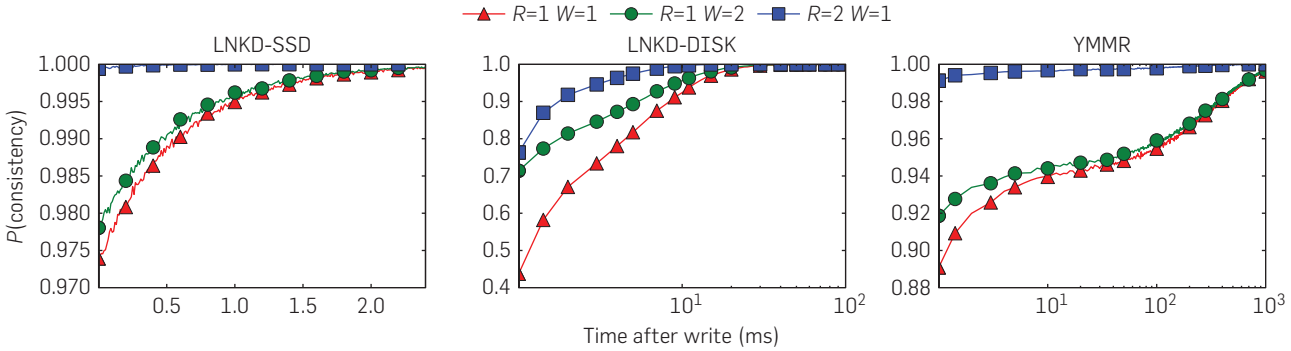
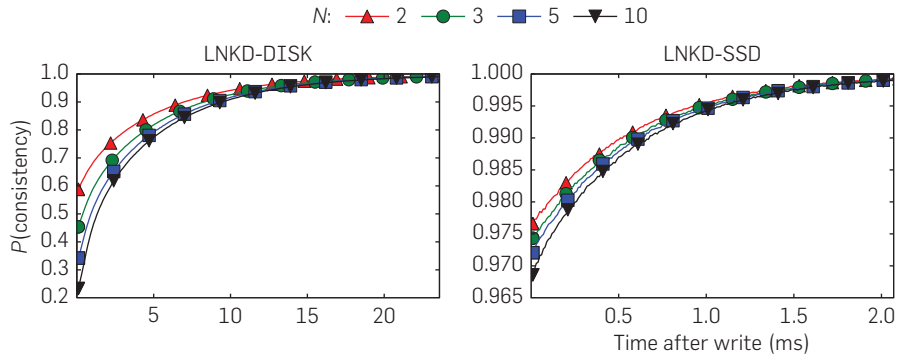


Figure 6. (Δ, p)-regular semantics for production operating latencies varying the number of replicas N when $R = W = 1$. Inconsistencies p are likely to resolve quickly (low Δ) even for large replication factors.



These results imply that maintaining a large number of replicas for availability or better performance results in a potentially large impact on consistency immediately after writing. However, the (Δ, p)-regular semantics probability (p) will still rise quickly (small Δ).

5.6. Latency vs. staleness

Choosing values for R and W is a trade-off between operation latency and consistency. To measure this trade-off, we compared 99.9th percentile operation latencies with the corresponding Δ at p = 99.9% for quorum configurations where N = 3, typical of deployments in the field.

Partial quorums often exhibit favorable latency-consistency trade-offs (Table 1). For `YMMR`, $R = W = 1$ results in low latency reads and writes (16.4 ms) but high Δ (1364 ms). However, setting $R = 2$ and $W = 1$ reduces Δ to 202 ms and the combined read and write latencies are 81.1% (186.7 ms) lower than the fastest strict quorum ($W = 1$, $R = 3$). Allowing p = 99.9%, Δ = 13.6 ms reduces `LNKD-DISK` read and write latencies by 16.5% (2.48 ms). For `LNKD-SSD`, across 10 M writes ("seven nines"), we did not observe staleness with $R = 2$, $W = 1$. $R = W = 1$ reduced latency by 59.5% (1.94 ms) with a corresponding Δ = 1.85 ms. In summary, lowering values of R and W can greatly improve operation latency but, even in the tail, the duration of inconsistency (Δ) is relatively small.

We omit full results here, but we have also experimented with heterogeneous replica behavior and with multi-item guarantees such as causal consistency and transactional atomicity.[7]

6. DISCUSSION AND FUTURE WORK

In this section, we discuss PBS design decisions, describe our experiences integrating PBS with real-world stores and end-user applications, and suggest areas for future work.

6.1. Prediction and verification

In this work, we have developed techniques for consistency prediction, which provide an expectation of system behavior given a set of input data about the system and the current operating environment. Given a trace, one can predict staleness after an arbitrary amount of time or number of versions without having to actually run any additional queries. Additionally, predictions allow users to easily perform "what-if" analysis across arbitrary replication configurations, request distributions (Δ and K), and hardware configurations (e.g., switching from SSDs to disks). We have found that prediction is computationally inexpensive and can be performed on a decentralized, per-replica basis. While prediction is flexible, it is only as good as the input traces. With representative input data, predictions will be accurate. However, with unrepresentative data (or bad models), prediction accuracy will suffer.

In contrast, verification[21] informs users how their data stores are performing with guaranteed certainty. If a user makes a change to their replication settings using a predictor, she may want to ensure that the change behaves as expected. While this verification is not well-suited to "what-if" analysis, it is an important complement to prediction. Verification effectively provides a metric that results from integrating the (K, Δ, p)-regular semantics density function weighted by the given read request rate (measured with respect to time since the last write). Additionally, verification is algorithmically complex,[11] but in our experience is not terribly difficult to implement.

We believe that both techniques will be increasingly useful as systems begin to treat consistency as a continuous, quantitative metric. Taken together, consistency prediction and verification techniques form a powerful toolkit.

6.2. White and black box approaches

In this work, we use a white box approach to consistency and exploit expert knowledge of replication protocols to provide quantitative insight. This requires translating from user-centric, declarative specifications of consistency anomalies into back-end protocol events (e.g., in the *WARS* model, the reordering between read and write responses). We could have alternatively attempted to reverse engineer system internals or provide an implementation-independent predictor, but this black box analysis would be substantially more complex. We believe that verification techniques are more amenable to black box techniques and that the portability benefits of black box techniques must be weighed against their potential inaccuracy. Given our experiences integrating prediction into existing data

Table 1. Δ for (Δ, p = 99.9%)-regular semantics and 99.9th percentile read (L_r) and write latencies (L_w), varying R and W with N = 3.

	LNKD-SSD			LNKD-DISK			YMMR		
	L_r	L_w	t	L_r	L_w	t	L_r	L_w	t
$R = 1, W = 1$	**0.66**	**0.66**	**1.85**	0.66	10.99	45.5	5.58	10.83	1364.0
$R = 1, W = 2$	0.66	1.63	1.79	0.65	20.97	43.3	5.61	427.12	1352.0
$R = 2, W = 1$	**1.63**	**0.65**	**0**	**1.63**	**10.9**	**13.6**	**32.6**	**10.73**	**202.0**
$R = 2, W = 2$	**1.62**	**1.64**	**0**	1.64	20.96	0	33.18	428.11	0
$R = 3, W = 1$	4.14	0.65	0	**4.12**	**10.89**	**0**	**219.27**	**10.79**	**0**
$R = 1, W = 3$	0.65	4.09	0	0.65	112.65	0	5.63	1870.86	0

Significant latency-staleness trade-offs are in bold.

stores and the large-scale adoption of open-source data stores, we believe that white box techniques are feasible, even if they require modifications to existing stores.

6.3. Real-world store integration

With the help of several open-source developers, we have developed patches for PBS functionality within two NoSQL stores: Cassandra and Voldemort. For Cassandra, we have taken two approaches: an invasive but more accurate implementation and an external but less accurate prediction module. For the former approach, we modified the Cassandra messaging layer to add a message creation timestamp in order to measure each of the `W`, `A`, `R`, `S` distributions. When tracing is enabled on a given server, the messaging layer logs per-operation timestamps in a separate PBS prediction module. The timestamps are stored in an in-memory circular buffer for each of the required message latencies. Subsequently, users can call the PBS predictor module via an externally accessible interface, which they can use to provide advanced functionality like dynamic replication configuration and monitoring (see below). This provides relatively accurate predictions at the expense of having to instrument the messaging layer. However, as data stores like Cassandra have expanded their user-accessible monitoring data (e.g., per-query latency tracing), we have more recently been exploring predictions outside of the database—the latter approach—which we have implemented for Voldemort. We have open-sourced implementations of both approaches.

6.4. PBS applications

PBS enables functionality not possible without quantitative consistency metrics. With PBS, we can automatically configure replication parameters by optimizing operation latency given constraints on staleness and minimum durability. Data store operators can subsequently provide service level agreements to applications and quantitatively describe latency-staleness trade-offs to users. Operators can dynamically configure replication using online latency measurements. This optimization also allows disentanglement of replication for durability from replication for reasons of low latency and higher capacity. For example, operators can specify a minimum replication factor for durability and availability but can also automatically increase N, decreasing tail latency for fixed R and W. We expanded upon these possibilities in a *SIGMOD 2013* demo that featured real-time predictions for a live Cassandra cluster and mock web service.[6]

6.5. *WARS* limitations and extensions

There are several limitations and potential extensions of the *WARS* model, which we sketch here and discuss in greater detail in extended versions of this work.[5, 7] *WARS* only models a single write and read and is therefore a conservative estimate for multi-write scenarios. Moreover, in our current treatment, *WARS* treats each distribution as IID. This is not fundamental to the model but is a limitation of our latency traces from industry. *WARS* also does not capture effects of additional, commonly employed anti-entropy processes (e.g., read-repair, Merkle-tree exchange) and may be a conservative estimate of staleness. It does not address system behavior under failures, which varies from store to store, and it assumes that clients contact a coordinator server instead of issuing requests themselves (e.g., Voldemort). We believe that these limitations are not fundamental and can be accounted for in a white box model but nonetheless remain future work. Finally, clients requiring staleness detection may do so asynchronously, enabling speculative reads and compensatory actions.[5, 7]

7. RELATED WORK

This research builds upon several related areas: quorum replication, consistency models, and approaches to quantifying consistency.

Managing replicated data is a long-studied problem in distributed systems and concurrent programming. There is a plethora of consistency models offering different trade-offs between semantics, performance, and availability. Traditional models such as serializability and linearizability as well as more recently proposed models such as timeline consistency[8] and parallel snapshot isolation[23] all provide "strong" semantics at the cost of high availability, or the ability to provide "always-on" response behavior at all replicas. In contrast, faced with a requirement for high availability and low latency, many production data stores have turned to weaker semantics to provide availability in the face of network partitions.[9, 26]

Our focus in this paper is on the semantics provided by existing, widely deployed systems. Due to the prevalence of "strong" consistency and eventual consistency models in practice (and the explicit choice between these two models in Dynamo-style systems), we largely focus on this dichotomy. However, there are a range of alternative but still "weak" models. As an example, the Bayou system provided a range of "session guarantees," including read-your-writes and monotonic reads consistency.[25] Similarly, a recent Technical Report from UT Austin claims that a variant of causal consistency is the strongest consistency model achievable in an available, one-way convergent (eventually consistent) system,[18] a model that has recently attracted systems implementations.[17] As we have hinted, probabilistic approaches are applicable to the consistency models beyond those we have considered here. Specific to staleness, prior research such as TACT[27] has examined how to provide deterministic staleness bounds. These deterministically bounded staleness systems represent the deterministic dual of PBS.

Our techniques for analyzing modern partial quorum systems draw on existing, largely theoretical literature. We briefly surveyed quorum replication[20] in Section 3. In this work, we specifically draw inspiration from probabilistic quorums[19] in analyzing expanding quorum systems and their consistency. We believe that revisiting probabilistic quorum systems—including non-majority quorum systems such as tree quorums—in the context of write propagation, anti-entropy, and Dynamo is a promising area for

theoretical work. While we study probabilistic guarantees for staleness, prior work on *k*-quorums[2, 3] have looked at *deterministic* guarantees that a partial quorum system will return values that are within *k* versions of the most recent write.[2]

Finally, recent research has focused on measuring and verifying the consistency of eventually consistent systems both theoretically and experimentally (Rahman et al. provide a brief survey[21]). This is useful for validating consistency predictions and understanding staleness violations.

8. CONCLUSION

In this paper, we introduced PBS, which models the expected staleness of data returned by eventually consistent data stores. PBS offers an alternative to the all-or-nothing consistency guarantees of many of today's systems by offering SLA-style consistency predictions. By extending prior theory on probabilistic quorum systems, we derived an analytical solution for the (K, p)-regular semantics of a partial quorum system, representing the expected staleness of a read operation in terms of versions. We also analyzed (Δ, p)-regular semantics, or expected staleness of a read in terms of real time, under Dynamo-style quorum replication. To do so, we developed the *WARS* latency model to explain how message reordering leads to staleness under Dynamo. To examine the effect of latency on (Δ, p)-regular semantics in practice, we used real-world traces from Internet companies to drive a Monte Carlo analysis. We find that eventually consistent Dynamo-style quorum configurations are often consistent after tens of milliseconds due in large part to their resilience to per-server latency variance. We conclude that eventually consistent partial quorum replication schemes frequently deliver consistent data during failure-free operation while offering significant latency benefits. We believe that continued study and deployment of quantitative consistency metrics will both enable useful end-user functionality and shed light on previously opaque and frequently controversial replication configurations.

Interactive demonstration

An interactive demonstration of Dynamo-style PBS is available at http://pbs.cs.berkeley.edu/#demo.

Acknowledgments

This work was greatly improved by feedback from many previously noted individuals.[5, 7] It was supported by gifts from Google, SAP, Amazon Web Services, Blue Goji, Cloudera, Ericsson, General Electric, Hewlett Packard, Huawei, IBM, Intel, MarkLogic, Microsoft, NEC Labs, NetApp, NTT Multimedia Communications Laboratories, Oracle, Quanta, Splunk, and VMware. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant DGE 1106400, National Science Foundation Grants IIS-0713661, CNS-0722077, and IIS-0803690, NSF CISE Expeditions award CCF-1139158, the Air Force Office of Scientific Research Grant FA95500810352, and by DARPA contract FA865011C7136.

References

1. Abadi, D.J. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Comput. 45*, 2 (2012), 37–42.
2. Aiyer, A., Alvisi, L., Bazzi, R.A. On the availability of non-strict quorum systems. In *DISC 2005*.
3. Aiyer, A.S., Alvisi, L., Bazzi, R.A. Byzantine and multi-writer k-quorums. In *DISC* (2006), 443–458.
4. Bailis, P., Ghodsi, A. Eventual consistency today: Limitations, extensions, and beyond. *ACM Queue 11*, 3 (Mar. 2013), 20:20–20:32.
5. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Probabilistically bounded staleness for practical partial quorums. *PVLDB 5*, 8 (2012), 776–787.
6. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. PBS at work: Advancing data management with consistency metrics. In *SIGMOD 2013 Demo*.
7. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Quantifying eventual consistency with PBS. *VLDB J.* (2014). (see http://link.springer.com/article/10.1007/s00778-013-0330-1).
8. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R. Pnuts: Yahoo!'s hosted data serving platform. In *VLDB 2008*.
9. Davidson, S., Garcia-Moina, H., Skeen, D. Consistency in partitioned networks. *ACM Comput. Surv. 17*, 3 (1985), 314–370.
10. 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 *SOSP 2007*, 205–220.
11. Golab, W., Li, X., Shah, M.A. Analyzing consistency properties for fun and profit. In *PODC* (2011), 197–206.
12. Hamilton, J. Perspectives: I love eventual consistency but... http://perspectives.mvdirona.com/2010/02/24/ILoveEventualConsistencyBut.aspx (24 Feb. 2010).
13. Herlihy, M., Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst. 12*, 3 (1990), 463–492.
14. Kirkell, J. Consistency or bust: Breaking a Riak cluster. http://www.oscon.com/oscon2011/public/schedule/detail/19762. Talk at O'Reilly OSCON 2011 (27 Jul. 2011).
15. Linden, G. Make data useful. https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt (29 Nov. 2006).
16. Linden, G. Marissa Mayer at Web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html (9 Nov. 2006).
17. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G. Stronger semantics for low-latency geo-replicated storage. In *NSDI 2013*.
18. Mahajan, P., Alvisi, L., Dahlin, M. *Consistency, Availability, Convergence*. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, 2011.
19. Malkhi, D., Reiter, M., Wool, A., Wright, R. Probabilistic quorum systems. *Inform. Commun. 170* (2001), 184–206.
20. Merideth, M., Reiter, M. Selected results from the latest decade of quorum systems research. In *Replication*, B. Charron-Bost, F. Pedone, and A. Schiper, eds. Volume 5959 of *LNCS* (2010). Springer, 185–206.
21. Rahman, M., Golab, W., AuYoung, A., Keeton, K., Wylie, J. Toward a principled framework for benchmarking consistency. In *Proceedings of the 8th Workshop on Hot Topics in System Dependability* (Hollywood, CA, 2012), USENIX.
22. Schurman, E., Brutlag, J. Performance related changes and their user impact. Presented at *Velocity Web Performance and Operations Conference* (San Jose, CA, Jun. 2009).
23. Sovran, Y., Power, R., Aguilera, M.K., Li, J. Transactional storage for geo-replicated systems. In *SOSP 2011*.
24. Stonebraker, M. Urban myths about SQL. http://voltdb.com/_pdf/VoltDB-MikeStonebraker-SQLMythsWebinar-060310.pdf. VoltDB Webinar (Jun. 2010).
25. Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B. Session guarantees for weakly consistent replicated data. In *PDIS 1994*.
26. Vogels, W. Eventually consistent. *CACM 52* (2009), 40–44.
27. Yu, H., Vahdat, A. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst. 20*, 3 (2002), 239–282.

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica ({pbailis, shivaram, franklin, hellerstein, istoica}@cs.berkeley.edu), University of California, Berkeley.

© 2014 ACM 0001-0782/14/08 $15.00

Technical University of Denmark

DTU

PROFESSOR

Software Engineering

DTU Compute invites applications for a professorship in Software Engineering. The professor will be affiliated with the Software Engineering Section, and will be in charge of strengthening and further developing the research of this section. The professor is also expected to actively take part in the definition, development and promotion of the future IT profile of DTU Compute, in particular by working together with other sections at DTU Compute.

Application deadline: **1 October 2014**

DTU is a technical university providing internationally leading research, education, innovation and public service. Our staff of 5.700 advance science and technology to create innovative solutions that meet the demands of society; and our 10.000 students are educated to address the technological challenges of the future. DTU is an independent academic university collaborating globally with business, industry, government, and public agencies.

Further details: career.dtu.de



The **Hasso Plattner Institute** for Software Systems Engineering (HPI) in Potsdam is Germany's university excellence center in Computer Science. Annually, the Institute's Research School seeks talented junior researchers and accordingly offers

10 Ph.D. Scholarships and Postdoc Scholarships

The **HPI Research School** focuses on the foundation and application of large-scale, highly complex and interconnected IT systems. With its interdisciplinary and international structure, the Research School interconnects the HPI research groups as well as its international branches at Cape Town University, Technion - Israel Institute of Technology and Nanjing University. The **HPI Future SOC Lab**, a state-of-the-art computer center, enriches the academic work at the HPI Research School.

The HPI professors and their research groups ensure high quality research and will supervise Ph.D. students in the following topic areas: Human Computer Interaction, Prof. Dr. Patrick Baudisch; Computer Graphics Systems, Prof. Dr. Jürgen Döllner; System Engineering and Modeling, Prof. Dr. Holger Giese; Software Architecture, Prof. Dr. Robert Hirschfeld; Internet Technologies and Systems, Prof. Dr. Christoph Meinel; Information Systems, Prof. Dr. Felix Naumann; Enterprise Platform and Integration Concepts, Prof. Dr. h.c. Hasso Plattner; Operating Systems and Middleware, Prof. Dr. Andreas Polze; Business Process Technology, Prof. Dr. Mathias Weske

If you have prior experience in any of these areas, you are invited to submit a full application with the following documents: curriculum vitae and copies of certificates/transcripts, brief research proposal, work samples/copies of relevant scientific work (e.g. master's thesis), and a letter of recommendation.

Applications must be submitted by August 15th of the respective year.
Positions are usually available at the beginning of October. Please send your applications to: **research-school-application@hpi.de**

For more information on HPI, the HPI Research School and openHPI see:
www.hpi.de/research-school
www.openHPI.de

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Ohio University
School of EECS
Assistant Professor

The Russ College of Engineering and Technology at Ohio University invites applications for a full-time, benefits-eligible, tenure track assistant professor position in computer science. The selected applicant will be expected to perform excellent research, teaching, and service in computer science. Candidates should have an earned doctorate in computer science or a related discipline. Candidates are expected to have strong research potential as well as an interest in teaching at both the undergraduate and graduate levels. Departmental support will include initial reduced teaching loads, competitive salary and generous start-up funds. Candidates from all research areas are welcomed, but preference will be given to candidates with expertise in secure and dependable software systems (i.e., software certification, verification, or validation; formal methods in software engineering; or cybersecurity).

Position will remain open until filled; for full consideration please apply by September 2, 2014. For details and to apply, go to http://www.ohiouniversityjobs.com/postings/9544

Stanford University
Graduate School of Business
Faculty Positions in Operations, Information and Technology

The Operations, Information and Technology (OIT) area at the Graduate School of Business, Stanford University, is seeking qualified applicants for full-time, tenure-track positions, starting September 1, 2015. All ranks and relevant disciplines will be considered. Applicants are considered in all areas of Operations, Information and Technology (OIT) that are broadly defined to include the analytical and empirical study of technological systems, in which technology, people, and markets interact. It thus includes operations, information systems/technology, and management of technology. Applicants are expected to have rigorous training in management science, engineering, computer science, economics, and/or statistical modeling methodologies. The appointed will be expected to do innovative research in the OIT field, to participate in the school's PhD program, and to teach both required and elective courses in the MBA program. Junior applicants should have or expect to complete a PhD by September 1, 2015.

Applicants are strongly encouraged to submit their applications electronically by visiting the web site http://www.gsb.stanford.edu/recruiting and uploading their curriculum vitae, research papers and publications, and teaching evaluations, if applicable, on that site. Alternatively, all materials may be sent by e-mail to faculty_recruiter@gsb.stanford.edu, or by postal mail (non-returnable) to Office of Faculty Recruiting, Graduate School of Business, Stanford University, 655 Knight Way, Stanford, CA 94305-7278. However, submissions via e-mail and postal mail can take 4-6 weeks for processing. For an application to be considered complete, each applicant must have three letters of recommendation emailed to the preceding email address, or sent via postal mail. The application deadline is November 15, 2014.

Stanford University is an equal opportunity employer and is committed to increasing the diversity of its faculty. It welcomes nominations of and applications from women, members of minority groups, protected veterans and individuals with disabilities, as well as from others who would bring additional dimensions to the university's research, teaching and clinical missions.



DOI:10.1145/2631198

Peter Winkler

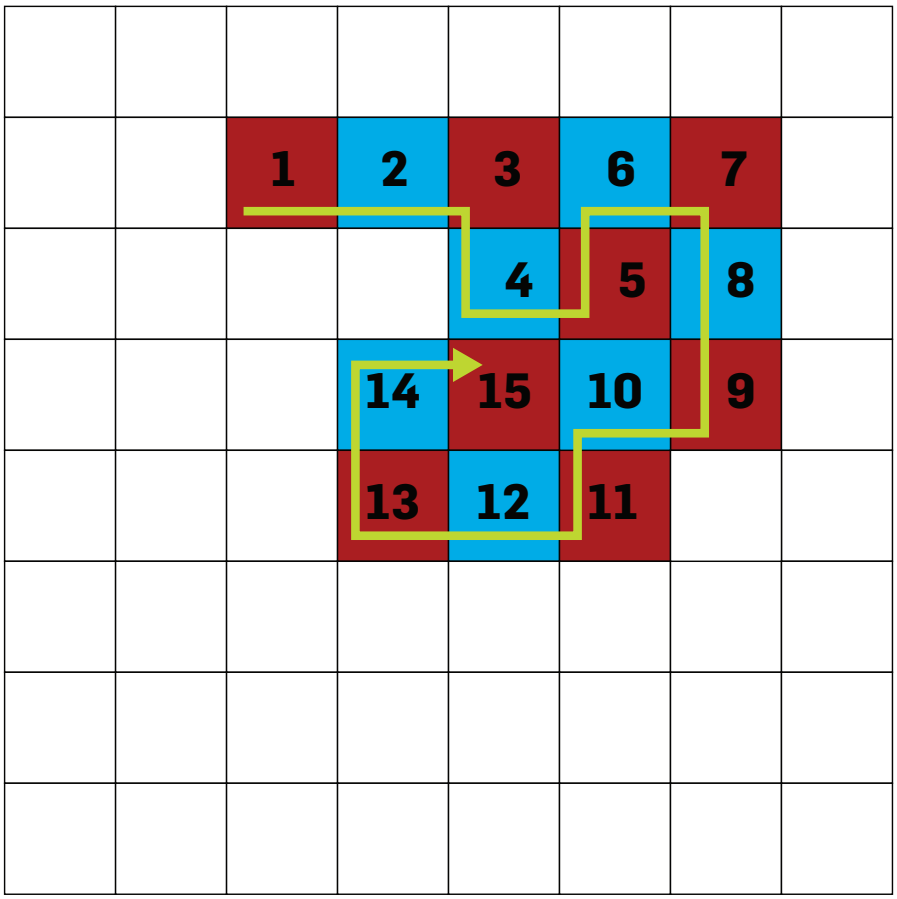
Puzzled
Paths and Matchings

Consider two simple games played by Alice and Bob on a checkerboard or, more generally, on a graph. The games look different, but, as we know, looks can be deceiving…

The Path Game. Alice and Bob alternately place a checker on an unoccupied square of an initially empty eight-by-eight checkerboard (see the figure here). The rule is that after Alice places her initial checker, every new checker must be (orthogonally) adjacent to the most recently placed checker. The players are in effect constructing a path of checkers. The last player to make a legal move wins the game. Your mission: find a winning strategy for Bob.

The Match Game. Suppose that Alice, when it is her turn, is no longer obliged to play next to Bob's last checker and may instead place her checker on any unoccupied square. Bob is still constrained to "match" Alice, that is, to play next to Alice's last move. As in the Path Game, the last player to make a legal move wins the game. So who wins now, with best play?

General graphs. Either the Path Game or the Match Game can be played on any finite graph. Alice begins by marking any vertex, and, subsequently, Bob and Alice alternate in marking vertices. In the Path Game, each marked vertex must be adjacent to the most recently marked vertex. In the Match Game, each vertex marked by Bob must be adjacent to the most recently marked vertex, but Alice can mark any unmarked vertex when it is her turn. As in the checkerboard versions, the last player to make a legal move wins the game. Now find a graph on which, with best play, Bob wins the Path Game and Alice wins the Match Game… if there is one.



Alice (red) beats Bob (blue) in a Path Game.

Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science at Dartmouth College, Hanover, NH.

Copyright held by author..

Computing Reviews

Connect with our Community of Reviewers

"Computing Reviews is worth reading because it makes me uncomfortable. It takes me away from my comfort zone into realities of which I was not aware."

- James O. Van Speybroeck

Association for Computing Machinery

ThinkLoud

www.computingreviews.com

Take a look at the **Internet's future.**

Find **the leaders** in data communication
and networking from industry and academia,
all in one place, once a year.

SIGCOMM
CHICAGO 2014



The **Annual Festival** of the **ACM** Special Interest Group on **Data Communication**

Chicago August 17 - 22, 2014

Association for Computing Machinery

acm sigcomm

conferences.sigcomm.org/sigcomm/2014