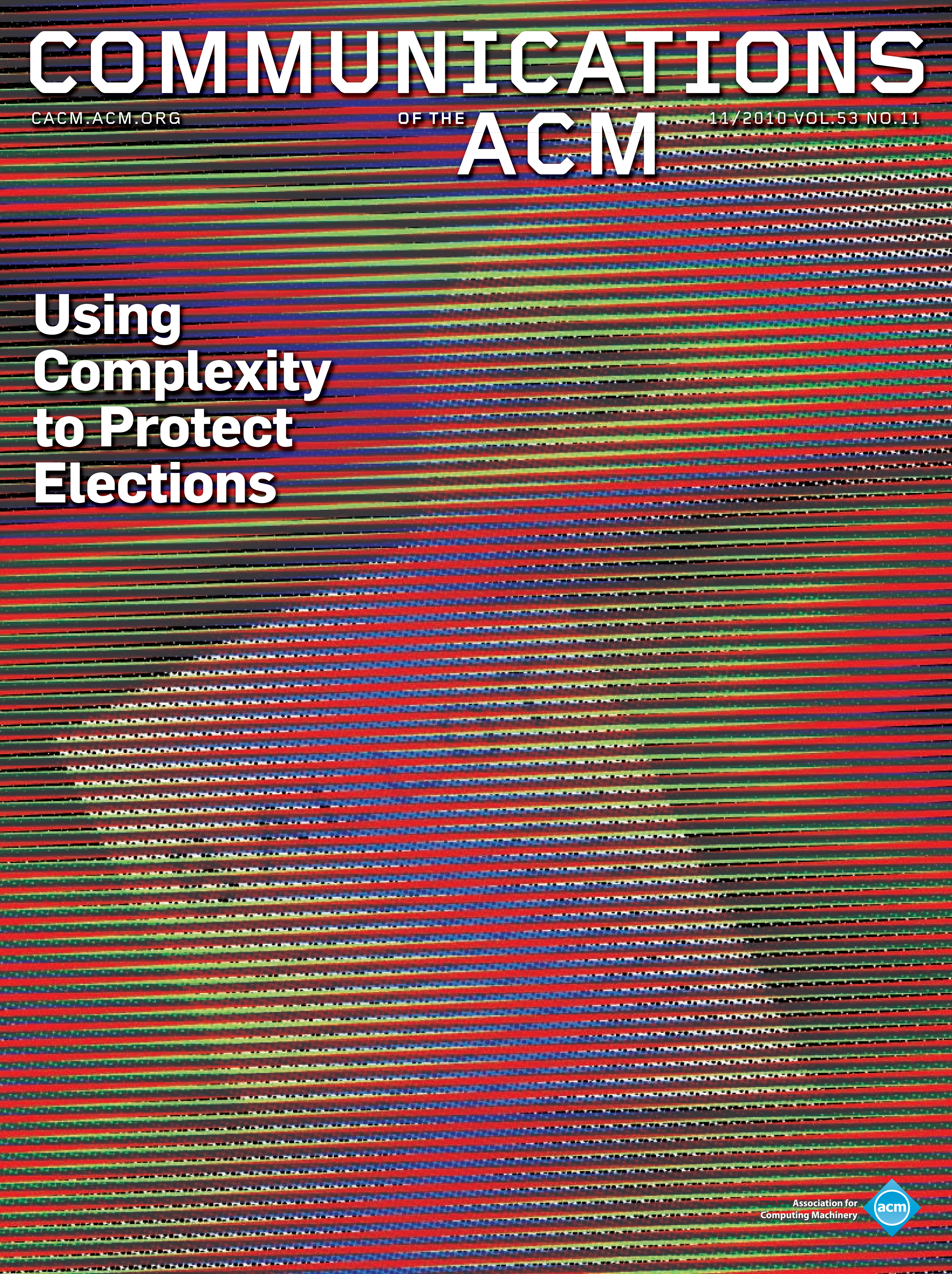
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

11/2010 VOL.53 NO.11



Using Complexity to Protect Elections

Association for Computing Machinery

CALL FOR PARTICIPATION

CTS 2011

Philadelphia, Pennsylvania, USA

The 2011 International Conference on Collaboration Technologies and Systems

May 23 – 27, 2011
The Sheraton University City Hotel
Philadelphia, Pennsylvania, USA

Important Dates:

Paper Submission Deadline	**December 15, 2010**
Workshop/Special Session Proposal Deadline	**November 15, 2010**
Tutorial/Demo/Panel Proposal Deadline	**January 8, 2011**
Notification of Acceptance	**February 1, 2011**
Final Papers Due	**March 1, 2011**

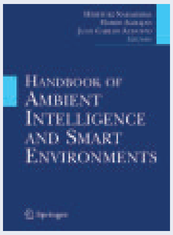
For more information, visit the CTS 2011 web site at:
http://cts2011.cisedu.info/

In cooperation with the ACM, IEEE, IFIP

Springer
the language of science

springer.com

Springer References & Key Library Titles



Handbook of Ambient Intelligence and Smart Environments

H. Nakashima, Future University, Hakodate, Hokkaido, Japan; **H. Aghajan**, Stanford University, Stanford, CA, USA; **J. C. Augusto**, University of Ulster at Jordanstown, Newtownabbey, UK (Eds.)

Provides readers with comprehensive, up-to-date coverage in this emerging field. Organizes all major concepts, theories, methodologies, trends and challenges into a coherent, unified repository. Covers a wide range of applications relevant to both ambient intelligence and smart environments. Examines case studies of recent major projects to present the reader with a global perspective of actual developments.

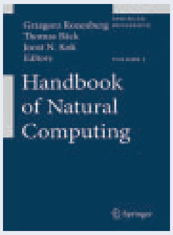
2010. XVIII, 1294 p. 100 illus. Hardcover
ISBN 978-0-387-93807-3 ► **$229.00**

Handbook of Multimedia for Digital Entertainment and Arts

B. Furht, Florida Atlantic University, Boca Raton, FL, USA (Ed.)

The first comprehensive handbook to cover recent research and technical trends in the field of digital entertainment and art. Includes an outline for future research directions within this explosive field. The main focus targets interactive and online games, edutainment, e-performance, personal broadcasting, innovative technologies for digital arts, digital visual and other advanced topics.

2009. XVI, 769 p. 300 illus., 150 in color. Hardcover
ISBN 978-0-387-89023-4 ► **$199.00**



Handbook of Natural Computing

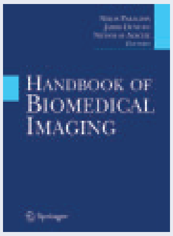
G. Rozenberg, **T. Bäck**, **J. N. Kok**, Leiden University, The Netherlands (Eds.)

We are now witnessing an exciting interaction between computer science and the natural sciences. Natural Computing is an important catalyst for this interaction, and this handbook is a major record of this important development.

2011. Approx. 1700 p. (In 3 volumes, not available seperately) Hardcover
ISBN 978-3-540-92909-3 ► **$749.00**

eReference
ISBN 978-3-540-92910-9 ► **$749.00**

Print + eReference
2011. Approx. 1700 p.
ISBN 978-3-540-92911-6 ► **$939.00**

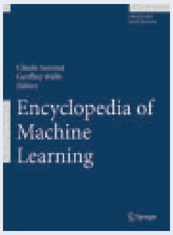


Handbook of Biomedical Imaging

N. Paragios, École Centrale de Paris, France; **J. Duncan**, Yale University, USA; **N. Ayache**, INRIA, France (Eds.)

This book offers a unique guide to the entire chain of biomedical imaging, explaining how image formation is done, and how the most appropriate algorithms are used to address demands and diagnoses. It is an exceptional tool for radiologists, research scientists, senior undergraduate and graduate students in health sciences and engineering, and university professors.

2010. Approx. 590 p. Hardcover
ISBN 978-0-387-09748-0 ► **approx. $179.00**



Encyclopedia of Machine Learning

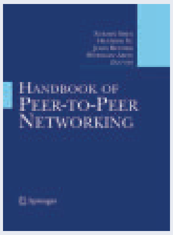
C. Sammut, **G. I. Webb** (Eds.)

The first reference work on Machine Learning Comprehensive A-Z coverage of this complex subject area makes this work easily accessible to professionals and researchers in all fields who are interested in a particular aspect of Machine LearningTargeted literature references provide additional value for researchers looking to study a topic in more detail.

2010. 800 p. Hardcover
ISBN 978-0-387-30768-8 ► **approx. $549.00**

eReference
2010. 800 p.
ISBN 978-0-387-30164-8 ► **approx. $549.00**

Print + eReference
2010. 800 p.
ISBN 978-0-387-34558-1 ► **approx. $689.00**



Handbook of Peer-to-Peer Networking

X. Shen, University of Waterloo, ON, Canada; **H. Yu**, Huawei Technologies, Bridgewater, NJ, USA; **J. Buford**, Avaya Labs Research, Basking Ridge, NJ, USA; **M. Akon**, University of Waterloo, ON, Canada (Eds.)

Offers elaborate discussions on fundamentals of peer-to-peer computing model, networks and applications. Provides a comprehensive study on recent advancements, crucial design choices, open problems, and possible solution strategies. Written by a team of leading international researchers and professionals.

2010. XLVIII, 1500 p. Hardcover
ISBN 978-0-387-09750-3 ► **$249.00**

Easy Ways to Order for the Americas ► **Write:** Springer Order Department, PO Box 2485, Secaucus, NJ 07096-2485, USA ► **Call: (toll free)** 1-800-SPRINGER ► **Fax:** 1-201-348-4505 ► **Email:** orders-ny@springer.com or **for outside the Americas** ► **Write:** Springer Customer Service Center GmbH, Haberstrasse 7, 69126 Heidelberg, Germany ► **Call:** +49 (0) 6221-345-4301 ► **Fax :** +49 (0) 6221-345-4229 ► **Email:** orders-hd-individuals@springer.com
► Prices are subject to change without notice. All prices are net prices.

014712x

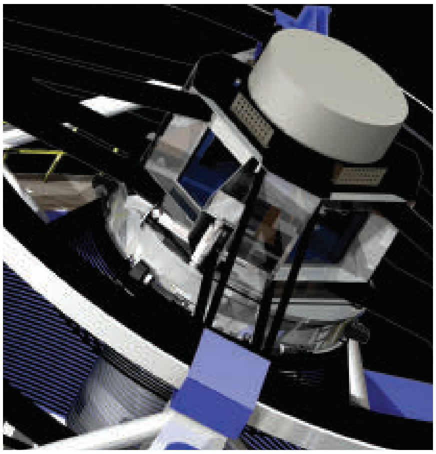
COMMUNICATIONS OF THE ACM

Departments

Last Byte

Association for Computing Machinery
Advancing Computing as a Science & Profession

News

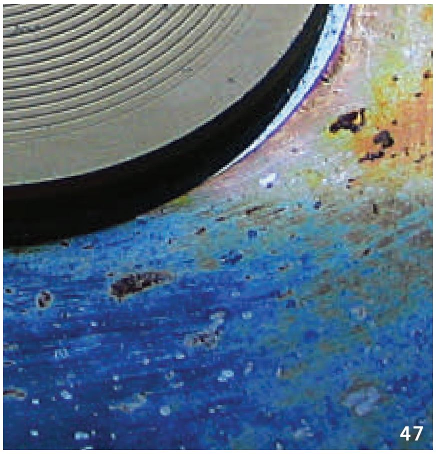


Viewpoints

IMAGE COURTESY OF LSST CORPORATION

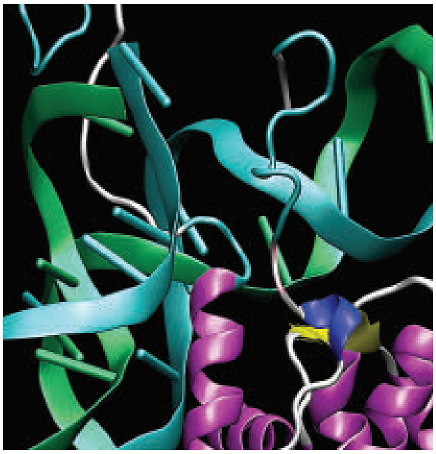
11/2010
VOL. 53 NO. 11

Practice



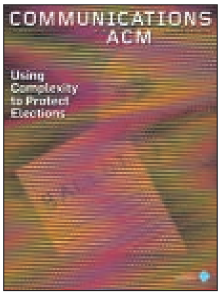
Articles' development led by acmqueue
queue.acm.org

Contributed Articles



Review Articles

Research Highlights



About the Cover: Vulnerabilities in the election process have been around for centuries. This month's cover story, beginning on p. 74, examines the use of computational complexity as an effective shield against election manipulation. Artist Melvin Galapon picks up that theme by colorfully depicting a protective layer of interference hovering over the election practice.

PHOTOGRAPH BY TARAN RAMPERSAD

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.

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

Computer Science Teachers Association
Chris Stephenson
Executive Director
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (800) 401-1799; F (541) 687-1840

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

CONTACT POINTS

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

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ADVERTISING

ACM ADVERTISING DEPARTMENT
2 Penn Plaza, Suite 701, New York, NY
10121-0701
T (212) 869-7440
F (212) 869-0481

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

Media Kit acmmediasales@acm.org

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) 626-0654.

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

Association for
Computing Machinery

Printed in the U.S.A.

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/1839676.1839677

Moshe Y. Vardi

On P, NP, and Computational Complexity

The second week of August was an exciting week. On Friday, August 6, Vinay Deolalikar announced a claimed proof that **P** ≠ **NP**. Slashdotted blogs broke the news on August 7 and 8, and suddenly the whole world was paying attention. Richard Lipton's August 15 blog entry at blog@CACM was viewed by about 10,000 readers within a week. Hundreds of computer scientists and mathematicians, in a massive Web-enabled collaborative effort, dissected the proof in an intense attempt to verify its validity. By the time the *New York Times* published an article on the topic on August 16, major gaps had been identified, and the excitement was starting to subside. The **P** vs. **NP** problem withstood another challenge and remained wide open.

During and following that exciting week many people have asked me to explain the problem and why it is so important to computer science. "If everyone believes that **P** is different than **NP**," I was asked, "why it is so important to prove the claim?'' The answer, of course, is that believing is not the same as knowing. The conventional "wisdom'' can be wrong. While our intuition does tell us that finding solutions ought to be more difficult than checking solutions, which is what the **P** vs. **NP** problem is about, intuition can be a poor guide to the truth. Case in point: modern physics.

While the **P** vs. **NP** quandary is a central problem in computer science, we must remember that a resolution of the problem may have limited practical impact. It is conceivable that **P** = **NP**, but the polynomial-time algorithms yielded by a proof of the equality are completely impractical, due to a very large degree of the polynomial or a very large multiplicative constant; after all, $(10n)^{1000}$ is a polynomial! Similarly, it is conceivable that **P** ≠ **NP**, but **NP** problems can be solved by algorithms with running time bounded by $n^{\log\log\log n}$—a bound that is not polynomial but incredibly well behaved.

Even more significant, I believe, is the fact that computational complexity theory sheds limited light on behavior of algorithms in the real world. Take, for example, the Boolean Satisfiability Problem (SAT), which is the canonical **NP**-complete problem. When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole. Garey and Johnson's classical textbook showed a long sad line of programmers who have failed to solve **NP**-complete problems. Guess what? These programmers have been busy! The August 2009 issue of *Communications* contained an article by Sharad Malik and Lintao Zhang (p. 76) in which they described SAT's journey from theoretical hardness to practical success. Today's SAT solvers, which enjoy wide industrial usage, routinely solve SAT instances with over one *million* variables. How can a scary **NP**-complete problem be so easy? What is going on?

The answer is that one must read complexity-theoretic claims carefully. Classical **NP**-completeness theory is about *worst-case* complexity.

Indeed, SAT does seem hard in the worst case. There are SAT instances with a few hundred variables that cannot be solved by any extant SAT solver. "So what?'' shrugs the practitioner, "these are artificial problems." Somehow, industrial SAT instances are quite amenable to current SAT-solving technology, but we have no good theory to explain this phenomenon. There is a branch of complexity theory that studies average-case complexity, but this study also seems to shed little light on practical SAT solving. How to design good algorithms is one of the most fundamental questions in computer science, but complexity theory offers only very limited guidelines for algorithm design.

An old cliché asks what the difference is between theory and practice, and answers that "in theory, they are not that different, but in practice, they are quite different." This seems to apply to the theory and practice of SAT and similar problems. My point here is not to criticize complexity theory. It is a beautiful theory that has yielded deep insights over the last 50 years, as well as posed fundamental, tantalizing problems, such as the **P** vs. **NP** problem. But an important role of theory is to shed light on practice, and there we have large gaps. We need, I believe, a richer and broader complexity theory, a theory that would explain both the difficulty and the easiness of problems like SAT. More theory, please!

Moshe Y. Vardi, EDITOR-IN-CHIEF

DOI:10.1145/1839676.1839678

How to Think About Objects

Though I agree with Mordechai Ben-Ari's Viewpoint "Objects Never? Well, Hardly Ever!" (Sept. 2010) saying that students should be introduced to procedural programming before object-oriented programming, dismissing OOP could mean throwing out the baby with the bathwater.

OOP was still in the depths of the research labs when I was earning my college degrees. I was not exposed to it for the first few years of my career, but it intrigued me, so I began to learn it on my own. The adjustment from procedural programming to OOP wasn't just a matter of learning a few new language constructs. It required a new way of thinking about problems and their solutions.

That learning process has continued. The opportunity to learn elegant new techniques for solving difficult problems is precisely why I love the field. But OOP is not the perfect solution, just one tool in the software engineer's toolbox. If it were the only tool, we would run the risk of repeating psychologist Abraham Maslow's warning that if the only tool you have is a hammer, every problem tends to look like a nail.

Learning any new software technique—procedural programming, OOP, or simply what's next—takes time, patience, and missteps. I have made plenty myself learning OOP, as well as other technologies, and continue to learn from and improve because of them.

For his next sabbatical, Ben-Ari might consider stepping back into the industrial world for a year or two. We've learned a great deal about OOP since he left for academia 15 years ago.

Jim Humelsine, Neptune, NJ

Evaluating Research: Hypercriticality vs. Radical Empiricism

In his Viewpoint "Is Computer Science Truly Scientific?" (July 2010), Gonzalo Génova suggested that computer science suffers from "radical empiricism," leading to rejection of research not supported by empirical evidence. We take issue with both his claim and (perhaps ironically) the evidence he used to support it.

Génova rhetorically asked "Must all scientific works be reasoned and demonstrable?," answering emphatically, "Yes, of course," to which we wholeheartedly agree. Broadly, there are two ways to achieve this goal: inference and deduction. Responding to the letter to the editor by Joseph G. Davis "No Straw Man in Empirical Research" (Sept. 2010, p. 7), Génova said theoretical research rests on definition and proof, not on evidence. Nonetheless, he appeared to be conflating inference and deduction in his argument that seminal past research would be unacceptable today. Many of the famous computer scientists he cited to support this assertion—Turing, Shannon, Knuth, Hoare, Dijkstra—worked (and proved their findings) largely in the more-theoretical side of CS. Even a cursory reading of the latest *Proceedings of the Symposium on Discrete Algorithms* or *Proceedings of Foundations of Computer Science* turns up many theoretical papers with little or no empirical content. The work of other pioneers Génova cited, including Meyer and Gamma, might have required more empirical evidence if presented today. Génova implied their work would not be accepted, and we would therefore be unable to benefit from it. The fact that they met the requirements of their time but (arguably) not of ours does not mean they would not have risen to the occasion had the bar been set higher. We suspect they would have, and CS would be none the poorer for it.

Génova's suggestion that CS suffers today from "radical empiricism" is an empirical, not deductive, claim that can be investigated through surveys and reviews. Still, he supported it via what he called "inductive justification," which sounds to us like argument by anecdote. Using the same inductive approach, conversations with our colleagues here at the University of California, Davis, especially those in the more theoretical areas of CS, lead us to conclude that today's reviews, though demanding and sometimes disappointing, are not "radically empirical." To the extent a problem exists in the CS review process, it is due to "hypercriticality," as Moshe Y. Vardi said in his "Editor's Letter" (July 2010, p. 5), not "radical empiricism."

Earl Barr and Christian Bird,
Davis, CA

Author's Response:

I'm glad to hear from Barr and Bird that there are healthy subfields in CS in this respect. I used "inductive justification" to support the claim that many classical works in the field are more theoretical and speculative than experimental, not to support an argument that CS suffers today from "radical empiricism." Investigating the latter through exhaustive empirical surveys of reviews would require surveyors being able to classify a reviewer as a "radical empiricist." If my column served this purpose, then I am content with it.

Gonzalo Génova, Madrid, Spain

Conclude with the Conclusions

The Kode Vicious Viewpoint "Presenting Your Project" by George V. Neville-Neil (Aug. 2010) made several debatable points about presentations, one of which was inexcusable: "...I always end with a Questions slide."

You have just given a 25-minute technical presentation to an educated, knowledgeable, technical audience. Using a series of slides, you have explained your problem, described your solutions, discussed your experiments, and finally concluded, displaying each slide for a minute or two. Your penultimate slide summarizes the whole presentation, including its "takeaway" message—everything you want your listeners to remember. Now you expect to spend four or five minutes answering questions. The slide you show as you answer will be on screen two or three times longer than any other slide.

So why remove the most useful slide in the whole presentation—the summary—and replace it with a content-free alternative showing perhaps a word or two. Is your audience so dense it cannot hear you say "Thank you" or ask for questions unless they're on the screen? Do you think the audience will forget to say something? Or is the problem with you, the presenter? Would you yourself forget to ask for questions if the slide wasn't on the screen in front of you?

Technical presentations should be held to a higher standard of information content and knowledge transfer than a sales pitch. My advice: Remove the "Thank You" and "Questions" slides, and leave up your "Conclusions" and "Summary" as long as possible.

Michael Wolfe, Hillsboro, OR

For Electronic Health Records, Don't Ignore VistA

Why did Stephen V. Cantrill's article "Computers in Patient Care: The Promise and the Challenge" (Sept. 2010) say nothing about the Veterans Health Information Systems and Technology Architecture (VistA) used for decades throughout the U.S. Department of Veterans Affairs (VA) medical system for its patients' electronic medical records? With 153 medical centers and 1,400 points of care, the VA in 2008 delivered care to 5.5 million people, registering 60 million visits (http://www1.va.gov/opa/publications/factsheets/fs_department_of_veterans_affairs.pdf).

In his book *The Best Care Anywhere* (http://p3books.com/bestcareanywhere) Phillip Longman documented VistA's role in delivering care with better outcomes than national averages to a population less healthy than national averages at a cost that has risen more slowly than national averages. Included was a long list of references (more than 100 in the 2010 second edition), yet Cantrill wrote "Although grand claims are often made about the potential improvements in the quality of care, decreases in cost, and so on, these are very difficult to demonstrate in a rigorous, scientific fashion."

Public-domain VistA also generalizes well outside the VA. For example, it has been deployed in the U.S. Indian Health Service, with additional functionality, including pediatrics. Speaking at the 2010 O'Reilly Open Source Convention (http://www.oscon.com/oscon2010/public/schedule/detail/15255), David Whiles, CIO of Midland Memorial Hospital, Midland, TX, described his hospital's deployment of VistA and how it has since seen a reduction in mortality rates of about two per month, as well as a dramatic 88% decrease in central-line infections entering at catheter sites (http://www.youtube.com/watch?v=ExoF_Tq14WY). Meanwhile, the country of Jordan (http://ehs.com.jo) is piloting an open source software stack deployment of VistA to provide electronic health records within its national public health care system.

[In the interests of full disclosure, I am an active member of the global VistA community, co-founding WorldVistA in 2002 (http://worldvista.org), a 501(c)(3) promoting affordable health care IT through VistA. Though now retired from an official role, I previously served as a WorldVistA Director.]

K.S. Bhaskar, Malvern, PA

Author's Response:

I appreciate Bhaskar's comments about the VA's VistA medical information system and applaud his efforts to generate a workable system in the public domain, but he misunderstood the intent of my article. It was not to be a comparison of good vs. bad or best vs. worst, but rather a discussion of many of the endemic issues that have plagued developers in the field since the 1960s. For example, MUMPS, the language on which VistA is based, was developed in the early 1970s for medical applications; VistA achieved general distribution in the VA in the late 1990s, almost 30 years later. Why so long? I tried to address some of these issues in the article. Also, VistA does not represent an integrated approach, but rather an interfaced approach with several proprietary subsystems.

Stephen V. Cantrill, M.D., Denver

Correction

The tribute "Robin Milner: The Elegant Pragmatist" by Leah Hoffmann (June 2010) requires a small correction. It said Milner "served as the first chair of the University of Cambridge Computer Laboratory." For all his many gifts, however, Milner was not precocious enough to have run a university laboratory at age three. He was born in 1934, and the University of Cambridge Computer Laboratory was founded (as the Mathematical Laboratory) in 1937. The source of the error is apparently Milner's official obituary, which noted he "held the first established Chair in Computer Science at the University of Cambridge." Translation to American: He held the first endowed professorship. In Britain, the word "Chair" refers to a professorship and should not be confused with "Head of Department."

Lawrence C. Paulson, Cambridge, England

Correction

Tom Geller's news story "Beyond the Smart Grid" (June 2010) should have cited Froehlich, J. Larson, E., Campbell, T., Haggerty, C., Fogarty, J., and Patel, S. "HydroSense: Infrastructure-Mediated Single-Point Sensing of Whole-Home Water Activity in *Proceedings of UbiComp 2009* (Orlando, FL, Sept. 30–Oct. 3, 2009) instead of Patel, S.N., Reynolds, M.S., and Abowd, G.D. "Detecting Human Movement By Differential Air Pressure Sensing in HVAC System Ductwork: An Exploration in Infrastructure-Mediated Sensing" in *Proceedings of Pervasive 2008, the Sixth International Conference on Pervasive Computing* (Sydney, Australia, May 19–22, 2008). We apologize for this error.

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

Coming Next Month in
COMMUNICATIONS

The Impact of Web 2.0 Technologies

Bayesian Networks

CS Education Week

Why We Need a Research Data Census

As well as the latest news on topic modeling, eye tracking and mobile phones, and crowdsourcing for social good.

Previous A.M. Turing Award Recipients

1966	A.J. Perlis
1967	Maurice Wilkes
1968	R.W. Hamming
1969	Marvin Minsky
1970	J.H. Wilkinson
1971	John McCarthy
1972	E.W. Dijkstra
1973	Charles Bachman
1974	Donald Knuth
1975	Allen Newell
1975	Herbert Simon
1976	Michael Rabin
1976	Dana Scott
1977	John Backus
1978	Robert Floyd
1979	Kenneth Iverson
1980	C.A.R Hoare
1981	Edgar Codd
1982	Stephen Cook
1983	Ken Thompson
1983	Dennis Ritchie
1984	Niklaus Wirth
1985	Richard Karp
1986	John Hopcroft
1986	Robert Tarjan
1987	John Cocke
1988	Ivan Sutherland
1989	William Kahan
1990	Fernando Corbató
1991	Robin Milner
1992	Butler Lampson
1993	Juris Hartmanis
1993	Richard Stearns
1994	Edward Feigenbaum
1994	Raj Reddy
1995	Manuel Blum
1996	Amir Pnueli
1997	Douglas Engelbart
1998	James Gray
1999	Frederick Brooks
2000	Andrew Yao
2001	Ole-Johan Dahl
2001	Kristen Nygaard
2002	Leonard Adleman
2002	Ronald Rivest
2002	Adi Shamir
2003	Alan Kay
2004	Vinton Cerf
2004	Robert Kahn
2005	Peter Naur
2006	Frances E. Allen
2007	Edmund Clarke
2007	E. Allen Emerson
2007	Joseph Sifakis
2008	Barbara Liskov
2009	Charles P. Thacker

Additional information on the past recipients of the A.M. Turing Award is available on: http://awards.acm.org/homepage.cfm?awd=140

ACM A.M. TURING AWARD NOMINATIONS SOLICITED

Nominations are invited for the 2010 ACM A.M. Turing Award. This, ACM's oldest and most prestigious award, is presented for contributions of a technical nature to the computing com¬munity. Although the long-term influences of the nominee's work are taken into consideration, there should be a particular outstanding and trendsetting technical achievement that constitutes the principal claim to the award. The award carries a prize of $250,000 and the recipient is expected to present an address that will be published in an ACM journal. Financial support of the Turing Award is provided by the Intel Corporation and Google Inc.

Nominations should include:

1) A curriculum vitae, listing publications, patents, honors, other awards, etc.

2) A letter from the principal nominator, which describes the work of the nominee, and draws particular attention to the contribution which is seen as meriting the award.

3) Supporting letters from at least three endorsers. The letters should not all be from colleagues or co-workers who are closely associated with the nominee, and preferably should come from individuals at more than one organization. Successful Turing Award nominations usually include substantive letters of support from a group of prominent individuals broadly representative of the candidate's field.

For additional information on ACM's award program please visit: www.acm.org/awards/

Additional information on the past recipients of the A.M. Turing Award is available on: http://awards.acm.org/homepage.cfm?awd=140

Nominations should be sent electronically by November 30, 2010 to: Vinton G. Cerf, c/o mcguinness@acm.org

Association for
Computing Machinery

DOI:10.1145/1839676.1839679

In the Virtual Extension

To ensure the timely publication of articles, Communications *created the Virtual Extension (VE) to expand the page limitations of the print edition by bringing readers the same high-quality articles in an online-only format. VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on merit. The following synopses are from articles now available in their entirety to ACM members via the Digital Library.*

contributed article

DOI: 10.1145/1839676.1839702

Supporting Ubiquitous Location Information in Interworking 3G and Wireless Networks

Massimo Ficco, Roberto Pietrantuono, and Stefano Russo

Users and wireless ISPs could tap location-based services across networks belonging to other ISPs to create ubiquitous personal networks.

Location-based services have emerged as a main interest of wireless ISPs, or WISPs, and network operators. Positioning mobile devices in the third generation (3G) of wireless communication networks (such as the Universal Mobile Telecommunications System) is crucial to many commercial services, including location applications that utilize accurate positioning (such as handset navigation tracking and locating points of interest); public and private emergency services, calling firefighters, medical teams, and emergency roadside assistance; and future applications (such as fraud detection, location-sensitive billing, and advertising).

However, positioning techniques vary by accuracy, implementation cost, and application scenarios (such as indoor and outdoor). WISPs can exploit their availability in order to locate their users in heterogeneous environments by using the most suitable positioning technique in a manner transparent to the user.

The recent interworking between 3G systems and wireless networks (such as IEEE 802.11 and Bluetooth) allows WISPs to leverage wireless networks for localization purposes. Wireless hotspots in public and private places (such as homes, offices, airports, shopping malls, arenas, hotels, and libraries), along with the new generation of mobile devices supporting multiple positioning technologies (such as GPS, Bluetooth, Wi-Fi, and RFID), fosters WISP development of integrated positioning systems.

contributed article

DOI: 10.1145/1839676.1839701

Relative Status of Journal and Conference Publications in Computer Science

Jill Freyne, Lorcan Coyle, Barry Smyth, and Padraig Cunningham

Citations represent a trustworthy measure of CS research quality—whether in articles in conference proceedings or in CS journals.

Though computer scientists agree that conference publications enjoy greater status in CS than in other disciplines, there is little quantitative evidence to support this view. The importance of journal publication in academic promotion makes it a highly personal issue, since focusing exclusively on journal papers misses many significant papers published by CS conferences.

This article aims to quantify the relative importance of CS journal and conference papers, showing that those in leading conferences match the impact of those in mid-ranking journals and surpass the impact of those in journals in the bottom half of the Thompson Reuters rankings (http://www.isiknowledge.com) for impact measured in terms of citations in Google Scholar. We also show that poor correlation between this measure and conference acceptance rates indicates conference publication is an inefficient market where venues equally challenging in terms of rejection rates offer quite different returns in terms of citations.

How to measure the quality of academic research and performance of particular researchers has always involved debate. Many CS researchers feel that performance assessment is an exercise in futility, in part because academic research cannot be boiled down to a set of simple performance metrics, and any attempt to introduce them would expose the entire research enterprise to manipulation and gaming. On the other hand, many researchers want some reasonable way to evaluate academic performance, arguing that even an imperfect system sheds light on research quality, helping funding agencies and tenure committees make more informed decisions.

viewpoint

DOI: 10.1145/1839676.1839703

In Support of Computer Science Teachers and the CSTA

Duncan Buell

A number of recent articles and comments have discussed the imbalance between enrollment and opportunities in computer science and the under-enrollments by minorities and women. An ongoing thread in Peter Denning's *Communications* columns and elsewhere concerns the identity of the discipline to which we belong. As the national representative from universities to the board of the Computer Science Teachers Association (CSTA), I continually see the question of the identity of our discipline both within and external to our field.

The identity of computer science is nowhere more important to the discipline of computer science than in the K–12 school system. We can instruct our own students in the nature of the discipline, but those we so instruct will only be those who first choose to come to us. If we want more students, and if we want to be understood for what we are, we must clarify the message about computer science that all students will receive as part of their K–12 education.

Even if one does not believe that "computer science" should be taught in the K–12 system, it is nonetheless necessary for us to be involved in defining for the schools that which is called "computer science." There *will* be courses in Photoshop, Web design, office tools, A+ certification, networking, and such, and there *will* be (a smaller number of) courses in Visual Basic, C++, or even Java. The simple fact is that these courses will exist in the schools, and there is nothing fundamentally wrong with that. What is a problem is for students to be misled into thinking that these are all indistinguishable and all equally describable as "computer science."

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:10.1145/1839676.1839680 http://cacm.acm.org/blogs/blog-cacm

Rethinking the Systems Review Process

Tessa Lau launches a discussion about the acceptance criteria for HCI systems papers at CHI, UIST, and other conferences.

Tessa Lau "What Makes a Good HCI Systems Paper?"

http://cacm.acm.org/blogs/blog-cacm/86066

There has been much discussion on Twitter, Facebook, and in blogs about problems with the reviewing system for HCI systems papers (see James Landay's blog post, "I give up on CHI/UIST" and the comment thread at http://dubfuture.blogspot.com/2009/11/i-give-up-on-chiuist.html). Unlike papers on interaction methods or new input devices, systems are messy. You can't evaluate a system using a clean little lab study, or show that it performs 2% better than the last approach. Systems often try to solve a novel problem for which there was no previous approach. The value of these systems might not be quantified until they are deployed in the field and evaluated with large numbers of users. Yet doing such an evaluation incurs a significant amount of time and engineering work, particularly compared to non-systems papers. The result, observed in conferences like CHI and UIST, is that systems researchers find it very difficult to get papers accepted. Reviewers reject messy systems papers that don't have a thorough evaluation of the system, or that don't compare the system against previous systems (which were often designed to solve a different problem).

At CHI 2010 there was an ongoing discussion about how to fix this problem. Can we create a conference/publishing process that is fair to systems work? Plans are afoot to incorporate iterative reviewing into the systems paper review process for UIST, giving authors a chance to have a dialogue with reviewers and address their concerns before publication.

However, I think the first step is to define a set of reviewing criteria for HCI systems papers. If reviewers don't agree on what makes a good systems paper, how can we encourage authors to meet a standard for publication?

Here's my list:

▸ A clear and convincing description of the problem being solved. Why isn't current technology sufficient? How many users are affected? How much does this problem affect their lives?

▸ How the system works, in enough detail for an independent researcher to build a similar system. Due to the complexities of system building, it is often impossible to specify all the parameters and heuristics being used within a 10-page paper limit. But the paper ought to present enough detail to enable another researcher to build a comparable, if not identical, system.

▸ Alternative approaches. Why did you choose this particular approach? What other approaches could you have taken instead? What is the design space in which your system represents one point?

▸ Evidence that the system solves the problem as presented. This does not have to be a user study. Describe situations where the system would be useful and how the system as implemented performs in those scenarios. If users have used the system, what did they think? Were they successful?

▸ Barriers to use. What would prevent users from adopting the system, and how have they been overcome?

▸ Limitations of the system. Under what situations does it fail? How can users recover from these failures?

What do you think? Let's discuss.

Readers' comments

I'd like to second your first recommendation. I've reviewed a number of systems papers that do not provide a sufficiently compelling motivation or use case—why should I or anyone care about this system? Without this, the paper often represents technology in search of a problem.

Now, having read Don Norman's provocative article "Technology First, Needs Last: The Research-Product Gulf" in the *recent issue of* interactions *magazine, I have a keener appreciation for the possible contribution of some technologies in search of problems, but I still believe these are more the exception than the norm … and that without adequately making the case for the human-centered value(s) the systems will help realize, such papers are probably more suitable for other venues.*

—Joseph McCarthy

One problem is that our field is moving so fast that we have to allow new ideas to cross evolve with other ideas rapidly. If we require evaluations of every paper, then we don't have the rapid turnaround required for innovations to cross paths with each other.

On the other hand, it seems wrong not to have some filter. Without filters, we might end up publishing ideas that seem interesting, but are actually quite useless.

I think you have a great start on a list of discussion points. One thing to keep in mind is that we should evaluate papers in whole rather in parts. I will often recommend accepting papers that are deficient in one area but very good in another.

—Ed Chi

I think it would be useful to some of us discussing your post if you could say more about the kinds of evidence you are referring to when you say "evidence that the system solves the problem" that are not user studies.

So, what are some examples of specific system problems ("clearly and convincingly presented"), and what would you consider appropriate evidence to show that your system solved the problem? Is it a set of usage scenarios that have been hard to address through previous designs and you show how a single interface design can address them completely? Is it a new, significantly more efficient algorithm or mechanism, for example, to handle complex preferences around group permissions, which would be useful to the builders of group systems to know about? (In the latter case, would evidence be performance testing, using logs of previous queries as data?) Is it a new approach for using skin-tapping as input?

—Dan Gruen

TESSA LAU

"If reviewers don't agree on what makes a good systems paper, how can we encourage authors to meet a standard for publication?"

I am a strong proponent of rigorous gatekeeping at conferences simply because I need some help figuring out which things are worth following in my limited time. At the same time, I think it is important to keep in mind all the different ways a systems paper can be really valuable and worth seeing at a conference like CHI. A systems paper could be interesting thanks to a thorough analysis of its deployment and usage (evaluation). Or it could be interesting thanks to a well-argued discussion of why it was built a particular way (design). Or it might just demonstrate that a given interesting capability could be created at all. Or it could be a careful argument about why a certain system would be really useful, even if it hasn't been built or evaluated yet (motivation/position paper). In the end, what I want are papers that stimulate thought and action. I'm not going to demand any particular levels of motivation, design, or evaluation; rather, I'm going to ask whether the whole is innovative enough. This is a highly subjective decision, which is why I greatly value wise program committees who can make such a judgment on my behalf.

—David Karger

I like your list, and think that the bullet points are representative of good evaluation criteria for systems papers across computer science.

The main sticking point is, as I see it, "Evidence that the system solves the problem as presented." In some other areas of empirical computer science, we have repositories of test problems, suites of agreed-upon performance metrics, testing harnesses for software, and so forth. Usability testing is seen as the gold standard in HCI, though, and it's much harder to leverage such tools to make evaluation by user testing efficient. The effort devoted to user testing of a new system can sometimes rival the effort to having built the system in the first place—okay, I might be exaggerating a bit, but still….

If we could agree on some reasonable substitutes, that would be good. Some of the papers I've worked on have included GOMS models of performance, for example, but not everyone buys into a purely analytical approach. Sometimes, even worse, what I'd like to convey in a paper is a conceptual shift, a different way of thinking about some kind of problem, and that's even harder to evaluate than pure performance.

—Robert St. Amant

I've suggested it before and will suggest it again. An easy start could be to make accepted interactivity demos "worth" as much as a full paper at CHI—same presentation length, and the associated paper (maybe six pages long) needs to be of the same "archival" status in the ACM Digital Library.

This could show a true commitment to systems research.

—Florian Mueller

Tessa Lau responds

Thank you all for the interesting discussion. My goal was to initiate a discussion within our community, not to have all the answers.

Along those lines, Dan, the question you raise about what constitutes "appropriate evidence" is one that I'll turn back to the community for a collective answer.

For what it's worth, though, I don't think of your examples as "systems." The first is usage scenarios or design. The second is an algorithm. The third is an interaction method. Each of those is fairly self-contained and possible to evaluate using a fairly closed study.

What I meant by "systems" is an implemented prototype that gives people access to new functionality that did not exist before. Examples of "systems" include CoScripter, Many Eyes, Landay's DENIM and SILK, Gajos's SUPPLE, Andrew Ko's Whyline. How can we show that each of these systems is "innovative enough" (in David's words) to merit publication?

Tessa Lau is a research staff member and manager at IBM's Almaden Research Center.

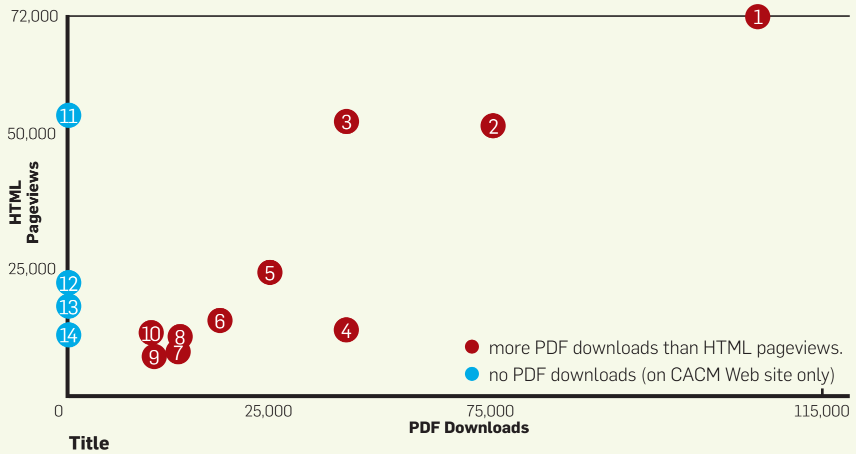
DOI:10.1145/1839676.1839681

David Roman

A Preference for PDF

When it comes to electronic formats, *Communications* readers prefer PDF to HTML, according to data on a year's worth of published articles. PDF's faithful reproduction of magazine pages appears to give it an edge. Legacy is another factor. Every article in *Communications*' 52-year history is available in PDF, while HTML versions date back to 1999. Also, members are accustomed to accessing ACM's many transactions, proceedings, and journals exclusively in PDF.

The table shows how the most popular articles published over a 12-month span were accessed from the DL and *Communications*' Web site.



	Title
1	A Few Billion Lines of Code Later http://cacm.acm.org/magazines/2010/2/69354
2	What Should We Teach New Software Developers? Why? http://cacm.acm.org/magazines/2010/1/55760
3	You Don't Know Jack About Software Maintenance http://cacm.acm.org/magazines/2009/11/48444
4	MapReduce: A Flexible Data Processing Tool http://cacm.acm.org/magazines/2010/1/55744
5	You're Doing it Wrong http://cacm.acm.org/magazines/2010/7/95061
6	Retrospective: An Axiomatic Basis for Computer Programming http://cacm.acm.org/magazines/2009/10/42360
7	An Interview With Edsger W. Dijkstra http://cacm.acm.org/magazines/2010/8/96632
8	Recent Progress in Quantum Algorithms http://cacm.acm.org/magazines/2010/2/69352
9	MapReduce and Parallel DBMSs: Friends or Foes? http://cacm.acm.org/magazines/2010/1/55743
10	Scratch: Programming for All http://cacm.acm.org/magazines/2009/11/48421
11	How we Teach Introductory Computer Science is Wrong http://cacm.acm.org/blogs/blog-cacm/45725
12	Are You Invisible? http://cacm.acm.org/blogs/blog-cacm/94307
13	The "NoSQL" Discussion has Nothing to Do With SQL http://cacm.acm.org/blogs/blog-cacm/50678
14	A Tale of A Serious Attempt At P≠NP http://cacm.acm.org/blogs/blog-cacm/97587

Methodology: Cited articles were published online or in the monthly print *Communications* between October 2009 through September 2010, and include the 10 titles with the most PDF downloads from the ACM Digital Library in the 12 months following their publication, and the 10 titles with the most HTML pageviews on the *Communications* Web site over the same period, according to Google Analytics.

ACM Member News

CHRIS STEPHENSON ON K–12 CS EDUCATION

ACM Member News recently interviewed Chris Stephenson, executive director of the Computer Science Teachers Association, about the status of K–12 CS education in the U.S. and how ACM members can help. "I think computer science teachers in the U.S. face a number of challenges that make it difficult for them to teach to their full potential," says Stephenson. "First, computer science is poorly understood in the U.S. school system, so administra-tors and decision-makers have no idea what CS teachers do or why it is important. This means teachers have to continually fight for their programs and their students and are often under-supported in terms of resources. Also, our teacher certification requirements are a complete mess, and CS teachers in many states must first be certified in some other discipline. This doubles the time and effort required. And it means CS teachers can be assigned to teach something other than computer science at any time. Finally, our discipline requires teachers to continually upgrade both their teaching and technical knowledge, and there is very little access to relevant and timely professional development.

"The most important thing ACM members can do is to advocate for computer science courses in their local schools.... The other really important action that U.S. members can do is to contact their congressional representatives and request that they support the Computer Science Education Act [HR 5929]. With enough support, this bill has the power to fundamentally change computer science education in this country."

A new joint ACM–CSTA report, *Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age*, is available at http://acm.org/Runningonempty/.

—*Jack Rosenberger*

Science | DOI:10.1145/1839676.1839682

Gregory Goth

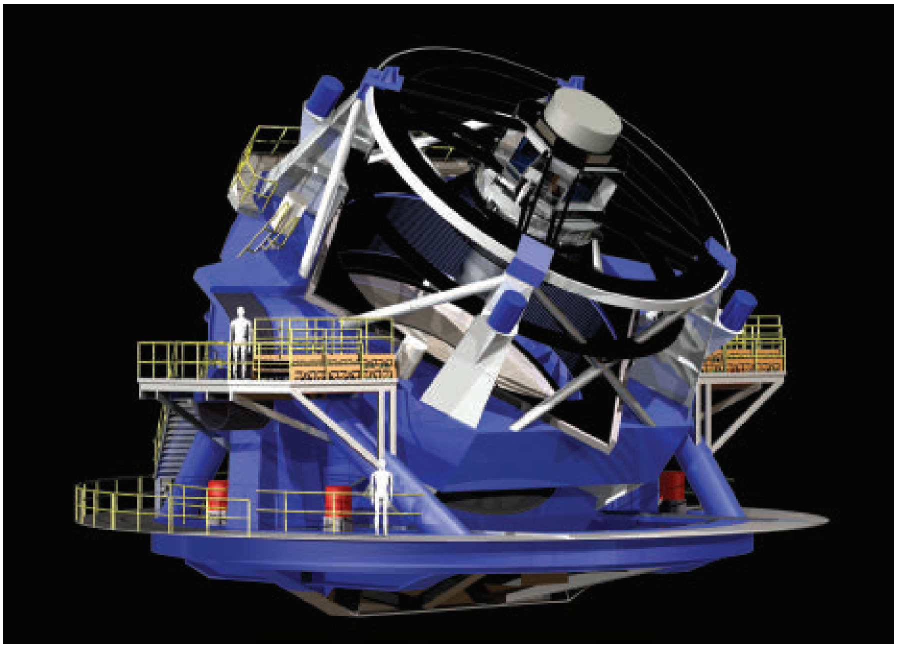
Turning Data Into Knowledge

Today's data deluge is leading to new approaches to visualize, analyze, and catalog enormous datasets.

THE AMOUNT OF data available to scientists of nearly every discipline has almost become a "Can you top this?" exercise in numbers.

The Sloan Digital Sky Survey (SDSS), for example, is often cited as a prime example. Since the survey's 2.5-meter telescope first went online in 1998, more than 2,000 refereed publications have been produced, but they use just 10% of the survey's available imaging data, according to a recent U.S. National Science Foundation workshop on data-enabled science in the mathematical and physical sciences. Once the next-generation, state-of-the-art Large Synoptic Survey Telescope (LSST) goes online in 2016, however, it is estimated to be capable of producing a SDSS-equivalent dataset every night for the next 10 years. Another often-cited example is the Large Hadron Collider. It will generate two SDSS's worth of data each day.

On the surface, then, the scientific community's mandate seems clear: create better computational tools to visualize, analyze, and catalog these enormous datasets. And to some extent, there is wide agreement these tasks must be pursued.



The Large Synoptic Survey Telescope will have the ability to survey the entire sky in only three nights.

IMAGE COURTESY OF LSST CORPORATION

Some leading computational research scientists believe, however, that progress in utilizing the vast expansion of data will best be attacked on a project-by-project basis rather than by a pan-disciplinary computational blueprint.

"In theory, you might think we should all be working together, and the reality might be that each of the people working on their own discipline are achieving the results they need to scientifically," says Dan Masys, M.D., chairman of biomedical informatics at Vanderbilt University. "There's a cost of communication that reaches an irreducible minimum when you work across disciplinary boundaries, and sometimes it's worth it.

"But the grander potential for synergy that's often spoken of at the level of federal funding agencies probably doesn't happen as much as people think would be best for science," Masys continues. "You can't push that rope all that well, because it depends on the art of the possible with respect to technologies and the vision of the scientists doing the work."

Tom Mitchell, chairman of the machine learning department at Carnegie Mellon University (CMU), concurs with Masys' assessment. "I think it starts from the bottom up and at some point you'll see commonalities across domains," he says. As an example, he cites time series algorithms being developed by CMU colleague Eric Xing that may also be useful for brain imaging work Mitchell is undertaking.

"There's an example I think is probably pretty representative of how it's going to go," Mitchell says. "People encounter problems and have to design algorithms to address them, but time series analysis is a pretty generic problem. So I think bottom up it will grow and then they will start connecting across [different disciplines]."

Vanderbilt's Masys is about to begin a collaboration with computational biologists from Oak Ridge National Laboratory. Masys says the Oak Ridge scientists' optimization of Vanderbilt's fundamental algorithms and the lab's teraflop-capable architecture will likely speed processing of problems involving multiplying "several million genome data points by several thousand people" from five days to three hours—a prime example of focused intradisciplinary collaboration and leading-edge hardware.

New Perspectives on Data

Both Mitchell and Randal Bryant, dean of the school of computer science at CMU, cite the influence of commercial companies for helping to expand the concept of what kind of data, and what kind of data storage and computational architectures, can produce useful scientific results.

"The commercial world, Google and its peers, have been the drivers on the data side, much more than the traditional sciences or universities," says Bryant, who cites the example of a Google cluster running a billion-word index that outperformed the Big Iron architecture of the "usual suspects" in a 2005 language-translation contest sponsored by the U.S. National Institute of Standards and Technology.

The availability of such large datasets can lead to serendipitous discoveries such as one made by Mitchell and his colleagues, using a trillion-word index Google had originally provided for machine translation projects. "We found we could build a computational model that predicts the neural activity that will show up in your brain when you think about an arbitrary noun," Mitchell says. "It starts by using a trillion-word collection of text provided to us by Google, and looks up the statistical properties of that word in the text; that is, if you give it the word 'telephone', it will look up how often 'telephone' occurs with words from a long list of verbs—for example, how often does it occur with 'hug', or 'eat', and so on.

"To Google's credit they put this out on the Web for anybody to use, but they were thinking it would be used by researchers working on translation—and it turned out to be useful for something else."

Meanwhile, the LSST project is planning multiple vectors by which its huge dataset—all of which will be publicly available in near-real time—will aid research by professional astronomers; programs at museums, secondary schools, and other institutions; and citizen scientists. The project's goal, say the organizers, is "open source, open data."

"We will develop methods for engaging the public so anyone with a Web browser can effectively explore aspects of the LSST sky that interest and impact the public," according to the LSST organizers. "We will work with the IT industry on enhanced visualization involving dynamic graphics overlays from metadata and provide tools for public query of the LSST database."

The LSST organization's hope, then, is that the distributed nature of allowing any researcher at any level to access the data will result in a plethora of projects—a kind of "given enough eye-

Ubiquitous Computing

Intel's Friendly, Smart Machines

Context-aware computing, in which devices understand what a user is doing and anticipate his or her needs without being asked, are the next step in the evolution of smart machines, says Justin Rattner, Intel vice president and chief technology officer.

In his keynote address at IDF2010, the recent Intel Developer Forum in San Francisco, Rattner laid out a vision in which computers use a variety of sensors—microphones, accelerometers, and global positioning systems (GPSs)—combined with "soft sensors" such as calendars and social networks, to track people's activity and figure out how the devices can help. For instance, a device might locate someone at her office, hear the sound of human voices, crosscheck her calendar, and conclude she's in a business meeting, then suggest to the husband trying to call her that this wouldn't be a good time to interrupt.

A television remote control, using unsupervised learning in which it continuously collects data and makes inferences about what's going on around it, could learn to recognize which person is holding it—based on how the user moves, what angle he holds it at, and how fast he presses the buttons—then make personalized recommendations for shows, based on past preferences. A prototype Personal Vacation Assistant, developed with Fodor's Travel, uses GPS location, time of day, and past behavior to recommend restaurants and tourist sites. Data, collected over time and shared among devices, is run through an inference algorithm that examines the input and generates confidence scores to determine what is likely going on.

Collecting this data will require giving users control over what gets shared, and allow them to turn off sensors, Rattner says. He gives no timeline for introducing such applications, but says, "We believe that context-aware computing is poised to fundamentally change the way we interact and relate to the devices that we use today."

—Neil Savage

balls" approach to massive datasets.

However, even massive datasets are sometimes not complete enough to deliver definitive results. Recent discoveries in biomedical research have revealed that even a complete index of the human genome's three billion pairs of chemical bases has not greatly accelerated breakthroughs in health care, because other crucial medical data is missing. A study of 19,000 women, led by researchers at Brigham and Women's Hospital in Boston, used data constructed from the National Human Genome Research Institute's catalog of genome-wide association study results published between 2005 and June 2009—only to find that the single biggest predictor of heart disease among the study's cohort is self-reported family history. Correlating such personal data with genetic indexes on a wide demographic scale today is nearly impossible as an estimated 80% of U.S.-based primary-care physicians do not record patient data in electronic medical records (EMRs). Recent government financial incentives are meant to spur EMR adoption, but for the immediate future, crucial data in biomedical research will not exist in digital form.

Another issue in biomedical research is the reluctance of traditionally trained scientists to accept datasets that were not created under the strict parameters required by, for example, epidemiologists and pharmaceutical companies.

CMU's Mitchell says this arena of public health research could be in the vanguard of what may be the true crux of the new data flood—the idea that the provenance of a given dataset should matter less than the provenance of a given hypothesis.

"The right question is, Do I have a scientific question and a method for answering it that is scientific, no matter what the dataset is?" Mitchell asks. Increasingly, he says, computational scientists will need to frame their questions and provide data for an audience that extends far beyond their traditional peers.

"We're at the beginning of the curve of a decades-long trend of increasingly evidence-based decision-making across society, that's been noticed by people in all walks of life," he says. "For example, the people at the public policy school at CMU came to the machine learning department and said, 'We want to start a joint Ph.D. program in public policy and machine learning, because we think the future of policy analysis will be increasingly evidence-based. And we want to train people who understand the algorithms for analyzing and collecting that evidence as well as they understand the policy side.'" As a result, the joint Ph.D. program was created at CMU.

"The right question is, Do I have a scientific question and a method for answering it that is scientific, no matter what the dataset is?" asks Tom Mitchell.

Further Reading

Duda, S.N. Cushman, C., and Masys, D.R.
An XML model of an enhanced dictionary to facilitate the exchange of pre-existing clinical research data in international studies, *Proceedings of the 12th World Congress on Health Informatics*, Brisbane, Australia, August 20–24, 2007.

Mitchell,T.M., Shinkareva, S.V., Carlson, A., Chang, K.-M., Malave, V.L., Mason, R.A., and Just, M.A.
Predicting human brain activity associated with the meanings of nouns, *Science 320*, 5880, May 30, 2008.

Murray-Rust, P.
Data-driven science—a scientist's view, NSF/JISC Repositories Workshop position paper, April 10, 2007.

Newman, H.B.
Data intensive grids and networks for high energy and nuclear physics: drivers of the formation of an information society, World Summit on the Information Society, Pan-European Ministerial Meeting, Bucharest, Romania, November 7–9, 2002.

Thakar, A.R.
The Sloan Digital Sky Survey: drinking from the fire hose, *Computing in Science and Engineering 10*, 1, Jan./Feb. 2008.

Gregory Goth is an Oakville, CT-based writer who specializes in science and technology.

Research & Development

Paper Chase

Due to enormous governmental investments in research and development, scientists in many Asian countries are steadily increasing their number of papers published in scientific journals.

The Asia-Pacific region increased its total of published science articles from 13% in the early 1980s to slightly more than 30% in 2009, according to the Thomson Reuters National Science Indicators, an annual database of the number of articles published in about 12,000 internationally recognized journals. China leads the pack with 11% in 2009, up from 0.4% in the early 1980s, followed by Japan with 6.7% and India with 3.4%. In contrast, the ratio of articles from scientists in the U.S. decreased to 28% in 2009, down from 40% in the early 1980s.

In all, 25 nations have increased their research, but none more so than Singapore. With a population of just five million, the nation published 8,500 articles in 2009, compared with only 200 in 1981. Singapore now allocates 3% of its gross domestic product to research and development, a figure expected to rise to 3.5% by 2015.

The increase in scientific publications, especially in East Asian countries, reflects a "phenomenal" increase in funding, Simon Marginson, a professor of higher education at the University of Melbourne, told *The New York Times*. Marginson attributed the increase in research output to governments' commitment to establishing knowledge-intensive economies. "It's very much not simply about knowledge itself—it's about its usefulness throughout the economy. I think that that economic vision is really the principal driver," Marginson said.

Another reason for increased publications is that many Asian universities now receive additional funding to have their papers translated into English, the language used by the majority of academic journals.

—Phil Scott

Technology | DOI:10.1145/1839676.1839683

Gary Anthes

Security in the Cloud

Cloud computing offers many advantages, but also involves security risks. Fortunately, researchers are devising some ingenious solutions.

"Computing may some day be organized as a public utility, just as the telephone system is a public utility," Massachusetts Institute of Technology (MIT) computer science pioneer John McCarthy noted in 1961.

We aren't quite there yet, but cloud computing brings us close. Clouds are all the rage today, promising convenience, elasticity, transparency, and economy. But with the many benefits come thorny issues of security and privacy.

The history of computing since the 1960s can be viewed as a continuous move toward ever greater specialization and distribution of computing resources. First we had mainframes, and security was fairly simple. Then we added minicomputers and desktop and laptop computers and client-server models, and it got more complicated. These computing paradigms gave way in turn to *n*-tier and grid computing and to various types of virtualization.

As hardware infrastructures grew more complicated and fragmented, so did the distribution of software and data. There seemed no end to the ways that users could split up their computing resources, and no end to the security problems that arose as a result. Part of the problem has been one of moving targets—just as one computing paradigm seemed solid, a new, more attractive one beckoned.

In a sense, cloud computing simplifies security issues for users by outsourcing them to another party, one that is presumed to be highly skilled at dealing with them. Cloud users may think they don't have to worry about the security of their software and data anymore, because they're in expert hands.

But such complacency is a mistake, say researchers at Hewlett-Packard (HP) Laboratories in Bristol, U.K. They are prototyping Cells as a Service, by which they hope to automate security management in the cloud. A cell, managed as a single administrative domain using common security policies, contains a bundle of virtual machines, storage volumes, and networks running across multiple physical machines. Around the cells HP inserts various sensors, detectors, and mitigators that look for viruses, intrusions, and other suspicious behavior. Virtualization enables these agents to be very close to the action without being part of it or observed by it, according to HP.

Cloud computing simplifies security issues for users by outsourcing them to companies such as Microsoft, which recently opened a $550 million data center in Chicago.

PHOTOGRAPH USED WITH PERMISSION FROM MICROSOFT

"People often think of virtualization as adding to security problems, but it is fundamentally the answer to a lot of those problems," says Martin Sadler, director of HP's Systems Security Lab. "You can do all sorts of things you can't do when these things are physical machines." For example, the sensors can watch CPU activity, I/O patterns, and memory usage and, based on models of past behavior, recognize suspicious activity. They can also assess the probability of certain events happening and take action accordingly. They might, for instance, throttle back the CPU, stop all I/O to a virtual machine (VM), or take a clone of the VM and move it elsewhere for evaluation. Agents could be deployed by cloud users, cloud service providers, or third parties such as a virus protection company, Sadler says.

But these agents introduce their own management challenges. There might be as many as 30 agents, interacting in various ways and with varying drains on system resources. HP Labs is developing analytic tools that can generate playbooks that script system behavior. These templates, tailorable by users, employ cost/benefit analyses and reflect what is most important to users and what cost they are willing to bear for various types of protection.

Virtual Machine Introspection

IBM Research is pursuing a similar approach called "virtual machine introspection." It puts security inside a protected VM running on the same physical machine as the guest VMs running in the cloud. The security VM employs a number of protective methods, including the whitelisting and blacklisting of guest kernel functions. It can determine the operating system and version of the guest VM and can start monitoring a VM without any beginning assumption of its running state or integrity.

Instead of running 50 virus scanners on a machine with 50 guest VMs, virtual machine introspection uses just one, which is much more efficient, says Matthias Schunter, a researcher at IBM Research's Zurich lab. "Another big advantage is the VM can't do anything against the virus scan since it's not aware it's being scanned," he says.

Another variation, called "lie detection," puts a tiny piece of software inside the VM to look at the list of running processes as seen by the user. Introspection software outside the VM can reliably determine all the processes actually running on the VM; if there is any difference between the two lists, some malware, such as a rootkit, is suspected of running on the VM.

Looking from both within the VM and without, the lie detector can also compare the lists of files on disk, the views of open sockets, the lists of loaded kernel modules, and so on. "Each of these lie tests improves the chances of detecting potential malware, but none of them can prove that no malware exists," says IBM researcher Klaus Julisch.

In a third application, a virtual intrusion detection system runs inside the physical machine to monitor traffic among the guest VMs. The virtual networks hidden inside a physical machine are not visible to conventional detectors because the detectors usually reside in a separate machine, Schunter says.

Indeed, snooping between VMs inside a machine was shown to be a real possibility by researchers last year. Computer scientists Thomas Ristenpart, Hovav Shacham, and Stefan Savage at the University of California, San Diego and Eran Tromer at MIT proved it was possible for an adversary to get his or her VM co-located with a target's VM on a cloud's physical machine 40% of the time. In a paper, "Hey, You, Get Off of My Cloud," they showed how the adversary could launch a side-channel attack based on the VM's sharing of physical resources such as CPU data caches. The researchers also outlined a number of mitigation steps, but concluded the only practical and foolproof protection is for cloud users to require that their VMs run on dedicated machines, which is potentially a costly solution.

> **"People often think of virtualization as adding to security problems, but it is fundamentally the answer to a lot of those problems," says Martin Sadler, director of HP's Systems Security Lab.**

Difficulties With Encryption

Encryption is sometimes seen as the ultimate security measure, but it also presents difficulties in the cloud. At present, processing encrypted data means downloading it and decrypting it for local use and then possibly uploading the results, which is a cumbersome and costly process.

The ability to process encrypted data in place has been a dream of cryptographers for years, but it is now demonstrating some progress. Last year, Craig Gentry, first at Stanford University and then at IBM Research, proved it is possible to perform certain operations on data without first decrypting it. The technique, called "fully homomorphic encryption," was hailed as a conceptual breakthrough, but is so computationally demanding that practical applications are years away, experts say.

Meanwhile, the more limited ability to search encrypted data is closer to reality. In "Cryptographic Cloud Stor-

Society

Pew Report on Mobile Apps

Although a greater number of adults are turning to mobile phones to text and access the Internet, age and gender differences exist, according to a report by Pew Research Center's Internet & American Life Project and The Nielsen Company.

The report, titled *The Rise of Apps Culture*, found that 35% of U.S. adults have software applications or apps on their phones, yet only 24% of adults use those apps. Overall, today's apps culture—essentially born a couple of years ago with the introduction of Apple's iPhone—is predominantly male, younger, and more affluent.

Eighteen to 29-year-olds comprise only 23% of the U.S. adult population but constitute 44% of the apps-using population. By contrast, 41% of the adult population is age 50 and older but this group makes up just 14% of apps users. Younger adopters also use apps, including games and social media, more frequently.

Gender differences were also apparent. Women are more likely to rely on social networking apps such as Facebook and Twitter while men are inclined to use productivity and financial apps.

Nevertheless, adoption is growing rapidly. The Nielsen Company found that the average number of apps on a smartphone has swelled from 22 in December 2009 to 27 today. Not surprisingly, iPhone owners top the list with an average of 40 apps, while Android users claim 25 and BlackBerry owners 14.

The next few years will likely usher in dramatic changes. "Every metric we capture shows a widening embrace of all kinds of apps by a widening population, states Roger Entner, coauthor of the report and senior vice president at Nielsen. "It's ... not too early to say that this is an important new part of the technology world."

—Samuel Greengard

age,” a paper published earlier this year, researchers Seny Kamara and Kristin Lauter of Microsoft Research described a virtual private storage service that aims to provide the security of a private cloud and the cost savings of a public cloud. Data in the cloud remains encrypted, and hence protected from the cloud provider, court subpoenas, and the like. Users index their data, then upload the data and the index, which are both encrypted, to the cloud. As needed, users can generate tokens and credentials that control who has access to what data.

Given a token for a keyword, an authorized user can retrieve pointers to the encrypted files that contain the keyword, and then search for and download the desired data in encrypted form. Unauthorized observers can’t know anything useful about the files or the keywords.

The experimental Microsoft service also offers users “proof of storage,” a protocol by which a server can prove to a client that it did not tamper with its encrypted data. The client encodes the data before uploading it and can verify the data’s integrity at will.

Not all cloud security risks arise from technology, says Radu Sion, a computer science professor at Stony Brook University. There is scant legal or regulatory framework, and few precedents, to deal with issues of liability among the parties in cloud arrangements, he notes. “What happens when your data is on a server in China but you outsourced to a cloud service in New York?” asks Sion. “Or what if you have the legal resources to fight a subpoena for your data, but they subpoena your cloud provider instead? *You* will be under scrutiny for moving to the cloud by your shareholders and everyone else.”

In “Cryptographic Cloud Storage,” Microsoft researchers Seny Kamara and Kristin Lauter describe a virtual private storage service that provides the security of a private cloud and the cost savings of a public cloud.

Nevertheless, Sion says all but the most sophisticated enterprises will be safer putting their computing resources in the expert hands of one of the major cloud providers. “Companies like Google and Amazon and Microsoft have hundreds of people devoted to security,” he says. “How many do you have?”

Further Reading

Christodorescu, M., Sailer, R., Schales, D., Sgandurra, D., and Zamboni, D.
Cloud security is not (just) virtualization security, *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, Chicago, IL, Nov. 13, 2009.

Gentry, C.
Fully homomorphic encryption using ideal lattices, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, Bethesda, MD, May 31–June 2, 2009.

Kamara, S. and Lauter, K.
Cryptographic cloud storage, *Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization*, Tenerife, Canary Islands, Spain, January 25–28, 2010.

Ristanpart, T., Tromer, E., Sacham, H., and Savage, S.
Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, *Proceedings of the 16th ACM Conference on Computer and Communications Security*, Chicago, IL, Nov. 9–13, 2009.

Shi, E., Bethencourt, J., Chan, T-H., Song, D., and Perrig, A.
Multi-dimensional range query over encrypted data, *Computer Science Technical Report CMU-CS-06-135R*, Carnegie Mellon University, March 2007.

Gary Anthes is a technology writer and editor based in Arlington, VA.

Distributed Computing

Math at Web Speed

“Many hands make light work,” goes the old adage. Now there’s data to prove it.

In recent weeks, both Yahoo! and Google have announced the results of separate mathematical experiments that demonstrate the computational power of large clusters of networked PCs.

At Yahoo!, a team led by researcher Tsz-Wo Sze broke the world record for calculating the digits of pi, crunching the famously irrational number to the two-quadrillionth bit by stitching together more than 1,000 computers to complete the calculation over a 23-day period. The researchers estimate that a typical computer would have taken at least 500 years to carry out the same operation.

Another group of researchers recently took advantage of Google’s distributed computing infrastructure to tackle another famously thorny computational challenge: Rubik’s Cube. The team developed an algorithm capable of solving any Rubik’s Cube configuration in 20 moves or less, resolving a conundrum that has puzzled mathematicians for three decades. The computers simulated all 43 quintillion possible combinations of the cube in just a few weeks, a task the researchers estimate would have taken a single computer 35 years.

Google has yet to release the details of its technical solution, but it probably bears some resemblance to the approach used at Yahoo!, where the team used Apache Hadoop, open-source software originally developed at Google (and later developed extensively by Yahoo!) that allows developers to stitch together thousands of computers over the network into a powerful cloud computer.

“We believe that our Hadoop clusters are already more powerful than many other supercomputers,” says Sze, who conceived of the project as part of an internal Yahoo! contest to demonstrate the capabilities of Hadoop.

In both cases, the mathematical problems proved particularly well-suited to distributed computing because the calculations can be parceled out over the network into much smaller operations, capable of running on a standard-issue PC. Making light work indeed.

—*Alex Wright*

Society | DOI:10.1145/1839676.1839684

Leah Hoffmann

Career Opportunities

What are the job prospects for today's—and tomorrow's—graduates?

LAST FALL, JIM Wordelman found himself in an enviable position. At a time when unemployment for recent U.S. college graduates was at the highest level since 1983, Wordelman, a senior at the University of Illinois at Urbana-Champaign's (UIUC's) Department of Computer Science, had several job offers from companies that wanted to hire him the following summer. Eventually, he took a job with Microsoft as a developer on its Internet Explorer team. "I did work for the company before and loved it," Wordelman explains. The job also put him in a position to follow his greatest passion: accessibility.

If recent data is any indication, Wordelman's case is not unique among computer science graduates in the U.S. (the job prospects for graduates in the United Kingdom, China, and India are discussed later). In fact, his fellow UIUC CS graduates received an average of 2.4 job offers this year. The mean starting salary: $68,650. "Our undergrads have had no trouble getting positions," says Rob Rutenbar, the department head. "Most of them are doing things like software development. Some launch entrepreneurial ventures."

At Carnegie Mellon University (CMU), the job outlook is equally rosy: 95% of this year's CS students had jobs waiting for them upon graduation. "Companies may be a little choosier, but they are still hiring," says Susanne Hambrusch, a computer science professor at Purdue University, where graduates enjoyed mean starting salaries of $66,875 last year.

According to projections from the U.S. Bureau of Labor Statistics (BLS), computing will be one of the fastest-growing job markets through 2018. Employment of software engineers, computer scientists, and network, database, and systems administrators is expected to grow between 24%–32% through 2018. They account for 71% of new jobs among the STEM (science, technology, engineering, and mathematics) fields. For a discipline that is still struggling with the public perception that its jobs are migrating offshore, such career predictions offer an important counterpoint.

Computer science graduates at Carnegie Mellon University, shown above, and other schools often have jobs waiting for them upon graduation.

PHOTOGRAPH BY DANIEL MIYAMOTO

Of the new jobs, according to BLS projections, 27% will be in software engineering, 21% in computing networking, and 10% in systems analysis. Software engineering alone is expected to add nearly 300,000 jobs in the next eight years.

Computer programmers will fare less well, with a projected decline in employment of 3% through 2018. The BLS cites advances in programming tools, as well as offshore outsourcing, as contributing factors to this decline. Nonetheless, the federal agency predicts employers will continue to need some local programmers, especially ones with strong technical skills. And many companies, having discovered that outsourcing is more challenging to manage than anticipated, are turning to domestic outsourcing to complete their programming projects, which is a trend the BLS expects to continue.

"The BLS projections are pretty compelling," says Peter Harsha, director of government affairs at the Computing Research Association (CRA). "We're optimistic."

College students seem to have picked up on that optimism, and are returning to the field after a steep six-year decline caused by the dot-com crash. According to the Taulbee Survey, an annual CRA study that gathers data for North American computer science and computer engineering programs, the number of computer science majors rose 8.1% in 2008 and another 5.5% in 2009. "It's a cautious uptrend," says Hambrusch. At some schools, the surge in interest is even more pronounced: applications to UIUC's CS program were up by 26% this year and increased by 32% at CMU.

The troubled economy has played a role in the uptick. Though the computing industry experienced a wave of layoffs at the height of the recession, it has been hit less hard than other sectors, and employment was up by an estimated 5% in the second quarter of 2010.

The Coolness Factor

According to a recent study conducted by the National Association of Colleges and Employers, the average salary for this year's crop of computer science grads stands at $61,112. And while it's too early to say for sure, some industry watchers predict an influx of students who might otherwise have majored in finance. Harsha, for example, cites David E. Shaw, a computer scientist turned hedge fund manager who made a fortune in quantitative trading, then returned to scientific research: "He's a model for a certain group."

There is also a coolness factor among a generation of students who grew up with computers and are deeply engaged with technologies like cellphones, Facebook and other social media, and the latest electronic devices from Apple and

other hardware companies. “For every popular trend in computing there's a spike in interest,” says Harsha, citing a similar boom-and-bust cycle that happened with the rise of the personal computer during the mid-1980s. Also, Harsha says, students may finally have realized that the stereotype of computing as a lonely career in which you sit in a cube and write code is not true.

“We all owe a non-trivial debt to companies like Google and Apple, who do cool work on cool products and don't look like your stereotypical guys in flannel suits,” says Lenny Pitt, a UIUC computer science professor.

Mark Stehlik, assistant dean for undergraduate education at CMU's School of Computer Science, has a different historical comparison: the space program of the 1960s, which fueled the imaginations and ambitions of a generation of schoolchildren. “There was such an enterprise built around it,” he recalls. Of course, to go to the moon, you had to be a rocket scientist. “And what if rocket science wasn't your thing?” Computer science majors, on the other hand, have a variety of career options to choose from once they graduate. “You can do software development across such a wide range of sectors,” notes Stehlik, as nowadays almost every industry has computing needs.

In spite of recent gains, the supply of CS graduates is still dwarfed by the projected number of jobs. According to the BLS projections, there will be more than twice as many new computing jobs per annum in the next eight years than the current level of 50,000 computing graduates will be able to fill. Nor can computer science departments, many of whom had trouble dealing with the influx of students in the late 1990s, expand as quickly as companies and universities might like. “We currently have about 775 undergrads, and we can add another couple hundred without a problem,” says UIUC's Rutenbar. “But we need to do some soul searching if we want to grow larger than that.”

Software engineering alone is expected to add nearly 300,000 jobs in the U.S. in the next eight years.

The International Outlook

The job prospects for computer science graduates in the United Kingdom, China, and India vary widely as does each country's educational and economic situations.

According to the United Kingdom's Higher Education Statistics Agency (HESA), 17% of 2009's CS graduates were unemployed six months later—more than any other discipline. Industry watchers caution that the figure should be taken with a grain of salt, however, since the category includes students who studied softer subjects like human-computer interaction and information development as well as traditional CS majors. “Higher-level things like software and systems design are a different picture,” says Bill Mitchell, director of the British Computer Society. “They are very much recruiting these types of people.”

Research institutions like the University of Southampton, which placed 94% of its computer science graduates in 2009, echo Mitchell's sentiment. “Companies still need people with really good skills, who have been exposed to different languages and platforms, who are confident and can code,” says Joyce Lewis, communications manager for the University of Southampton's School of Electronics and Computer Science. And while the University of Southampton and other members of the Russell Group—an association of 20 universities that's often referred to as the U.K.'s Ivy League—have no trouble filling spaces in their computer science programs, educators are nonetheless concerned by a massive nationwide drop in interest in the field. “Enrollment has dropped by nearly 60% over the past eight years,” says Mitchell, who is working to reform the national IT curriculum and reverse the trend. “Companies tell us they have to bring people in from Silicon Valley.”

Recent computer science graduates in China are also struggling with a demanding job market. According to a study conducted by the MyCOS Institute, a Beijing-based think tank, computer science, English, and law have topped a list of majors with the most unemployed graduates for the past three years. In 2009, the most recent year for which data is available, computer science was second only to English in the number of unemployed graduates.

Here, too, the figures underlie a more complicated picture. Thanks to governmental encouragement, the number of university graduates in China has risen dramatically during the past 10 years. In 2008, more than six million students graduated nationwide; in 2002, the total number was below 1.5 million. Such increases, education experts contend, were not matched with employment prospects, particularly in technical fields, where market needs are highly specialized. Therefore, students must work hard to distinguish themselves from a glut of applicants. Often, that means earning a graduate degree. "Companies get a lot of applicants, and to make it easier, some use the degree as a filter," says Xiaoge Wang, an associate professor in the department of computer science and technology at Tsinghua University. At Tsinghua, 83% of 2009's computer science graduates enrolled in graduate programs at home or abroad, up from 78% in 2008 and 66% in 2007. "Our students would like to go to companies like Microsoft or IBM, which require a Ph.D. or a master's," says Wang.

In India, the IT industry is doing well after a slowdown brought on by the global downturn. According to the country's National Association of Software and Services Companies, the IT services sector, still the dominant source of computing jobs, grew nearly 16.5% in 2009, and software exports are expected to increase by 14.4% in the current fiscal year. Job placements at the country's top engineering schools are robust, with many students receiving multiple offers upon graduation. One concern, however, is the growing lack of educators to teach the next generation of software engineers—a shortage of up to 70,000 teachers, according to some estimates. University pay scales are low compared to the private sector, and few students pursue the advanced degrees that would qualify them for university positions. As a report published in the *International Journal of Engineering Studies* explained, "The teaching load of professors in the top research-intensive schools has increased, and talented potential research students are being attracted by high-paying private-sector jobs, or by research opportunities at better-funded institutions abroad." Those students who do pursue advanced degrees, according to the study's authors, often do so to improve their market value in the job market.

One concern in India is the growing lack of educators to teach the next generation of software engineers—a shortage of up to 70,000 teachers, according to some estimates.

Further Reading

National Association of Software and Services Companies
The IT-BPO Sector in India: Strategic Review 2009, http://www.nasscom.in/Nasscom/templates/NormalPage.aspx?id=55816.

Solanki, K., Dalal, S., and Bharti, V.
Software engineering education and research in India: a survey, *International Journal of Engineering Studies 1*,3, 2009.

U.S. Bureau of Labor Statistics
Occupational Outlook Handbook, 2010-11 edition, http://www.bls.gov/oco/.

Universities & Colleges Admissions Service
Unistats From Universities and Colleges in the U.K., http://unistats.direct.gov.uk/retrieveColleges_en.do.

Zhang, M. and Virginia M.L.
Undergraduate computer science education in China, *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, March 10–13, 2010, Milwaukee, WI.

Leah Hoffmann is a Brooklyn, NY-based technology writer.

Milestones

David Kuck Wins Ken Kennedy Award

David Kuck, an Intel Fellow, is the recipient of the second annual ACM-IEEE Computer Society Ken Kennedy Award for his four decades of contributions to compiler technology and parallel computing, which have improved the cost-effectiveness of multiprocessor computing.

In an email interview, Kuck discussed his current research. "I'm working on hardware/software codesign at a very comprehensive level, considering system cost, performance, and operating cost (in terms of energy), as well as applications sensitivity. I'm working on theory and tools to support codesign. I introduced a computational capacity model in the 1970s and pushed it much further in the past few years. Measured bandwidth and capacity (bandwidth used) for a set of architectural nodes for each phase in a computation provide capacity ratios that are invariant across hardware changes. This leads to very fast simulations of new machines, solving linear programming and related problems to satisfy design goals."

Asked about the next important innovation with compilers, Kuck said, "Compiler optimization transformations are well developed, but where and how to apply them is still a mystery. I believe that building a large repository of codelets could remove much of the mystery related to sequential, vector, parallel, and energy-aware compilation. Many trade-offs must be made, so a pre-analyzed repository would allow phases, and sequences of them, to be matched to codelets for optimal compilation. This is a combined static and dynamic approach to compilation."

The Ken Kennedy Award recognizes substantial contributions to programmability and productivity in computing and substantial community service or mentoring contributions, and includes a $5,000 honorarium.

—*Jack Rosenberger*

Emerging Technology | DOI:10.1145/1839676.1839704

Neil Savage

Wide Open Spaces

The U.S. Federal Communications Commission's decision to open frequencies in the broadcast spectrum could enable broadband networks in rural areas, permit smart electric grids, and more.

U.S. Federal Communications Commission Chairman Julius Genachowski.

PHOTOGRAPH BY J.D. LASICA/SOCIALMEDIA.BIZ

NEWLY AVAILABLE FREQUENCIES in the broadcast spectrum should increase Internet access and spur innovation, U.S. Federal Communications Commission (FCC) Chairman Julius Genachowski said in opening those frequencies to unlicensed commercial use.

When TV broadcasters switched to narrower digital channels in 2009, they freed up frequencies below 700 megahertz, the so-called "white spaces" between channels. Signals at these frequencies can travel about three times as far as the traditional Wi-Fi frequency of 2.4 gigahertz, and easily penetrate buildings and other physical obstacles.

The new FCC rules create room for two classes of devices: fixed, high-power ones that transmit at up to 4 watts and portable, low-power devices limited to 100 milliwatts. Harold Feld, legal director of Public Knowledge, a Washington, D.C., public interest group, says thousands of small wireless Internet service providers in rural areas, underserved by broadband connections, will be quick to take advantage of the range and penetration, to achieve good coverage with fewer towers.

But that will not happen tomorrow. "It's going to take a minimum of 18 months to get even the most basic devices approved and out there," Feld says.

Neeraj Srivastava, vice president of marketing at Spectrum Bridge, a wireless networking company in Lake Mary, FL, that works with white spaces, expects the first enhanced Wi-Fi systems, using fixed devices, could start appearing in the first quarter of 2011. Under experimental licenses from the FCC, Srivastava says Spectrum Bridge has run several pilot programs that demonstrate the near-term uses.

One project provided the connectivity to deploy smart grid power monitors across the electrical system in Plymouth, CA. By placing white-space radios at substations, Spectrum Bridge created a wireless network to keep track of power usage and simultaneously supplied the town's residents with wireless broadband. A project in Wilmington, NC, provided wireless links to inaccessible water-quality monitors and traffic-monitoring bridge cameras. At a hospital in Logan, OH, where concrete walls blocked Wi-Fi and the building's structure made cabling difficult, Spectrum Bridge created a wireless network to monitor patients, share data, and access security cameras. They also brought broadband access to rural Claudville, VA.

The 4-watt applications can use existing standards; Spectrum Bridge used a modified WiMAX radio in the Logan hospital, for example. Meanwhile, IEEE is working on a standard for the low-power devices, comparable to its 802.11 standard for Wi-Fi. That will determine the design of chips for the smartphones and laptops that will use them, Srivastava says, so it may be more than two years before the appearance of the first low-power applications.

Feld believes that, as the utility of white spaces becomes apparent, the FCC will look for more spectrum to release. "As people think about how you could have radios that are more cognitive, more sensitive to their spectrum environments and act accordingly, people are going to want to see that technology become more widely available," he says.

Srivastava notes that when the FCC made the spectrum now used by Wi-Fi available in 1985, the popular wireless applications were garage door openers and baby monitors. Nobody had thought of Bluetooth, Wi-Fi, or home networking. "There's always a third class of devices, and those are the unknown ones. Those probably have the most promise, but I can't tell you what they are," Srivastava says.

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

DOI:10.1145/1839676.1839685

Pablo J. Boczkowski

Economic and Business Dimensions

The Divergent Online News Preferences of Journalists and Readers

Reading between the lines of the thematic gap between the supply and demand of online news.

The political body, like the biological one, needs the right combination of nutrients to function adequately. One such key ingredient is news about public affairs that is necessary to inform political deliberation and encourage educated participation among the citizenry. In most liberal democratic societies, this news is largely provided by elite news organizations in print, broadcast, and online media. But, at least on the Web, while these organizations have supplied this kind of news in considerable quantities, the demand for news among online readers has gravitated toward other kinds of content also provided on these sites, such as information about weather, sports, crime, gossip, and entertainment. That frames an interesting dilemma for online news, and also for society as a whole.

Measuring Divergence in Online News

I did not look for the dilemma in online news. Once I found it in one place, however, I went looking for it elsewhere and found it everywhere.

For a book on imitation in online and offline news,[1] I measured the amount of readers' news choices and the thematic composition of their choices. I found a large, double-digit difference between supply (preferences of journalists) and demand (preferences of readers).

More precisely, I calculated the degree of similarity in the events covered in the stories the three leading online news sites in Argentina considered the most important ones in any given news cycle. This meant collecting each homepage's first 10 stories counting from left to right and from the top down in a grid-like manner. The analysis sought to determine whether an event covered in one site was also covered in at least one of the others.

The high level of similarity in news products could not be explained by patterns in the nature of demand. If anything, consumers seemed to want more differentiated news products and also news marked by a different thematic composition than what was offered to them. These sites shared 52% of the events presented on their respective top 10 stories. Furthermore, almost 59% of these stories shared by more than one site dealt with political, business, economic, and international matters.

By contrast, the most popular stories among the readers of these sites were quite dissimilar: only 36% of the hard news in the top 10 most viewed stories on one site were also among the top 10 most viewed stories in at least

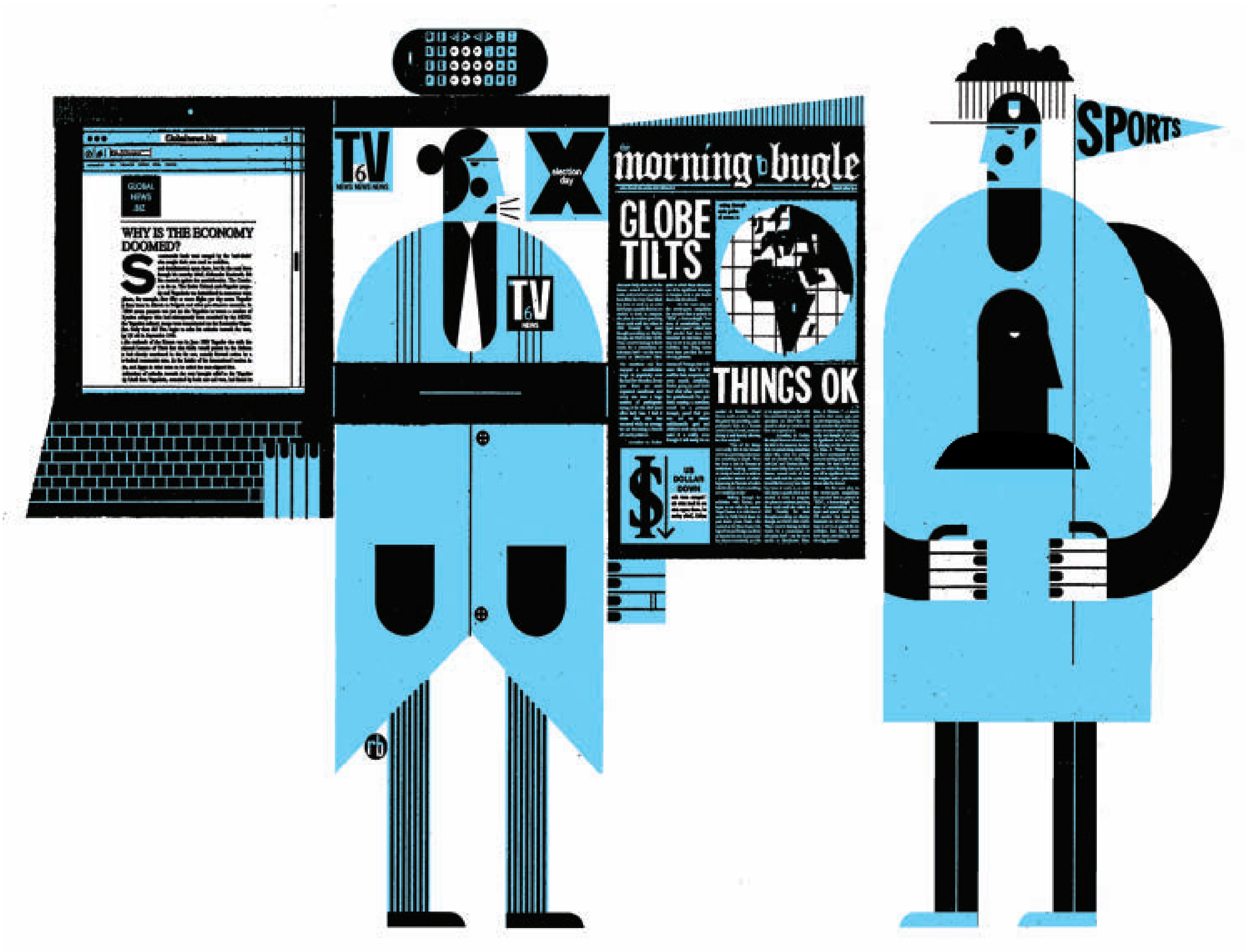


ILLUSTRATION BY RAYMOND BIESINGER

one of the other two sites, and only 32% of these stories popular in more than one site dealt with politics, business, economic, and international matters. This amounted to a 16 percentage point gap between the levels of similarity of the top news choices of journalists and consumers, and a 27 percentage point gap between the thematic composition of these similar choices.

That initial study begged a follow-up question: do similar patterns arise elsewhere? A second study showed that the mismatch also applies to the leading, elite media in the U.S. My collaborator Limor Peer of Yale University and I conducted a study that compared the concurrent news choices of journalists and consumers of four leading, U.S.-based sites: CNN, Yahoo News, *Chicago Tribune*, and the now-defunct *Seattle Post-Intelligencer*.[3] We chose these sites to represent broadcast, online-only, and newspaper parent companies, and also different geographic orientations. The first two sites were national-global, while the second two were local.

In all cases journalists selected more news about politics, economics, business, and international matters than readers, who, in turn, were more interested in topics such as sports, weather, entertainment, and crime. On each data collection day, research assistants gathered information on the top 10 stories selected by journalists and by consumers, respectively, as operationalized in the previous study. A comparison of the thematic composition of journalists' and consumers' top story preferences per site revealed 13 percentage point gaps on *Seattle Post-Intelligencer* and Yahoo, 14 percentage point gaps on CNN, and 17 percentage point gaps on *Chicago Tribune*.

We worried that supply and demand are interdependent: journalists might prioritize certain stories because they perceive that consumers could find them appealing, and consumers might click more often on stories that receive major treatment by journalists. To try to get past these interdependencies, we conducted a second analysis on each site that excluded the stories that were selected by both journalists and consumers. Focusing on the stories that journalists chose irrespective of their level of popularity among consumers, and those that consumers chose even though they were not prominently displayed on the homepage, would give us a stronger measure of each group's independent preference.

The results suggest that independent from the influence of each other, the choices of journalists and consumers would follow strikingly divergent trajectories. The gaps between the thematic composition of the top stories selected by journalists and consumers increased by an average of 19 percentage points.

Does the divergence between supply and demand depend on geography or ideology? Results from a third study, undertaken in collaboration with Northwestern University graduate students Eugenia Mitchelstein and Martin Walter, show that the mismatch is a widespread phenomenon that cuts across media from countries and regions with disparate histories, cultural makeup, and ideological positions.[2]

This study deployed the design of the second study to examine 11 sites in Western Europe and Latin America: *The Guardian* and *The Times* in the United Kingdom; *El País* and *El Mundo* in Spain; *Die Welt* and *Der Tagesspiegel* in Germany; *La Reforma* and *El Universal* in Mexico; *Clarín* and *La Nación* in Argentina; and *Folha de Sao Paulo* in Brazil (there was only one Brazilian site because data from a second comparable site was not publicly available). All the sites were the online presence of leading generalist, mainstream, and elite newspapers with national reach in their respective countries. Moreover, in the five countries from which two news sites were sampled, the pairs had somewhat divergent ideological outlooks—either conservative and centrist or conservative and liberal.

Once again, there was a sizable thematic gap between the supply and demand of online news, with the journalists leaning more toward stories about politics, business, economics, and international matters than readers. The differences between the top news choices of journalists and consumers ranged from 30 percentage points in *The Guardian* to 9 percentage points in *Clarín*, with an average of 19 percentage points. In addition, there were no major patterns of variance in this gap by geographical region or ideological preference. First, journalists choose news about politics, economics, business, and international matters 20 percentage points more often than readers in Western Europe and 19 percentage points more often in Latin America. Second, while on conservative sites the thematic difference between journalists' and consumers' choices was 21 percentage points, on centrist/liberal sites this difference was 19 percentage points.

It is unlikely that the mismatch between supply and demand of news in the elite media began with the Web.

The Future of Media and Democracy

It is unlikely that the mismatch between supply and demand of news in the elite media began with the Web. As the noted sociologist and former journalist Robert Park wrote many decades ago, "The things which most of us would like to publish are not the things that most of us want to read. We may be eager to get into print what is, or seems to be, edifying, but we want to read what is interesting." But the strong market position of these elite media meant that because advertisers had to go through them to reach potential consumers, journalists could get away with fulfilling their sense of civic duty by disseminating "edifying" news despite their limited appeal among the general public.

But in the highly competitive contemporary media environment, few news organizations enjoy the kind of natural monopoly or oligopoly position that newspapers and television networks had in the past. Perhaps, none do. Of all media markets, the Web is the most competitive one because of low geographic and distribution barriers and the very high number of players.

In addition, the Web enables organizations to automatically track the number of clicks garnered by each story. This has meant that personnel at elite online news sites are deeply aware of the extent to which supply and demand don't meet. They must confront the dilemma introduced at the beginning of this column on a daily basis.

What should they do? If they stay the course and the nature of consumer preferences does not change (and there is no reason to suspect it might), the mismatch between supply and demand will further erode their economic sustainability. If they change course and give consumers more of what they want, they will likely pay the price of becoming a different kind of news organization and having to compete in an already crowded space of "populist media." Either way, the future does not bode well for them.

The potential implications of these trends for democracy are not encouraging either. As noted earlier, the leading, elite news organizations are major contributors of the kind of information about political, economic, and international matters that is essential for well-informed democratic deliberation and participation. This information is much more difficult to find in other sources such as tabloid media and even blogs (which largely amplify the information that originates in elite news organizations).

Society's appetite for information might get satiated by news about weather, sports, crime, gossip, and entertainment. But the contributions of these symbolic nutrients for the healthy functioning of the body politic will surely be lacking. If, in part, we are what we eat, we should be aware that we also are the news that we consume. And when the supply and demand of online news does not meet, it is not just elite media organizations that might suffer, but also all of us.

References

1. Boczkowski, P. *News at Work: Imitation in an Age of Information Abundance*. University of Chicago Press, Chicago, IL, 2010.
2. Boczkowski, P., Mitchelstein, E., and Walter, M. (in press). Convergence across divergence: Understanding the gap in the online news choices of journalists and consumers in Western Europe and Latin America. *Communication Research*.
3. Boczkowski, P. and Peer, L. (in press). The choice gap: The divergent online news preferences of journalists and consumers. *Journal of Communication*.

Pablo J. Boczkowski (pjb9@northwestern.edu) is a professor in the Department of Communication Studies at Northwestern University and also, during the 2010–2011 academic year, a visiting scholar at the University of Chicago Booth School of Business. He is the author of *Digitizing the News: Innovation in Online Newspapers* (MIT Press, 2004) and *News at Work: Imitation in an Age of Information Abundance* (University of Chicago Press, 2010).

I would like to thank Shane Greenstein for the invitation to write this column and Shane and Eugenia Mitchelstein for feedback on earlier versions.

DOI:10.1145/1839676.1839686

Stephen Cooper, Lance C. Pérez, and Daphne Rainey

Education

K–12 Computational Learning

Enhancing student learning and understanding by combining theories of learning with the computer's unique attributes.

In "Computational Thinking,"[14] Jeannette Wing struck a chord that has resonated strongly (generating positive as well as negative responses) with many computer scientists and non-computer scientists. Wing has subsequently defined computational thinking as the process of abstraction,[15] guided by various engineering-type concerns including efficiency, correctness, and several software engineering "-ilities" (maintainability, usability, modifiability, and so forth). Some have interpreted computational thinking as an attempt to capture the set of computer science skills essential to virtually every person in a technological society, while others view it as a new description of the fundamental discipline that represents computer science and its intersection with other fields. The National Academies report[1] captures both of these views, as well as presenting others.

Berkeley public elementary school students on a field trip learn how computers enable science.

PHOTOGRAPH BY ROY KALTSCHMIDT, LAWRENCE BERKELEY NAT'L LAB

While we can live with such definitions/descriptions in the higher education arena, we struggle with these notions of computational thinking in the K–12 arena (note that we primarily consider K–12 education within the U.S.). Several concerns spring to mind:

1. Computer science does not appear within the core topics covered in high school. We would have a tough time justifying a computer science course, even the "great ideas" AP Principles course (being developed as part of Denning[4]) replacing Algebra 2, Biology, or American Government. K–12 education is a zero-sum game. If one wishes to add a course, one must also propose a course to be removed.

2. Even as an elective topic, computer science tends to be disproportionately available to those wealthy suburban schools. Margolis et al.[6] explore this situation in depth within the urban Los Angeles school district.

3. Too few K–12 computing teachers are available to implement a national-scale computing requirement. CUNY's ambitious 10,000 teacher project[2] will not produce sufficient numbers of computing teachers required to instruct *all* schoolchildren in the U.S. It would not even get one qualified teacher into each of the nation's 30,000+ high schools (see http://nces.ed.gov/programs/digest/d09/tables/dt09_086.asp).

4. It is not clear to us how teachers in other K–12 subjects would take advantage of school children who had been trained in computational thinking.

5. The most common definitions of computational thinking are confusing when explained to non-computer scientists. And many K–12 computing teachers are not computer scientists.

We find the above definitions of computational thinking not especially useful when considered in the context of K–12 education, and, more specifically, K–12 science, technology, engineering, and mathematics (STEM) education. We propose an alternative to the common definition of computational thinking we believe is appropriate for operationalization in K–12 education and consider its broader implications.

A K–12 View of Computational Thinking

We have struggled with how computational thinking might be different from mathematical thinking, algorithmic thinking, quantitative reasoning, design thinking, and several other models of math, science, and even engineering to critical thinking and problem solving. It was after struggling with the latest type of thinking that we realized that perhaps even the term "computational thinking" was misleading (from a K–12 perspective), and we were approaching the definition incorrectly. Rather than considering computational thinking as a part of the process for problem solving, we instead developed a model of computational learning that emphasizes the central role that a computer (and possibly its abstraction) can play in enhancing the learning process and improving achievement of K–12 students in STEM and other courses. The figure here depicts our current working model of computational learning. It should be noted that this model is explicit in its use of a computer and specifically excludes non-cognitive uses of technology (Powerpoint, wikis, blogs, clickers, and so forth).

Similar to Wing's original vision of computational thinking, we see computational learning as an iterative and interactive process between the human (the K–12 student in our case) and the computer (or, in a more theoretical construct, a model of computation). We also make explicit the two consequences of the human cognitive process, namely, the capacity for abstraction and for problem formulation, and two strengths of the computer, namely, their ability to present complex data sets, often visually, and their capacity for storing factual and relational knowledge. These four elements frame and establish the boundaries of the iterative interaction between the human being and the computer. Note that the accompanying figure does not explicitly include a teacher, not because we believe teachers are unnecessary, but rather because the role of the teacher in this model is complex and requires further investigation.

In developing this model, we observed that it includes other extant models in scientific learning and inquiry. For example, one can view computational science as the interaction between the human and the computer that is contained within the box where a human being formulates a problem and provides a representation suitable for a computer. The computer then acts on this representation and returns the results of these actions to the human being through, for example, a visual representation. Computational learning expands this interaction by allowing the computer to add foundational knowledge, not just data, unknown to the human and by having the results of the computer's actions represented in a form compatible with the human's current capacity for abstraction. In the more interesting instances of computational learning, both of these processes are likely to be adaptive and personalized to the individual.

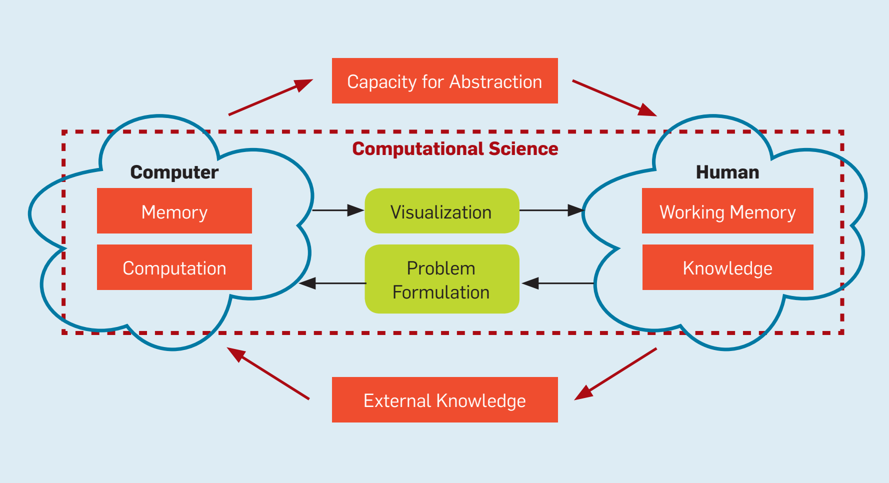
We make several observations about computational learning:

- Computational learning is iterative, requiring interaction between the computer and the human.
- In computational learning, the computer can compensate for a human's lack of factual and relational knowledge and mathematical and scientific sophistication.
- The computer's ability to quickly compute multiple examples and present them via a modality appropriate to a human's current development stage and level of meta-cognitive awareness can leverage the human's inherent, though perhaps not fully conscious, capacity for abstraction.

This model for computational learning differs significantly from other proposed notions of computational thinking. For example, algorithmic thinking does not require a computer and mathematical thinking is almost solely dependent on the human's formalization capacity for abstraction.[2]

To better understand how our proposed computational learning model can be operationalized, we present two examples: one from middle school computing and one from high school biology.

A view of computational learning.



Digitizing Data

Cutler and Hutton[3] modified a CS Unplugged activity on image representation (see http://csunplugged.org/sites/default/files/activity_pdfs_full/unplugged-02-image_representation.pdf) to enable middle school students to work interactively with a computer program as they learn about how computers digitize images. The purpose of these activities is to help students to understand what it means for a computer to digitally represent an image. More importantly, the students learn to move from concrete representations of images to more abstract representations of those images (as a digital representation), and from representation of images in 2D to representing objects in 3D. And this ability to abstract is important across all STEM disciplines.

Students work interactively with the computer program, receiving feedback to their attempts at digitizing their data. Over the series of lessons, they develop an initial ability to abstract away from the physical representation of an image to its digital representation. They are then further able to develop their ability to abstract as they move from 2D images to 3D objects. Finally, students develop a further ability to abstract. As part of the creation of a single 3D object, say a chair, the students are then challenged to place many chairs in a room. They need to be able to recognize that representing a 3D chair consists of two parts: the relative coordinates of each of the parts of the chair, and the absolute location of one part (say the bottom-right corner of the front-right leg).

Evolutionary Biology

The second example involves the teaching of evolution using computational learning. Our vision is of a 3D visualization system that could simulate evolution. A student could specify an organism, with primitive appendages (arms, legs, joints, and other attributes) to accomplish locomotion. Then, by providing an environment, the student could run the simulation to watch how the organism's ability to move evolves over time as a function of its current locomotion capability coupled with the impact of that organism's environment. Students could change the appendages and/or the environment to observe how such changes lead to a difference in the organism's evolution over time. In computer science terms, this example is similar to passing a program and an initial state as input to a Universal Turing Machine.

Such a simulation allows the student to work interactively with the computer program. The student learns both from the impact of the changes she makes to the initial configuration of the organism and to the initial environment (which will lead to the organism evolving the ability to move differently) as well as by the ability to observe the simulation/visualization as it is running. In science, researchers have found that visualization is central to increasing conceptual understanding and prompting the formation of dynamic mental models of particulate matter and processes (see [5,7,9,12]). Visualization and computer interaction through animation allow students to engage more in the cognitive process, and to select and organize more relevant information for problem-solving.[7] Computer animations incorporated into interactive simulations offer the user a chance to manipulate variables to observe the effect on the system's behavior (see [9,10,13]).

This model for computational learning differs significantly from other proposed notions of computational thinking.

While we know of no tool that provides the exact support/simulation we are describing, there are several available visualization systems that can simulate/model the world. Two of these systems have helped to shape our vision of the above-mentioned simulation: The 3D visualization system, Framsticks (www.framsticks.com) can be used for modeling evolution, and the 2D simulation system, NetLogo (http://ccl.northwestern.edu/netlogo/) has many available pre-built simulations, including those that model evolution albeit in a different manner than what we describe.

Conclusion

Most papers we've seen on computational thinking represent attempts at repackaging computing science concepts, especially in the form of algorithmic thinking and introductory programming, sometimes in other domains. Though this may be useful in some contexts, it is unlikely such a simple approach will have significant impact on student learning—of computer science or other disciplines—in the K–12 setting. The proposed model of computational learning combines theories of learning with the computer's superiority in dealing with complexity and variability and its ability to present results using modalities that appeal to the learner in order to enhance student learning and understanding. We believe that computational learning can be framed within various theories of learning, where the computer plays a similar role as Vygotsky's More Knowledgeable Other (though the computer obviously cannot think at a higher level than the student),[11] or even fits within Newell and Simon's Information Processing Theory framework.[8] Our hope is that by considering our model of computational learning, we can better educate and prepare teachers to benefit from computing in and outside the classroom, and that approaches and computing tools can be identified and built to improve K–12 student STEM learning.

References

1. Committee for the Workshops on Computational Thinking, National Research Council. *Report of a Workshop on the Scope and Nature of Computational Thinking.* National Academies Press, Washington, D.C., 2010.
2. Cuny, J. Finding 10,000 teachers. CSTA Voice, 5, 6 (2010), 1–2; http://www.csta.acm.org/Communications/sub/CSTAVoice_Files/csta_voice_01_2010.pdf
3. Cutler, R. and Hutton, M. Digitizing data: Computational thinking for middle school students through computer graphics. In *Proceedings of the 31st Annual Conference of the European Association for Computer Graphics EG 2010—Education Papers.* (Norrköping, Sweden, May 2010), 17–24.
4. Denning, P. Beyond computational thinking. *Commun. ACM 52*, 6 (June 2009), 28–30.
5. Gilbert, J.K., Justi, R., and Aksela, M. The visualization of models: A metacognitive competence in the learning of chemistry. Paper presented at the 4th Annual Meeting of the European Science Education Research Association, Noordwijkerhout, The Netherlands, 2003.
6. Margolis, J. et al. *Stuck in the Shallow End: Education, Race, and Computing.* The MIT Press, 2008.
7. National Science Foundation. Molecular visualization in science education. Report from the molecular visualization in science education workshop. NCSA access center, National Science Foundation, Arlington, VA, 2001.
8. Newell, A., and Simon, H.A. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ, 1972.
9. Rotbain, Y., Marbach-Ad, G., and Stavy, R. Using a computer animation to teach high school molecular biology. *Journal of Science Education and Technology 17* (2008), 49–58.
10. Sewell R., Stevens, R., and Lewis, D. Multimedia computer technology as a tool for teaching and assessment of biological science. *Journal of Biological Education 29* (1995), 27–32.
11. Vygotsky, L.S. *Mind and Society: The Development of Higher Mental Processes.* Harvard University Press, Cambridge, MA, 1978.
12. Williamson V.M. and Abraham, M.R. The effects of computer animation on the particulate mental models of college chemistry students. *Journal of Research in Science Teaching 32* (1995), 522–534.
13. Windschitl, M.A. A practical guide for incorporating computer-based simulations into science instruction. *The American Biology Teacher 60* (1998), 92–97.
14. Wing, J.M. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.
15. Wing, J.M. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, 366 (2008), 3717–3725.

Stephen Cooper (coopersc@purdue.edu) is an associate professor (teaching) in the Computer Science Department at Stanford University.

Lance C. Pérez (lperez@unl.edu) is an associate professor in the Department of Electrical Engineering at the University of Nebraska-Lincoln where he also holds the position of Associate Vice Chancellor for Academic Affairs.

Daphne Rainey (raineyd4@gmail.com) is a biologist currently serving in a temporary assignment as a program director at NSF's Division of Undergraduate Education within its Education and Human Resources Directorate.

DOI:10.1145/1839676.1839687

Pamela Samuelson

Legally Speaking

Why Do Software Startups Patent (or Not)?

Assessing the controversial results of a recent empirical study of the role of intellectual property in software startups.

Two-thirds of the approximately 700 software entrepreneurs who participated in the 2008 Berkeley Patent Survey report that they neither have nor are seeking patents for innovations embodied in their products and services. These entrepreneurs rate patents as the least important mechanism among seven options for attaining competitive advantage. Even software startups that hold patents regard them as providing only a slight incentive to innovate.

These are three of the most striking findings from a recently published article, "High Technology Entrepreneurs and the Patent System: Results of the 2008 Berkeley Patent Survey."[1] After providing some background about the survey, I will discuss some key findings about how software startup firms use and are affected by the patent system.

While the three findings highlighted above might seem to support a software patent abolitionist position, it is significant that one-third of the software entrepreneur respondents reported having or seeking patents, and that they perceive patents to be important to persons or firms from whom they hope to obtain financing.

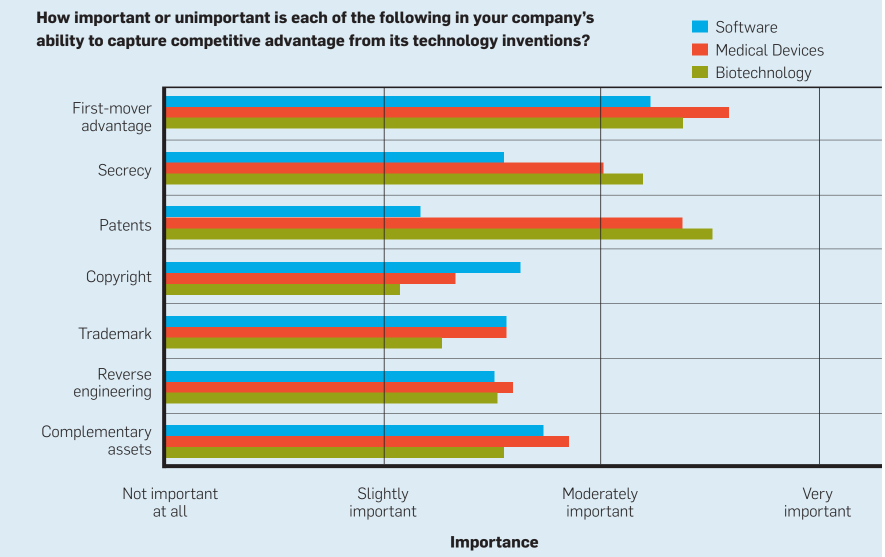
Some Background on the Survey

More than 1,300 high-technology entrepreneurs in the software, biotechnology, medical devices, and computer hardware fields completed the Berkeley Patent Survey. All of these firms were no more than 10 years old before the survey was conducted. We drew our sample from a general population of high-tech firms registered with Dun & Bradstreet (D&B) and from the VentureXpert (VX) database that has a rich data set on venture-backed startups. (Just over 500 of the survey software respondents were D&B firms; just under 200 respondents were VX firms.)

Eighty percent of the software respondents were either the CEOs or CTOs of their firms, and most had experience in previous startups. The average software firm had 58 employees, half of whom were engineers. Between 10%–15% of the software startup respondents among the D&B respondents were venture-backed firms. Among the software respondents, only 2% had experienced an initial public offering (IPO), while 9% had been acquired by another firm.

Our interest in conducting this survey arose because high-technology entrepreneurs have contributed significantly to economic growth in recent decades. They build firms that create new products, services, organizations, and opportunities for complementary economic activities. We were curious to know the extent to which high-tech

Measures of capturing "competitive advantage" from inventions.



startups were utilizing the patent system, as well as to learn their reasons for choosing to avail themselves of the patent system—or not.

The basic economic principle underlying the patent system is that technology innovations are often expensive, time-consuming, and risky to develop, although once developed, these innovations are often inexpensive and easy to copy; in the absence of intellectual property rights (IPRs), innovative high-tech firms may have insufficient incentives to invest in innovation insofar as they cannot recoup their research and development (R&D) expenses and justify further investments in innovation because of cheap copies that undermine the firms' recoupment strategy.

Although this economic principle applies to all companies, early-stage technology firms might, we conjectured, be more sensitive to IPRs than more mature firms. The former often lack various kinds of complementary assets (such as well-defined marketing channels and access to cheap credit) that the latter are more likely to enjoy. We decided it would be worthwhile to test this conjecture empirically. With generous funding from the Ewing Marion Kauffman Foundation, three colleagues and I designed and carried out the survey and have begun analyzing the results.

Why Startups Patent

The most important reasons for seeking patents, as reported by software executives who responded to the Berkeley Patent Survey, were these: to prevent competitors from copying the innovation (2.3 on a 4 point scale, where 2 was moderately important), to enhance the firms' reputation (2.2), and to secure investment and improve the likelihood of an IPO (1.96 and 1.97 respectively).

The importance of patents to investors was also evident from survey data showing striking differences in the rate of patenting among the VX and the D&B software companies.

Three-quarters of the D&B firms had no patents and were not seeking them. Because the D&B firms are, we believe, fairly typical of the population of software startup firms in the U.S., their responses may well be representative of patenting rates among software startups generally. It is, in fact, possible that the overall rate of software startup patenting is lower than this given that patent-holders may have been more likely than non-patent-holders to take time to fill out a Berkeley Patent Survey.

It is an article of faith among many IP lawyers that patents provide significant incentives for firms to engage in R&D and develop new products.

In striking contrast to the D&B respondents, over two-thirds of the VX software startup respondents in the sample, all venture-backed, had or were seeking patents. We cannot say why these venture-backed firms were more likely to seek patents than other firms. Perhaps venture capitalists (VCs) are urging firms they fund to seek patents; or VCs may be choosing to fund the development of software technologies that VCs think are more amenable to patenting.

Interestingly, the rate of patenting did not vary by the age of the firm (that is, older firms did not patent more than younger firms).

Why Forego Patenting?

The survey asked two questions about decisions to forego patenting: For the last innovation for which the firm chose not to seek a patent, what factors influenced this decision, and what was the most important factor in the decision?

The costs of obtaining and of enforcing patents emerged as the first and second most frequent explanation. Twenty-eight percent of the software startup executives reported that the costs of obtaining patents had been the most important factor in this decision, and 12% said that the costs of enforcing patents was the most important factor. (They reported that average cost of getting a software patent was just under $30,000.)

Calendar of Events

November 17–19
Asian Internet Engineering Conference,
Bangkok, Thailand,
Contact: Kanchanasut Kanchana,
Email: kk@cs.ait.ac.th

November 17–19
Advances in Computer Entertainment Technology Conference,
Taipei, Taiwan,
Contact: Duh Henry B.L.,
Email: eledbl@nus.edu.sg

November 22–23
Conference on Decision and Game Theory for Security,
Berlin, Germany,
Contact: Tansu Alpcan,
Email: alpcan@sec.t-labs.tu-berlin.de

November 22–24
The 17th ACM Symposium on Virtual Reality Software and Technology,
Hong Kong,
Contact: George Baciu,
Email: csgeorge@comp.polyu.edu.hk

December 1–3
9th International Conference on Mobile and Ubiquitous Multimedia,
Limassol, Cyprus,
Contact: Angelides Marios,
Email: marios.angelides@brunel.ac.uk

December 4–8
The 43rd Annual IEEE/ACM International Symposium on Microarchitecture,
Atlanta, GA,
Sponsored: SIGMICRO,
Contact: Sudhakar Yalamanchili,
Email: suhdha@ece.gatech.edu

December 5–8
Winter Simulation Conference,
Baltimore, Maryland,
Sponsored: SIGSIM ,
Contact: Joe Hugan,
Email: jhugan@gmail.com

December 12–13
Virtual Reality Continuum and its Applications in Industry,
Seoul, Republic of Korea,
Sponsored: SIGGRAPH ,
Contact: Hyunseung Yang,
Email: hsyang@cs.kaist.ac.kr

Ease of inventing around the innovation and satisfaction with secrecy also influenced software startup decisions not to seek patents, although only rarely were these factors considered the most important.

Intriguingly, more than 40% of the software respondents cited the unpatentability of the invention as a factor in decisions to forego patenting. Almost a quarter of them rated this as the most important factor. Indeed, unpatentability ranked just behind costs of obtaining patents as the most frequently cited "most important factor" for not seeking patents.

It is difficult to know what to make of the unpatentability finding. One explanation may be that the software respondents believed that patent standards of novelty, non-obviousness, and the like are so rigorous that their innovation might not have satisfied patent requirements. Yet, because the patentability of software innovations has been contentious for decades, it may also be that a significant number of these entrepreneurs have philosophical or practical objections to patents in their field.

How Important Are Patents to Competitive Advantage?

One of the most striking findings of our study is that software firms ranked patents dead last among seven strategies for attaining competitive advantage, as the accompanying figure shows. (The relative unimportance of patents for competitive advantage in the software field contrasts sharply with the perceived importance of patents in the biotech industry, where patents are ranked the most important means of attaining such advantage.)

As shown in the figure on page 30, software startups regard first-mover advantage as the single most important strategy for attaining competitive advantage. The next most important strategy was complementary assets (for example, providing services for licensed software or offering a proprietary complement to an open source program).

Among IPRs, copyrights and trademarks—closely followed by secrecy and difficulties of reverse engineering—outranked patents as means of attaining competitive advantage among software respondents by a statistically significant margin.

What Incentive Effects Do Patents Have?

The Berkeley Patent survey asked startup executives to rate the incentive effects of patents on a scale, where 0 = no incentive, 1 = weak incentive, 2 = moderate incentive, and 3 = strong incentive, for engaging in four types of innovation: (1) inventing new products, processes, or services, (2) conducting initial R&D, (3) creating internal tools or processes, and (4) undertaking the risks and costs of commercializing the innovation.

We were surprised to discover the software respondents reported that patents provide only weak incentives for engaging in core activities, such as invention of new products (0.96) and commercialization (0.93). By contrast, biotech and medical device firms reported just above 2 (moderate incentives) for these same questions.

Interestingly, the results did not change significantly when considering only responses from software entrepreneurs whose firms hold at least one patent or application. Even patent-holding software entrepreneurs reported that patents provide just above a weak incentive for engaging in these innovation-related activities.

Resolving a Paradox

If patents provide only weak incentives for investing in innovation among software startups, why did two-thirds of the VX respondents and at least one-quarter of the D&B respondents seeking patents? The answer may lie in the perception among software entrepreneurs that patents may be important to potential funders, such as VCs, angel investors, other firms, commercial banks, and friends and family. Sixty percent of software startup respondents who had negotiated with VCs reported that they perceived VC decisions about whether to make the investments to be affected by patents. Between 40% and 50% of the software respondents reported that patents were important to other types of investors, such as angels, investment banks, and other companies.

Controversy Over Survey Findings

It is an article of faith among many IP lawyers that patents provide significant incentives for firms to engage in R&D and develop new products. Most would also expect, as we did, that high-tech startup companies would regard patents as more important as an inducement to innovation than large firms, given that the latter have lots of other assets for achieving and maintaining success in the marketplace.

Anecdotes highlighting the importance of patents to high-tech entrepreneurs are relatively easy to find. Because data from the Berkeley Patent Survey suggests that software entrepreneurs regard patents as quite unimportant, the reaction of some prominent patent lawyers to our article about the survey has been sharply negative. We believe, however, that our analysis is sound and these critiques are off-base. We encourage readers to read the full article and make their own judgments.

Future Research

Over the next several years, the co-authors of the Berkeley Patent Survey article expect to analyze further data from this survey and to report new findings. We will look more closely, for example, at differences in patenting rates among those in different sectors of the software industry and differences between patent holders and non-patent holders. We know already that product innovators seek patents more often than process innovators.

The findings reported here suggest that software entrepreneurs do not find persuasive the canonical story that patents provide strong incentives to engage in technology innovation. These executives regard first-mover advantage and complementary assets as more important than IPRs in conferring competitive advantage upon their firms. Moreover, among IPRs, copyrights and trademarks are perceived to be more important than patents. Still, about one-third of the software entrepreneur respondents reported having or seeking patents, and their perception that their investors care about patents seems to be a key factor in decisions to obtain patents. C

Reference

1. Graham, S.J.H., Merges, R.P., Samuelson, P., and Sichelman, T. High technology entrepreneurs and the patent system: Results of the 2008 Berkeley patent survey. *Berkeley Technology Law Journal 25*, 4 (2010), 1255–1327; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1429049.

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

DOI:10.1145/1839676.1839688

Joel F. Brenner

Privacy and Security
Why Isn't Cyberspace More Secure?

Evaluating governmental actions—and inactions—toward improving cyber security and addressing future challenges.

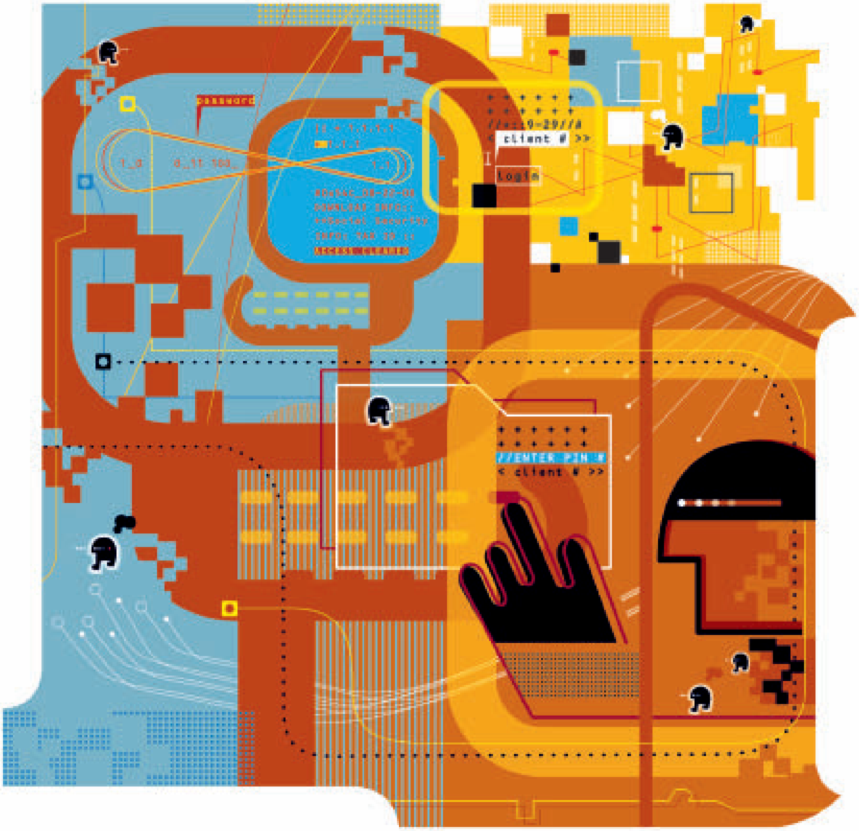


ILLUSTRATION BY CELIA JOHNSON

In cyberspace it's easy to get away with criminal fraud, easy to steal corporate intellectual property, and easy to penetrate governmental networks. Last spring the new Commander of USCYBERCOM, NSA's General Keith Alexander, acknowledged for the first time that even U.S. classified networks have been penetrated.[2] Not only do we fail to catch most fraud artists, IP thieves, and cyber spies—we don't even know who most of them are. Yet every significant public and private activity—economic, social, governmental, military—depends on the security of electronic systems. Why has so little happened in 20 years to alter the fundamental vulnerability of these systems? If you're sure this insecurity is either (a) a hoax or (b) a highly desirable form of anarchy, you can skip the rest of this column.

Presidential Directives to Fix This Problem emerge dramatically like clockwork from the White House echo chamber, chronicling a history of executive torpor. One of the following statements was made in a report to President Obama in 2009, the other by President George H.W. Bush *in 1990*. Guess which is which:

“Telecommunications and information processing systems are highly susceptible to interception, unauthorized electronic access, and related forms of technical exploitation, as well as other dimensions of the foreign intelligence threat.”

“The architecture of the Nation's digital infrastructure, based largely on the Internet, is not secure or resilient. Without major advances in the security of these systems or significant change in how they are constructed or operated, it is doubtful that the United States can protect itself from the growing threat of cybercrime and state-sponsored intrusions and operations.”

Actually, it doesn't much matter which is which.[a] In between, for the sake of nonpartisan continuity, Presi-

[a] The first quotation is from President G.H.W. Bush's National Security Directive 42, July 5, 1990, redacted for public release, April 1, 1992; http://www.fas.org/irp/offdocs/nsd/nsd_42.htm. The second quotation is from the preface to “Cyberspace Policy Review: Assuring a Trusted and Resilient Information and Communications Infrastructure,” May 2009; http://www.whitehouse.gov/assets/documents/Cyberspace_Policy_Review_final.pdf.

dent Clinton warned of the insecurities created by cyber-based systems and directed in 1998 that "no later than five years from today the United States shall have achieved and shall maintain the ability to protect the nation's critical infrastructures from intentional acts that would significantly diminish" our security.[6] Five years later would have been 2003.

In 2003, as if in a repeat performance of a bad play, the second President Bush stated that his cybersecurity objectives were to "[p]revent cyber attacks against America's critical infrastructure; [r]educe national vulnerability to cyber attacks; and [m]inimize damage and recovery time from cyber attacks that do occur."[7]

These Presidential pronouncements will be of interest chiefly to historians and to Congressional investigators who, in the aftermath of a disaster that we can only hope will be relatively minor, will be shocked, *shocked* to learn that the nation was electronically naked.

Current efforts in Washington to deal with cyber insecurity are promising—but so was Sisyphus' fourth or fifth trip up the hill. These efforts are moving at a bureaucratically feverish pitch—which is to say, slowly—and so far they have produced nothing but more declarations of urgency and more paper. Why?

Lawsuits and Markets

Change in the U.S. is driven by three things: liability, market demand, and regulatory (usually federal) action. The role and weight of these factors vary in other countries, but the U.S. experience may nevertheless be instructive transnationally since most of the world's intellectual property is stored in the U.S., and the rest of the world perceives U.S. networks as more secure than we do.[4] So let's examine each of these three factors.

Liability has been a virtually nonexistent factor in achieving greater Internet security. This may be surprising until you ask: Liability for what, and who should bear it? Software licenses are enforceable, whether shrink-wrapped or negotiated, and they nearly always limit the manufacturer's liability to the cost of the software. So suing the software manufacturer for allegedly lousy security would not be worth the money and effort expended. What are the damages, say, from finding your computer is an enslaved member of a botnet run out of Russia or Ukraine? And how do you prove the problem was caused by the software rather than your own sloppy online behavior?

Deciding what level of imperfection is acceptable is not a task you want your Congressional representative to perform.

Asking Congress to make software manufacturers liable for defects would be asking for trouble: *All* software is defective, because it's so astoundingly complicated that even the best of it hides surprises. Deciding what level of imperfection is acceptable is not a task you want your Congressional representative to perform. Any such legislation would probably drive some creative developers out of the market. It would also slow down software development—which would not be all bad if it led to higher security. But the general public has little or no understanding of the vulnerabilities inherent in poorly developed applications. On the contrary, the public clamors for rapidly developed apps with lots of bells and whistles, so an equipment vendor that wants to control this proliferation of vulnerabilities in the name of security is in a difficult position.

Banks, merchants, and other holders of personal information do face liability for data breaches, and some have paid substantial sums for data losses under state and federal statutes granting liquidated damages for breaches. In one of the best known cases, Heartland Payments Systems may end up paying approximately $100 million as a result of a major breach, not to mention millions more in legal fees. But the defendants in such cases are buyers, not makers and designers, of the hardware and software whose deficiencies create many (but not all) cyber insecurities. Liability presumably makes these companies somewhat more vigilant in their business practices, but it doesn't make hardware and software more secure.

Many major banks and other companies already know they have been persistently penetrated by highly skilled, stealthy, and anonymous adversaries, very likely including foreign intelligence services and their surrogates. These firms spend millions fending off attacks and cleaning their systems, yet no forensic expert can honestly tell them that all advanced persistent intrusions have been defeated. (If you have an expert who will say so, fire him right away.)

In an effective liability regime, insurers play an important role in raising standards because they tie premiums to good practices. Good automobile drivers, for example, pay less for car insurance. Without a liability dynamic, however, insurers play virtually no role in raising cyber security standards.

If liability hasn't made cyberspace more secure, what about market demand? The simple answer is that the consuming public buys on price and has not been willing to pay for more secure software. In some cases the aftermath of identity theft is an ordeal. In most instances of credit card fraud, however, the bank absorbs 100% of the loss, so their customers have little incentive to spend more for security. (In Britain, where the customer rather than the bank usually pays, the situation is arguably worse because banks are in a better position than customers to impose higher security requirements.) Most companies also buy on price, especially in the current economic downturn.

Unfortunately we don't know whether consumers or corporate customers would pay more for security if they knew the relative insecurities of the products on the market. As J. Alex Halderman of the University of Michigan recently noted, "most customers don't have enough information to accurately gauge software quality, so secure software and insecure software tend to sell for about the same price."[3] This could be fixed, but doing so would require agreed metrics for judging products and either the systematic disclosure of insecurities or a widely accepted testing and evaluation service that enjoyed the public's confidence. *Consumer Re-*

ports plays this role for automobiles and many other consumer products, and it wields enormous power. The same day *Consumer Reports* issued a "Don't buy" recommendation for the 2010 Lexus GX 460, Toyota took the vehicle off the market. If the engineering and computer science professions could organize a software security laboratory along the lines of *Consumer Reports*, it would be a public service.

Federal Action

Absent market- or liability-driven improvement, there are eight steps the U.S. federal government could take to improve Internet security, and none of them would involve creating a new bureaucracy or intrusive regulation:

1. Use the government's enormous purchasing power to require higher security standards of its vendors. These standards would deal, for example, with verifiable software and firmware, means of authentication, fault tolerance, and a uniform vocabulary and taxonomy across the government in purchasing and evaluation. The Federal Acquisition Regulations, guided by the National Institute of Standards and Technology, could drive higher security into the entire market by ensuring federal demand for better products.

2. Amend the Privacy Act to make it clear that Internet Service Providers (ISPs) *must* disclose to one another and to their customers when a customer's computer has become part of a botnet, regardless of the ISP's customer contract, and *may* disclose that fact to a party that is not its own customer. ISPs may complain that such a service should be elective, at a price. That's equivalent to arguing that cars should be allowed on the highway without brakes, lights, and seatbelts. This requirement would generate significant remedial business.

3. Define behaviors that would permit ISPs to block or sequester traffic from botnet-controlled addresses—not merely from the botnet's command-and-control center.

4. Forbid federal agencies from doing business with any ISP that is a hospitable host for botnets, and publicize the list of such companies.

5. Require bond issuers that are subject to the jurisdiction of the Federal Energy Regulatory Commission to disclose in the "Risk Factors" section of their prospectuses whether the command-and-control features of their SCADA networks are connected to the Internet or other publicly accessible network. Issuers would scream about this, even though a recent McAfee study plainly indicates that many of them that do follow this risky practice think it creates an "unresolved security issue."[1] SCADA networks were built for isolated, limited access systems. Allowing them to be controlled via public networks is rash. This point was driven home forcefully this summer by discovery of the "Stuxnet" computer worm, which was specifically designed to attack SCADA systems.[4] Yet public utilities show no sign of ramping up their typically primitive systems.

6. Increase support for research into attribution techniques, verifiable software and firmware, and the benefits of moving more security functions into hardware.

7. Definitively remove the antitrust concern when U.S.-based firms collaborate on researching, developing, or implementing security functions.

8. Engage like-minded governments to create international authorities to take down botnets and make naming-and-addressing protocols more difficult to spoof.

Political Will

These practical steps would not solve all problems of cyber insecurity but they would dramatically improve it. Nor would they involve government snooping and or reengineering the Internet or other grandiose schemes. They would require a clear-headed understanding of the risks to privacy, intellectual property, and national security when an entire society relies for its commercial, governmental, and military functions on a decades-old information system designed for a small number of university and government researchers.

Translating repeated diagnoses of insecurity into effective treatment would also require the political will to marshal the resources and effort necessary to do something about it. The Bush Administration came by that will too late in the game, and the Obama Administration has yet to acquire it. After his inauguration, Obama dithered for nine months over the package of excellent recommendations put on his desk by a nonpolitical team of civil servants from several departments and agencies. The Administration's lack of interest was palpable; its hands are full with a war, health care, and a bad economy. In difficult economic times the President naturally prefers invisible risk to visible expense and is understandably reluctant to increase costs for business. In the best of times cross-departmental (or cross-ministerial) governance would be extremely difficult—and not just in the U.S. Doing it well requires an interdepartmental organ of directive power that can muscle entrenched and often parochial bureaucracies, and in the cyber arena, we simply don't have it. The media, which never tires of the cliché, told us we were getting a cyber "czar," but the newly created cyber "Coordinator" actually has no directive power and has yet to prove his value in coordinating, let alone governing, the many departments and agencies with an interest in electronic networks.

And so cyber-enabled crime and political and economic espionage continue apace, and the risk of infrastructure failure mounts. As for me, I'm already drafting the next Presidential Directive. It sounds a lot like the last one.

References

1. Baker, S. et al. *In the Crossfire: Critical Infrastructure in the Age of Cyber War*, CSIS and McAfee, (Jan. 28, 2010), 19; http://img.en25.com/Web/McAfee/NA_CIP_RPT_REG_2840.pdf. See also P. Kurtz et al., *Virtual Criminology Report 2009: Virtually Here: The Age of Cyber Warfare*, McAfee and Good Harbor Consulting, 2009, 17; http://iom.invensys.com/EN/pdfLibrary/McAfee/WP_McAfee_Virtual_Criminology_Report_2009_03-10.pdf.
2. Gertz, B. 2008 intrusion of networks spurred combined units. *The Washington Times*, (June 3, 2010); http://www.washingtontimes.com/news/2010/jun/3/2008-intrusion-of-networks-spurred-combined-units/.
3. Halderman, J.Q. To strengthen security, change developers' incentives. *IEEE Security and Privacy* (Mar./Apr. 2010), 79.
4. Krebs, B. "Stuxnet" worm far more sophisticated than previously thought. *Krebs on Security*, Sept. 14, 2010; http://krebsonsecurity.com/2010/09/stuxnet-worm-far-more-sophisticated-than-previously-thought/.
5. McAfee. *Unsecured Economies: Protecting Vital Information*. 2009, 4, 13–14; http://www.cerias.purdue.edu/assets/pdf/mfe_unsec_econ_pr_rpt_fnl_online_012109.pdf.
6. Presidential Decision Directive 63, (May 22, 1998); http://www.fas.org/irp/offdocs/pdd/pdd-63.htm.
7. *The National Strategy to Secure Cyberspace 2003.* U.S. Department of Homeland Security.

Joel F. Brenner (jbrenner@cooley.com) of the law firm Cooley LLP in Washington, D.C., was the U.S. National Counterintelligence Executive from 2006–2009 and the Inspector General of the National Security Agency from 2002–2006.

DOI:10.1145/1839676.1839690

Matt Welsh

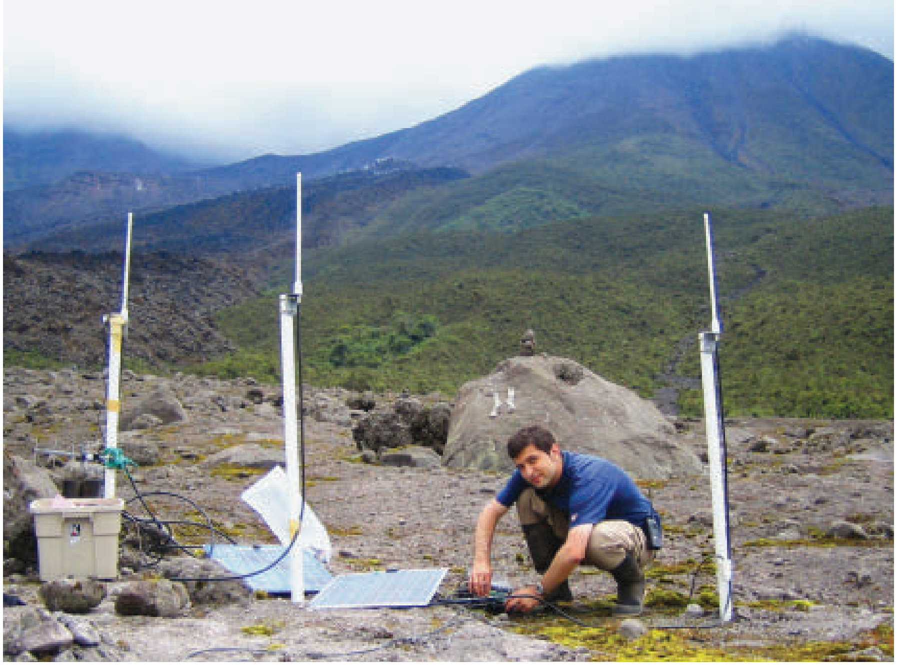
Viewpoint
Sensor Networks for the Sciences

Lessons from the field derived from developing wireless sensor networks for monitoring active and hazardous volcanoes.

MUCH COMPUTER SCIENCE research is interdisciplinary, bringing together experts from multiple fields to solve challenging problems in the sciences, engineering, and medicine. One area where the interface between computer scientists and domain scientists is especially strong is wireless sensor networks, which offer the opportunity to apply computer science concepts to obtaining measurements in challenging field settings. Sensor networks have been applied to studying vibrations on the Golden Gate Bridge,[1] tracking zebra movements,[2] and understanding microclimates in redwood canopies.[4]

Our own work on sensor networks for volcano monitoring[6] has taught us some valuable lessons about what's needed to make sensor networks successful for scientific campaigns. At the same time, we find a number of myths that persist in the sensor network literature, possibly leading to invalid assumptions about what field conditions are like, and what research problems fall out of working with domain scientists. We believe these lessons are of broad interest to "applied computer scientists" beyond the specific area of sensor networks.

Our group at Harvard has been collaborating with geophysicists at New Mexico Tech, UNC, and the Instituto Geofísico in Ecuador for the last five years on developing wireless sensor networks for monitoring active and hazardous volcanoes (see Figure 1). We have deployed three sensor networks on two volcanoes in Ecuador: Tungurahua and Reventador. In each case, wireless sensors measured seismic and acoustic signals generated by the volcano, and digitized signals are collected at a central base station located at the volcano observatory. This application pushes the boundaries of conventional sensor network design in terms of the high data rates involved (100Hz or more per channel); the need for fine-grained time synchronization to compare signals collected across different nodes; the need for reliable, complete signal collection over the lossy wireless network; and the need to discern "interesting" signals from noise.



Harvard University Ph.D. student Konrad Lorincz installing sensors at Reventador volcano.

PHOTOGRAPH BY KONRAD LORINCZ

These deployments have taught us many lessons about what works and what doesn't in the field, and what the important problems are from the perspective of the domain scientists. Interestingly, many of these problems are not the focus of much of the computer science research community. Our view is that a sensor network should be treated as a scientific instrument, and therefore subject to the same high standards of data quality applied to conventional scientific instrumentation.

Some Myths

First, let us dispel a few common myths about sensor network field deployments.

Myth #1: Nodes are deployed randomly. A common assumption in sensor network papers is that nodes will be randomly distributed over some spatial area (see Figure 2). An often-used idiom is that of dropping sensor nodes from an airplane. (Presumably, this implies that the packaging has been designed to survive the impact and there is a mechanism to orient the radio antennas vertically once they hit the ground.)

Such a haphazard approach to sensor siting would be unheard of in many scientific campaigns. In volcano seismology, sensor locations are typically chosen carefully to ensure good spatial coverage and the ability to reconstruct the seismic field. The resulting topologies are fairly irregular and do not exhibit the spatial uniformity often assumed in papers. Moreover, positions for each node must be carefully recorded using GPS, to facilitate later data analysis. In our case, installing each sensor node took nearly an hour (involving digging holes for the seismometer and antenna mast), not to mention the four-hour hike through the jungle just to reach the deployment site.

Myth #2: Sensor nodes are cheap and tiny. The original vision of sensor networks drew upon the idea of "smart dust" that could be literally blown onto a surface. While such technology is still an active area of research, sensor networks have evolved around off-the-shelf "mote" platforms that are substantially larger, more power hungry, and expensive than their hypothetically aerosol counterparts ("smart rocks" is a more apt metaphor). The notion that sensor nodes are disposable has led to much research that assumes it is possible to deploy many more sensor nodes than are strictly necessary to meet scientific requirements, leveraging redundancy to extend network battery lifetime and tolerate failures.

It should be emphasized that the cost of the attached sensor can outstrip the mote itself. A typical mote costs approximately $100, sometimes with onboard sensors for temperature, light, and humidity. The inexpensive sensors used on many mote platforms many not be appropriate for scientific use, confounded by low resolution and the need for calibration. While the microphones used in our volcano sensor network cost pennies, seismometers cost upward of thousands of dollars. In our deployments, we use a combination of relatively inexpensive ($75 or so) geophones with limited sensitivity, and more expensive ($1,000) seismometers. The instruments used by many volcano deployments are in the tens of thousands of dollars, so much that many research groups borrow (rather than buy) them.

Working with domain scientists has taught us some valuable lessons about sensor network design.

Myth #3: The network is dense. Related to the previous myths is the idea that node locations will be spatially homogeneous and dense, with each node having on the order of 10 or more neighbors in radio range. Routing protocols, localization schemes, and failover techniques often leverage such high density through the power of many choices.

This assumption depends on how closely aligned the spatial resolution of the desired network matches the radio range, which can be hundreds of meters with a suitably designed antenna configuration. In volcanology, the propagation speed of seismic waves (on the order of kilometers per second) dictates sensor placements hundreds of meters apart or more, which is at the practical limit of the radio range. As a result, our networks have typically featured nodes with at most two or three radio neighbors, with limited opportunities for redundancy in the routing paths. Likewise, the code-propagation protocol we used worked well in a lab setting when all of the nodes were physically close to each other; when spread across the volcano, the protocol fell over, probably due to the much higher degree of packet loss.

Lessons Learned

Working with domain scientists has taught us some valuable lessons about sensor network design. Our original intentions were to leverage the collaboration as a means of furthering our own computer science research agenda, assuming that whatever we did would be satisfactory to the geophysicists. In actuality, their data requirements ended up driving our research in several new directions, none of which we anticipated when we started the project.

Lesson #1: It's all about the data. This may seem obvious, but it's interesting how often the actual data produced by a sensor network is overlooked when designing a clever new protocol or programming abstraction. To first approximation, scientists simply want *all of the data produced by all of the sensors, all of the time*.

The approach taken by such scientists is to go to the field, install in-

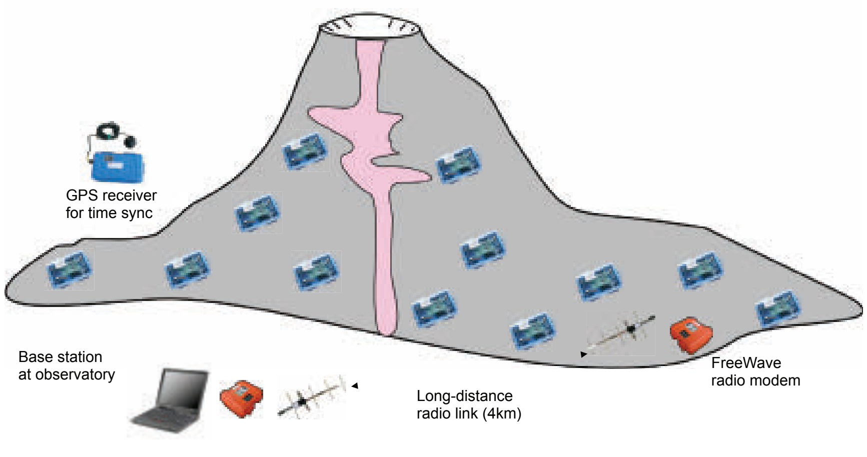


Figure 1. Sensor network design for monitoring active volcanoes.

Figure 2a. Myth (taken from Sankarasubramaniam[3]).

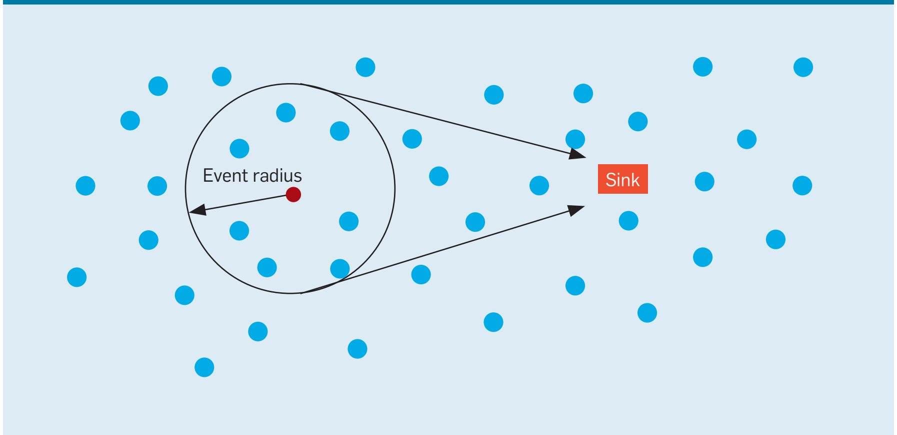
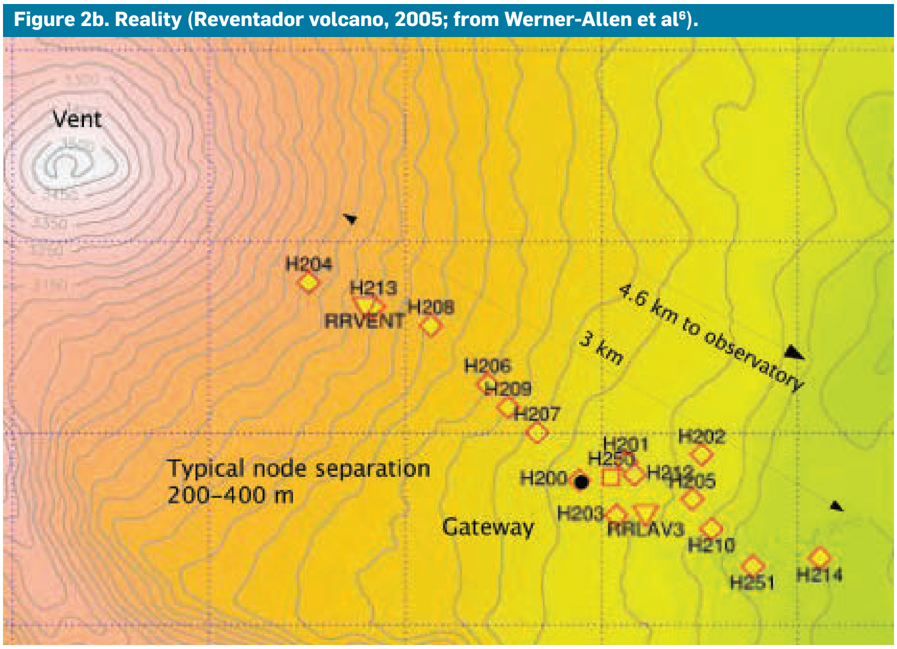


Figure 2b. Reality (Reventador volcano, 2005; from Werner-Allen et al[6]).



struments, collect as much data as possible, and then spend a considerable amount of time analyzing it and writing journal papers. After all, data collection is expensive and time consuming, and requires working in dirty places without a decent Internet connection. Scientists have a vested interest in getting as much "scientific value" as possible out of a field campaign, even if this requires a great deal of effort to understand the data once it has been collected. In contrast, the sensor network community has developed a wide range of techniques to perform data processing on the fly, aggregating and reducing the amount of data produced by the network to satisfy bandwidth and energy constraints. Many of these techniques are at odds with the domain scientists' view of instrumentation. No geophysicist is interested in the "average seismic signal" sampled by multiple nodes in the network. We advocate a two-pronged approach to this problem. The first is to incorporate large flash memories onto sensor nodes: it is now possible to build multi-gigabyte SD or Compact Flash memory into every node, allowing for months of continuous sensor data to be stored locally.

Though this converts the sensor node into a glorified data logger, it also ensures that all of the data will be available for (later) analysis and validation of the network's correct operation. It is often necessary to service nodes in the field, such as to change batteries, offering an early opportunity to retrieve the data manually by swapping flash cards. The second approach is to perform data collection with the goal of maximizing scientific value while satisfying resource constraints, such as a target battery lifetime. Our work on the Lance system[5] demonstrated it is possible to drive signal downloads from a sensor network in a manner that achieves *near optimal* data quality subject to these constraints. Figure 3 shows the rectification of raw signals collected from the network.

Inherent in this approach is the assumption that not all data is created equal: there must be some domain-specific assignment of "value" to the signals collected by the network to drive the process. In volcano seismology, scientists are interested in signals corresponding to geophysical events (earthquakes, tremors, explosions) rather than the quiet lull that can last for hours or days between such events. Fortunately, a simple amplitude filter running on each sensor node can readily detect seismic events of interest.

Lesson #2: Computer scientists and domain scientists need common ground. It should come as no surprise that the motivations of computer scientists and "real" scientists are not always aligned. Domain scientists are largely interested in obtaining high-quality data (see Lesson #1 above); whereas computer scientists are driven by the desire to do "cool stuff:" *new* protocols, *new* algorithms, *new* programming models. Our field thrives on novelty whereas domain scientists have an interest in measured conservatism. Anything new we computer scientists throw into the system potentially makes it harder, not easier, for the domain scientists to publish papers based on the results.

Finding common ground is essential to making such collaborations work. Starting small can help. Our first volcano deployment involved just three nodes running for two days, but in the process we learned an incredible amount about how volcanologists do field work (and what a donkey will—and will not—carry on its back). Our second deployment focused on collecting data with the goal of making the geophysicists happy with the fidelity of the instrument. The third was largely driven by CS goals, but with an eye toward meeting the scientists' data requirements. Writing joint grant proposals can also help to get everyone on the same page.

Lesson #3: Don't forget about the base station! The base station is a critical component of any sensor network architecture: it is responsible for coordinating the network's operation, monitoring its activity, and collecting the sensor data itself. Yet it often gets short shrift, perhaps because of the false impression the base station code will be easy to write or that it is uninteresting.

The vast majority of our development efforts focused on the sensor node software, which is fairly complex and uses nonstandard programming languages and tools. The base station, in our case a laptop located at the volcano observatory, was mostly an afterthought: some slapped-together Perl scripts and a monolithic Java program acting as a combined network controller, data logger, monitor, and GUI. The base station code underwent a major overhaul in the first two days after arriving in the field, mostly to add features (such as logging) that we didn't anticipate needing during our lab testing. We paid for the slapdash nature of the base station software. One race condition in the Java code (for which the author takes full credit) led to an 8-hour outage, while everyone was asleep. (We also assumed that the electricity supply at the observatory would be fairly reliable, which turned out not to be true.)

Figure 3a. Original data: bad timing, unnormalized signals.

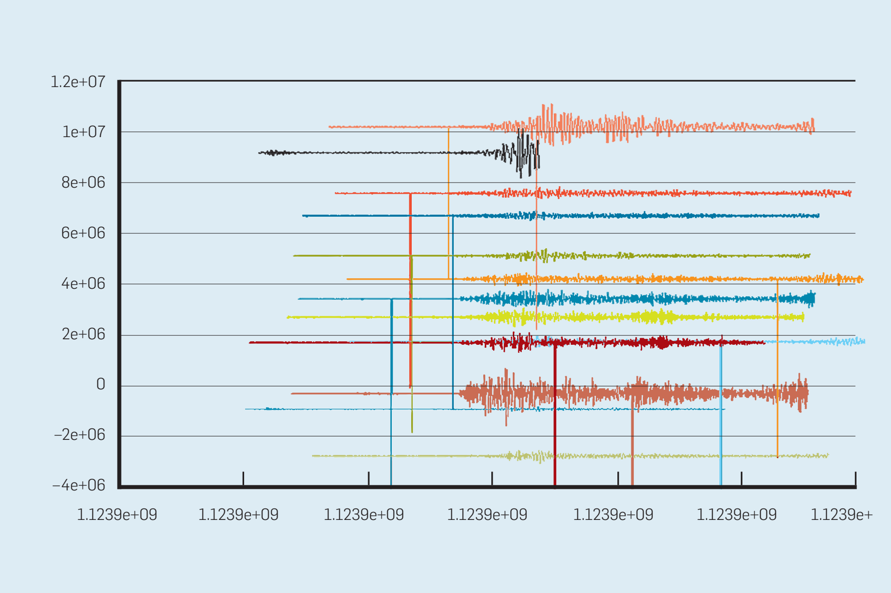
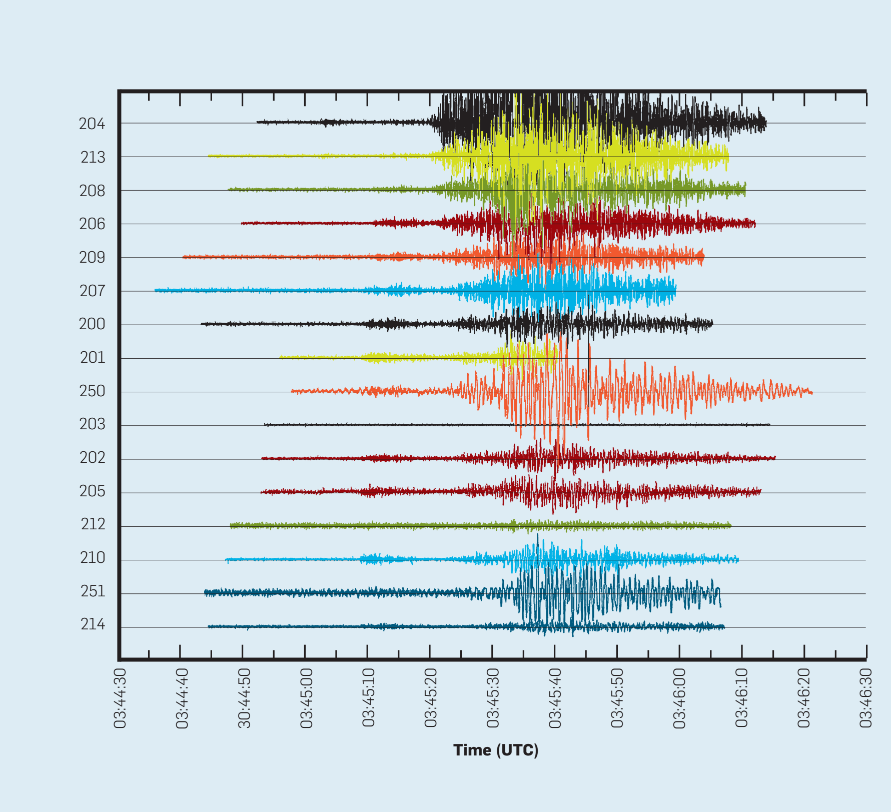


Figure 3b. Data after cleanup.



Our redesign for the 2007 Tungurahua deployment involved modularizing the base station code, so that each component can fail independently. One program communicates with the network; another acts as a GUI; another logs the sensor data; and another runs the algorithm for scheduling downloads. Bugs can be fixed and each of these programs can be restarted at any time without disrupting the other programs.

Conclusion

Scientific discovery is increasingly driven by advances in computing technology, and sensor networks are an important tool to enhance data collection in many scientific domains. Still, there is a gap between the stereotype of a sensor network in the literature and what many scientists need to obtain good field data. Working closely with domain scientists yields tremendous opportunities for furthering a computer science research agenda driven by real-world problems.

References

1. Kim, S. et al. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of IPSN 2007* (Cambridge, MA, Apr. 2007).
2. Liu, T. et al. Implementing software on resource-constrained mobile sensors: Experiences with Impala and ZebraNet. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSYS'04)*, June 2004.
3. Sankarasubramaniam, Y., Akan, O., and Akyildiz, I. ESRT: Event-to-sink reliable transport in wireless sensor networks. In *Proceedings of MobiHoc'03*, 2003.
4. Tolle, G. et al. A macroscope in the redwoods. In *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems* (SenSys 2005).
5. Werner-Allen, G., Dawson-Haggerty, S., and Welsh, M. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems* (SenSys'08), Nov. 2008.
6. Werner-Allen, G. et al. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (OSDI 2006), Nov. 2006.

Matt Welsh (mdw@eecs.harvard.edu) is a professor of computer science at Harvard University in Cambridge, MA.

ACM, Advancing Computing as a Science and a Profession

Dear Colleague,

The power of computing technology continues to drive innovation to all corners of the globe, bringing with it opportunities for economic development and job growth. ACM is ideally positioned to help computing professionals worldwide stay competitive in this dynamic community.

ACM provides invaluable member benefits to help you advance your career and achieve success in your chosen specialty. Our international presence continues to expand and we have extended our online resources to serve needs that span all generations of computing practitioners, educators, researchers, and students.

ACM conferences, publications, educational efforts, recognition programs, digital resources, and diversity initiatives are defining the computing profession and empowering the computing professional.

This year we are launching Tech Packs, integrated learning packages on current technical topics created and reviewed by expert ACM members. The Tech Pack core is an annotated bibliography of resources from the renowned ACM Digital Library – articles from journals, magazines, conference proceedings, Special Interest Group newsletters, videos, etc. – and selections from our many online books and courses, as well an non-ACM resources where appropriate.

BY BECOMING AN ACM MEMBER YOU RECEIVE:

Timely access to relevant information

Communications of the ACM magazine • ACM Tech Packs • *TechNews* email digest • Technical Interest Alerts and ACM Bulletins • ACM journals and magazines at member rates • full access to the *acmqueue* website for practitioners • ACM SIG conference discounts • the optional ACM Digital Library

Resources that will enhance your career and follow you to new positions

Career & Job Center • online books from Safari® featuring O'Reilly and Books24x7® • online courses in multiple languages • virtual labs • e-mentoring services • *CareerNews* email digest • access to ACM's 34 Special Interest Groups • an acm.org email forwarding address with spam filtering

ACM's worldwide network of more than 97,000 members ranges from students to seasoned professionals and includes many renowned leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Alain Chesnais

Alain Chesnais

President
Association for Computing Machinery

Association for
Computing Machinery

Advancing Computing as a Science & Profession

Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: AD10

You can join ACM in several easy ways:

Online	Phone	Fax
http://www.acm.org/join	*+1-800-342-6626 (US & Canada)* *+1-212-626-0500 (Global)*	*+1-212-944-1318*

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
http://www.acm.org/membership/L2-3/

Special rates for members of sister societies:
http://www.acm.org/membership/dues.html

Please print clearly

Name

Address

City State/Province Postal code/Zip

Country E-mail address

Area code & Daytime phone Fax Member number, if applicable

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

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/serving/ethics.html

choose one membership option:

PROFESSIONAL MEMBERSHIP:

o **ACM Professional Membership: $99 USD**

o **ACM Professional Membership plus the ACM Digital Library: $198 USD ($99 dues + $99 DL)**

o **ACM Digital Library: $99 USD (must be an ACM member)**

STUDENT MEMBERSHIP:

o **ACM Student Membership: $19 USD**

o **ACM Student Membership plus the ACM Digital Library: $42 USD**

o **ACM Student Membership PLUS Print *CACM* Magazine: $42 USD**

o **ACM Student Membership w/Digital Library PLUS Print *CACM* Magazine: $62 USD**

**All new ACM members will receive an
ACM membership card.
For more information, please visit us at www.acm.org**

Professional membership dues include $40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

o Visa/MasterCard o American Express o Check/money order

o Professional Member Dues ($99 or $198) $ ____________________

o ACM Digital Library ($99) $ ____________________

o Student Member Dues ($19, $42, or $62) $ ____________________

Total Amount Due $ ____________________

Card # Expiration date

Signature

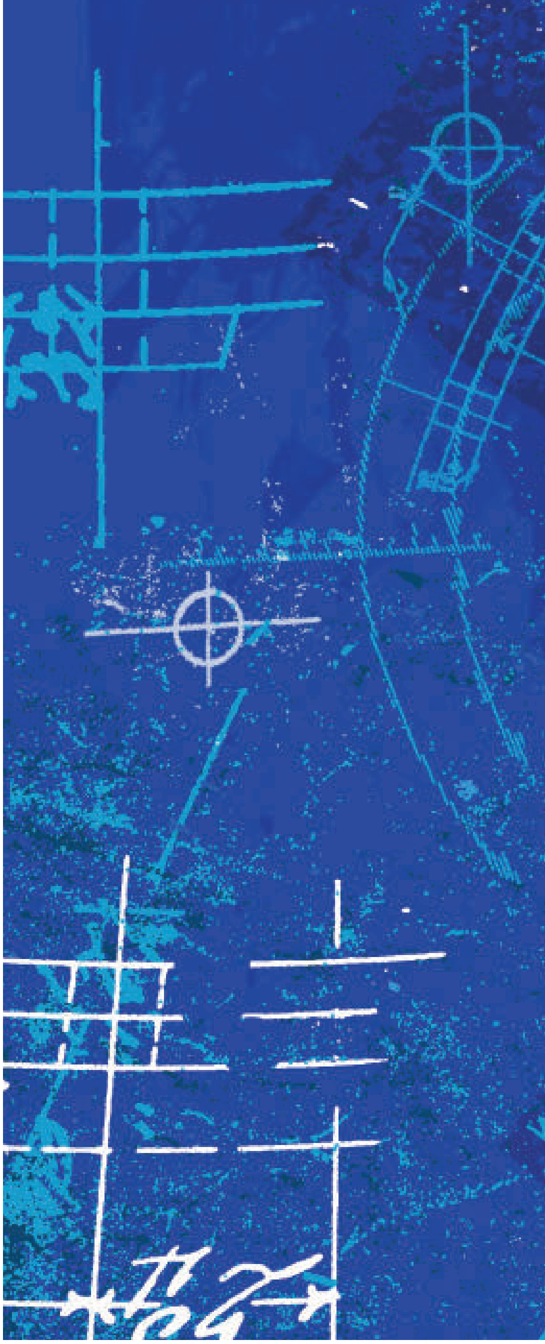
DOI:10.1145/1839676.1839691

Article development led by acmqueue
queue.acm.org

Want to keep your users?
Just make it easy for them to leave.

BY BRIAN W. FITZPATRICK AND JJ LUECK

The Case Against Data Lock-in



ENGINEERS EMPLOY MANY different tactics to focus on the user when writing software: for example, listening to user feedback, fixing bugs, and adding features that their users are clamoring for. Since Web-based services have made it easier for users to move to new applications, it is becoming even more important to focus on building and retaining user trust. We have found that an incredibly effective—although certainly counterintuitive—way to earn and maintain user trust is to make it easy for users to leave your product with their data in tow. This not only prevents lock-in and engenders trust, but also forces your team to innovate and compete on technical merit. We call this *data liberation*.

Until recently, users rarely asked whether they could quickly and easily get their data out before they put reams of personal information into a new Internet service. They were more likely to ask questions such as: "Are my friends using the service?" "How reliable is it?" and "What are the odds that the company providing the service is going to be around in six months or a year?" Users are starting to realize, however, that as they store more of their personal data in services that are not physically accessible, they run the risk of losing vast swaths of their online legacy if they do not have a means of removing their data.

It is typically a lot easier for software engineers to pull data out of a service that they use than it is for regular us-

ILLUSTRATION BY GARY NEILL

ers. If APIs are available, we engineers can cobble together a program to pull our data out. Without APIs, we can even whip up a screen scraper to get a copy of the data. Unfortunately, for most users this is not an option, and they are often left wondering if they can get their data out at all.

Locking your users in, of course, has the advantage of making it more difficult for them to leave you for a competitor. Likewise, if your competitors lock their users in, it is harder for those users to move to your product. Nonetheless, it is far preferable to spend your engineering effort on innovation than it is to build bigger walls and stronger doors that prevent users from leaving. Making it easier for users to experiment today greatly increases their trust in you, and they are more likely to return to your product line tomorrow.

Locking users in may suppress a company's need to innovate as rapidly as possible. Instead, your company may decide—for business reasons—to slow down development on your product and move engineering resources to another product. This makes your product vulnerable to other companies that innovate at a faster rate. Lock-in allows your company to have the appearance of continuing success when, without innovation, it may in fact be withering on the vine.

If you do not—or cannot—lock your users in, the best way to compete is to innovate at a breakneck pace. Let's use Google Search as an example. It's a product that *cannot* lock users in: users do not have to install software to use it; they do not have to upload data to use it; they do not have to sign two-year contracts; and if they decide to try another search engine, they merely type it into their browser's location bar, and they are off and running.

How has Google managed to get users to keep coming back to its search engine? By focusing obsessively on constantly improving the quality of its results. The fact that it is so easy for users to switch has instilled an incredible sense of urgency in Google's search quality and ranking teams. At Google we think that if we make it easy for users to leave any of our products, failure to improve a product results in immediate feedback to the engineers, who respond by building a better product.

What Data Liberation Looks Like

At Google, our attitude has always been that users should be able to control the data they store in any of our products, and that means they should be able to get their data *out* of any product. Period. There should be no additional monetary cost to do so, and perhaps most importantly, the amount of effort required to get the data out should be constant, regardless of the amount of data. Individually downloading a dozen photos is no big inconvenience, but what if a user had to download 5,000 photos, one at a time, to get them out of an application? That could take weeks of their time.

Even if users have a copy of their data, it can still be locked in if it is in a proprietary format. Some word processor documents from 15 years ago cannot be opened with modern software because they are stored in a proprietary format. It is important, therefore, not only to have access to data, but also to have it in a format that has a publicly available specification. Furthermore, the specification must have reasonable license terms: for example, it should be royalty-free to implement. If an open format already exists for the exported data (for example, JPEG or TIFF for photos), then that should be an option for bulk download. If there is no industry standard for the data in a product (for example, blogs do not have a standard data format), then at the very least the format should be publicly documented—bonus points if your product provides an open source reference implementation of a parser for your format.

The point is that users should be in control of their data, which means they need an easy way of accessing it. Providing an API or the ability to download 5,000 photos one at a time does not exactly make it easy for your average user to move data in or out of a product. From the user-interface point of view, users should see data liberation merely as a set of buttons for import and export of all data in a product.

Google is addressing this problem through its Data Liberation Front, an engineering team whose goal is to make it easier to move data in and out of Google products. The data liberation effort focuses specifically on data that could hinder users from switching to another service or competing product—that is, data that users create in or import into Google products. This is all data stored intentionally via a direct action—such as photos, email, documents, or ad campaigns—that users would most likely need a copy of if they wanted to take their business elsewhere. Data indirectly created as a side effect (for example, log data) falls outside of this mission, as it is not particularly relevant to lock-in.

It is preferable to spend your engineering effort on innovation than it is to build bigger walls and stronger doors that prevent users from leaving. Making it easier for users to experiment today greatly increases their trust, and they are more likely to return to your product line tomorrow.

Another "non-goal" of data liberation is to develop new standards: we allow users to export in existing formats where we can, as in Google Docs where users can download word processing files in OpenOffice or Microsoft Office formats. For products where there is no obvious open format that can contain all of the information necessary, we provide something easily machine readable such as XML (for example, for Blogger feeds, including posts and comments, we use Atom), publicly document the format, and, where possible, provide a reference implementation of a parser for the format (see the Google Blog Converters AppEngine project for an example[a]). We try to give the data to the user in a format that makes it easy to import into another product. Since Google Docs deals with word processing documents and spreadsheets that predate the rise of the open Web, we provide a few different formats for export; in most products, however, we assiduously avoid the rat hole of exporting into every known format under the sun.

The User's View

There are several scenarios where users might want to get a copy of their data from your product: they may have found another product that better suits their needs and they want to bring their data into the new product; you have announced that you are going to stop supporting the product they are using; or, worse, you may have done something to lose their trust.

Of course, just because your users want a copy of their data does not necessarily mean they are abandoning your product. Many users just feel safer hav-

[a] http://code.google.com/p/google-blog-converters-appengine/wiki/BloggerExportTemplate; and http://code.google.com/apis/blogger/docs/2.0/reference.html#LinkCommentsToPosts.

ing a local copy of their data as a backup. We saw this happen when we first liberated Blogger: many users started exporting their blogs every week while continuing to host and write in Blogger. This last scenario is more rooted in emotion than logic. Most data that users have on their computers is not backed up at all, whereas hosted applications almost always store multiple copies of user data in multiple geographic locations, accounting for hardware failure in addition to natural disasters. Whether users' concerns are logical or emotional, they need to feel their data is safe: it's important that your users trust you.

Case Study: Google Sites

Google Sites is a Web site creator that allows WYSIWYG editing through the browser. We use this service inside of Google for our main project page, as it is really convenient for creating or aggregating project documentation. We took on the job of creating the import and export capabilities for Sites in early 2009.

Early in the design, we had to determine what the external format of a Google Site should be. Considering that the utility Sites provides is the ability to create and collaborate on Web sites, we decided that the format best suited for true liberation would be XHTML. HTML, as the language of the Web, also makes it the most portable format for a Web site: just drop the XHTML pages on your own Web server or upload them to your Web service provider. We wanted to make sure this form of data portability was as easy as possible with as little loss of data as possible.

Sites uses its internal data format to encapsulate the data stored in a Web site, including all revisions to all pages in the site. The first step to liberating this data was to create a Google Data API. A full export of a site is then provided through an open source Java client tool that uses the Google Sites Data API and transforms the data into a set of XHTML pages.

The Google Sites Data API, like all Google Data APIs, is built upon the AtomPub specification. This allows for RPC (remote procedure call)-style programmatic access to Google Sites data using Atom documents as the wire format for the RPCs. Atom works well for the Google Sites use case, as the data fits fairly easily into an Atom envelope.

Figure 1 is a sample of one Atom entry that encapsulates a Web page within Sites. This can be retrieved by using the Content Feed to Google Sites.

We have highlighted (in red) the actual data that is being exported, which includes an identifier, a last update time in ISO 8601 format, title, revision number, and the actual Web-page content. Mandatory authorship elements and other optional information included in the entry have been removed to keep the example short.

Once the API was in place, the second step was to implement the transformation from a set of Atom feeds into a collection of portable XHTML Web pages. To protect against losing any data from the original Atom, we chose to embed all of the metadata about each Atom entry right into the transformed XHTML. Not having this metadata in the transformed pages poses a problem during an import—it becomes unclear which elements of XHTML correspond to the pieces of the original Atom entry. Luckily, we did not have to invent our own metadata embedding technique; we simply used the hAtom microformat.

To demonstrate the utility of microformats, Figure 2 shows the same sample after being converted into XHTML with hAtom microformat embedded:

The highlighted class attributes map directly to the original Atom elements, making it very explicit how to reconstruct the original Atom when importing this information back into Sites. The microformat approach also has the side benefit that any Web page can be imported into Sites if the author is willing to add a few class attributes to data within the page. This ability to reimport

Figure 1. Atom entry encapsulating a Web page within Sites.

```
<entry xmlns:sites=”http://schemas.google.com/sites/2008”>
  <id>https://sites.google.com/feeds/content/site/...</id>
  <updated>2009-02-09T21:46:14.991Z</updated>
  <category scheme=”http://schemas.google.com/g/2005#kind”
         term=”http://schemas.google.com/sites/2008#webpage”
         label=”webpage”/>
  <title>Maps API Examples</title>
  <sites:revision>2</sites:revision>
  <content type=”xhtml”>
    <div xmlns=”http://www.w3.org/1999/xhtml”>
    ... PAGE CONTENT HERE ...
    </div>
  </content>
</entry>
```

Figure 2. Atom entry converted into XHTML.

```
<div class=”hentry webpage”
     id=”https://sites.google.com/feeds/content/site/...”>
  <h3>
    <span class=”entry-title”>Maps API Examples</span>
  </h3>
  <div>
    <div class=”entry-content”>
      <div xmlns=”http://www.w3.org/1999/xhtml”>
      ... PAGE CONTENT HERE ...
      </div>
    </div>
  </div>
  <small>
    Updated on
    <abbr class=”updated” title=”2009-02-09T21:46:14.991Z”>
      Feb 9, 2009
     </abbr>
    (Version <span class=”sites:revision”>2</span>)
  </small>
</div>
```

a user's exported data in a lossless manner is key to data liberation—it may take more time to implement, but we think the result is worthwhile.

Case Study: Blogger

One of the problems we often encounter when doing a liberation project is catering to the power user. These are our favorite users. They are the ones who love to use the service, put a lot of data into it, and want the comfort of being able to do very large imports or exports of data at any time. Five years of journalism through blog posts and photos, for example, can easily extend beyond a few gigabytes of information, and attempting to move that data in one fell swoop is a real challenge. In an effort to make import and export as simple as possible for users, we decided to implement a one-click solution that would provide the user with a Blogger export file that contains all of the posts, comments, static pages, and even settings for any Blogger blog. This file is downloaded to the user's hard drive and can be imported back into Blogger later or transformed and moved to another blogging service.

One mistake we made when creating the import/export experience for Blogger was relying on one HTTP transaction for an import or an export. HTTP connections become fragile when the size of the data you are transferring becomes large. Any interruption in that connection voids the action and can lead to incomplete exports or missing data upon import. These are extremely frustrating scenarios for users and, unfortunately, much more prevalent for power users with lots of blog data. We neglected to implement any form of partial export as well, which means power users sometimes need to resort to silly things such as breaking up their export files by hand in order to have better success when importing. We recognize this is a bad experience for users and are hoping to address it in a future version of Blogger.

A better approach, one taken by rival blogging platforms, is not to rely on the user's hard drive to serve as the intermediary when attempting to migrate lots of data between cloud-based Blogging services. Instead, data *liberation* is best provided through APIs, and data *portability* is best provided by building code using those APIs to perform cloud-to-cloud migration. These types of migrations require multiple RPCs between services to move the data piece by piece, and each of these RPCs can be retried upon failure automatically without user intervention. It is a much better model than the one transaction import. It increases the likelihood of total success and is an all-around better experience for the user. True cloud-to-cloud portability, however, works only when each cloud provides a liberated API for all of the user's data. We think cloud-to-cloud portability is really good for users, and it's a tenet of the Data Liberation Front.

Challenges

As you have seen from these case studies, the first step on the road to data liberation is to decide exactly what users need to export. Once you have covered data that users have imported or created by themselves into your product, it starts to get complicated. Take Google Docs, for example: a user clearly owns a document that he or she created, but what about a document that belongs to another user, then is edited by the user currently doing the export? What about documents to which the user has only read access? The set of documents the user has read access to may be considerably larger than the set of documents the user has actually read or opened if you take into account globally readable documents. Lastly, you have to take into account document metadata such as access control lists. This is just one example, but it applies to any product that lets users share or collaborate on data.

Another important challenge to keep in mind involves security and authentication. When you are making it very easy and fast for users to pull their data out of a product, you drastically reduce the time required for an attacker to make off with a copy of all your data. This is why it's a good idea to require users to re-authenticate before exporting sensitive data (such as their search history), as well as over-communicating export activity back to the user (for example, email notification that an export has occurred). We are exploring these mechanisms and more as we continue liberating products.

Large data sets pose another challenge. An extensive photo collection, for example, which can easily scale into multiple gigabytes, can pose difficulties with delivery given the current transfer speeds of most home Internet connections. In this case, either we have a client for the product that can sync data to and from the service (such as Picasa), or we rely on established protocols and APIs (for example, POP and IMAP for Gmail) to allow users to sync incrementally or export their data.

Conclusion

Allowing users to get a copy of their data is just the first step on the road to data liberation: we have a long way to go to get to the point where users can easily move their data from one product on the Internet to another. We look forward to this future, where we as engineers can focus less on schlepping data around and more on building interesting products that can compete on their technical merits—not by holding users hostage. Giving users control over their data is an important part of establishing user trust, and we hope more companies will see that if they want to retain their users for the long term, the best way to do that is by setting them free.

Acknowledgments

Thanks to Bryan O'Sullivan, Ben Collins-Sussman, Danny Berlin, Josh Bloch, Stuart Feldman, and Ben Laurie for reading drafts of this article.

Related articles on queue.acm.org

Other People's Data
Stephen Petschulat
http://queue.acm.org/detail.cfm?id=1655240

Why Cloud Computing Will Never Be Free
Dave Durkee
http://queue.acm.org/detail.cfm?id=1772130

Brian Fitzpatrick started Google's Chicago engineering office in 2005 and is the engineering manager for the Data Liberation Front and the Google Affiliate Network. A published author, frequent speaker, and open source contributor for more than 12 years, Fitzpatrick is a member of the Apache Software Foundation and the Open Web Foundation, as well as a former engineer at Apple and CollabNet.

JJ Lueck joined the software engineering party at Google in 2007. An MIT graduate and prior engineer at AOL, Bose, and startups Bang Networks and Reactivity, he enjoys thinking about problems such as cloud-to-cloud interoperability and exploring the depths and potentials of virtual machines.

DOI:10.1145/1839676.1839692

Article development led by acmqueue
queue.acm.org

As storage systems grow larger and larger, protecting their data for long-term storage is becoming ever more challenging.

BY DAVID S.H. ROSENTHAL

Keeping Bits Safe: How Hard Can It Be?

THESE DAYS, WE are all data pack rats. Storage is cheap, so if there is a chance the data could possibly be useful, we keep it. We know that storage isn't completely reliable, so we keep backup copies as well. But the more data we keep, and the longer we keep it, the greater the chance that some of it will be unrecoverable when we need it.

There is an obvious question we should be asking: how many copies in storage systems with what reliability do we need to get a given probability that the data will be recovered when we need it? This may be an obvious question to ask, but it is a surprisingly difficult question to answer. Let's look at the reasons why.

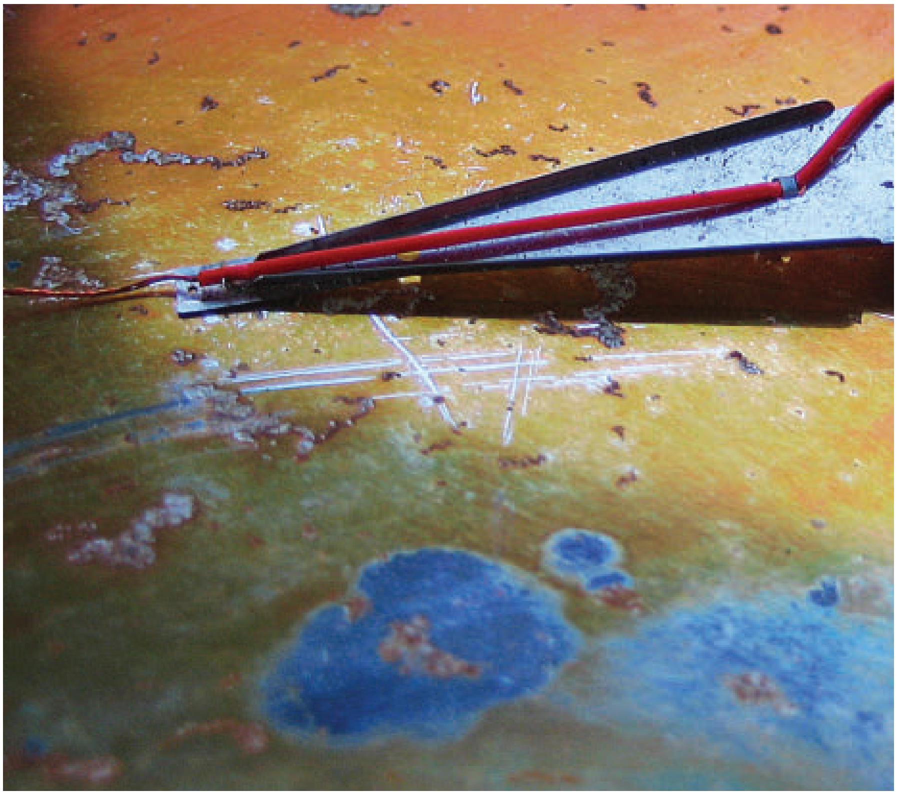
To be specific, let's suppose we need to keep a petabyte for a century and have a 50% chance that every bit will survive undamaged. This may sound like a lot of data and a long time, but there are already data collections bigger than a petabyte that are important to keep forever. The Internet Archive is already multiple petabytes.

The state of our knowledge about keeping bits safe can be summarized as:

- *The more copies, the safer.* As the size of the data increases, the per-copy cost increases, reducing the number of backup copies that can be afforded.
- *The more independent the copies, the safer.* As the size of the data increases, there are fewer affordable storage technologies. Thus, the number of copies in the same storage technology increases, decreasing the average level of independence.
- *The more frequently the copies are audited, the safer.* As the size of the data

increases, the time and cost needed for each audit to detect and repair damage increases, reducing their frequency.

At first glance, keeping a petabyte for a century is not difficult. Storage system manufacturers make claims for their products that far exceed the reliability we need. For example, Sun claimed that its ST5800 Honeycomb product had an MTTDL (mean time to data loss) of 2.4×10^{6} years.[a,41] Off-the-shelf solutions appear so reliable that backups are unnecessary. Should we believe these claims? Where do they come from?



PHOTOGRAPH BY TARAN RAMPERSAD

Before using Sun's claim for the ST5800 as an example, I should stipulate that the ST5800 was an excellent product. It represented the state of the art in storage technology, and Sun's marketing claims represented the state of the art in storage marketing. Nevertheless, Sun did not guarantee that data in the ST5800 would last 2.4×10^{6} years. Sun's terms and conditions explicitly disclaimed any liability whatsoever for loss of, or damage to, the data the ST5800 stores[40] whenever it occurs.

All that Sun was saying was if you watched a large number of ST5800 systems for a long time, recorded the time at which each of them first suffered a data loss, and then averaged these times, the result would be 2.4×10^{6} years. Suppose Sun watched 10 ST5800s and noticed that three of them lost data during the first year, four of them lost data after 2.4×10^{6} years, and the remaining three lost data after 4.8×10^{6} years; Sun would be correct that the MTTDL was 2.4×10^{6} years. But we would not consider a system with a 30% chance of data loss in the first year was adequate to keep a petabyte safe for a century. A single MTTDL number is not a useful characterization of a solution.

Let's look at the slightly more scientific claim made at the recent launch of the SC5800 by the marketing department of Sirius Cybernetics:[b] "SC5800 has an MTTDL of $(2.4\pm0.4)\times10^{6}$ years." Sirius implicitly assumes the failures are normally distributed and thus claims that about two-thirds of the failures would occur between 2.0×10^{6} and 2.8×10^{6} years after the start of the experiment. As Sirius did not start watching a batch of SC5800s 2.8 million years ago, how would they know?

Sirius says it will sell 2×10^{4} SC5800s per year at $\$5\times10^{4}$ each (a $1 billion-a-year business), and it expects the product to be in the market for 10 years. The SC5800 has a service life of 10 years. So if Sirius watched the entire production of SC5800s ($\$10^{10}$ worth of storage systems) over their entire service life, the experiment would end 20 years from now after accumulating about 2×10^{6} system-years of data. If its claim were correct, Sirius would have about a 17% chance of seeing a single data-loss event.

In other words, Sirius claims the probability that *no SC5800 will ever lose any data* is more than 80%. Or, since each SC5800 stores 5×10^{13} bytes, there is an 80% probability that 10^{19} bytes of data will survive 10 years undamaged.

If one could believe Sirius' claim, the petabyte would look pretty safe for a century. But even if Sirius were to base its claim on an actual experiment, it would not provide results for 20 years and even when it did, would not validate the number in question. In fact, claims such as those of Sun and Sirius are not the result of experimentation at all. No feasible experiment could validate them. They are *projections* based on models of how components of the system such as disks and software behave.

Models

The state of the art in this kind of modeling is exemplified by the Pergamum project at UC Santa Cruz.[39] Its model includes disk failures at rates derived from measurements[30,35] and sector failures at rates derived from disk vendor specifications. This system attempts to conserve power by spinning the disks down whenever possible; it makes an allowance for the effect of doing so on disk lifetime, but it is not clear upon what this allowance is based. The Pergamum team reports that the simulations were difficult:

"This lack of data is due to the extremely high reliability of these configurations—the simulator modeled many failures, but so few caused data loss that the simulation ran very slowly.

[a] Numbers are expressed in powers-of-10 notation to help readers focus on the scale of the problems and the extraordinary level of reliability required.

[b] Purveyors of chatty doors, existential elevators, and paranoid androids to the nobility and gentry of this galaxy.[1]

This behavior is precisely what we want from an archival storage system: it can gracefully handle many failure events without losing data. Even though we captured fewer data points for the triple inter-parity configuration, we believe the reported MTTDL is a reasonable approximation."[39]

Although the Pergamum team's effort to obtain "a reasonable approximation" to the MTTDL of its system is praiseworthy, there are a number of reasons to believe it overestimates the reliability of the system in practice:

- The model draws its failures from exponential distributions. The team thus assumes that both disk and sector failures are uncorrelated, although all observations of actual failures[5,42] report significant correlations. Correlated failures greatly increase the probability of data loss.[6,13]
- Other than a small reduction in disk lifetime from each power-on event, the Pergamum team assumes that failure rates observed in always-on disk usage translate to the mostly off environment. A study[43] published after the Pergamum paper reports a quantitative accelerated life test of data retention in almost-always-off disks. It shows that some of the 3.5-inch disks anticipated by the Pergamum team have data life dramatically worse in this usage mode than 2.5-inch disks using the same head and platter technology.
- The team assumes that disk and sector failures are the only failures contributing to the system failures, although a study[17] shows that other hardware components contribute significantly.
- It assumes that its software is bug-free, despite several studies of file and storage implementations[14,20,31] that uniformly report finding bugs capable of causing data loss in all systems studied.
- It also ignores all other threats to stored data[34] as possible causes of data loss. Among these are operator error, insider abuse, and external attack. Each of these has been the subject of anecdotal reports of actual data loss.

What can such models tell us? Their results depend on both of the following:

- The details of the simulation of the system being studied, which, one hopes, accurately reflect its behavior.
- The data used to drive the simulation, which, one hopes, accurately reflects the behavior of the system's components.

The more data we keep, and the longer we keep it, the greater the chance that some of it will be unrecoverable when we need it.

Under certain conditions, it is reasonable to use these models to compare different storage-system technologies. The most important condition is that the models of the two systems use the same data. A claim that modeling showed system A to be more reliable than system B when the data used to model system A had much lower failure rates for components such as disk drives would not be credible.

These models may well be the best tools available to evaluate different techniques for preventing data loss, but they aren't good enough to answer our question. We need to know the *maximum* rate at which data will be lost. The models assume things, such as uncorrelated errors and bug-free software, that all real-world studies show are false. The models exclude most of the threats to which stored data is subject. In those cases where similar claims, such as those for disk reliability,[30,35] have been tested, they have been shown to be optimistic. The models thus provide an estimate of the *minimum* data loss rate to be expected.

Metrics

Even if we believed the models, the MTTDL number does not tell us how much data was lost in the average data-loss event. Is petabyte system A with an MTTDL of 10^6 years better than a similar-size system B with an MTTDL of 10^3 years? If the average data-loss event in system A loses the entire petabyte, where the average data-loss event in system B loses a kilobyte, it would be easy to argue that system B was 10^9 times better.

Mean time to data loss is not a useful metric for how well a system stores bits through time, because it relates to time but not to bits. Nor is the UBER (unrecoverable bit error rate) typically quoted by disk manufacturers; it is the probability that a bit will be read incorrectly regardless of how long it has been sitting on the disk. It relates to bits but not to time. Thus, we see that we lack even the metric we would need to answer our question.

Let us oversimplify the problem to

get a clearer picture. Suppose we had eliminated all possible sources of correlated data loss, from operator error to excess heat. All that remained would be *bit rot*, a process that randomly flips the bits the system stores with a constant small probability per unit time. In this model we can treat bits as radioactive atoms, so that the time after which there is a 50% probability that a bit will have flipped is the *bit half-life*.

The requirement of a 50% chance that a petabyte will survive for a century translates into a bit half-life of 8×10^{17} years. The current estimate of the age of the universe is 1.4×10^{10} years, so this is a bit half-life approximately 6×10^{7} times the age of the universe.

This bit half-life requirement clearly shows the high degree of difficulty of the problem we have set for ourselves. Suppose we want to know whether a system we are thinking of buying is good enough to meet the 50% chance of keeping a petabyte for a century. Even if we are sublimely confident that every source of data loss other than bit rot has been totally eliminated, we still have to run a benchmark of the system's bit half-life to confirm it is longer than 6×10^{7} times the age of the universe. And this benchmark has to be complete in a year or so; it can't take a century.

So we take 10^{3} systems just like the one we want to buy, write a petabyte of data into each so we have an exabyte of data altogether, wait a year, read the exabyte back, and check it. If the system is just good enough, we might see five bit flips. Or, because bit rot is a random process, we might see more, or less. We would need either a lot more than an exabyte of data or a lot more than a year to be reasonably sure the bit half-life was long enough for the job. But even an exabyte of data for a year costs 10 times as much as the system we want to buy.

What this thought-experiment tells us is we are now dealing with such large numbers of bits for such a long time that we are never going to know whether the systems we use are good enough:

- The known causes of data loss are too various and too highly correlated for models to produce credible projections.
- Even if we ignore all those causes, the experiments that would be needed to be reasonably sure random bit rot is not significant are too expensive, or take too long, or both.

Our inability to compute how many backup copies we need to achieve a reliability target is something we are just going to have to live with. We are not going to have enough backup copies, and stuff will get lost or damaged.

Measuring Failures

It wasn't until 2007 that researchers started publishing studies of the reliability that actual large-scale storage systems were delivering in practice. Enterprises such as Google[9] and institutions such the Sloan Digital Sky Survey[37] and the Large Hadron Collider[8] were collecting petabytes of data with long-term value that had to remain online to be useful. The annual cost of keeping a petabyte online was more than $1 million.[27] It is easy to see why questions of the economics and reliability of storage systems became the focus of researchers' attention.

Papers at the 2007 File and Storage Technologies (FAST) conference used data from NetApp[35] and Google[30] to study disk-replacement rates in large storage farms. They showed that the manufacturer's MTTF numbers were optimistic. Subsequent analysis of the NetApp data[17] showed that all other components contributed to the storage system failures and that:

"Interestingly, [the earlier studies] found disks are replaced much more frequently (2–4 times) than vendor-specified [replacement rates]. But as this study indicates, there are other storage subsystem failures besides disk failures that are treated as disk faults and lead to unnecessary disk replacements."[17]

Two studies, one at CERN (European Organization for Nuclear Research)[18] and one using data from NetApp,[5] greatly improved on earlier work using data from the Internet Archive.[6,36] They studied *silent data corruption*—events in which the content of a file in storage changes with no explanation or recorded errors—in state-of-the-art storage systems.

The NetApp study looked at the incidence of silent storage corruption in individual disks in RAID arrays. The data was collected over 41 months from NetApp's filers in the field, covering more than 1.5×10^{6} drives. The study found more than 4×10^{5} silent corruption incidents. More than 3×10^{4} of them were not detected until RAID restoration and could thus have caused data loss despite the replication and auditing provided by NetApp's row-diagonal parity RAID.[11]

The CERN study used a program that wrote large files into CERN's various data stores, which represent a broad range of state-of-the-art enterprise storage systems (mostly RAID arrays), and checked them over a period of six months. A total of about 9.7×10^{16} bytes was written and about 1.92×10^{8} bytes were found to have suffered silent corruption, of which about two-thirds were persistent; rereading did not return good data. In other words, about 1.2×10^{-9} of the data written to CERN's storage was permanently corrupted within six months. We can place an upper bound on the bit half-life in this sample of current storage systems by assuming the data was written instantly at the start of the six months and checked instantly at the end; the result is 2×10^{8} or about 10^{-2} times the age of the universe. Thus, to reach the petabyte for a century requirement we would need to improve the performance of current enterprise storage systems by a factor of at least 10^{9}.

Tolerating Failures

Despite manufacturers' claims, current research shows that state-of-the-art storage systems fall so many orders of magnitude below our bit preservation requirements that we cannot expect even dramatic improvements in technology to fill the gap. Maintaining a single replica in a single storage system is not an adequate solution to the bit preservation problem.

Practical digital preservation systems must therefore:

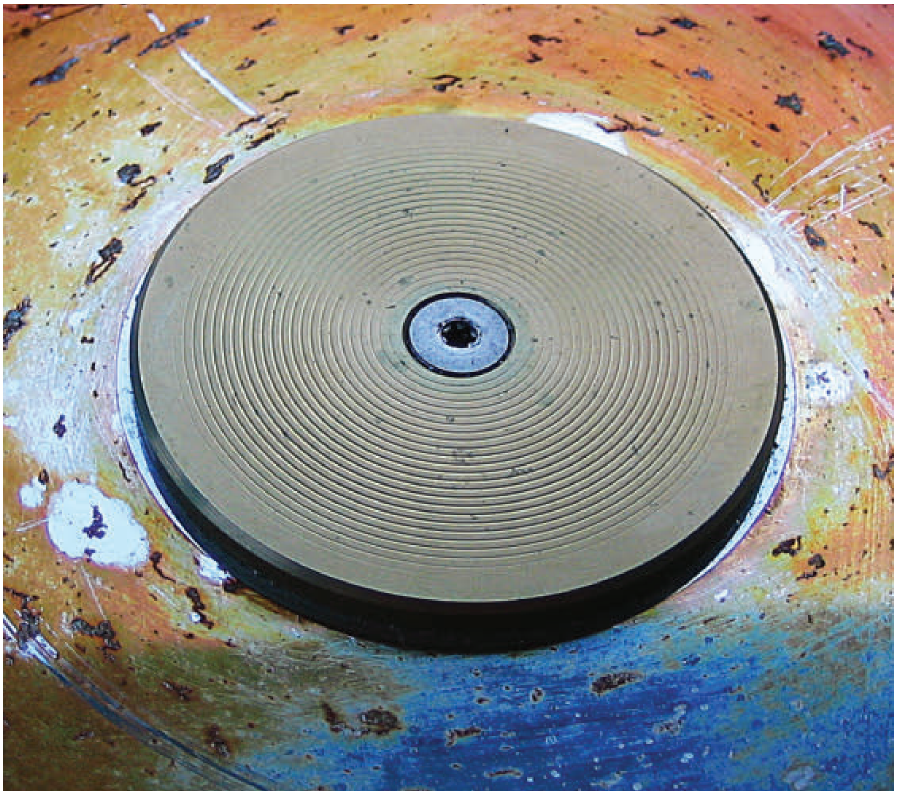
- Maintain more than one copy by *replicating* their data on multiple, ideally different, storage systems.
- Audit or (*scrub*) the replicas to detect damage, and repair it by overwriting the known-bad copy with data from another.

The more replicas and the more frequently they are audited and repaired, the longer the bit half-life we can expect. This is, after all, the basis for the backups and checksums technique in common use. In fact, current storage systems already use such techniques internally—for example, in the form of RAID.[29] Despite this, the bit half-life they deliver is inadequate. Unfortunately, adding the necessary inter-storage-system replication and scrubbing is expensive.

Cost figures from the San Diego Supercomputer Center[c] in 2008 show that maintaining a single online copy of a petabyte for a year costs about $\$1.05\times10^{6}$. A single near-line copy on tape costs about $\$4.2\times10^{5}$ a year. These costs decrease with time, albeit not as fast as raw disk costs. The British Library estimates a 30% per annum decrease. Assuming this rate continues for at least a decade, if you can afford about 3.3 times the first year's cost to store an extra replica for a decade, you can afford to store it indefinitely. So, adding a second replica of a petabyte on disk would cost about $\$3.5\times10^{6}$ and on tape about $\$1.4\times10^{6}$. Adding cost to a preservation effort to increase reliability in this way is a two-edged sword: doing so necessarily increases the risk that preservation will fail for economic reasons.

Further, without detailed understanding of the rates at which different mechanisms cause loss and damage, it still is not possible to answer the question we started with and to know how many replicas would make us as safe as we need to be, and thus the cost of the necessary replication. At small scales the response to this uncertainty is to add more replicas, but as the scale increases this rapidly becomes unaffordable.

Replicating among identical systems is much less effective than replicating among diverse systems. Identical systems are subject to common mode failures—for example, those caused by a software bug in all the systems damaging the same data in each. On the other hand, purchasing and operating a number of identical systems will be considerably cheaper than operating a set of diverse systems.



PHOTOGRAPH BY TARAN RAMPERSAD

Each replica is vulnerable to loss and damage. Unless they are regularly audited they contribute little to increasing bit half-life. The bandwidth and processing capacity needed to scrub the data are both costly, and adding these costs increases the risk of failure. Custom hardware[25] could compute the SHA-1[28] checksum of a petabyte of data in a month, but doing so requires impressive bandwidth—the equivalent of three gigabit Ethernet interfaces running at full speed the entire month. User access to data in long-term storage is typically infrequent; it is therefore rarely architected to pro-

[c] Figures for 2007 are in Moore et al.[27]

vide such high-bandwidth read access. System cost increases rapidly with I/O bandwidth, and the additional accesses to the data (whether on disk or on tape) needed for scrubbing themselves potentially increase the risk of failure.

The point of writing software that reads and verifies the data-systems store in this way is to detect damage and exploit replication among systems to repair it, thereby increasing bit half-life. How well can we do this? RAID is an example of a software technique of this type applied to disks. In practice, the CERN study[18] looking at real RAID systems from the outside showed a significant rate of silent data corruption, and the NetApp study[5] looking at them from the inside showed a significant rate of silent disk errors that would lead to silent data corruption. A study[20] of the full range of current algorithms used to implement RAID found flaws leading to potential data loss in all of them. Both this study, and another from IBM,[16] propose improvements to the RAID algorithms but neither claims it can eliminate silent corruption, or even accurately predict its incidence:

PHOTOGRAPH BY TARAN RAMPERSAD

"While we attempt to use as realistic probability numbers as possible, the goal is not to provide precise data-loss probabilities, but to illustrate the advantage of using a model checker, and discuss potential trade-offs between different protection schemes."[20]

Thus, although intersystem replication and scrubbing are capable of decreasing the incidence of data loss, they cannot eliminate it completely. And the replication and scrubbing software itself will contain bugs that can cause data loss. It must be doubtful that we can implement these techniques well enough to increase the bit half-life of systems with an affordable number of replicas by 10^9.

Magic Media

Considering the difficulties facing disk-drive technology,[12] the reliability storage systems achieve is astonishing, but it clearly isn't enough. News sites regularly feature stories reporting claims that some new storage medium has solved the problem of long-term data storage. Synthetic stone DVDs[23] claimed to last 1,000 years were a recent example. These claims should be treated as skeptically as those of Sun and other storage system manufacturers. It may well be that the media in question are more reliable than their competitors, but as we have seen, raw media reliability is only a part of the story. Our petabyte would be a stack of 2×10^5 stone DVDs. A lot can happen to a stack that big in 100 years. Truly magic media that are utterly reliable would make the problems better, but they would not make them go away completely.

I remember magnetic bubble memory, so I have a feeling of déjà vu, but it is starting to look possible that flash memory, or possibly more exotic solid-state technologies such as memristors or phase-change memory, may supplant disks. There is a lot to like about these technologies for long-term storage, but will they improve storage reliability?

Again, we don't know the answer yet. Despite flash memory's ubiquity, it is not even clear yet how to measure its UBER:

"UBER values can be much better than 10^{-15} but UBER is a strong function of program/erase cycling and subsequent retention time, so UBER specifications must be coupled with maximum specifications for these quantities."[26]

In other words, it depends how you use it, which does not appear to be the case for disk. Flash memory used for long-term data storage, which is written once and read infrequently, should in principle perform very well. And the system-level effects of switching from hard disk to flash can be impressive:

"FAWN [fast array of wimpy nodes] couples low-power embedded CPUs to small amounts of local flash storage, and balances computation and I/O capabilities to enable efficient, massively parallel access to data. ...FAWN clusters can handle roughly 350 key-value queries per joule of energy—two orders of magnitude more than a disk-based system."[3]

Fast CPUs, fast RAM, and fast disks all use lots of power, so the low power draw of FAWN is not a surprise. But the high performance comes from another aspect of disk evolution. Table 1 shows how long it would take to read the whole of a state-of-the-art disk of various generations.

Disks have been getting bigger but they have not been getting equivalently faster. This is to be expected; the data rate depends on the inverse of the diameter of a bit, but the capacity depends on the inverse of the area of a bit. FAWN nodes can read their entire contents very quickly, useful for scrubbing, as well as answering queries.

This is all encouraging, but once it became possible to study the behavior of disk storage at a large scale, it became clear that system-level reliability fell far short of the media specifications. This should make us cautious about predicting a revolution from flash or any other new storage technology.

Economics

Ever since Clayton Christensen published *The Innovator's Dilemma*[10] it has been common knowledge that disk-drive cost per byte halves every two years. So you might argue that you don't need to know how many copies you need to keep your data safe for the long term, you just need to know how many you need to keep it safe for the next few years. After that, you can keep more copies.

In fact, what has happened is the capacity at constant cost has been doubling every two years, which is not quite the same thing. As long as this exponential grows faster than you generate new data, adding copies through time is a feasible strategy.

Alas, exponential curves can be deceiving. Moore's Law has continued to deliver smaller and smaller transistors. A few years ago, however, it effectively ceased delivering faster and faster CPU clock rates. It turned out that, from a business perspective, there were more important things to spend the extra transistors on than making a single CPU faster. Like putting multiple CPUs on a chip.

At a recent Library of Congress meeting, Dave Anderson of Seagate warned[4] that something similar is about to happen to hard disks. Technologies—HAMR (heat-assisted magnetic recording) and BPM (bit pattern media)—are in place to deliver the 2013 disk generation (that is, a consumer 3.5-inch drive holding 8TB). But the business case for building it is weak. The cost of the transition to BPM in particular is daunting.[24] Laptops, netbooks, and now tablets are destroying the market for the desktop boxes that 3.5-inch drives go into. And very few consumers fill up the 2009 2TB disk generation, so what value does having an 8TB drive add? Let alone the problem of how to back up an 8TB drive on your desk!

What is likely to happen—indeed, is already happening—is that the consumer market will transition rather quickly to 2.5-inch drives. This will eliminate the high-capacity $100 3.5-inch drive, since it will no longer be produced in consumer quantities. Consumers will still buy $100 drives, but they will be 2.5 inches and have perhaps one-third the capacity. For a while the $/byte curve will at best flatten, and more likely go up. The problem this poses is that large-scale disk farms are currently built from consumer 3.5-inch drives. The existing players in the market have bet heavily on the exponential cost decrease continuing; if they're wrong, it will be disruptive.

Table 1. The time to read an entire disk of various generations.

1990	240
2000	720
2006	6450
2009	8000
2013	12800

Disks have been getting bigger but they have not been getting equivalently faster. This is to be expected; the data rate depends on the inverse of the diameter of a bit, but the capacity depends on the inverse of the area of a bit.

The Bigger Picture

Our inability to compute how many backup copies we need to achieve a reliability target is something we are just going to have to live with. In practice, we are not going to have enough backup copies, and stuff will get lost or damaged. This should not be a surprise, but somehow it is. The fact that bits can be copied correctly leads to an expectation that they always *will* be copied correctly, and then to an expectation that digital storage will be reliable. There is an odd cognitive dissonance between this and people's actual experience of digital storage, which is that loss and damage are routine occurrences.[22]

The fact that storage is not reliable enough to allow us to ignore the problem of failures is just one aspect of a much bigger problem looming over computing as it continues to scale up. Current long-running petascale high-performance computer applications require complex and expensive checkpoint and restart schemes, because the probability of a failure during execution is so high that restarting from

Table 2. The four cases of message digest comparison.

Digest	Match	No Match
Unchanged	Data OK	Data bad
Changed	Deliberate alteration	Data and/or digest bad

scratch is infeasible. This approach will not scale to the forthcoming generation:

"…it is anticipated that exascale systems will experience various kinds of faults many times per day. It is also anticipated that the current approach for resilience, which relies on automatic or application-level checkpoint-restart, will not work because the time for checkpointing and restarting will exceed the mean time to failure of a full system. …

"Some projections estimate that, with the current technique, the time to checkpoint and restart may exceed the mean time to interrupt of top supercomputers before 2015. This not only means that a computation will do little progress; it also means that fault-handling protocols have to handle multiple errors—current solutions are often designed to handle single errors."[7]

Just as with storage, the numbers of components and interconnections are so large that the incidence of failures is significant, and the available bandwidths are relatively so low that recovering from the failures is time consuming enough that multiple failure situations have to be handled. There is no practical, affordable way to mask the failures from the applications. Application programmers will need to pay much more attention to detecting and recovering from errors in their environment. To do so they will need both the APIs and the system environments implementing them to become much more failure-aware.

API Enhancements

Storage APIs are starting to move in this direction. Recent interfaces to storage services[2] allow the application's write call to provide not just a pointer to the data and a length, but also, optionally, the application's message digest of the data. This allows the storage system to detect whether the data was damaged during its journey from the application to the device, or while it was sitting in the storage device, or being copied back to the application. Recent research has shown the memory buffers[44] and data paths[17] between the application and the storage devices contribute substantially to errors.

Let's take the Amazon S3 (Simple Storage Service) REST API[2] as an example to show that, while these developments are welcome, they are far from a panacea. The PUT request supports an optional (and recommended) Content-MD5 (Message-Digest algorithm 5) header containing the application's digest of the data. The responses to most requests, including PUT, include an ETag (entity tag) header with the service's MD5 of the object. The application can remember the digest it computed before the PUT and, when the PUT returns, verify that the service's digest matches.

Doing so is a wise precaution, but all it really tells the application is that the service knows what the application thinks is the correct digest. The service knows this digest, not because it computed the digest of the correct data, but because the application provided it in the Content-MD5 header. A malign or malfunctioning service could respond correctly to PUT and HEAD requests by remembering the application's digest, without ever storing the data or computing its digest.

The application could try to detect a malign or malfunctioning service by using a GET to obtain the stored data, computing the digest (a) of the returned data, and comparing that with (b) either the digest in the response's ETag header or the digest it computed before the original PUT and remembered (which should be the same). It might seem that there are two cases: if the two message digests match, then the data is OK;[d] otherwise it isn't. There are actually four cases, as shown in Table 2, depending on whether the digest (b) is unchanged or not. The four cases illustrate two problems:

- The bits forming the digest are no different from the bits forming the data; neither is magically incorruptible. A malign or malfunctioning service could return bad data with a digest in the ETag header that matched the data but was not the digest originally computed. Applications need to know whether the digest has been changed. A system for doing so without incorruptible storage is described in Haber et al.[15]
- Given the pricing structure for cloud storage services such as Amazon S3, it is too expensive to extract the entire data at intervals to confirm it is being stored correctly. Some method in which the service computes the digest of the data is needed, but simply asking the service to return the digest of a stored object is not an adequate check.[33] The service must be challenged to prove its object is good. The simplest way to do this is to ask the service to compute the digest of a nonce (a random string of bits) and the object; because the service cannot predict the nonce, a correct response requires access to the data after the request is received. Systems using this technique are described in Maniatis et al.[21] and Shah et al.[38]

Early detection is a good thing: the shorter the time between detection and repair, the smaller the risk that a second error will compromise the repair. But detection is only part of the solution; the system also has to be able to repair the damaged data. It can do so only if it has replicated the data elsewhere—and some deduplication layer has not optimized away this replication.

Conclusion

It would be nice to end on an upbeat note, describing some technological fix that would allow applications to ignore the possibility of failures in their environment, and specifically in long-term storage. Unfortunately, in the real world, failures are inevitable. As systems scale up, failures become more frequent. Even throwing money at the problem can only reduce the incidence of failures, not exclude them entirely.

[d] Assuming the digest algorithm has not been broken, not a safe assumption for MD5.[19]

Applications in the future will need to be much more aware of, and careful in responding to, failures.

The high-performance computing community accurately describes what needs to be done:

"We already mentioned the lack of coordination between software layers with regards to errors and fault management. Currently, when a software layer or component detects a fault it does not inform the other parts of the software running on the system in a consistent manner. As a consequence, fault-handling actions taken by this software component are hidden to the rest of the system. ...In an ideal wor[l]d, if a software component detects a potential error, then the information should propagate to other components that may be affected by the error or that control resources that may be responsible for the error."[7]

In particular, as regards storage, APIs should copy Amazon's S3 by providing optional data-integrity capabilities that allow applications to perform end-to-end checks. These APIs should be enhanced to allow the application to provide an optional nonce that is prepended to the object data before the message digest reported to the application is computed. This would allow applications to exclude the possibility that the reported digest has been remembered rather than recomputed.

Acknowledgments

Grateful thanks are due to Eric Allman, Kirk McKusick, Jim Gettys, Tom Lipkis, Mark Compton, Petros Maniatis, Mary Baker, Fran Berman, Tsutomu Shimomura, and the late Jim Gray. Some of this material was originally presented at iPRES 2008 and subsequently published in the *International Journal of Digital Curation*.[32]

This work was funded by the member libraries of the LOCKSS Alliance, and the Library of Congress' National Digital Information Infrastructure and Preservation Program. Errors and omissions are the author's own.

Related articles on queue.acm.org

Triple-Parity RAID and Beyond
Adam Leventhal
http://queue.acm.org/detail.cfm?id=1670144

Hard Disk Drives: The Good, the Bad and the Ugly!
Jon Elerath
http://queue.acm.org/detail.cfm?id=1317403

You Don't Know Jack about Disks
Dave Anderson
http://queue.acm.org/detail.cfm?id=864058

References

1. Adams, D. *The Hitchhiker's Guide to the Galaxy.* British Broadcasting Corp., 1978.
2. Amazon. Amazon S3 API Reference (Mar. 2006); http://docs.amazonwebservices.com/AmazonS3/latest/API/.
3. Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V. FAWN: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (2009), 1–14.
4. Anderson. D. Hard drive directions (Sept. 2009); http://www.digitalpreservation.gov/news/events/other_meetings/storage09/docs/2-4_Anderson-seagate-v3_HDtrends.pdf.
5. Bairavasundaram, L., Goodson, G., Schroeder, B., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. An analysis of data corruption in the storage stack. In *Proceedings of 6th Usenix Conference on File and Storage Technologies*, (2008).
6. Baker, M., Shah, M., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T.J., Bungale, P. A fresh look at the reliability of long-term digital storage. In *Proceedings of EuroSys2006*, (Apr. 2006).
7. Cappello, F., Geist, A., Gropp, B., Kale, S., Kramer, B., Snir, M. Toward exascale resilience. Technical Report TR-JLPC-09-01. INRIA-Illinois Joint Laboratory on Petascale Computing, (July 2009).
8. CERN. Worldwide LHC Computing Grid, 2008; http://lcg.web.cern.ch/LCG/.
9. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., Grube, R.E. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th Usenix Symposium on Operating System Design and Implementation*, (2006), 205–218.
10. Christensen, C.M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Harvard Business School Press (June 1997), Cambridge, MA.
11. Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., Sankar, S. Row-diagonal parity for double disk failure correction. In *3rd Usenix Conference on File and Storage Technologies* (Mar. 2004).
12. Elerath. J. Hard-disk drives: The good, the bad, and the ugly. *Commun. ACM 52*, 6 (June 2009).
13. Elerath, J.G., Pecht, M. Enhanced reliability modeling of RAID storage systems. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (2007), 175–184.
14. Engler, D. A system's hackers crash course: techniques that find lots of bugs in real (storage) system code. In *Proceedings of 5th Usenix Conference on File and Storage Technologies*, (Feb. 2007).
15. Haber, S., Stornetta, W.S. How to timestamp a digital document. *Journal of Cryptology 3*, 2 (1991), 99–111.
16. Hafner, J.L., Deenadhayalan, V., Belluomini, W., Rao, K. Undetected disk errors in RAID arrays. *IBM Journal of Research & Development 52*, 4/5, (2008).
17. Jiang, W., Hu, C., Zhou, Y., Kanevsky, A. Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics. In *Proceedings of 6th Usenix Conference on File and Storage Technologies*, (2008).
18. Kelemen, P. Silent corruptions. In *8th Annual Workshop on Linux Clusters for Super Computing*, (2007)
19. Klima, V. Finding MD5 collisions—A toy for a notebook. Cryptology ePrint Archive, Report 2005/075; http://eprint.iacr.org/2005/075.
20. Krioukov, A., Bairavasundaram, L.N., Goodson, G.R., Srinivasan, K., Thelen, R., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. Parity lost and parity regained. In Proceedings of 6th Usenix Conference on File and Storage Technologies, (2008).
21. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M., Muliadi, Y. Preserving peer replicas by rate-limited sampled voting. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, (Oct. 2003), 44–59.
22. Marshall, C. "It's like a fire. You just have to move on:" Rethinking personal digital archiving. In *6th Usenix Conference on File and Storage Technologies*, (2008).
23. Mearian, L. Start-up claims its DVDs last 1,000 years. *Computerworld*, (Nov. 2009).
24. Mellor, C. Drive suppliers hit capacity increase difficulties. *The Register*, (July 2010).
25. Michail, H.E., Kakarountas, A.P., Theodoridis, G., Goutis, C.E. A low-power and high-throughput implementation of the SHA-1 hash function. In *Proceedings of the 9th WSEAS International Conference on Computers*, (2005).
26. Mielke, N., Marquart, T., Wu1, N., Kessenich, J., Belgal, H., Schares, E., Trivedi, F., Goodness, E., Nevill, L.R. Bit error rate in NAND flash memories. In *46th Annual International Reliability Physics Symposium*, (2008), 9–19.
27. Moore, R. L., D'Aoust, J., McDonald, R. H., Minor, D. Disk and tape storage cost models. In *Archiving 2007*.
28. National Institute of Standards and Technology (NIST). *Federal Information Processing Standard Publication 180-1: Secure Hash Standard*, (Apr. 1995).
29. Patterson, D. A., Gibson, G., Katz, R.H. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (June 1988), 109–116.
30. Pinheiro, E., Weber, W.-D., Barroso, L. A. Failure trends in a large disk drive population. In *Proceedings of 5th Usenix Conference on File and Storage Technologies*, (Feb. 2007).
31. Prabhakaran, V., Agrawal, N., Bairavasundaram, L., Gunawi, H., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. IRON file systems. In *Proceedings of the 20th Symposium on Operating Systems Principles*, (2005).
32. Rosenthal, D.S.H. Bit preservation: A solved problem? *International Journal of Digital Curation 1*, 5 (2010).
33. Rosenthal, D.S.H. LOCKSS: Lots of copies keep stuff safe. In *NIST Digital Preservation Interoperability Framework Workshop*, (Mar. 2010).
34. Rosenthal, D.S.H., Robertson, T.S., Lipkis, T., Reich, V., Morabito, S. Requirements for digital preservation systems: a bottom-up approach. *D-Lib Magazine 11*, 11 (2005).
35. Schroeder, B., Gibson, G. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of 5th Usenix Conference on File and Storage Technologies* (Feb. 2007).
36. Schwarz, T., Baker, M., Bassi, S., Baumgart, B., Flagg, W., van Imngen, C., Joste, K., Manasse, M., Shah, M. Disk failure investigations at the Internet Archive. In *Work-in-Progress Session, NASA/IEEE Conference on Mass Storage Systems and Technologies*, (2006).
37. SDSS (Sloan Digital Sky Survey), 2008; http://www.sdss.org/.
38. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R. Auditing to keep online storage services honest. In *11th Workshop on Hot Topics in Operating Systems*, (May 2007).
39. Storer, M.W., Greenan, K. M., Miller, E.L., Voruganti, K. Pergamum: replacing tape with energy-efficient, reliable, disk-based archival storage. In *Proceedings of 6th Usenix Conference on File and Storage Technologies*, (2008).
40. Sun Microsystems. Sales Terms and Conditions, Section 11.2, (Dec. 2006); http://store.sun.com/CMTemplate/docs/legal_terms/TnC.jsp#11.
41. Sun Microsystems. ST5800 presentation. Sun PASIG Meeting, (June 2008).
42. Talagala, N. *Characterizing large storage systems: error behavior and performance benchmarks.* Ph.D. thesis, Computer Science Division, University of California at Berkeley, (Oct. 1999).
43. Williams, P., Rosenthal, D. S. H., Roussopoulos, M., Georgis, S. Predicting the archival life of removable hard disk drives. In *Archiving 2008*, (June 2008).
44. Zhang, Y., Rajimwale, A., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. End-to-end data integrity for file systems: A ZFS case study. In *8th Usenix Conference on File and Storage Technologies*, (2010).

David S.H. Rosenthal has been an engineer in Silicon Valley for a quarter of a century, including as a Distinguished Engineer at Sun Microsystems and employee #4 at NVIDIA. For the last decade he has been working on the problems of long-term digital preservation under the auspices of the Stanford Library.

DOI:10.1145/1839676.1839693

Article development led by acmqueue
queue.acm.org

To move forward with programming languages we must first break free from the tyranny of ASCII.

BY POUL-HENNING KAMP

Sir, Please Step Away from the ASR-33!

ONE OF THE naughty details of my Varnish software is that the configuration is written in a domain-specific language that is converted into C source code, compiled into a shared library, and executed at hardware speed. That obviously makes me a programming language syntax designer, and just as obviously I have started to think more about how we express ourselves in these syntaxes.

Rob Pike recently said some very pointed words about the Java programming language, which if you think about it, sounded a lot like the pointed words James Gosling had for C++, and remarkably similar to what Bjarne Stroustrup said about good ol' C.

I have always admired Pike. He was already a giant in the field when I started, and his ability to foretell the future has been remarkably consistent.[1] In front of me I have a tough row to hoe, but I will attempt to argue that this time Pike is merely rearranging the deckchairs of the *Titanic* and that he missed the next big thing by a wide margin.

Pike got fed up with C++ and Java and did what any self-respecting hacker would do: he created his own language—better than Java, better than C++, better than C—and he called it Go.

But did he *go* far enough?

The Go language does not in any way look substantially different from any of the other programming languages. Fiddle a couple of glyphs here and there and you have C, C++, Java, Python, Tcl, or whatever.

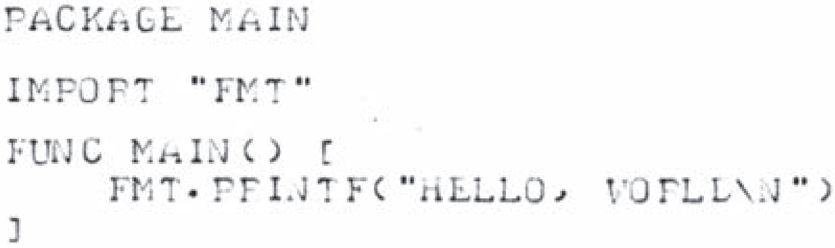
Programmers are a picky bunch when it comes to syntax, and it is a sobering thought that one of the most rapidly adopted programming languages of all time, Perl, barely had one for the longest time. Ironically, what syntax designers are really fighting about is not so much the proper and best syntax for the expression of ideas in a machine-understandable programming language as it is the proper and most efficient use of the ASCII table real estate.

It's all ASCII to me…

There used to be a programming language called ALGOL, the lingua franca of computer science back in its heyday. ALGOL was standardized around 1960 and dictated about a dozen mathematical glyphs such as ×, ÷, ¬, and the very readable subscripted 10 symbol, for use in what today we call scientific notation. Back then computers were built by hand and had one-digit serial numbers. Having a teletypewriter customized for your programming language was the least of your worries.

A couple of years later came the APL programming language, which included an extended character set containing a lot of math symbols. I am told that APL still survives in certain obscure corners of insurance and economics modeling.

Then ASCII happened around 1963, and ever since, programming languages have been trying to fit into it. (Wikipedia claims that ASCII grew the backslash [\]



```
PACKAGE MAIN

IMPOFT "FMT"

FUNC MAIN() [
    FMT.PFINTF("HELLO, VOFLL\N")
]
```

Result of the Go Language ASR33 Compatibility Test.

specifically to support ALGOL's /\ and \/ Boolean operators. No source is provided for the claim.)

The trouble probably started for real with the C programming language's need for two kinds of `and` and `or` operators. It could have used just `or` and `bitor`, but | and || saved one and three characters, which on an ASR-33 teletype amounts to 1/10 and 3/10 second, respectively.

It was certainly a fair trade-off—just think about how fast you type yourself—but the price for this temporal frugality was a whole new class of hard-to-spot bugs in C code.

Niklaus Wirth tried to undo some of the damage in Pascal, and the bickering over `begin` and `end` would no } take.

C++ is probably the language that milks the ASCII table most by allowing templates and operator overloading. Until you have inspected your data types, you have absolutely no idea what + might do to them (which is probably why there never was enough interest to stage an International Obfuscated C++ Code Contest, parallel to the IOCCC for the C language).

C++ stops short of allowing the programmer to create new operators. You cannot define :-: as an operator; you have to stick to the predefined set. If Bjarne Stroustrup had been more ambitious on this aspect, C++ could have beaten Perl by 10 years to become the world's second write-only programming language, after APL.

How desperate the hunt for glyphs is in syntax design is exemplified by how Guido van Rossum did away with the canonical scope delimiters in Python, relying instead on indentation for this purpose. What could possibly be of such high value that a syntax designer would brave the controversy this caused? A high-value pair of matching glyphs, { and }, for other use in his syntax could. (This decision also made it impossible to write Fortran programs in Python, a laudable achievement in its own right.)

The best example of what happens if you do the opposite is John Ousterhout's Tcl programming language. Despite all its desirable properties—such as being created as a language to be embedded in tools—it has been widely spurned, often with arguments about excessive use of, or difficult-to-figure-out placement of, {} and [].

My disappointment with Rob Pike's Go language is that the rest of the world has moved on from ASCII, but he did not. Why keep trying to cram an expressive syntax into the straitjacket of the 95 glyphs of ASCII when Unicode has been the new black for most of the past decade?

Unicode has the entire gamut of Greek letters, mathematical and technical symbols, brackets, brockets, sprockets, and weird and wonderful glyphs such as "Dentistry symbol light down and horizontal with wave" (0x23c7). Why do we still have to name variables OmegaZero when our computers now know how to render 0x03a9+0x2080 properly?

The most recent programming language syntax development that had anything to do with character sets apart from ASCII was when the ISO-C standard committee adopted trigraphs to make it possible to enter C source code on computers that do not have ASCII's 95 characters available—a bold and decisive step in the wrong direction.

While we are at it, have you noticed that screens are getting wider and wider these days, and that today's text processing programs have absolutely no problem with multiple columns, insert displays, and hanging enclosures being placed in that space?

But programs are still decisively vertical, to the point of being horizontally challenged. Why can't we pull minor scopes and subroutines out in that right-hand space and thus make them supportive to the understanding of the main body of code?

And need I remind anybody that you cannot buy a monochrome screen anymore? Syntax-coloring editors are the default. Why not make color part of the syntax? Why not tell the compiler about protected code regions by putting them on a framed light gray background? Or provide hints about likely and unlikely code paths with a green or red background tint?

For some reason computer people are so conservative we still find it more uncompromisingly important for our source code to be compatible with a Teletype ASR-33 terminal and its 1963-vintage ASCII table than it is for us to be able to express our intentions clearly.

And, yes, me too: I wrote this in vi(1), which is why the article does not have all the fancy Unicode glyphs in the first place. **C**

Related articles on queue.acm.org

A Conversation with Arthur Whitney
http://queue.acm.org/detail.cfm?id=1531242

You're Doing It Wrong
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=1814327

How Not to Write Fortran in Any Language
Donn Seeley
http://queue.acm.org/detail.cfm?id=1039535

Reference

1. Pike, R. Systems software research is irrelevant; http://herpolhode.com/rob/utah2000.pdf.

Poul-Henning Kamp (phk@FreeBSD.org) has programmed computers for 26 years and is the inspiration behind bikeshed.org. His software has been widely adopted as "under the hood" building blocks. His most recent project is the Varnish HTTP accelerator, which is used to speed up large Web sites such as Facebook.

DOI:10.1145/1839676.1839694

For workloads with abundant parallelism, GPUs deliver higher peak computational throughput than latency-oriented CPUs.

BY MICHAEL GARLAND AND DAVID B. KIRK

Understanding Throughput-Oriented Architectures

MUCH HAS BEEN written about the transition of commodity microprocessors from single-core to multicore chips, a trend most apparent in CPU processor families. Commodity PCs are now typically built with CPUs containing from two to eight cores, with even higher core counts on the horizon. These chips aim to deliver higher performance by exploiting modestly parallel workloads arising from either the need to execute multiple independent programs or individual programs that themselves consist of multiple parallel tasks, yet maintain the same level of performance as single-core chips on sequential workloads.

A related architectural trend is the growing prominence of throughput-oriented microprocessor architectures. Processors like Sun's Niagara and NVIDIA's graphics processing units, or GPUs, follow in the footsteps of earlier throughput-oriented processor designs but have achieved far broader use in commodity machines. Broadly speaking, they focus on executing parallel workloads while attempting to maximize total throughput, even though sacrificing the serial performance of a single task may be required. Though improving total throughput at the expense of increased latency on individual tasks is not always a desirable trade-off, it is unquestionably the right design decision in many problem domains that rely on parallel computations, including real-time computer graphics, video processing, medical-image analysis, molecular dynamics, astrophysical simulation, and gene sequencing.

Modern GPUs are fully programmable and designed to meet the needs of a problem domain—real-time computer graphics—with tremendous inherent parallelism. Furthermore, real-time graphics places a premium on the total amount of work that can be accomplished within the span of a single frame (typically lasting 1/30 second). Due to their historical development, GPUs have evolved as exemplars of throughput-oriented processor architecture. Their emphasis on throughput optimization and their expectation of abundant available parallelism is more aggressive than many other throughput-oriented architectures. They are also widely available and easily programmable. NVIDIA

» key insights

- **Throughput-oriented processors tackle problems where parallelism is abundant, yielding design decisions different from more traditional latency-oriented processors.**
- **Due to their design, programming throughput-oriented processors requires much more emphasis on parallelism and scalability than programming sequential processors.**
- **GPUs are the leading exemplars of modern throughput-oriented architecture, providing a ubiquitous commodity platform for exploring throughput-oriented programming.**

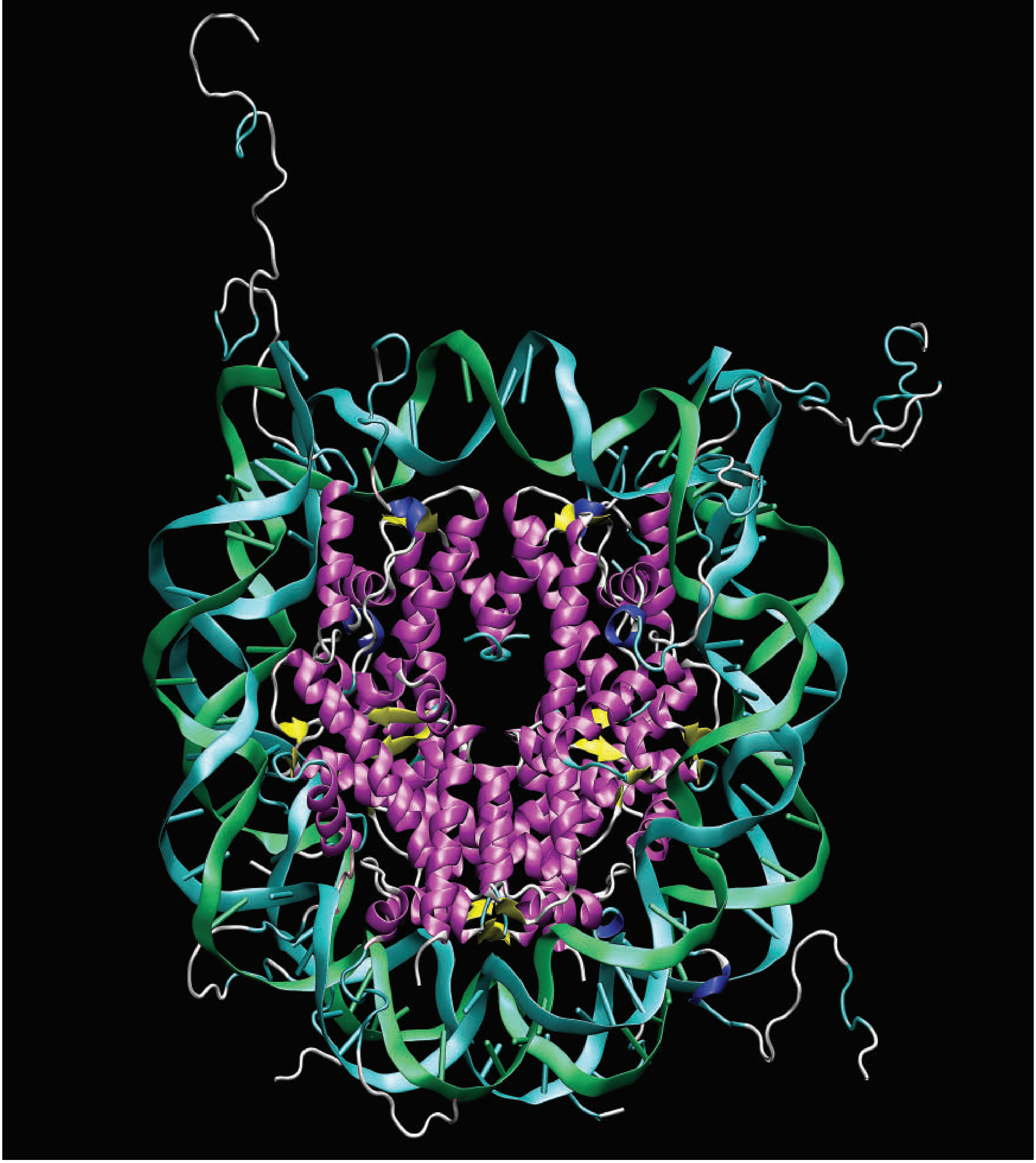


Figure 1. Throughput-oriented processors like the NVIDIA Tesla C2050 deliver substantially higher performance on intrinsically parallel computations, including molecular dynamics simulations.

released its first GPU supporting the CUDA parallel computing architecture in 2006 and is currently shipping its third-generation CUDA architecture, code-named "Fermi,"[24] released in 2010 in the Tesla C2050 and other processors.

Figure 1 is a nucleosome structure (with 25,095 atoms) used in benchmarking the AMBER suite of molecular dynamics simulation programs. Many core computations performed in molecular dynamics are intrinsically parallel, and AMBER recently added CUDA-accelerated computations (http://ambermd.org/gpus/). Its Generalized Born implicit solvent calculation for this system running on the eight cores of a dual four-core Intel Xeon E5462 executes at a rate of 0.06 nanoseconds of simulation time per day of computation. The same calculation running on an NVIDIA Tesla C2050 executes the simulation at a rate of 1.04 ns/day, roughly 144 times more work per day than a single sequential core and just over 17 times the throughput of all eight cores.

Using the GPU as a case study, this article explores the fundamental architectural design decisions differentiating throughput-oriented processors from their more traditional latency-oriented counterparts. These architectural differences also lead to an approach to parallel programming that is qualitatively different from the parallel thread models prevalent on today's CPUs.

Throughput-Oriented Processors

Two fundamental measures of processor performance are task latency (time elapsed between initiation and completion of some task) and throughput (total amount of work completed per unit time). Processor architects make many carefully calibrated trade-offs between latency and throughput optimization, since improving one could degrade the other. Real-world processors tend to emphasize one over the other, depending on the workloads they are expected to encounter.

Traditional scalar microprocessors are essentially latency-oriented architectures. Their goal is to minimize the running time of a single sequential program by avoiding task-level latency whenever possible. Many architectural techniques, including out-of-order execution, speculative execution, and sophisticated memory caches, have been developed to help achieve it. This traditional design approach is predicated on the conservative assumption that the parallelism available in the workload presented to the processor is fundamentally scarce. Single-core scalar CPUs typified by the Intel Pentium IV were aggressively latency-oriented. More recent multicore CPUs (such as the Intel Core2 Duo and Core i7) reflect a trend toward somewhat less-aggressive designs that expect a modest amount of parallelism.

Throughput-oriented processors, in contrast, arise from the assumption that they will be presented with workloads in which parallelism is abundant. This fundamental difference leads to architectures that differ from traditional sequential machines. Broadly speaking, throughput-oriented processors rely on three key architectural features: emphasis on many simple processing cores, extensive hardware multithreading, and use of single-instruction, multiple-data, or SIMD, execution. Aggressively throughput-oriented processors, exemplified by the GPU, willingly sacrifice single-thread execution speed to increase total computational throughput across all threads.

No successful processor can afford to optimize aggregate task throughput while completely ignoring single-task latency or vice versa. Different processors may also vary in the degree they emphasize one over the other; for instance, individual throughput-oriented architectures may not use all three architectural features just listed. Also worth noting is that several architectural strategies, including pipelining, multiple issue, and out-of-order execution, avoid task-level latency by improving instruction-level throughput.

Hardware multithreading. A computation in which parallelism is abundant can be decomposed into a collection of concurrent sequential tasks that may potentially be executed in parallel, or simultaneously, across many threads. We view a thread as a virtualized scalar processor, typically meaning each thread has a program counter, register file, and associated processor state. A thread is thus able to execute the instruction stream corresponding to a single sequential task. Note this model of threads says nothing about the way concurrent threads are scheduled; for instance, whether they are scheduled fairly (any thread ready to run is eventually executed) is a separate issue.

It is well known that multithreading, whether in hardware[31] or software,[4] provides a way of tolerating latency. If a given thread is prevented from running because it is waiting for an instruction to make its way through a pipelined functional unit, data to arrive from external DRAM or some other event, a multithreaded system can allow another unblocked thread to run. That is, the long-latency operations of a single thread can be hidden or covered by ready-to-run work from another thread. This focus on tolerating latency, where processor utilization does not suffer simply because a fraction of the active threads are blocked, is a hallmark of throughput-oriented processors.

Hardware multithreading as a design strategy for improving aggregate performance on parallel workloads has a long history. The peripheral processors of the Control Data Corp. CDC 6600 developed in the 1960s and the Heterogeneous Element Processor (HEP) system[28] developed in the late 1970s are notable examples of the early use of hardware multithreading. Many more multithreaded processors have been designed over the years[31]; for example, the Tera,[1,2] Sun Niagara,[18] and NVIDIA GPU22 architectures all use aggressive multithreading to achieve high-throughput performance on parallel workloads, all with interleaved multithreading.[21] Each is capable of switching between threads at each cycle. Thus the execution of threads is interleaved at extremely fine granularity, often at the instruction level.

Blocking multithreading is a coarser-grain strategy in which a thread might run uninterrupted until encountering a long-latency operation, at which point a different thread is selected for execution. The streaming processors Imagine,[16] Merrimac,[9] and SPI Storm[17] are notable examples of throughput-oriented architectures adopting this strategy. These machines explicitly partition programs into bulk load/store operations on entire data blocks and "kernel" tasks in which memory accesses are restricted to on-chip blocks loaded on their behalf. When a kernel finishes processing its on-chip data, a different task in which required memory blocks have been loaded onto the chip is executed. Overlapping the bulk data transfer for one or more tasks while another is executing hides memory-access latency. Strategic placement of kernel boundaries where context switches occur can also substantially reduce the amount of state that must be retained between task executions.

A third strategy called simultaneous multithreading[30] allows different threads to simultaneously issue instructions to independent functional units and is used to improve the efficiency of superscalar sequential processors without having to find instruction-level parallelism within a single thread. It is likewise used by NVIDIA's Fermi architecture[24] in place of intra-thread dual issue to achieve higher utilization.

The design of the HEP,[28] Tera,[2] and NVIDIA G80[22] processors highlights an instructive characteristic of some throughput-oriented processors: none provides a traditional cache for load/store operations on external memory, unlike latency-oriented processors (such as typical CPUs) that expend substantial chip area on sophisticated cache subsystems. These machines are able to achieve high throughput in the absence of caches because they assume there is sufficient parallel work available to hide the latency of off-chip memory accesses. Unlike previous NVIDIA processors, the Fermi architecture provides a cache hierarchy for external memory accesses but still relies on extensive multithreading for latency tolerance.

Many simple processing units. The high transistor density in modern semiconductor technologies makes it feasible for a single chip to contain multiple processing units, raising the question of how to use the available area on the chip to achieve optimal performance: one very large processor, a handful of large processors, or many small processors?

Designing increasingly large single-processor chips is unattractive.[6] The strategies used to obtain progressively higher scalar performance (such as out-of-order execution and aggressive speculation) come at the price of rapidly increasing power consumption; incremental performance gains incur increasingly large power costs.[15] Thus, while increasing the power consumption of a single-threaded core is physically possible, the potential performance improvement from more aggressive speculation appears insignificant by comparison. This analysis has led to an industrywide transition toward multicore chips, though their designs remain fundamentally latency-oriented. Individual cores maintain roughly comparable scalar performance to earlier generations of single-core chips.

Throughput-oriented processors achieve even higher levels of performance by using many simple, and hence small, processing cores.[10] The individual processing units of a throughput-oriented chip typically execute instructions in the order they appear in the program, rather than trying to dynamically reorder instructions for out-of-order execution. They also generally avoid speculative execution and branch prediction. These architectural simplifications often reduce the speed with which a single thread completes its computation. However, the resulting savings in chip area allow for more parallel processing units and correspondingly higher total throughput on parallel workloads.

SIMD execution. Parallel processors frequently employ some form of single-instruction, multiple-data, or SIMD, execution[12] to improve their aggregate throughput. Issuing a single instruction in a SIMD machine applies the given operation to potentially many data operands; SIMD addition might, for example, perform pairwise addition of two 64-element sequences. As with multithreading, SIMD execution has a long history dating to at least the 1960s.

Aggressively throughput-oriented processors, exemplified by the GPU, willingly sacrifice single-thread execution speed to increase total computational throughput across all threads.

Most SIMD machines can be classified into two basic categories. First is the SIMD processor array, typified by the ILLIAC IV developed at the University of Illinois,[7] the Thinking Machines CM-2,[29] and the MasPar Computer Corp. MP-1.[5] All consisted of a large array of processing elements (hundreds or thousands) and a single control unit that would consume a single instruction stream. The control unit would broadcast each instruction to all processing elements that would then execute the instruction in parallel.

The second category is the vector processor, exemplified by the Cray-1[25] and numerous other machines[11] that augment a traditional scalar instruction set with additional vector instructions operating on data vectors of some fixed width—64-element vectors in the Cray-1 and four-element vectors in the most current vector extensions (such as the x86 Streaming SIMD Extensions, or SSE). The operation of a vector instruction, like vector addition, may be performed in a pipelined fashion (as on the Cray-1) or in parallel (as in current SSE implementations). Several modern processor families, including x86 processors from Intel and AMD and the ARM Cortex-A series, provide vector SIMD instructions that operate in parallel on 128-bit (such as four 32-bit integer) values. Programmable GPUs have long made aggressive use of SIMD; current NVIDIA GPUs have a SIMD width of 32. Many recent research designs, including the Vector IRAM,[19] SCALE,[20] and Imagine and Merrimac streaming processors,[9,16] have also used SIMD architectures to improve efficiency.

SIMD execution is attractive because, among other things, it increases the amount of resources that can be devoted to functional units rather than control logic. For instance, 32 floating-point arithmetic units coupled with a single control unit takes less chip area than 32 arithmetic units with 32 separate control units. The desire to amortize the cost of control logic over numerous functional units was the key motivating factor behind even the earliest SIMD machines.[7]

However, devoting less space to control comes at a cost. SIMD execution delivers peak performance when parallel tasks follow the same execution trace and can suffer when heterogeneous tasks follow completely different execution traces. The efficiency of SIMD architectures depends on the availability of sufficient amounts of uniform work. In practice, sufficient uniformity is often present in abundantly parallel workloads, since it is more likely that a pool of 10,000 concurrent tasks consists of a small number of task types rather than 10,000 completely disparate computations.

GPUs

Programmable GPUs are the leading exemplars of aggressively throughput-oriented processors, taking the emphasis on throughput further than the vast majority of other processors and thus offering tremendous potential performance on massively parallel problems.[13]

Historical perspective. Modern GPUs have evolved according to the needs of real-time computer graphics, two aspects of which are of particular importance to understanding the development of GPU designs: it is an extremely parallel problem, and throughput is its paramount measure of performance.

Visual applications generally model the environments they display through a collection of geometric primitives, with triangles the most common. The most widely used techniques for producing images from these primitives proceed through several stages where processing is performed on each triangle, triangle corner, and pixel covered by a triangle. At each stage, individual triangles/vertices/pixels can be processed independently of all others. An individual scene can easily paint millions of pixels at a time, thus generating a great deal of completely parallel work. Furthermore, processing an element generally involves launching a thread to execute a program—usually called a shader—written by the developer. Consequently, GPUs are specifically designed to execute literally billions of small user-written programs per second.

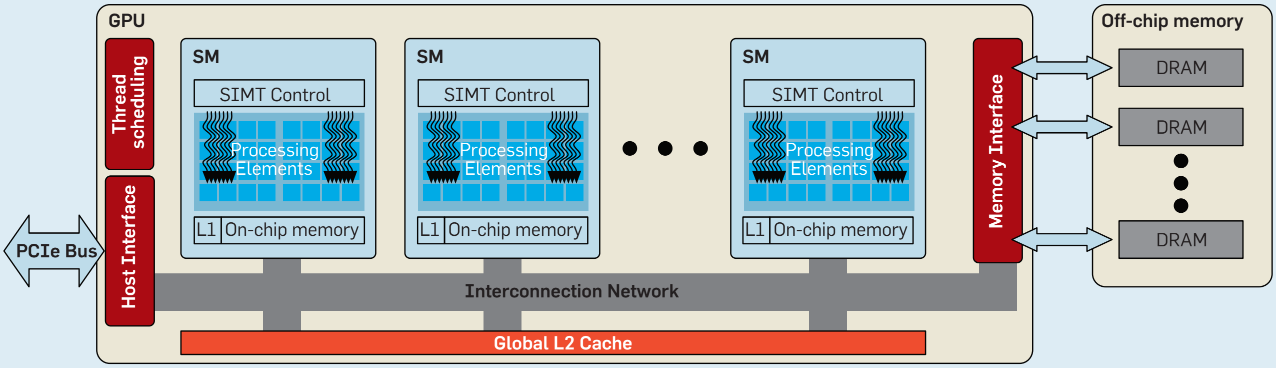
Most real-time visual applications are designed to run at a rate of 30–60 frames per second. A graphics system is therefore expected to generate, render, and display images of visually complex worlds within 33ms. Since it must complete many millions of independent tasks within this timeframe, the time to complete any one of these tasks is relatively unimportant. But the total amount of work that can be completed within 33ms is of great importance, as it is generally closely correlated with the visual richness of the environment being displayed.

Their role in accelerating real-time graphics has also made it possible for GPUs to become mass-market devices, and, unlike many earlier throughput-oriented machines, they are also widely available. Since late 2006, NVIDIA has shipped almost 220 million CUDA-capable GPUs—several orders of magnitude more than historical massively parallel architectures like the CM-2 and MasPar machines.

NVIDIA GPU architecture. Beginning with the G80 processor released in late 2006, all modern NVIDIA GPUs support the CUDA architecture for parallel computing. They are built around an array of multiprocessors, referred to as streaming multiprocessors, or SMs.[22,24] Figure 2 diagrams a representative Fermi-generation GPU like the GF100 processor used in the Tesla C2050. Each multiprocessor supports on the order of a thousand co-resident threads and is equipped with a large register file, giving each thread its own dedicated set of registers. A high-end GPU with many SMs can thus sustain tens of thousands of threads simultaneously. Multiprocessors contain many scalar processing elements that execute the instructions issued by the running threads. Each multiprocessor also contains high-bandwidth, low-latency on-chip shared memory, while at the same time providing its threads with direct read/write access to off-chip DRAM. The Fermi architecture can configure its 64KB of per-SM memory as either a 16KB L1 cache and 48KB RAM or a 48KB L1 cache and 16KB RAM. It also provides a global 768KB L2 cache shared by all SMs. The table here summarizes the capacity of a single SM for the three generations of NVIDIA CUDA-capable GPUs.

The SM multiprocessor handles all thread creation, resource allocation, and scheduling in hardware, interleaving the execution of threads at the instruction-level with essentially zero overhead. Allocation of dedicated registers to all active threads means there is no state to save/restore when switching between threads. With all thread management performed in hardware, the cost of employing many threads is minimal. For example, a Tesla C2050 executing the increment() kernel in Figure 3 will create, execute, and retire threads at a rate of roughly 13 billion threads/sec.

Figure 2. NVIDIA GPU consisting of an array of multithreaded multiprocessors.



To manage its large population of threads efficiently, the GPU employs a single-instruction, multiple-thread, or SIMT, architecture in which threads resident on a single SM are executed in groups of 32, called warps, each executing a single instruction at a time across all its threads. Warps are the basic unit of thread scheduling, and in any given cycle the SM is free to issue an instruction from any runnable warp. The threads of a warp are free to follow their own execution path, and all such execution divergence is handled automatically in hardware. However, it is obviously more efficient for threads to follow the same execution path for the bulk of the computation. Different warps may follow different execution paths without penalty.

While SIMT architectures share many performance characteristics with SIMD vector machines, they are, from the programmer's perspective, qualitatively different. Vector machines are typically programmed with either vector intrinsics explicitly operating on vectors of some fixed width or compiler autovectorization of loops. In contrast, SIMT machines are programmed by writing a scalar program describing the action of a single thread. A SIMT machine implicitly executes groups of independent scalar threads in a SIMD fashion, whereas a vector machine explicitly encodes SIMD execution in the vector operations in the instruction stream it is given.

CUDA programming model. The CUDA programming model[23,26] provides a minimalist set of abstractions for parallel programming on massively multithreaded architectures like the NVIDIA GPU. A CUDA program is organized into one or more threads executing on a host processor and one or more parallel kernels that can be executed by the host thread(s) on a parallel device.

Individual kernels execute a scalar sequential program across a set of parallel threads. The programmer organizes the kernel's threads into thread blocks, specifying for each kernel launch the number of blocks and number of threads per block to be created. CUDA kernels are thus similar in style to a blocked form of the familiar single-program, multiple-data, or SPMD, paradigm. However, CUDA is somewhat more flexible than most SPMD systems in that the host program is free to customize the number of threads and blocks launched for a particular kernel at each invocation. A thread block is a group of parallel threads that may synchronize with one another at a per-block barrier and communicate among themselves through per-block shared memory. Threads from different blocks may coordinate with one another via atomic operations on variables in the global memory space visible to all threads. There is an implicit barrier between successive dependent kernels launched by the host program.

The NVIDIA CUDA Toolkit (http://www.nvidia.com/cuda) includes a C compiler equipped with a small set of language extensions for writing CUDA programs. Figure 3 sketches a simple CUDA program fragment illustrating these extensions. The `__global__` modifier indicates the `increment()` function is a kernel entry point and may be called only when launching a kernel. Unmodified functions and those functions explicitly marked `__host__` are normal C functions. The host program launches kernels using the function-call-like syntax `increment<<<B, T>>>(...)`, indicating the function `increment()` will be launched in parallel across `B` blocks of `T` threads each. The blocks of a kernel are numbered using two-dimensional indices visible to the kernel as the special variables `blockIdx.x` and `blockIdx.y`, ranging from `0` to `gridDim.x-1` and `gridDim.y-1`, respectively. Similarly, the threads of a block are numbered with three-dimensional indices `threadIdx.x`, `threadIdx.y`, `threadIdx.z`; the extent of the block in each dimension is given by `blockDim.x`, `blockDim.y`, and `blockDim.z`.

The function `parallel_increment()` accepts an array *x* of *n* elements and launches a parallel kernel with at least one thread for each element organized into blocks of 256 threads each. Since the data in the example is one-dimensional, the code in Figure 3 uses one-dimensional indices. Also, since every thread in this computation is completely independent, deciding to use 256 threads per block in our implementation was largely arbitrary. Every thread of the kernel computes a globally unique index *i* from its local `thread-`

Figure 3. Trivial CUDA C kernel for incrementing each element of an array.

```
__global__ void increment(float *x, int n)
{
    // Each thread will process 1 element, which
    // is determined from the thread's index.
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    if( i<n )  x[i] = x[i] + 1;
}

__host__ void parallel_increment(float *x, int n)
{
    // Launch increment() kernel with 1 thread
    // per element, grouped into ⌈n/256⌉ blocks
    // of 256 threads each.
    increment<<<ceil(n/256), 256>>>(x, n);
}
```

Capacity of each SM over three GPU generations.

	G8x/G9x	GT2xx	GF100
Registers (32-bit)	8192	16384	32768
Co-resident threads	768	1024	1536
Independent warps	24	32	48
Shared memory (KB)	16	16	48/16
L1 cache (KB)	—	—	16/48
L2 cache (KB per chip)	—	—	768

Idx and the blockIdx of its block. It increments the value of x_i by 1 if $i < n$—a conditional check required since n need not be a multiple of 256.

Throughput-Oriented Programming

Scalability is the programmer's central concern in designing efficient algorithms for throughput-oriented machines. Today's architectural trends clearly favor increasing parallelism, and effective algorithmic techniques must scale with hardware parallelism. Some techniques suitable for four parallel threads may be entirely unsuitable for 4,000 parallel threads. Running thousands of threads at a time, GPUs are a powerful platform for exploring scalable algorithms and a leading indicator for algorithm design on future throughput-oriented architectures.

Abundant parallelism. Throughput-oriented programs must expose substantial amounts of fine-grain parallelism, fulfilling the expectations of the architecture. Exploiting multicore CPUs obviously requires exposing parallelism as well, but a programmer's mental model of parallelism on a throughput-oriented processor is qualitatively different from multicore. A four-core CPU can be fully utilized by four to eight threads. Thread creation and scheduling are computationally heavyweight, since they can involve the saving and restoration of processor state and relatively expensive calls to the operating system kernel. In contrast, a GPU typically requires thousands of threads to cover memory latency and reach full utilization, while thread scheduling is essentially without cost.

Consider computing the product $y=Ax$, where A is a $n \times n$ matrix, and x is an n-element vector. For sparse problems, because the vast majority of matrix entries is 0, A is best represented using a data structure that stores only its non-zero elements. The algorithm for sparse matrix-vector multiplication (SpMV) would look like this:

```
procedure spmv(y, A, x):
   for each row i:
    y[i] = 0
     for each non-zero column
j:
       y[i] += A[i,j] * x[j]
```

Since each row is processed independently, a simple CUDA implementation would assign a separate thread to each row. For large matrices, this could easily expose millions of threads of parallelism. However, for smaller matrices with only a few thousand rows, this level of parallelism might be insufficient, so an efficient implementation could instead assign multiple threads to process each row. In the most extreme case, each non-zero element could be assigned to a separate thread.

GPUs are specifically designed to execute literally billions of small user-written programs per second.

Figure 4 plots an experiment measuring the performance of three different parallel granularities: one thread/row, 32 threads/row, and one thread/non-zero.[3] These tests use synthetic matrices with a constant number of entries distributed across a variable number of rows ranging from one row with four million entries on the left to four million rows of one entry each on the right. The maximal parallelism resulting from assigning one thread per non-zero element yields the most efficient implementation when there are few rows but suffers from lower absolute performance due to its need for inter-thread synchronization. For intermediate row counts, assigning 32 threads per row is the best solution, while assigning one thread per row is best when the number of rows is sufficiently large.

Calculation is cheap. Computation generally costs considerably less than memory transfers, particularly external memory transfers that frequently require hundreds of cycles to complete. The fact that the cost of memory access has continued to increase and is now quite high relative to the cost of computation is often referred to as the "memory wall." The energy required to move data between the chip and external DRAM is also far higher than required to operate an on-chip functional unit. In a 45nm process, a 64-bit integer addition unit expends roughly 1pJ (picojoule), and a 64-bit floating point fused multiply add, or FMA, unit requires around 100pJ. In contrast, reading a 64-bit value from external DRAM requires on the order of 2,000pJ.[8]

The high relative cost of accessing memory affects both latency and throughput-oriented processors, since the cost is the result of the physical properties of semiconductor technology. However, the performance consequences of external memory references for throughput-oriented processors can be more significant; these processors are designed to reach a higher peak computational throughput and may have a higher peak throughput-to-bandwidth ratio than latency-oriented processors. More important, they seek to tolerate rather than avoid latency. To hide the latency of frequent movement of data to/from main memory requires either more threads or more work per thread, generally requiring larger data sets.

The best performance is typically achieved when calculation is more common than data access. Performing roughly 10 to 20 operations per word of data loaded from memory is ideal, and it may be preferable to locally recompute values rather than store frequently needed values in external memory. Consider a simple example of computing a moderately expensive function like sin θ for 256 unique values of θ. Tabulating all 256 possible values would require little space, but accessing them from external memory would require hundreds of cycles. In the same amount of time, a thread could execute perhaps 50 to 100 instructions that could be used to compute the result and leave the memory bandwidth available for other uses.

Divide and conquer. Divide-and-conquer methods often yield effective parallel algorithms, even for apparently serial problems. Consider the merging of two sorted sequences *A* and *B*, a common problem for which most computer science students learn a sequential solution like this:

```
function merge1(A, B):
 if empty(A): return B
 if empty(B): return A

 if first(A)<first(B):
     return first(A) +
     merge1(rest(A), B)
 else:
     return first(B) +
     merge1(A, rest(B))
```

A related divide-and-conquer algorithm picks an element *s* from either *A* or *B* and partitions both sequences into those elements A_1,B_1 that are less than *s* and elements A_2,B_2 that are not less than *s*. Having split the input sequences, constructing the merged sequence is simply a matter of recursively merging A_i with B_i. The code for doing it would look like this:

```
function merge2(A, B):
 if empty(A): return B
 if empty(B): return A

 s = select_an_element(A,B)
 A1, A2 = partition(A, s)
 B1, B2 = partition(B, s)

 return merge2(A1,B1) + [s] +
 merge2(A2,B2)
```

This approach is reminiscent of quicksort though more efficient since *A* and *B* are both sorted. If *s* is drawn from *A*, "partitioning" *A* is trivial, since the elements less than *s* are simply those preceding *s*, and the corresponding point at which *s* splits *B* can be found through binary search.

This divide-and-conquer method can lead to an inherently parallel algorithm by picking a sorted sequence of *k* splitting elements s_1, ..., s_k. These splitters partition both *A* and *B* into *k*+1 subsequences that can be merged independently like this:

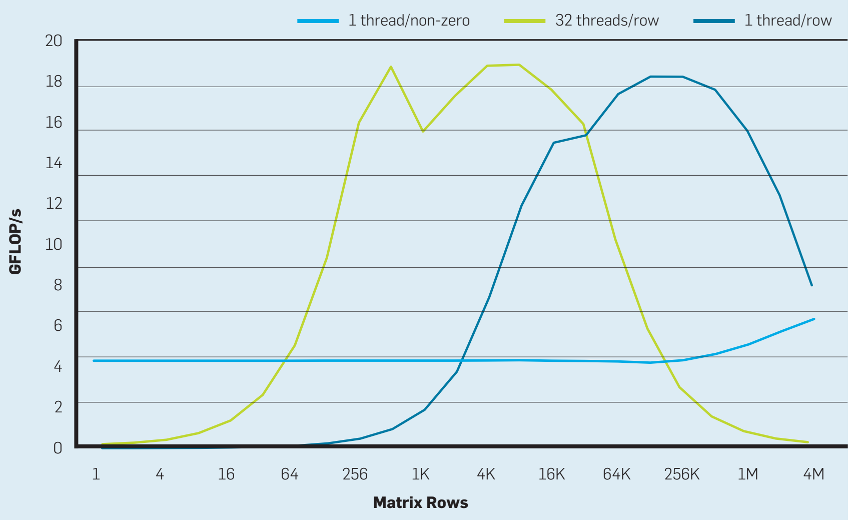
```
function merge3(A, B):
 if empty(A): return B
 if empty(B): return A

 // Pick k elements from A
 and B, in sorted order
 S = [s1, ..., sk] = select_
 elements(A, B, k)
 A0, ..., Ak] = partition(A,
 S)
 [B0, ..., Bk] = partition(B,
 S)

 return merge3(A0,B0) + [s1] +
 ... + [sk] + merge3(Ak,Bk)
```

Since each recursive merge is independent, this is an intrinsically parallel algorithm. Merge algorithms of this form have been used in the parallel programming literature for decades[14] and can be used to build efficient merge sort routines in CUDA.[27]

Hierarchical synchronization. More often than not, parallel threads must synchronize with one another at various times, but excessive synchronization can undermine the efficiency of parallel programs. Synchronization must be treated carefully on massively parallel throughput-oriented processors where thousands of threads could potentially contend for a lock.

To avoid unnecessary synchronization, threads should synchronize hierarchically, keeping parallel computations independent as long as possible. When parallel programs are decomposed hierarchically, as in divide-and-conquer methods, the synchronization of threads can also be organized in a hierarchical fashion. For example, when evaluating `merge3(A,B)` earlier, each recursive parallel merge step can pro-

Figure 4. Double-precision throughput of SpMV strategies on an NVIDIA GeForce GTX 285 GPU; all matrices have exactly 4 x 2^{10} nonzeros but differing row counts.



ceed independently of all the others. Any synchronization required within these subtasks can be localized to the subtasks. Only at the end, when all subsequent operations are merged, must the parallel subtasks be synchronized with one another. Organizing synchronization hierarchically also aligns well with the physical cost of synchronizing threads spread across different sections of a given system. It is natural to expect that threads executing on a single core can be synchronized much more cheaply than threads spread across an entire processor, just as threads on a single machine can be synchronized more cheaply than threads across the multiple nodes of a cluster.

Conclusion

The transition from single-core to multicore processors and the increasing use of throughout-oriented architectures signal greater emphasis on parallelism as the driving force for higher computational performance. Yet these two kinds of processors differ in the degree of parallelism they expect to encounter in a typical workload. Throughput-oriented processors assume parallelism is abundant, rather than scarce, and their paramount design goal is maximizing total throughput of all parallel tasks rather than minimizing the latency of a single sequential task.

Emphasizing total throughput over the running time of a single task leads to a number of architectural design decisions. Among them, the three primary architectural trends typical of throughput-oriented processors are hardware multithreading, many simple processing elements, and SIMD execution. Hardware multithreading makes managing the expected abundant parallelism cheap. Simple in-order cores forgo out-of-order execution and speculation, and SIMD execution increases the ratio of functional units to control logic. Simple core design and SIMD execution reduce the area and power cost of control logic, leaving more resources for parallel functional units.

These design decisions are all predicated on the assumption that sufficient parallelism exists in the workloads the processor is expected to handle. The performance of a program with insufficient parallelism may therefore suffer. A fully general-purpose chip (such as a CPU) cannot afford to aggressively trade for increased total performance at the cost of single-thread performance. The spectrum of workloads presented to it is simply too broad, and not all computations are parallel. For computations that are largely sequential, latency-oriented processors perform better than throughput-oriented processors. On the other hand, a processor specifically intended for parallel computation can accept this trade-off and realize significantly greater total throughput on parallel problems as a result.

As the differences between these architectures appear durable rather than transient, the ideal system is thus heterogeneous, where a latency-oriented processor (such as a CPU) and a throughput-oriented processor (such as a GPU) work in tandem to address the heterogeneous workloads presented to them.

References

1. Alverson, G., Alverson, R., Callahan, D., Koblenz, B., Porterfield, A., and Smith, B. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In *Proceedings of the Sixth international Conference on Supercomputing* (Washington, D.C., July 19–24). ACM Press, New York, 1992, 188–197.
2. Alverson, R., Callahan, D., Cummings, D., Koblenz, B., Porterfield, A., and Smith, B. The Tera computer system. In *Proceedings of the Fourth international Conference on Supercomputing* (Amsterdam, The Netherlands, June 11–15). ACM Press, New York, 1990, 1–6
3. Bell, N. and Garland, M. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (Portland, OR, Nov. 14–20). ACM Press, New York, 2009, 1–11.
4. Birrell, A.D. *An Introduction to Programming with Threads.* Research Report 35. Digital Equipment Corp. Systems Research, Palo Alto, CA, 1989.
5. Blank, T. The MasPar MP-1 architecture. In *Proceedings of Compcon* (San Francisco, CA, Feb. 26–Mar. 2). IEEE Press, 1990, 20–24.
6. Borkar, S., Jouppi, N.P., and Stenstrom, P. Microprocessors in the era of terascale integration. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Nice, France, Apr. 16–20). EDA Consortium, San Jose, CA, 2007, 237–242.
7. Bouknight, W.J., Denenberg, S.A., McIntyre, D.E., Randall, J.M., Sameh, A.H., and Slotnick, D.L. The Illiac IV system. *Proceedings of the IEEE 60*, 4 (Apr. 1972), 369–388.
8. Dally, W. Power efficient supercomputing. Presented at the Accelerator-based Computing and Manycore Workshop (Lawrence Berkeley National Laboratory, Berkeley, CA, Nov. 30–Dec. 2, 2009); http://www.lbl.gov/cs/html/Manycore_Workshop09/GPU Multicore SLAC 2009/dallyppt.pdf
9. Dally, W.J., Labonte, F., Das, A., Hanrahan, P., Ahn, J., Gummaraju, J., Erez, M., Jayasena, N., Buck, I., Knight, T. J., and Kapasi, U.J. Merrimac: Supercomputing with streams. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing* (Nov. 15–21). IEEE Computer Society, Washington, D.C., 2003.
10. Davis, J.D., Laudon, J., and Olukotun, K. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th international Conference on Parallel Architectures and Compilation Techniques* (Sept. 17–21). IEEE Computer Society, Washington, D.C., 2005, 51–62.
11. Espasa, R., Valero, M., and Smith, J.E. Vector architectures: Past, present and future. In *Proceedings of the 12th international Conference on Supercomputing* (Melbourne, Australia). ACM Press, New York, 1998, 425–432.
12. Flynn, M.J. Very high-speed computing systems. *Proceedings of the IEEE 54*, 12 (Dec. 1966), 1901–1909.
13. Garland, M., Grand, S.L., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V. Parallel computing experiences with CUDA. *IEEE Micro 28*, 4 (July 2008), 13–27.
14. Gavril, F. Merging with parallel processors. *Commun. ACM 18*, 10 (Oct. 1975), 588–591.
15. Grochowski, E., Ronen, R., Shen, J., and Wang, H. Best of both latency and throughput. In *Proceedings of the IEEE international Conference on Computer Design* (Oct. 11–13). IEEE Computer Society, Washington, D.C., 2004, 236–243.
16. Kapasi, U., Dally, W.J., Rixner, S., Owens, J.D., and Khailany, B. The Imagine stream processor. In *Proceedings of the 2002 IEEE International Conference on Computer Design* (Sept. 16–18). IEEE Computer Society, Washington, D.C., 2002, 282–288.
17. Khailany, B.K., Williams, T., Lin, J., Long, E.P., Rygh, M., Tovey, D.W., and Dally, W.J. A programmable 512 GOPS stream processor for signal, image, and video processing. *IEEE Journal of Solid-State Circuits 43*, 1 (Jan. 2008), 202–213.
18. Kongetira, P., Aingaran, K., and Olukotun, K. Niagara: A 32-way multithreaded Sparc processor. *IEEE Micro 25*, 2 (Mar./Apr. 2005), 21–29.
19. Kozyrakis, C. and Patterson, D. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture* (Istanbul, Turkey, Nov. 18–22). IEEE Computer Society Press, Los Alamitos, CA, 2002, 283–293.
20. Krashinsky, R., Batten, C., Hampton, M., Gerding, S., Pharris, B., Casper, J., and Asanovic, K. The vector-thread architecture. *SIGARCH Computer Architecture News 32*, 2 (Mar. 2004), 52–63.
21. Laudon, J., Gupta, A., and Horowitz, M. Interleaving: A multithreading technique targeting multiprocessors and workstations. In *Proceedings of the Sixth International Conference on Architectural Support For Programming Languages and Operating Systems* (San Jose, CA, Oct. 5–7). ACM Press, New York, 1994, 308–318.
22. Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro 28*, 2 (Mar./Apr. 2008), 39–55.
23. Nickolls, J., Buck, I., Garland, M., and Skadron, K. Scalable parallel programming with CUDA. *Queue 6*, 2 (Mar./Apr. 2008), 40–53.
24. NVIDIA. *NVIDIA's Next-Generation CUDA Compute Architecture: Fermi*, Oct. 2009; http://www.nvidia.com/fermi
25. Russell, R.M. The Cray-1 computer system. *Commun. ACM, 21*, 1 (Jan. 1978), 63–72.
26. Sanders, J. and Kandrot, E. *CUDA By Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley, July 2010.
27. Satish, N., Harris, M., and Garland, M. Designing efficient sorting algorithms for manycore GPUs. In *Proceedings of the 2009 IEEE international Symposium on Parallel & Distributed Processing* (May 23–29). IEEE Computer Society, Washington, D.C., 2009, 1–10.
28. Smith, B.J. Architecture and applications of the HEP multiprocessor computer system. *Proceedings of the International Society for Optical Engineering 298* (Aug. 1981), 241–248.
29. Tucker, L.W. and Robertson, G.G. Architecture and applications of the Connection Machine. *Computer 21*, 8 (Aug. 1988), 26–38.
30. Tullsen, D.M., Eggers, S.J., and Levy, H.M. Simultaneous multithreading: maximizing on-chip parallelism. In *Proceedings of the 22nd Annual international Symposium on Computer Architecture* (S. Margherita Ligure, Italy, June 22–24). ACM Press, New York, 1995, 392–403.
31. Ungerer, T., Robič, B., and Šilc, J. A survey of processors with explicit multithreading. *ACM Computing Surveys 35*, 1 (Mar. 2003), 29–63.

Michael Garland (mgarland@nvidia.com) is a senior research scientist in NVIDIA Research, Santa Clara, CA.

David B. Kirk (dk@nvidia.com) is an NVIDIA Fellow and former chief scientist of NVIDIA Research, Santa Clara, CA.

DOI:10.1145/1839676.1839695

Concerns about biased manipulation of search results may require intervention involving government regulation.

BY PATRICK VOGL AND MICHAEL BARRETT

Regulating the Information Gatekeepers

IN 2003, 2BIGFEET, an Internet business specializing in the sale of oversize shoes ranked among the top results in Google searches for its products. Its prime location on the virtual equivalent of New York's high-end shopping mecca Fifth Avenue brought a steady stream of clicks and revenue. But success was fleeting:

That November, Google's engineers modified their search engine's algorithms, an update later dubbed "Florida" by the search-engine community. 2bigfeet's rankings dropped abruptly just before the Christmas selling season, and this Internet success story was suddenly on the brink of bankruptcy.[2]

Search engines have established themselves as critical gatekeepers of information. However, despite an increasingly monopolistic Internet search market, they and the implicit filtering process in their rankings remain largely beyond public scrutiny and control. This has inspired us to explore an increasingly topical question: Should search-engine ranking be regulated?

Search engines generally work by indexing the Web through so-called crawler programs. When a user types in a request, search algorithms determine the most relevant results in the index. Although the precise workings of these algorithms are kept at least as secret as Coca-Cola's formula they are usually based on two main functions: keyword analysis (for evaluating pages along such dimensions as frequency of specific words) and link analysis (based on the number of times a page is linked to from other sites and the rank of these other sites) (see Figure 1).

» key insights

- **With search engines guiding access to critical Web-based information flow, many users are increasingly concerned over possible targeted manipulation of search results.**
- **With markets and technology both unlikely to ensure unbiased results, regulation may be the only alternative.**
- **One promising way forward is clearer guidelines for search-engine optimization through self-regulation.**

Appreciating the value of top rankings, Webmasters have learned to optimize their pages so big search engines rank them more highly. This has spawned a worldwide industry of search-engine-optimization consultants, whose techniques are grouped into two categories: white-hat, ensuring that search engines easily analyze a site and are accepted by search engines; and black-hat, including hidden text, as in white text on a white background, considered illicit by most search engines and upon discovery generally punished with degraded ranking.

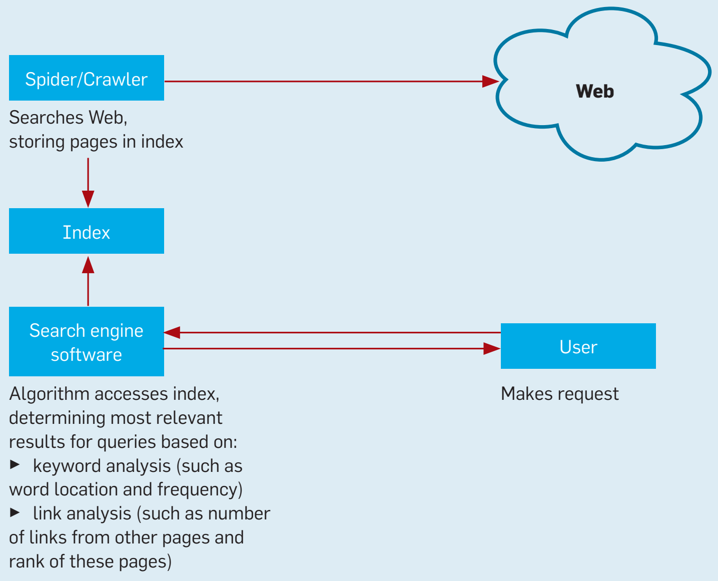
Search engines clearly have a legitimate interest in fighting inappropriate third-party optimization techniques to ensure their search-result quality; for instance, sites with no other purpose than linking to specific sites to increase page rank (link farms) are black-hat and must be dealt with accordingly, though punishment can be problematic for multiple reasons:

First, sudden ranking demotion and resulting diminished inflow of visitors have major effects on businesses, as illustrated by the cases of Skyfacet (which reportedly lost $500,000 of revenue in 2006) and MySolitaire (which reportedly lost $250,000 the same year[14]). Only a few cases, including the companies SearchKing[18] and Kinderstart[11] involving lawsuits over page rankings and German car manufacturer BMW, received notable media attention. Many more cases of dropped ranking have been condemned to virtual silence, among them search-engine optimizer Bigmouthmedia.

Second, though market-leader Google has published guidelines on creating "Google-friendly" pages, the line between permitted and illicit practices is blurry at best.[20] For example, Google's guidelines rightly warn against cloaking, "the practice of presenting different content [...] to users and search engines."[13] However, to a certain extent cloaking can be justified and used with good intent by major sites without penalty.[8] For instance, the *Wall Street Journal* uses it to show full versions of pay-per-view articles to Google's indexing program.[8]

The difficulty of straddling the line between permitted and illicit practices is further illustrated by a case involving paid links: In February 2009 Google punished its subsidiary Google Japan through a page rank demotion for paying for online reviews of a new widget.[23] While Google's attempt to play by its own rules is positive the case highlights the difficulty of distinguishing permitted from illicit optimization techniques. A leading U.S. commentator in online search asked: "If Google itself [...] found itself in this situation, how are ordinary Web sites to be expected to know the 'rules' about what they can or cannot do?"[23]

Figure 1. How ranking works.


Spider/Crawler
Searches Web, storing pages in index
Web
Index
Search engine software
User
Algorithm accesses index, determining most relevant results for queries based on:
► keyword analysis (such as word location and frequency)
► link analysis (such as number of links from other pages and rank of these pages)
Makes request

Third, our research supports the idea that there is no established process of announcement or appeal prior to rank demotion. Companies affected usually realize their fate only through a sudden loss of traffic or revenue. In a personal interview [2008], the CEO of an educational company told us: "The office called me and told me [...] that revenue was down [...], so I checked our logs and our history [...] It was all on one day. We were up to 14 million pageviews per month, and on one day it dropped 70% and [stayed there], and that was it."

Fourth, options are limited for companies affected by ranking demotion. One interviewee recalls his company got no response, even though he personally went to the search engine firm's headquarters for assistance.

Fifth, several allegations in the blogosphere claim large players are treated better than their less-powerful counterparts. For example, in May 2008, Hewlett-Packard began offering free blog templates, including hidden links to its own pages, a quick way to gather "high-quality" links and clicks.[25] However, there was no evidence of punishment by major search engines, sparking significant controversy in the community.[25]

Finally, search engines have punished Web sites using search-engine optimization, as well as the search-engine-optimization companies themselves. In the case of SearchKing, a search-engine consulting company in Oklahoma, U.S. courts have found that Google "knowingly and intentionally" dropped the company's Web sites in its rankings to punish what it deemed illicit ranking manipulation that SearchKing had carried out for its clients.[18]

Rationale for Regulation

Several researchers have pointed to the dangers of targeted manipulation, arguing it undermines values like free speech, fairness, economic efficiency, and autonomy, as well as the institution of democracy. Concerning democracy and free speech, Introna and Nissenbaum[17] argued a decade ago that search engines' broad structural bias can lead to underrepresentation of niches and minority interests and a loss of variety. Drawing on Anderson's theory of ethic limitations to markets,[1] they made a solid case that this lack of pluralism does not correspond to society's "liberal commitments to freedom, autonomy, and welfare."[1] Introna and Nissenbaum viewed the Internet as a "political good" due to its role as "conveyor of information" and function like "traditional public spaces" as a "forum for political deliberation."[17] Consequently, just as schools and national heritage are not left to the mercy of free markets, they argued the Internet is a public good requiring special protection.[17] Similarly, Bracha and Pasquale[3] made the case in 2007 that targeted manipulation of search engine results "threatens the openness and diversity of the Internet as a system of public expression."[3]

Another approach based on democratic values emphasizes the right of free speech. Chandler[5] argued it protects not only the ability to listen and speak, but that intermediaries cannot impose "discriminatory filters that the listener would not otherwise have used."[5] Since extraneous bias introduces different discrimination criteria, free speech is undermined by targeted search-engine-result manipulation.[5]

Fairness might also be undermined. Since search-engine rankings have enormous influence on business performance, ranking manipulation can cause significant harm both arbitrarily and more deliberately. Search engines as private entities are generally free to conduct business as they wish within the limitations applicable to all companies. However, Webmasters cannot simply opt out of their dependence on search engines, perhaps representing an "inescapable influence,"[10] taking their relationship from the private to the public sphere where more dependable accountability is expected.[10]

Other concerns involve economic efficiency, deception, and autonomy. Targeted manipulation can limit the availability of information, causing market inefficiencies and barriers to entry.[3] Manipulating search results, while leading users "to believe that search results are based on relevancy alone,"[16] could be deceptive, while search engines shaping user options and information could limit user autonomy.[3]

Companies affected usually notice their fate only through a sudden loss of traffic or revenue.

Regulation and alternatives. Traditional justification for regulation includes control of monopoly power and excess profits, compensation for externalities, inadequate information, unequal bargaining power, and scarcity of essential products.[4] While none perfectly fits the case of targeted results manipulation, intervention could be supported through two rationales:

Information asymmetries. Elkin-Koren and Salzberger[7] pointed out that the Internet's common market failure is an overload rather than undersupply of information.[7] However, the solution to this problem—the Internet search market—suffers from a lack of information. Just as laymen cannot easily assess the services of doctors or the effects of a particular medicine,[4] users of search engines are unable to adequately evaluate search services.

Market power. While the Internet search market shows monopolistic tendencies (extremely strong market positions of a few key players), there is no strong case for regulatory intervention on the standard antitrust argument of abuse of monopoly power (such as lack of competition leading to excessive pricing). However, a case can be based on Breyer's argument[4] of an "unjustifiably discriminatory exercise of personal power" combined with the "concentration of substantial social and political power" in a private entity that controls an "essential product"[4] (see also Bracha and Pasquale[3] for a similar view on the application of essential-facility arguments to search engines). These arguments were developed in the early 20th century U.S. business environment when a group of railway companies in control of access to the city of St. Louis prevented competitors from offering services in the same area.[21] Even if these arguments do not apply to users of search engines, they may apply to Webmasters unable to choose the search engines the public uses to find information.

Regulation must be compared to other solutions, particularly free markets. The basic market argument is that search engines have an incentive to produce the most relevant search results; otherwise, users would switch to their competitors.[17] However, markets alone are unlikely to address the concerns of targeted manipulation for three reasons:

Proprietary algorithms. Users would not be able to detect targeted manipulation in most cases, as search engines keep their algorithms secret.[3] Moreover, even if users are more aware of results manipulation, their expectation of what they are searching for is shaped during the search process.[17] Consequently, they cannot objectively evaluate search-engine quality[17];

Concentrated market. While the Internet search market is highly concentrated, a less monopolistic market is unlikely to emerge due to high economies of scale.[3] Furthermore, incumbents benefit from their existing user base by, say, collecting user data through such products as the Google toolbar. Moreover, the emergence of new big players is also unlikely, since promising start-ups could be acquired by such dominant search giants as Google and Microsoft; and

User inertia. While switching might seem easy (users simply type another address), evidence suggests that personal habit is a key factor when selecting a search engine[22]; moreover, new technologies like personalized search are likely to raise switching costs significantly.

Technological development is also often mentioned in the context of search-engine bias.[12] However, while new technologies may alleviate some concern (such as reinforcement of popular sites), no technology in sight is likely to cure the problem of targeted manipulation.

Two counterarguments often brought up against regulation of search-engine bias are that search results are free speech and therefore cannot be regulated, and search engines are not essential facilities, as they do not fulfil the criteria of essential facilities accepted by U.S. courts.

While these arguments have merit, they are insufficient for rejecting intervention for several reasons: Though the courts have acknowledged First Amendment rights for search engines, a number of legal scholars have argued against this view.[3,5] Both arguments emerge from a U.S.-centric context, which, as the Microsoft antitrust case in the 1990s showed, is not the only legal arena for regulation. Moreover, in legal circles it has been suggested that the existing regulatory frameworks may be inadequate for something as groundbreaking as Internet search. Given the state of the law, governments and multinational bodies may need to create a new regulatory framework.

Markets alone are unlikely to be sufficient to address the concerns about targeted manipulation.

Search and Its Stakeholders

Who are the stakeholders and what are their interests? Mitchell wrote[19] that stakeholders are characterized by power, legitimacy, and urgency. Building on a broad investigation of stakeholder interests in search-engine law by Grimmelmann,[15] we see four main actors—users, search engines, Webmasters, and search-engine optimizers—that are, to some extent, characterised by Mitchell's attributes.

Table 1 is an overview of key stakeholder interests in Internet search bias and stakeholder recognition of possible manipulation. Search-engine optimizers are particularly conflicted. On the one hand, they stand to gain from greater transparency in Internet search, as their business would be easier and more efficient, and a clearer picture of accepted practices would enable them to guarantee their clients that search engines tolerate their techniques. On the other hand, they profit from lack of transparency, as it raises the value of their key assets—expertise and inside knowledge. Moreover, search-engine optimizers with good contacts and community-wide attention can profit from their influence with search-engine companies in "curing" cases of rank demotion. Therefore, top performers arguably have little interest in greater transparency.

Commentators have made several proposals for regulating targeted manipulations. We comment first on the two most promising, then introduce a new approach (outlined in Table 2).

Obligation to provide reasons for rank demotion. One proposal[20] suggested establishing an obligation to provide a reason for rank demotion to increase transparency in the relationship between Webmasters and search engines. In this context, Pasquale[20] drew an interesting analogy with credit-reporting agencies providing reasons for adverse credit information to consumers.[20] It would favor the interests of users and especially Webmasters, because it would support Website optimization for search engines. Search-engine optimizers would probably be divided between the less influential that gain and top performers that lose from greater transparency. Building on the credit-agency analogy, Pasquale[20] argued that the cost to search engines and risk of algorithmic reverse engineering would be low.[20] However, as the number of potential queries on rank demotion is arguably much higher than the number on adverse credit ratings, cost to the search engines would likely be significant. Moreover, search engines would likely oppose any obligation to provide precise reasons for each rank demotion, as it would increase the risk of lawsuits.

Installation of ombudsmen and process of appeal. Taking the previous proposal a step further, some have called for an appeal process against rank demotion.[24] For example, Forsyth[10] emphasized if search engines were public entities with ensuing accountability, a process of appeal would probably already have been established.[10] While Google offers a way to submit pages for "reconsideration,"[8] such appeals are judged internally without transparency, and, while successful in some cases, a Webmaster's only option might be to attract enough publicity to get a search-engine representative to take up the case internally.[9]

Installing a formal, transparent appeals process would clearly be in the interest of both users and Webmasters, while assisting some search engine optimizers but diminishing the value of top optimizer contacts. On the other hand, search engines would arguably incur somewhat higher costs for installing such a process. Moreover, a formal process with a clear chance of success would facilitate appeals, thereby increasing numbers of requests and probably encouraging "appeal gaming."

Clearer guidelines for search engine optimization. Current guidelines between black-hat and white-hat search engine optimizers are gray, often forcing Webmasters and optimizers to speculate as to which techniques would be punished through ranking degradation. In a personal interview on search engine optimization, one search-engine consultant said the key to differentiating between black-hat and white-hat optimization techniques in unclear cases is "implication of intent." However, in the 2008 case of Hewlett-Packard mentioned earlier, comments by search-engine optimizers indicate the existence of at least some bias distinguishing between permitted and illicit optimization methods.

One approach to increasing transparency in the relationship between Webmasters and search engines is to establish clearer guidelines distinguishing black-hat from white-hat optimization. The gray area in between would be diminished, giving Webmasters and optimizers a better idea of what to expect from search engines. The diminished gray area would lead to more consistent application of ranking degradation, as questionable sites would fall more clearly into one category or the other. Moreover, new guidelines could also cover search engines' assessment of intent in questionable cases, further promoting consistent treatment of market players. This approach is promising because it has advantages for all stakeholders (see Figure 2).

Clearer guidelines. A self-regulatory approach initiated by regulators would be the easiest and most efficient way to achieve clearer guidelines. There is ample precedence that self-regulation works in cyberspace; for example, in the early days of Internet search, many search engines did not distinguish between organic and paid results (advertisements). However, following a letter by the U.S. Feder-

Table 1. Stakeholder interests in the regulation of search bias.

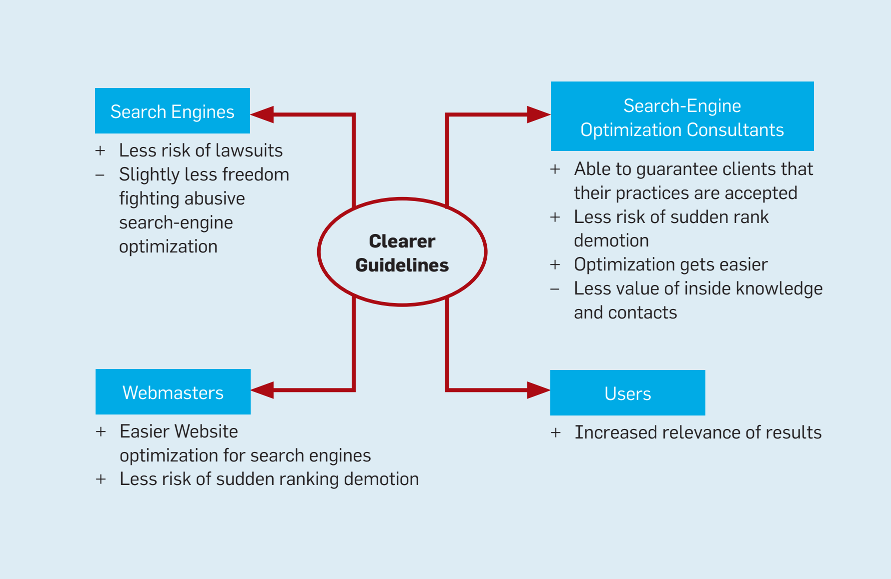
	Interests	Awareness of manipulation
Users	► High-quality content, particularly quick query response and inclusion of new pages; clear, well-structured results; transparent construction of results ► Pleasant search experience, easy usability	Low
Search Engines	► Protection of trade secrets, particularly algorithms ► Freedom to make changes to algorithms for preventing abuse of search-engine optimization ► Low cost of regulation	High
Webmasters	► Consistent, transparent ranking of pages in search engines	Low, some among professionals
Search-Engine Optimizers	► Top group: low transparency ► Low-level search-engine optimizers: high transparency	High

Table 2. Regulatory proposals and stakeholder interests.

++ strongly supportive
+ supportive
— opposed
— — strongly opposed

Regulatory Proposal	Users	Search Engines	Content Providers	Search-Engine Optimizers
Obligation to provide reasons for rank demotion	++	—	++	divided
Installation of ombudsmen and appeal process	++	—	++	divided
Clearer guidelines for search-engine optimization	++	+	++	divided

Figure 2. How clearer search-engine optimization guidelines would affect stakeholders.


Clearer Guidelines
Search Engines
+ Less risk of lawsuits
– Slightly less freedom fighting abusive search-engine optimization
Search-Engine Optimization Consultants
+ Able to guarantee clients that their practices are accepted
+ Less risk of sudden rank demotion
+ Optimization gets easier
– Less value of inside knowledge and contacts
Webmasters
+ Easier Website optimization for search engines
+ Less risk of sudden ranking demotion
Users
+ Increased relevance of results

al Trade Commission in 2002 recommending search engines ensure that "paid [...] results are distinguished from non-paid results with clear and conspicuous disclosures,"[16] all major search engines implemented such practices. Also, self-regulation would help the process of regulation keep up with the pace of technological change. Finally, as clearer guidelines would arguably favor all stakeholders, search engines would at least join the public dialogue on self-regulation.

Policymakers could signal the importance of targeted manipulation and initiate a dialogue on self-regulation by creating a committee of key stakeholders to examine cases of rank demotion and recommend ways to improve today's optimization guidelines. In addition, the topic of search-engine regulation could be put on the agenda of the next United Nations Internet Governance Forum (www.intgovforum.org).

Self-regulation alone may not alleviate concern about rank demotion. One idea from the Internet's early days may chart another way forward. As disputes over domain names became more heated in the 1990s and U.S. trademark law proved insufficient, the Internet Corporation for Assigned Names and Numbers (www.icann.org) and the World Intellectual Property Organization (www.wipo.int) developed the Uniform Domain-Name Dispute-Resolution Policy (www.icann.org/en/udrp/udrp.htm) to promote quick and inexpensive resolution of domain-name conflicts.[6] A similar body could help establish new optimization guidelines and manage the mediation process.

Conclusion

Here, we've argued for the need to open a debate on how to regulate targeted ranking manipulation that hinders search-engine optimization. These practices threaten democracy and free speech, fairness, market efficiency, autonomy, and freedom from deception. Making the case for regulation can be based on the search market's failure of information asymmetries and concentration of market power over an essential product in a private entity. Our analysis of specific regulatory proposals and their implications for stakeholders highlights the benefits of establishing clearer guidelines for optimizers. C

References

1. Anderson, E. *Value in Ethics and Economics.* Harvard University Press, Cambridge, MA, 1993.
2. Battelle, J. *The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture.* Nicholas Brealey Publishing, London, 2005.
3. Bracha, O. and Pasquale, F. *Federal Search Commission? Access, Fairness and Accountability in the Law of Search.* University of Texas Public Law Research Paper No. 123, Austin, TX, 2007.
4. Breyer, S. Typical justifications for regulation. In *A Reader on Regulation*, R. Baldwin, C. Scott, and C. Hood, Eds. Oxford University Press, New York, 1998, 59–93.
5. Chandler, J. A right to reach an audience: An approach to intermediary bias on the Internet. *Hofstra Law Review 35*, 3 (Spring 2007), 1095–1139.
6. Davis, G. The ICANN uniform domain name dispute resolution policy after nearly two years of history. *e-OnTheInternet* (Jan./Feb. 2002); http://www.isoc.org/oti/articles/1201/icann.html
7. Elkin-Koren, N. *Law, Economics and Cyberspace.* Edward Elgar Publishing, Cheltenham, U.K., 2004.
8. Fishkin, R. White-hat cloaking: It exists, it's permitted, it's useful. *SEOMoz Blog* (June 30, 2008); http://www.seomoz.org/blog/white-hat-cloaking-it-exists-its-permitted-its-useful
9. Fontenot, D. Matt Cutts, Why am I still being punished? *SEO Scoop* (Jan. 24, 2008); http://www.seo-scoop.com/2008/01/24/matt-cutts-why-am-i-still-being-punished/
10. Forsyth, H. Google MP? *How the Internet Is Challenging Our Notions of Political Power.* Presentation at Pembroke College Ivory Tower Society, Cambridge, U.K. (Jan. 28, 2008).
11. Goldman, E. *KinderStart v. Google* dismissed: With sanctions against KinderStart's counsel. *Technology & Marketing Law Blog* (Mar. 20, 2007); http://blog.ericgoldman.org/archives/2007/03/kinderstart_v_g_2.htm
12. Goldman, E. Search engine bias and the demise of search engine utopianism. *Yale Journal of Law & Technology 8* (Spring 2006), 188–200.
13. Google, Inc. *Webmaster Tools Help Cloaking, Sneaky Javascript Redirects, and Doorway Pages* (Dec. 4, 2008); http://www.google.com/support/webmasters/bin/answer.py?answer=66355&topic=15263.
14. Greenberg, A. Condemned to Google hell. *Forbes* (Apr. 4, 2007); http://www.forbes.com/2007/04/29/sanar-google-skyfacet-tech-cx_ag_0430googhell.html
15. Grimmelmann, J. The structure of search engine law. *Iowa Law Review 93*, 1 (Nov. 2007), 3–63.
16. Hippsley, H. *Letter from FTC to Search Engines Regarding Commercial Alert Complaint Requesting Investigation of Various Internet Search Engine Companies for Paid Placement and Paid Inclusion Programs.* Federal Trade Commission, Washington, D.C., June 27, 2002; http://www.ftc.gov/os/closings/staff/commercialalertattatch.shtm
17. Introna, L.D. and Nissenbaum, H. Shaping the Web: Why the politics of search engines matters. *The Information Society 16*, 3 (2000), 169–185.
18. Miles-LaGrange, V. *SearchKing, Inc. v. Google Technology, Inc.* CIV-02-1457-M.U.S. District Court for the Western District of Oklahoma, 2003.
19. Mitchell, R.K., Agle, B.R., and Wood, D.J. Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. *The Academy of Management Review 22*, 4 (Oct. 1997), 853–886.
20. Pasquale, F. *Rankings, Reductionism, and Responsibility.* Seton Hall Public Law Research Paper. Seton Hall University, South Orange, NJ, Feb. 25, 2006.
21. Pitofsky, R. *The Essential Facility Doctrine Under United States Antitrust Law.* Federal Trade Commission, Washington, D.C., 2001; http://www.ftc.gov/os/comments/intelpropertycomments/pitofskyrobert.pdf
22. Sherman, C. Search engine users: Loyal or blasé? *Search Engine Watch* (Apr. 19, 2004); http://searchenginewatch.com/3342041
23. Sullivan, D. Google penalizes Google Japan for buying links. *Search Engine Land* (Feb. 11, 2009); http://searchengineland.com/google-penalizes-google-japan-16541
24. Sullivan, D. Google ombudsman? Search ombudsman? Great idea: Bring them on! *Search Engine Watch* (July 2006); http://blog.searchenginewatch.com/blog/060706-075235
25. Wall, A. Strategic content as marketing for link building. *SEO Book* (May 8, 2008); http://www.seobook.com/content-marketing-win

Patrick Vogl (pvogl@gmx.at) is a graduate of the MPhil in Technology Policy at the Judge Business School, University of Cambridge, U.K.

Michael Barrett (m.barrett@jbs.cam.ac.uk) is a Reader in Information Technology and Innovation and Director of Programmes at the Judge Business School, University of Cambridge, U.K.

DOI:10.1145/1839676.1839696

Computational complexity may truly be the shield against election manipulation.

BY PIOTR FALISZEWSKI, EDITH HEMASPAANDRA, AND LANE A. HEMASPAANDRA

Using Complexity to Protect Elections

FOR THOUSANDS OF years, people—and more recently, electronic agents—have been conducting elections. And surely for just as long, people—or more recently, electronic agents—have been trying to affect the outcomes of those elections. Such attempts take many forms. Often and naturally, actors may seek to change the structure of the election, for example, by attracting new voters, suppressing turnout, recruiting candidates, or setting election district boundaries. Sometimes voters may even be bribed to vote a certain way. And a voter may try to manipulate an election by casting an insincere vote that may yield a more favorable outcome than would the voter's sincere vote: Not all people who preferred Ralph Nader in the 2004 U.S. presidential election actually voted for him.

One might hope that by choosing a particularly wonderful election system, one can perfectly block such attacks. However, classic work from economics and political science proves that every reasonable election system sometimes gives voters an incentive to vote insincerely (see Duggan[17] and the references therein). Reasonable election systems cannot make manipulation impossible. However, they can make manipulation computationally infeasible.

This article is a nontechnical introduction to a startling approach to protecting elections: using computational complexity as a shield. This approach seeks to make the task of whoever is trying to affect the election *computationally* prohibitive. To better understand the cases in which such protection cannot be achieved, researchers in this area also spend much of their time working for the Dark Side: trying to build polynomial-time algorithms to attack election systems.

This complexity-based approach to protecting elections was pioneered in a stunning set of papers, about two decades ago, by Bartholdi, Orlin, Tovey, and Trick.[2,3,5] The intellectual fire they lit smoldered for quite a while, but in recent years has burst into open flame. Computational complexity may truly be the key to defending elections from manipulation.

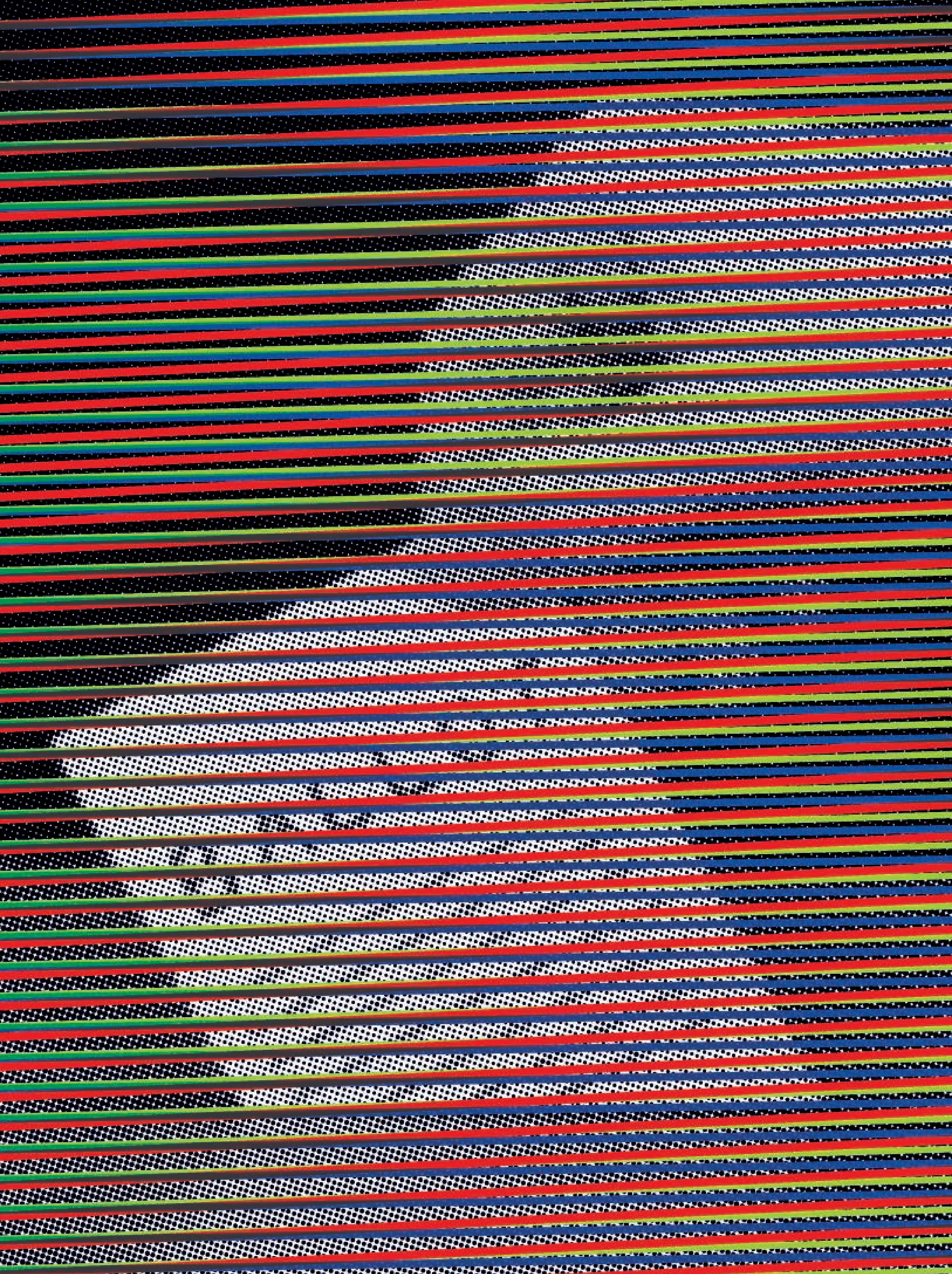
Preliminaries and the Complexity of the Winner Problem

In the introduction, we focused on

» key insights

- **Algorithms can be used to seek attacks on elections, and complexity can serve to protect elections from attacks. For some election systems, manipulation has been proven NP-hard.**
- **Dichotomy theorems pinpoint what it is about an election system that makes it computationally resistant to manipulation.**
- **It is natural to consider an election system's computational weaknesses and strengths as one factor, among many, when selecting a system for a given task. In particular, one must consider which types of attacks one most needs to thwart.**

ILLUSTRATION BY MELVIN GALAPON



protecting elections, rather than on why and in what settings elections are used for aggregating preferences in the first place. The latter issue could itself fill a survey—but not this survey. However, before moving on we briefly mention a few varied examples of how elections can be useful in aggregating preference. In daily life, humans use elections to aggregate preferences in tasks ranging from citizens choosing their political representatives to an academic department's faculty members selecting which job candidate to hire to conference business meeting attendees selecting future locations for their conference. In electronic settings, elections often can take on quite different, yet also interesting and important, challenges. For example, one can build a metasearch engine based on combining underlying search engines, in order to seek better results and be more resistant to "Web spam."[18] One can use voting as an approach to building recommender systems[41] and to planning.[20] Voting was already very important before computers and the internet existed, and in the modern world, where multiagent settings abound, the importance of voting is greater still.

Voting was already very important before computers and the Internet existed, and in the modern world, where multiagent settings abound, the importance of voting is greater still.

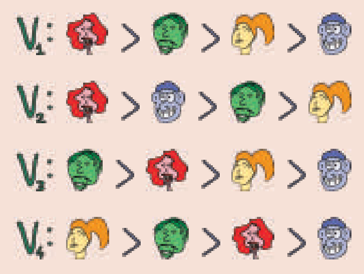
In this article, we will discuss the successes and failures to date in using complexity to defend against three important classes of attacks on election systems: (structural) control attacks, (voter) manipulation, and bribery. In these three settings, high computational complexity is the goal. But first, we briefly discuss a case so surprising that one might not even think of it, namely, the case in which an election system is so complex that even determining who won is intractable.

We must first introduce the model of elections we will use throughout this article. While doing so, we will also define some election systems, such as plurality rule. An election consists of a candidate set C and a list V of votes (ballots) over those candidates. In almost all the election systems we discuss, a vote is simply a strict ordering of all the candidates, for example, "Nader > Gore > Bush" if the voter likes Nader most, Gore next most, and Bush least. An exception is approval voting, in which each vote is a bit-vector giving a thumbs-up or thumbs-down to each candidate.

An election system is simply a mapping from (C, V) to a "winner set" W, $\emptyset \subseteq W \subseteq C$. Perhaps the most famous and common election system is plurality, in which each candidate who most often comes at the top of voters' orders is put into W. We will focus quite a bit on plurality in this article, since it has been extensively studied with respect to using complexity to protect elections. Plurality is itself a special case of a broad class of election systems known as scoring systems or scoring-rule systems. In these, each candidate gets from each voter a certain number of points based on where he or she falls in the voter's ordering, and whoever gets the most points wins. For example, the scoring point system for plurality (in k-candidate elections) is that a voter's favorite candidate gets one point from that voter and the other k–1 candidates get zero points from that voter. In the Borda election system, proposed in the 18th century, the points from favorite to least favorite are k–1, k–2, . . . , 0. In veto elections, the points are 1, 1, 1, . . . , 1, 0; that is, the voter in effect votes against one candidate. Scoring systems are a flexible, important class of voting systems and, as we will see, they are a class whose manipulation complexity (for fixed numbers of candidates) is completely analyzed. There are many other important election systems, but to move the article along, we will introduce them as we need.

An election system that immediately merits note is the Condorcet rule. In Condorcet elections, a candidate is a winner exactly if he or she beats each other candidate in head-to-head majority-rule elections under the voters' preferences. Consider the election shown in Figure 1. In that election there is no Condorcet winner, since is beaten by 3-to-1, is beaten by 4-to-0, and

Figure 1. An election.



in a 2-to-2 tie, 🔴 fails to beat 🟢.

Although this example shows that Condorcet elections sometimes have no winner, some election systems—the so-called Condorcet-consistent systems—so value the naturalness of the notion of being a Condorcet winner that they ensure that when a Condorcet winner exists, he or she is the winner in their system. One particularly interesting system is the election system proposed by the famous author and mathematician Lewis Carroll in 1876. Carroll took an approach that should warm the hearts of computer scientists. He said, in effect, that whoever had the smallest edit distance from being a Condorcet winner was the winner in his election system. His edit distance was with respect to the number of sequential exchanges of adjacent candidates in voter orderings. So in the Figure 1 example, 🔴 and 🟢 tie as Carroll winners, since either of them with one adjacent exchange can become a Condorcet winner (for example, if we by one exchange turn voter 1's preference list into 🟢 > 🔴 > 🟠 > 🔵, then 🟢 becomes a Condorcet winner), but 🔵 for example would take seven adjacent exchanges to become a Condorcet winner.

Lewis Carroll's system is quite lovely in that it focuses on the closeness to Condorcet winnerhood. Carroll's paper has been included in books collecting the most important social choice papers of all time. However, Carroll's system has one glaring flaw: It is computationally intractable to tell who won! This was first shown in a paper by Bartholdi, Tovey, and Trick,[4] who showed that this problem was **NP**-hard. Later, Hemaspaandra, Hemaspaandra, and Rothe[30] precisely classified the problem's complexity as "complete" (that is, in a certain formal sense the hardest problem) for the class of problems that can be solved by parallel access to **NP** (a class that forms the Θ_2^p level of the polynomial hierarchy).[a]

On its face, this result is a disaster for Lewis Carroll's election system. Although we want manipulation of elections to be difficult, we do not want to achieve this by migrating to election systems so opaque that we cannot efficiently compute who won.

This disaster may not be quite as severe as it first seems. Recent work on Lewis Carroll elections seeks to slip around the edges of the just-mentioned intractability result. In particular, two recent papers show that simple polynomial-time greedy algorithms correctly find the Lewis Carroll winner all but an asymptotically exponentially vanishing portion of the time when the number of voters is more than quadratically larger than the number of candidates and the inputs are drawn from the uniform probability distribution.[35,39] In fact, that algorithm can even be made "self-knowingly correct"—it almost always declares that its answer is correct, and when it does so it is never wrong.[35] Another way of arguably bypassing the hardness results for the Lewis Carroll winner problem is through approximation algorithms. For example, Caragiannis et al.[9] have recently developed two approximation algorithms for computing candidates' scores in Carroll's system. And a third way to sidestep the hardness results is to change the framework, namely, to assume that the number of candidates or the number of voters is bounded by a fixed constant, and to seek polynomial-time algorithms in that setting.[b] The seminal paper of Bartholdi, Tovey, and Trick[4] successfully pursued this line, as

[a] We will not provide here a discussion of *NP*-hardness/*NP*-completeness/Θ_2^p-completeness, but suffice it to say that complexity theorists broadly believe any problem that has any one of these properties is intractable, that is, does not have a deterministic polynomial-time algorithm. However, these notions are worst-case notions. In the section "Using Complexity to Block Election Manipulation" we will discuss how their worst-case nature is itself a worry when using them to protect election systems.

[b] Many real-life settings have relatively few candidates. And a particularly interesting setting with few voters but many candidates comes from Dwork et al.,[18] who suggested building a search engine for the Web that would simply query other search engines and then conduct an election given the search engines' answers as votes.

Table 1. The computational complexity of control in Condorcet, Copeland, Llull, and plurality elections.

The results regarding constructive control in Condorcet and plurality elections are due to Bartholdi et al.,[5] the results on destructive control for Condorcet and plurality are due to Hemaspaandra et al.,[31] and the results regarding Llull and Copeland are due to Faliszewski et al.[25] Adding (Unlimited Number of) Candidates has not been explicitly studied in Bartholdi et al.[5] and Hemaspaandra et al.,[31] but the results on this for Condorcet and plurality elections are corollaries to these papers' proofs.

Election System	Condorcet		Copeland		Llull		Plurality	
Control Type	Const. Control	Dest. Control	Const. Control	Dest. Control	Const. Control	Dest. Control	Const. Control	Dest. Control
Adding (Unlimited Number of) Candidates	I	V	R	V	V	V	R	R
Adding Candidates	I	V	R	V	R	V	R	R
Deleting Candidates	V	I	R	V	R	V	R	R
Run-off Partition of Candidates (Ties Promote)	V	I	R	V	R	V	R	R
Run-off Partition of Candidates (Ties Eliminate)	V	I	R	V	R	V	R	R
Partition of Candidates (Ties Promote)	V	I	R	V	R	V	R	R
Partition of Candidates (Ties Eliminate)	V	I	R	V	R	V	R	R
Partition of Voters (Ties Eliminate)	R	V	R	R	R	R	V	V
Partition of Voters (Ties Promote)	R	V	R	R	R	R	R	R
Adding Voters	R	V	R	R	R	R	V	V
Deleting Voters	R	V	R	R	R	R	V	V

Figure 2. Ramon Llull, 13th-century mystic and philosopher.

have more recent papers.[7,11,24,25] However, their polynomial-time algorithms sometimes involve truly astronomical multiplicative constants.

Finally, we mention that in the years since the work showing Lewis Carroll's election system to have a winner problem that is complete for parallel access to **NP**, a number of other systems, most notably those of Kemeny and Young, have also been shown to be complete for parallel access to **NP**.[33,43] Naturally, researchers have sought to bypass these hardness results as well (for examples, see[6,9,12,36]).

Using Complexity to Block Election Control

Can our choice of election systems—and not merely nasty ones with hard winner problems but rather natural ones with polynomial-time winner problems—be used to make influencing election outcomes costly? We start the discussion of this issue by considering problems of election *control*, introduced by Bartholdi, Tovey, and Trick[5] in 1992. In election control, some actor who has complete knowledge of all the votes seeks to achieve a desired outcome—either making a favored candidate be the sole winner ("constructive control") or precluding a despised candidate from being a unique winner ("destructive control")—via changing the structure of the election. The types of structural changes that Bartholdi, Tovey, and Trick proposed are adding or deleting candidates, adding or deleting voters, or partitioning candidates or voters into a two-round election structure. Between these types and the different tie-breaking rules that can be used to decide which candidates move forward from the preliminary rounds of two-round elections in the case of ties (that is, whether all the tied people move forward or none of them do), there now are eleven types of control that are typically studied—each having both a constructive and a destructive version.

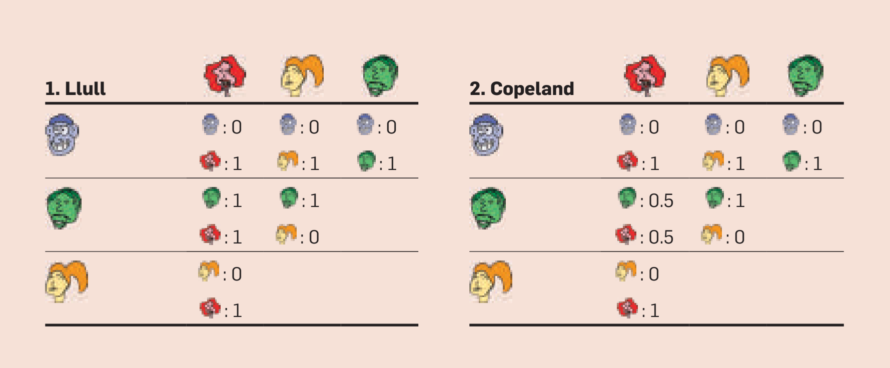
For reasons of space we will not define all 11 types here. We will define one type explicitly and will mention the motivations behind most of the others. Let us consider Control by Deleting Voters. In this control scenario, the input to the problem is the election (C, V), a candidate $p \in C$, and an integer k. The question is whether by removing at most k votes from V one can make p be the sole winner (for the constructive case) or can preclude p from being a unique winner (for the destructive case). Control by Deleting Voters is loosely inspired by vote suppression: It is asking whether by the targeted suppression of at most k votes the given goal can be reached. (By discussing vote suppression we are in no way endorsing it, and indeed we are discussing paths toward making it computationally infeasible.) So, for a given election system E, we are interested in the complexity of the set composed of all inputs (C, V, p, k) for which the goal can be reached.[c]

The other control types similarly are motivated as abstractions of real-world actions—many far more savory than vote suppression. For example, Control by Adding Voters abstracts such actions as get-out-the-vote drives, positive advertising campaigns, providing vans to drive elderly people to the polls, registration drives, and so on. Control by Adding Candidates and Control by Deleting Candidates reflect the effect of recruiting candidates into—and pressuring them to withdraw from—the race. The memory of the 2000 U.S. presidential race suggests that whether a given small-party candidate—say Ralph Nader—enters a race can change the outcome. The partition models loosely capture other behaviors, such as gerrymandering.

Table 1 summarized the constructive and destructive control results for four election systems whose behavior is completely known: Plurality, Condorcet, Llull, and Copeland. The mystic and philosopher Ramon Llull (Figure 2) defined the Llull system in the 1200s, and the Copeland system is a closely related system defined in modern times. In both of these systems one considers each pair of candidates and awards one point to the winner in their head-to-head majority-rule contest, and if the head-to-head contest is a tie, in Copeland each gets half a point but in Llull each still gets one point. So, for example, in Copeland one gets $\|C\|–1$ points exactly if one is a Condorcet winner. The Llull/Copeland system is used in the group stage

[c] As to *who* is seeking to do the control, that is external to the model. For example, it can be some central authority or a candidate's campaign committee. In fact, in the real world there often are competing control actors. But results we will soon cover show that even a single control actor faces a computationally infeasible problem. Also, the reader may naturally feel uncomfortable with the model's assumption that the control actor knows all the votes of all the voters. But note that that just makes the "shield" results stronger: they show that even if one had perfect information about the votes, finding a control action is *still* intractable.

Figure 3. The points assigned by the Llull/Copeland systems in the head-to-head contests of the election of Figure 1.



of the World Cup soccer tournament, except there (after rescaling) wins get one point and ties get one third of a point. Figure 3 shows how the election from Figure 1 comes out under the Llull and Copeland systems. In Table 1, I (immunity) means one can never change the outcome with that type of control attack—a dream case; R (resistance) means it is **NP**-hard to determine whether a given instance can be successfully attacked—still a quite good case; and V (vulnerability) means there is a polynomial-time algorithm to detect whether there is a successful attack of the given type (and indeed to produce the exact attack)—the case we wish to avoid.

Remarkably, given that Llull created his system in the 1200s, among all natural systems based on preference orders, Llull and Copeland are the systems that currently have the greatest numbers of proven resistances to control. As one can see from Table 1, Copeland is perfectly resistant to the constructive control types and to all voter-related control types (but is vulnerable to the destructive, candidate-related control types). And Llull's 13th-century system is almost as good. Ramon Llull, the mystic, truly was ahead of his time.

If one wants an even greater number of resistances than Copeland/Llull provides, one currently can do that in two different ways. Recently, Erdélyi, Nowak, and Rothe[22] showed that a voting system whose votes are in a different, richer model—each voter provides both an approval vector and a strict ordering—has a greater number of control resistances, although in achieving that it loses some of the particular resistances of Copeland/Llull. And Hemaspaandra, Hemaspaandra, and Rothe[32] constructed a hybridization scheme that allows one to build an election system whose winner problem—like the winner problem of all four systems from Table 1—is computationally easy, yet the system is resistant to all 22 control attacks. Unfortunately, that election system is in a somewhat tricky manner "built on top of" other systems each of which will in some cases determine the winner, and so the system lacks the attractiveness and transparency that real-world voters reasonably expect.

The current push-pull between using complexity as a shield and seeking holes in and paths around that shield is a natural part of the drama of science.

To conclude our discussion of control, we mention one other setting, that of choosing a whole assembly or committee through an election. Such assembly-election settings introduce a range of new challenges. For example, the voters will have preferences over assemblies rather than over individual candidates. We point the reader to the work of Meir et al.[40] for results on the complexity of controlling elections of this type.

Using Complexity to Block Election Manipulation

Manipulation is often used informally as a broad term for attempts to affect election outcomes. But in the literature, manipulation is also used to refer just to the particular attack in which a voter or a coalition of voters seeks to cast their votes in such a way as to obtain a desired outcome, for example, making some candidate win. In formulating such problems, one often studies the case in which each voter has a weight, as is the case in the electoral college and in stockholder votes. The input to such problems consists of the weights of all voters, the votes of the nonmanipulators, and the candidate the manipulators are trying to make a winner.

Manipulation problems have been studied more extensively than either control or bribery problems, and so the literature is too broad to survey in any detail. But we now briefly mention a few of the key themes in this study, including using complexity to protect, using algorithms to attack, studying approximations to bypass protections, and analyzing manipulation properties of random elections.

The seminal papers on complexity of manipulation are those of Bartholdi, Orlin, Tovey, and Trick.[2,3] Bartholdi, Tovey, and Trick[3] gave polynomial-time algorithms for manipulation and proved a hardness-of-manipulation result (regarding so-called second-order Copeland voting). Bartholdi and Orlin[2] showed that for "single transferable vote," a system that is used for some countries' elections, whether a given voter can manipulate the election is **NP**-complete, even in the unweighted case.

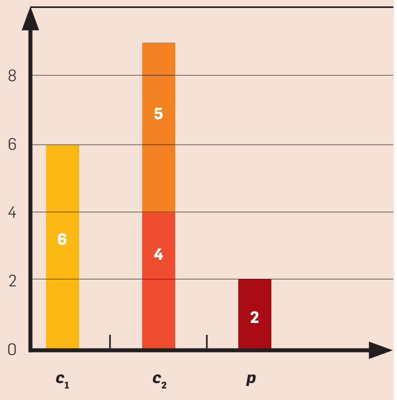
Even if election systems are proven intractable to manipulate in general, it remains possible that if one allows only a certain number of candidates, the manipulation problem becomes easy. Conitzer, Sandholm, and Lang[15] provide a detailed study of this behavior, showing for each of many election systems the exact number of candidates necessary to make its (constructive, weighted, coalitional) manipulation problem computationally infeasible. For example, in this setting manipulation is easy for Borda with up to two candidates, but becomes infeasible when restricted even to three candidates.

In contrast, it is well known that manipulation is simple for plurality elections regardless of the number of candidates. That is unfortunate, since plurality elections are the most common and most important elections in the real world.

What holds for scoring-rule election systems other than plurality? One could try analyzing scoring systems one at a time to see which are subject to manipulation, but it might be a long slog since there are an infinite number of scoring systems. This motivates us to look toward an excellent general goal: finding a dichotomy theorem that in one fell swoop pinpoints what it is about an election system that makes it vulnerable to manipulation or that makes manipulation computationally prohibitive. For scoring systems, this was achieved in Hemaspaandra and Hemaspaandra[29] (see also the closely related work[15,42]), which showed that scoring systems are **NP**-complete to manipulate (in the weighted setting) precisely if they allow "diversity of dislike" (that is, the point values for the second favorite and least favorite candidates differ), and that all other scoring systems are easy to manipulate. From this it follows that the only easily manipulable scoring systems are an infinite collection of trivial systems, plurality, and an infinite collection of systems that are disguised, transformed versions of plurality; all other scoring systems are **NP**-hard to manipulate.

Figure 4. An example of a weighted plurality election.

Each bar represents a weighted vote for a particular candidate. We can make *p* a winner by bribing the weight-5 voter to vote for *p*, but bribing only the heaviest voter to vote for *p* would not be sufficient.



There has been an intense effort to circumvent such hardness results. Indeed, the seminal paper on manipulation[3] provided a greedy single-voter manipulation algorithm that was later proved to also work in an interesting range of coalitional-manipulation settings.[42,49] An influential paper of Conitzer and Sandholm[14] shows that voting systems and distributions that on a large probability weight of the inputs satisfy certain conditions have a manipulability-detection algorithm that is correct on at least that same set of inputs. A different line of research focuses on analyzing the probability with which a randomly selected election is susceptible to a given form of manipulation.[16,28,47,48] In the standard probabilistic model used in this line of work,[d] for many natural election systems the probability that a voter can affect the result of an election by simply casting a random vote is small but nonnegligible.

This work is motivated by perhaps the greatest single worry related to using **NP**-hardness to protect elections—a worry that applies to **NP**-hardness results not just about manipulation, but also about control and bribery. That worry is that **NP**-hardness is a worst-case theory, and it is in concept possible that **NP**-hard sets may be easily solved on many of their input instances even if **P** and **NP** differ.[e] Levin has famously developed the theory of average-case **NP**-hardness,[37] and although that theory is difficult to apply and is tied to what distributions one uses, it would be extremely interesting to establish that the manipulation, control, and bribery problems for important election systems are average-case **NP**-hard with respect to some appropriate and compellingly natural distribution.

A very exciting new path toward circumventing hardness-of-manipulation results (and, potentially, toward more generally circumventing hardness results about election-related issues) is to look at restricted domains for the collections of votes the electorate may cast. In particular, there is a very important political science notion called "single-peaked preferences," in which the candidates are modeled along an axis, such as liberal to conservative, and as one goes away from each voter's most preferred candidate in either of the axis's directions the voter prefers the candidates less and less. Walsh[46] raised the fascinating question of whether hard election-manipulation problems remain hard even for electorates that follow the single-peaked model, and he provided natural examples in which manipulation problems remain hard even when restricted to single-peaked electorates. In contrast, and inspired by a different part of Walsh's paper that showed some profile completion problems are easy for single-peaked electorates, a recent paper by Faliszewski et al.[27] shows that for single-peaked electorates many **NP**-hard manipulation and control problems have polynomial-time algorithms. The point of—and threat of—this research line is that for electorates that are single-peaked,

[d] This model is called *impartial culture*. In impartial culture each vote is chosen uniformly at random from the set of all permutations of the candidates.

[e] There are a number of results in theoretical computer science that are related to this issue, while as a practical matter not resolving it for the concrete cases we care about. For example, by an easy "padding" trick one can see that *every* **NP**-hard set can have its instances transformed into questions about (in the jargon, can be many-one polynomial-time reduced to) a set that is easy on overwhelmingly many of its instances.[21] Unfortunately, this does not necessarily imply that the original set is easy on overwhelmingly many of its instances. In fact, it is known that relative to a random "oracle" (black box), there are **NP** sets on which no polynomial-time heuristic algorithm can do well.[34] Also, it is well known that if any **NP**-hard set has a polynomial-time heuristic algorithm that is correct on all but a "sparse" amount of its input, then **P** = **NP**.[44] However, "sparse" in that research line is so small as to not reassure us here. And, finally, there has been much interest in distributions, problems, and settings that remove the gap between worst-case and average-case complexities.[1,38]

NP-hardness results simply may fail to hold. And the reason that can happen is the assumption of single-peaked preferences is so restrictive that it can rule out some of the collections of votes used in the outputs of reductions in general-case **NP**-hardness proofs.

Yet another path toward circumventing hardness-of-manipulation results leads to relaxing the notion of solving a manipulation problem. Procaccia and Rosenschein[42] initiated this approach by showing that the heuristic from the seminal work of Bartholdi, Tovey, and Trick,[3] when extended to a coalitional manipulation setting, works correctly on an interesting class of scoring-system manipulation instances. By an even more careful analysis, together with Zuckerman, they later extended this result to a number of other election systems,[49] and they obtained approximation results and results that for manipulable instances are guaranteed to return a manipulation that will work if one is allowed to add a certain number and weight of additional manipulators. Brelsford et al.[8] provide their own framework for studying approximability of manipulation problems (as well as approximability of bribery and control problems) and for a large class of scoring systems gives approximation algorithms for manipulation.

Returning to playing defense, what can we do if a system has a polynomial-time manipulation algorithm? Can we somehow feed the system a can of spinach and turn it fearsome? To a surprising extent the answer is yes, as studied in work of Conitzer and Sandholm[13] and Elkind and Lipmaa.[19] They variously do this by adding an elimination "pre-round" (that may or may not be based on a hypothetical one-way function) or by changing the election into a long series of rounds of candidate elimination. The good news is that this approach often boosts the complexity, and the bad news is that these multiround election systems are simply not the same intuitively attractive animals that they are built from.

Using Complexity to Block Bribery in Elections

The complexity-theoretic study of bribery in elections was proposed by Faliszewski, Hemaspaandra, and Hemaspaandra,[24] and started far more recently than did the complexity-theoretic study of control and manipulation of elections. Bribery comes in many variants, but the basic pattern is just what the term brings to mind. The briber has a certain budget, the voters (who depending on the model may or may not have weights) each have a price for which their vote can be bought, and depending on the model voters may or may not be required to each have unit cost (the former case is referred to as the "without prices" case). And the question is whether the briber can achieve his or her goal—typically, to make a preferred candidate p be a winner—within the budget. Note that bribery has aspects of both control and manipulation. Like some types of control one has to choose which collection of voters to act on, but like manipulation one is altering votes.

For reasons of space, we cover bribery only briefly. We do so by giving a few examples focusing on plurality elections and Llull elections.

For plurality elections, the complexity of bribery turns out to be very sensitive to the model. For plurality, bribery is **NP**-complete when voters have weights and prices, but is in polynomial time if voters have only weights, only prices, or neither weights nor prices.[24,f] For the weighted and the weighted-and-priced cases, these results can be extended to dichotomy theorems that completely classify which scoring-rule election systems have **NP**-complete bribery problems and which have feasible bribery problems.[24] Also, for plurality, there is an efficient algorithm that can approximately solve the problem up to any given precision[23]—a so-called fully polynomial-time approximation scheme.

For Llull elections, the results again are very sensitive to the model. On one hand, both with and without weights, and both with and without voter prices, the bribery problem for Llull elections is **NP**-complete. On the other hand, if one changes one's model and associates a cost not to each voter, but rather to each pairwise preference of each voter (so the more one changes a given voter's vote, the more one has to pay—so-called "microbribery"), Llull bribery (without weights) can be done, in a slightly different model that allows irrational preferences, in polynomial time.[25]

Summary

In this article, we discussed some of the main streams—control, manipulation, and bribery—in the study of how complexity can be used as a shield to protect elections (see Faliszewski et al.[26] for a more technical survey). This line was started by the striking insight of Bartholdi, Orlin, Tovey, and Trick (see also Simon[45] for even earlier roots) that although economics proves we cannot make manipulation impossible, we can seek to make it computationally infeasible. As we have seen, many hardness results have been obtained, as have many polynomial-time attacks. Election systems and settings vary greatly in the behaviors one can establish. It is natural to consider an election system's computational weaknesses and strengths, as one factor among many, when choosing an election system for a given task, and in particular to choose a system carefully in light of the types of attacks one most needs it to thwart. Yet the work on computational protection of elections has also energized the search for end runs around that protection, such as approximation algorithms and heuristics having provably frequent good performance, and one must also worry about such potential end runs when making one's election-system choice.

This work all falls within the emerging area known as computational social choice (see Chevaleyre et al.[10] for a superb survey), an area that links AI, systems, and theory within computer science, as well as economics, political science, mathematics, and operations research. Elections have been important for thousands of years, and with the current and anticipated increase of electronic agency, elections become more important—and more open to attacks—with each passing year. The current push-pull between using complexity

[f] The bribery algorithms are far from trivial. For example, Figure 4 shows an election (without prices) where the very natural heuristic of first bribing the heaviest voter yields a suboptimal solution. Similarly, it is easy to find examples where bribing the heaviest voter of a current winner does not lead to an optimal solution.

as a shield and seeking holes in and paths around that shield is a natural, exciting part of the drama of science, and is likely to continue for decades to come as new models, techniques, and attacks are formulated and studied. This study will clearly benefit from the broadest possible participation, and we urge any interested readers—and most especially those early in their careers—to bring their own time and skills to bear on the many problems that glimmer in the young, important, challenging study of the complexity of elections.

Acknowledgments

We are deeply grateful to Preetjot Singh, *Communications* editors Georg Gottlob, Moshe Vardi, and Andrew Yao, and the anonymous referees for helpful suggestions and encouragement.

Piotr Faliszewski was supported in part by Polish Ministry of Science and Higher Education grant N-N206-378637, AGH-UST grant 11.11.120.865, and by the Foundation for Polish Science's Homing Program. Edith Hemaspaandra was supported in part by NSF grant IIS-0713061 and a Friedrich Wilhelm Bessel Research Award from the Alexander von Humboldt Foundation. Lane A. Hemaspaandra was supported in part by NSF grants CCF-0426761 and CCF-0915792 and a Friedrich Wilhelm Bessel Research Award from the Alexander von Humboldt Foundation.

References

1. Ajtai, M. Worst-case complexity, average-case complexity and lattice problems. *Documenta Mathematica*, Extra Volume ICM III (1998), 421–428.
2. Bartholdi, III, J. and Orlin, J. Single transferable vote resists strategic voting. *Social Choice and Welfare 8*, 4 (1991) 341–354.
3. Bartholdi, III, J., Tovey, C. and Trick, M. The computational difficulty of manipulating an election. *Social Choice and Welfare 6*, 3 (1989) 227–241.
4. Bartholdi, III, J., Tovey, C. and Trick, M. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare 6*, 2 (1989), 157–165.
5. Bartholdi, III, J., Tovey, C. and Trick, M. How hard is it to control an election? *Mathematical and Computer Modeling 16*, 8/9 (1992), 27–40.
6. Betzler, N., Fellows, M., Guo, J., Niedermeier, R. and Rosamond, F. Fixed-parameter algorithms for Kemeny scores. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*. Lecture Notes in Computer Science #5034 (June 2008). Springer-Verlag, 60–71.
7. Betzler, N., Guo, J., Niedermeier, R. Parameterized computational complexity of Dodgson and Young elections. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory*. Lecture Notes in Computer Science #5124 (July 2008). Springer-Verlag, 402–413.
8. Brelsford, E., Faliszewski, P., Hemaspaandra, E., Schnoor, H., and Schnoor, I. Approximability of manipulating elections. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (July 2008). AAAI Press, 44–49.
9. Caragiannis, I., Covey, J., Feldman, M., Homan, C. Kaklamanis, C., Karanikolas, N., Procaccia, A. and Rosenschein, J. On the approximability of Dodgson and Young elections. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms* (Jan. 2009). Society for Industrial and Applied Mathematics, 1058–1067.
10. Chevaleyre, Y., Endriss, U., Lang, J. and Maudet, N. A short introduction to computational social choice. In *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science*. Lecture Notes in Computer Science #4362 (Jan. 2007). Springer-Verlag, 51–69.
11. Christian, R., Fellows, M., Rosamond, F. and Slinko, A. On complexity of lobbying in multiple referenda. *Review of Economic Design 11*, 3 (2007), 217–224.
12. Conitzer, V., Davenport, A. and Kalagnanam, J. Improved bounds for computing Kemeny rankings. In *Proceedings of the 21st National Conference on Artificial Intelligence* (July 2006). AAAI Press, 620–626.
13. Conitzer, V. and Sandholm, T. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Aug. 2003). Morgan Kaufmann, 781–788.
14. Conitzer, V. and Sandholm, T. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the 21st National Conference on Artificial Intelligence* (July 2006). AAAI Press, 627–634.
15. Conitzer, V., Sandholm, T. and Lang, J. When are elections with few candidates hard to manipulate? *Journal of the ACM 54*, 3 (2007), Article 14.
16. Dobzinski, S. and Procaccia, A. Frequent manipulability of elections: The case of two voters. In *Proceedings of the 4th International Workshop on Internet and Network Economics*. (Dec. 2008). Springer-Verlag *Lecture Notes in Computer Science #5385*, 653–664
17. Duggan, J. and Schwartz, T. Strategic manipulability without resoluteness or shared beliefs: Gibbard–Satterthwaite generalized. *Social Choice and Welfare 17*, 1 (2000), 85–93
18. Dwork, C., Kumar, R., Naor, M. and Sivakumar, D. Rank aggregation methods for the Web. In *Proceedings of the 10th International World Wide Web Conference* (Mar. 2001). ACM Press, NY, 613–622.
19. Elkind, E. and Lipmaa, H. Hybrid voting protocols and hardness of manipulation. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation* (Dec. 2005). Springer-Verlag *Lecture Notes in Computer Science #3872*, 206–215.
20. Ephrati, E. and Rosenschein, J. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence 20*, 1-4 (1997), 13–67.
21. Erdélyi, G., Hemaspaandra, L., Rothe, J. and Spakowski, H. Generalized juntas and NP-hard sets. *Theoretical Computer Science 410*, 38–40 (2009), 3995–4000.
22. Erdélyi, G., Nowak, M. and Rothe, J. Sincere-strategy preference-based approval voting fully resists constructive control and broadly resists destructive control. *Mathematical Logic Quarterly 55*, 4 (2009), 425–443.
23. Faliszewski, P. Nonuniform bribery (short paper). In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems* (May 2008), 1569–1572.
24. Faliszewski, P., Hemaspaandra, E. and Hemaspaandra, L. How hard is bribery in elections? *Journal of Artificial Intelligence Research 35* (2009), 485–532.
25. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research 35* (2009), 275–341.
26. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. A richer understanding of the complexity of election systems. In *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz.* S. Ravi and S. Shukla, eds. Springer, 2009, 375–406.
27. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. In *Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge* (July 2009). ACM Digital Library, 118–127.
28. Friedgut, E., Kalai, G. and Nisan, N. Elections can be manipulated often. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science* (Oct. 2008). IEEE Computer Society, 243–249.
29. Hemaspaandra, E., Hemaspaandra, L. Dichotomy for voting systems. *Journal of Computer and System Sciences 73*, 1 (2007), 73–83.
30. Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM 44*, 6 (1997), 806–825.
31. Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence 171*, 5–6 (2007), 255–285.
32. Hemaspaandra, E., Hemaspaandra, L. and Rothe, J. Hybrid elections broaden complexity-theoretic resistance to control. *Mathematical Logic Quarterly 55*, 4 (2009), 397–424.
33. Hemaspaandra, E., Spakowski, H. and Vogel, J. The complexity of Kemeny elections. *Theoretical Computer Science 349*, 3 (2005), 382–391.
34. Hemaspaandra, L. and Zimand, M. Strong self-reducibility precludes strong immunity. *Mathematical Systems Theory 29*, 5 (1996), 535–548.
35. Homan, C. and Hemaspaandra, L. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. *Journal of Heuristics 15*, 4 (2009), 403–423.
36. Kenyon-Mathieu, C. and Schudy, W. How to rank with few errors. In *Proceedings of the 39th ACM Symposium on Theory of Computing* (June 2007). ACM Press, 95–103.
37. Levin, L. Average case complete problems. *SIAM Journal on Computing 15*, 1 (1986), 285–286.
38. Li, M. and Vitányi, P. Average case complexity under the universal distribution equals worst-case complexity. *Information Processing Letters 42*, 3 (1992), 145–149.
39. McCabe-Dansted, J., Pritchard, G. and Slinko, A. Approximability of Dodgson's rule. *Social Choice and Welfare 31*, 2 (2008), 311–330.
40. Meir, R., Procaccia, A., Rosenschein, J. and Zohar, A. The complexity of strategic behavior in multi-winner elections. *Journal of Artificial Intelligence Research 33* (2008), 149–178.
41. Pennock, D., Horvitz, E. and Giles, C. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence* (July/Aug. 2000). AAAI Press, 729–734.
42. Procaccia, A. and Rosenschein, J. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research 28* (2007), 157–181.
43. Rothe, J., Spakowski, H. and Vogel, J. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems 36*, 4 (2003), 375–386.
44. Schöning, U. Complete sets and closeness to complexity classes. *Mathematical Systems Theory 19*, 1 (1986), 29–42.
45. Simon, H. *The Sciences of the Artificial*. MIT Press, 1969. Third edition, 1996.
46. Walsh, T. Uncertainty in preference elicitation and aggregation. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (July 2007). AAAI Press, 3–8.
47. Xia, L. and Conitzer, V. Generalized scoring rules and the frequency of coalitional manipulability. In *Proceedings of the 9th ACM Conference on Electronic Commerce* (July 2008). ACM Press, NY, 109–118.
48. Xia, L. and Conitzer, V. A sufficient condition for voting rules to be frequently manipulable. In *Proceedings of the 9th ACM Conference on Electronic Commerce* (July 2008). ACM Press, NY, 99–108.
49. Zuckerman, M. Procaccia, A. and Rosenschein, J. Algorithms for the coalitional manipulation problem. *Artificial Intelligence 173*, 2 (2009), 392–412.

Piotr Faliszewski (faliszew@agh.edu.pl) is an assistant professor at the AGH University of Science and Technology, Kraków, Poland.

Edith Hemaspaandra (eh@cs.rit.edu) is a professor at the Rochester Institute of Technology, Rochester, NY.

Lane A. Hemaspaandra (lane@cs.rochester.edu) is a professor at the University of Rochester, Rochester, NY.

DOI:10.1145/1839676.1839697

Technical Perspective
Data Races are Evil with No Exceptions

By Sarita Adve

EXPLOITING PARALLELISM HAS become the primary means to higher performance. Shared memory is a pervasively used programming model, where parallel tasks or threads communicate through a global address space. Popular languages, such as C, C++, and Java have (or will soon have) standardized support for shared-memory threads. Unfortunately, shared-memory programs are notoriously prone to subtle bugs, often due to data races.

Languages that allow data races obfuscate true communication and synchronization. Since any load or store may communicate or synchronize with another thread through a data race, it becomes difficult to reason about sections of code in isolation. Data races also create non-determinism since the ordering between two racing accesses is timing-dependent. Racy programs therefore require reasoning about many interleavings of individual memory accesses and their executions are difficult to reproduce. Data races have therefore been widely considered as symptoms of bugs and there is much research to automatically detect them.

An arguably more fundamental problem with data races concerns semantics. Every programming language must specify what value a load can return, also called the *memory model*. It has been surprisingly difficult to specify an acceptable model that balances ease-of-use and performance for languages that allow data races.[1] Programmers usually expect a sequential interleaving-based model called sequential consistency. For data-race-free programs, it is straightforward to provide sequential consistency and high performance. With racy code, however, routine compiler and hardware optimizations can result in surprising behaviors. Consequently, the upcoming C and C++ memory models specify "undefined" behavior for programs with data races. This gives freedom to implementers but poses challenges for debugging racy code. Java's safety requirements preclude the use of "undefined" behavior. The Java memory model, therefore, specifies very weak semantics for racy programs, but is extremely complex and currently has known bugs.

Despite the debugging and semantics difficulties, some prior work refers to certain data races (for example, some unsynchronized reads) as benign and useful for performance. Unfortunately, reasonable semantics for programs with any type of data race remain elusive; C, C++, and Java do not provide usable semantics with any (benign or not) data race.

A natural conclusion is to strive for languages that eliminate data races. Although there is much progress, such general-purpose languages,[2] are not yet commercially available. The following two papers on Goldilocks and FastTrack suggest alternative solutions that use always-on data race detection for current languages.

Goldilocks was the first work to propose a data race be treated as a language-level exception, just like null pointer dereferences. This insight cleans up the most difficult part of the memory models mess—executions are either sequentially consistent or throw an exception. No complicated semantics of Java and no unsafe behavior of C/C++!

The challenge, compared to much prior work on race detection, is that an always-on language-level exception mechanism must be both fast and precise. The well-established race detection algorithm based on vector clocks is precise, but incurs orders-of-magnitude slowdown. Faster algorithms (for example, based on a lockset approach) exist, but produce false positives, which are unacceptable for enforcing language-level semantics.

Goldilocks extends the lockset approach to make it precise, at a much faster speed than the previous precise vector clock algorithm. FastTrack followed up by optimizing the vector clock approach to make it run faster, without losing precision and performing even better than Goldilocks. For the first time, these papers made it possible to believe that "data race as an exception" may be a viable solution to the vexing debugging and semantics problems of shared memory.

The absolute slowdown of both techniques is still quite significant, but they expose an exciting research agenda. An immediate challenge is to improve performance. Another concerns mapping races detected in compiler-generated code to source code. Others are exploring hardware for similar goals.[3]

More broadly, these papers encourage a fundamental rethinking of programming models, languages, compilers, and hardware. Should languages be designed to eliminate data races by design?[2] Or should the runtime automatically detect races? Or should the hardware?[3] Can similar techniques be used for enforcing even stronger properties such as determinism and atomicity? Does this impact how we view shared memory as a programming model? The answer to all these questions is probably "yes."

It is unlikely that any of language, runtime, or hardware techniques alone will strike the best balance between ease-of-use, generality, performance, and complexity, but designing systems that combine the strengths of such techniques remains challenging. It is also unclear what high-level language properties such systems will finally ensure; for example, race elimination is certainly a critical factor in ensuring determinism and atomicity. What is clear is these papers provide key insights to shape the final solutions and are important steps toward making parallel programming amenable to the masses. C

References

1. Adve, S.V. and Boehm, H-J. Memory models: A case for rethinking parallel languages and hardware. *Commun. ACM 53*, 8 (Aug. 2010) 90–101.
2. Bocchino, R.L. et al. A type and effect system for deterministic parallel Java. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2009.
3. Lucia, B. et al. Conflict exceptions: Providing simple concurrent language semantics with precise hardware exceptions. In *Proceedings of the International Symposium on Computer Architecture*, 2010.

Sarita Adve (sadve@illinois.edu) is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign.

DOI:10.1145/1839676.1839698

Goldilocks: A Race-Aware Java Runtime

By Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran

Abstract

We present Goldilocks, a Java runtime that monitors program executions and throws a `DataRaceException` when a data race is about to occur. This prevents racy accesses from taking place, and allows race conditions to be handled before they cause errors that may be difficult to diagnose later. The `DataRaceException` is a valuable debugging tool, and, if supported with reasonable computational overhead, can be an important safety feature for deployed programs. Experiments by us and others on race-aware Java runtimes indicate that the `DataRaceException` may be a viable mechanism to enforce the safety of executions of multithreaded Java programs.

An important benefit of `DataRaceException` is that executions in our runtime are guaranteed to be race free and thus sequentially consistent as per the Java Memory Model. This strong guarantee provides an easy-to-use, clean semantics to programmers, and helps to rule out many concurrency-related possibilities as the cause of errors. To support the `DataRaceException`, our runtime incorporates the novel Goldilocks algorithm for precise dynamic race detection. The Goldilocks algorithm is general, intuitive, and can handle different synchronization patterns uniformly.

1. INTRODUCTION

When two accesses by two different threads to a shared variable are enabled simultaneously, i.e., at the same program state, a race condition is said to occur. An equivalent definition involves an execution in which two threads make conflicting accesses to a variable without proper synchronization actions being executed between the two accesses. A common way to ensure race freedom is to associate a lock with every shared variable, and to ensure that threads hold this lock when accessing the shared variable. The lock release by one thread and the lock acquire by the next establish the required synchronization between the two threads.

Data races are undesirable for two key reasons. First, a race condition is often a symptom of a higher-level logical error such as an atomicity violation. Thus, race detectors serve as a proxy for more general concurrency-error detection when higher-level specifications such as atomicity annotations do not exist. Second, a race condition makes the outcome of certain shared variable accesses nondeterministic. For this and other reasons, both the Java Memory Model (JMM)[10] and the C++ Memory Model (C++MM)[2] define well-synchronized programs to be programs whose executions are free of race conditions. For race-free executions, these models guarantee sequentially consistent semantics; in particular, every read deterministically returns the value of the "last" write. This semantics is widely considered to be the only simple-enough model with which writing useful concurrent programs is practical. For executions containing race conditions, the semantics is either completely undefined (as is the case for C++MM[2]) or is complicated enough that writing a useful and correct program with "benign races" is a challenge.

Detection and/or elimination of race conditions has been an area of intense research activity. The work presented in this paper (and initially presented in Elmas et al.[6]) makes two important contributions to this area.

First, for the first time in the literature, we propose that race conditions should be language-level exceptions just like null pointer dereferences and indexing an array out of its bounds. The Goldilocks runtime for Java provides a new exception, `DataRaceException`,[a] that is thrown precisely when an access that causes an actual race condition is about to be executed. Since a racy execution is never allowed to take place, this guarantees that the execution remains sequentially consistent.

The `DataRaceException` brings races to the programmer's attention explicitly. When this exception is caught, the recommended course of action is to terminate the program gracefully. This is because, for racy Java programs, a wide range of compiler optimizations are allowed by the JMM, and this makes it complicated to relate program executions to source code. If the exception is not caught, the Goldilocks runtime terminates the thread that threw the exception. While not recommended, the programmer could also choose to make optimistic use of a `DataRaceException` by, for instance, retrying the code block that led to the race, hoping that the conflicting thread has performed the synchronization necessary to avoid a race in the meantime. Since the first paper on the Goldilocks runtime,[6] the idea that certain concurrency errors, especially ones that result in sequential consistency violations, should result in exceptions has gained significant support and several implementations of the idea have been investigated.[9, 11]

To support a `DataRaceException`, a runtime must

[a] We define `DataRaceException` as a subclass of the `RuntimeException` class in Java.

The original version of this paper was published in the *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI)*, June 2007.

incorporate a precise yet efficient race detection mechanism. In this context, false positives in race detection cannot be tolerated. The second contribution of our work is the Goldilocks algorithm, a novel, precise, and general algorithm for detecting data races at runtime. In Elmas et al.,[6] we presented an implementation of the Goldilocks algorithm in a Java Virtual Machine (JVM) called Kaffe.[19] Our experiments with Goldilocks on benchmarks brought up the new possibility that the overhead of post-deployment precise race detection in a runtime may be tolerable. There has been significant progress in the efficiency of precise race detection since the Goldilocks runtime was first published (see Flanagan and Freund, and Pozmiansky and Schuster,[7,14] for example) and this idea appears viable today.

The Goldilocks algorithm is based on an intuitive, general representation for the happens-before relationship as a generalized lockset (Goldilockset) for each variable. In the traditional use of the term, a lockset for a shared variable x at a point in an execution contains a set of locks. A thread can perform a race-free access to x at that point by first acquiring a lock in this lockset. A Goldilockset is a generalization of a lockset. In Java, locks and volatile variables are synchronization objects, and acquiring and releasing a lock, as well as reading from and writing to a volatile variable, are synchronization operations. To reflect this, at each point in an execution, the Goldilockset for a shared variable x may contain thread ids, locks, and volatile variables. A thread can perform a race-free access to x iff its thread id is in the Goldilockset, or if it first acquires a lock that is in the Goldilockset, or reads a volatile variable that is in the Goldilockset. In other words, the Goldilockset indicates the threads that have the ownership of x and the synchronization objects that protect access to x at that point. The Goldilockset is updated during the execution as synchronization operations are performed. As a result, Goldilocksets are a compact, intuitive way to precisely represent the happens-before relationship. Thread-local variables, variables protected by different locks at different points of the execution, and event-based synchronization with condition variables are all uniformly handled by Goldilocks. Furthermore, Goldilocksets can easily be generalized to handle other synchronization primitives such as software transactions[18] and adapted to handle memory models of languages other than Java, such as C++MM. To facilitate this, in this paper (differently from Elmas et al.[6]) we present Goldilocks on a generic memory model and then show how the algorithm can be specialized to JMM.

1.1. Related work

Dynamic Race Detection: There are two approaches to dynamic data-race detection, one based on locksets and the other based on the happens-before relation. Eraser[16] is a well-known lockset-based algorithm for detecting race conditions dynamically by enforcing the locking discipline that every shared variable is protected by a unique lock. In spite of the numerous papers that refined the Eraser algorithm to reduce the number of false alarms, there are still cases, such as dynamically changing locksets, that cannot be handled precisely. Precise lockset algorithms exist for Cilk programs,[3] but they cannot handle concurrency patterns implemented using volatile variables such as barrier synchronization.

There is a significant body of research on dynamic data-race detection based on computing the happens-before relation[4,7,14,15,17] using vector clocks.[12] Hybrid techniques[14,20] combine lockset and happens-before analysis. For example, RaceTrack[20] uses a basic vector clock algorithm to capture thread-local accesses to objects thereby eliminating unnecessary and imprecise applications of the Eraser algorithm. Similarly, MultiRace[14] presents djit$^+$, a vector clock algorithm with several optimizations to reduce the number of checks at an access, including keeping distinct vector clocks for reads and writes and using a lockset algorithm as a fast-path check. To the best of our knowledge, FastTrack,[7] which builds on djit$^+$, is the best-performing vector clock-based algorithm in the literature. By exploiting some access patterns, FastTrack reduces the cost of vector clock updates to $O(1)$ on average. We provide a qualitative comparison of the Goldilocks and FastTrack algorithms in Section 4.3. Vector clock and Goldilocks are both precise, but the generalized locksets in Goldilocks provide an intuitive representation of how shared variables are protected at each point the execution.

Concurrency-Related Exceptions: Since proposed first by the authors in Elmas et al.,[6] the idea that programming platforms should be able to guarantee sequential consistency for *all* programs has gained significant support. Marino et.al.[11] and Lucia et.al.[9] provide platforms with explicit memory model exceptions. Both platforms define stronger but simpler contracts than JMM and C++MM, which enable efficient hardware implementations that support the memory model exceptions.

2. A GENERIC MEMORY MODEL

In this section, we present a generic memory model and express the JMM as a special case of it. This generic model allows a uniform treatment of the various synchronization constructs in Java. We also believe that memory models at different levels (e.g., the hardware level) and for different languages (e.g., C++MM) can be expressed as instances of this model. This allows Goldilocks to be applied in these settings directly.

Variables and Actions: Program variables are separated into two categories: data variables (*Data*) and synchronization variables (*Sync*). We use x, and o to refer to data and synchronization variables, respectively. Threads in a program execute actions from the following categories:

- *Data variable accesses*: $\mathsf{read}(t, x, v)$ by thread t reads the current value v of a data variable x, and $\mathsf{write}(t, x, v)$ by thread t writes the value v to x.
- *Synchronization operations*: When threads synchronize using a synchronization mechanism, a thread t_i executes a notification action, which is then observed by other threads t_j. Such a notification–observation pair defines a "synchronizes-with" edge from the former action to the latter. We classify actions that serve as sources and sinks of a synchronizes-with edge as synchronization source and sink actions, respectively.

- *Synchronization source actions*: sync-source(t, o) by thread t creates a synchronizes-with source by writing to a synchronization variable o. Lock releases and volatile variable writes in Java are synchronization source actions.
- *Synchronization sink actions*: sync-sink(t, o) by thread t creates a synchronizes-with sink by reading from a synchronization variable o. Lock acquires and volatile variable reads in Java are synchronization sink actions.

Multithreaded Executions: An execution E is represented by a tuple $E = \langle Tid, Act, W, \xrightarrow{po}., \xrightarrow{so}.\rangle$.

- Tid is the set of identifiers for threads involved in the execution. Each newly forked thread is given a new unique id from Tid.
- Act is the set of actions that occur in this execution. $Act|_t$ is the set of actions performed by $t \in Tid$, and $Act|_x$ (resp. $Act|_o$) are the sets of actions performed on data variable x (resp. synchronization variable o).
- W is a total function that maps each read(t, x, v) in Act to a write(u, x, v) in Act. W is used to model the write-seen relationship between a read of x and the write to x it sees. In a race-free, sequentially consistent execution, this is the last write before read(t, x, v). In order to make the function W total, we assume an initial write for each variable before any reads.
- $\xrightarrow{po}_t$ is the program order per thread t. For each thread t, $\xrightarrow{po}_t$ is a total order over $Act|_t$ and gives in which order the actions were issued to execute. This order is sometimes referred to as the observed execution order.
- $\xrightarrow{so}_o$ is the synchronization order per synchronization variable $o \in Sync$. For each $o \in Sync$, $\xrightarrow{so}_o$ is a total order over $Act|_o$.

Synchronizes-With and Happens-Before: Given an execution with program and synchronization orders, we extract two additional orders called the synchronizes-with ($\xrightarrow{sw}$) and happens-before ($\xrightarrow{hb}$) orders. Data races are defined using these orders.

A synchronization operation α_1 by thread t_1 *synchronizes with* α_2 by thread t_2, denoted $\alpha_1 \xrightarrow{sw} \alpha_2$, if α_1 is a sync-source on some synchronization variable o, α_2 is a sync-sink on o, and $\alpha_1 \xrightarrow{so}_o \alpha_2$.

The *happens-before* partial order $\xrightarrow{hb}$ on the execution E is defined as the transitive closure of the program orders $\xrightarrow{po}_t$ for all $t \in Tid$ and the synchronizes-with order $\xrightarrow{sw}$.

In this paper, we focus only on *well-formed* executions,[10] which respect (i) the intra-thread semantics and (ii) the semantics of the synchronization variables and operations. In addition, well-formed executions satisfy two essential requirements for data-race detection:

- *Happens-before consistency*: This property makes use of the happens-before order to restrict the write-seen relationship. For example, for a read action α, $\alpha \xrightarrow{hb} W(\alpha)$ cannot happen, and $W(\alpha)$ cannot be overwritten by another write action β such that $W(\alpha) \xrightarrow{hb} \beta \xrightarrow{hb} \alpha$.
- The union of the program orders for all $t \in Tid$ and the synchronization orders for all variables $o \in Sync$ is a valid partial order. During an execution, our data-race detection algorithm examines a linearization of this partial order and identifies the happens-before edges between data accesses.

Sequential Consistency: Sequential consistency is a property that allows programmers to use an interleaving model of execution where accesses from different threads are interleaved into a total order, and every read sees the value of the most recent write. Sequential consistency is widely considered to be the only simple-enough model with which writing useful concurrent programs is practical. Formally, an execution $E = \langle Tid, A, W, \xrightarrow{po}., \xrightarrow{so}.\rangle$ is *sequentially consistent* if there exists a total order $\xrightarrow{SC}$ over Act satisfying the following:

- For every thread $t \in Tid$, $\xrightarrow{SC}$ respects the program order $\xrightarrow{po}_t$, i.e., $\xrightarrow{po}_t \subseteq \xrightarrow{SC}$.
- Every read α = read(x) sees the most recent write to x in $\xrightarrow{SC}$, i.e., there is no other β = write(x) such that $W(\alpha) \xrightarrow{SC} \beta \xrightarrow{SC} \alpha$.

Data Races: Two data variable accesses are called *conflicting* if they refer to the same shared data variable and at least one of them is a write access.

One frequently used definition of a race condition involves a program state in which two conflicting accesses by two different threads to a shared data variable are simultaneously enabled. To distinguish this definition from others, let us refer to this condition as a *simultaneity race*. The definition of a race condition used in most work on dynamic race detection is what we call a *happens-before race* and involves two conflicting accesses not ordered by the happens before relationship, i.e., not separated by proper synchronization operations. For C++, these two definitions of a race condition have been shown to be equivalent.[2] This proof also generalizes to Java executions. Formally, an execution $E = \langle Tid, Act, W, \xrightarrow{po}, \xrightarrow{so}.\rangle$ contains a happens-before race if there are two conflicting actions, α, $\beta \in Act|_x$ accessing a data variable x, such that neither $\alpha \xrightarrow{hb} \beta$ nor $\beta \xrightarrow{hb} \alpha$ holds. Conversely, the execution is *race free* if every pair of conflicting accesses to a data variable are ordered by happens-before.

The well-formedness of an execution guarantees that if the execution has no race conditions, then it is sequentially consistent. The Goldilocks runtime makes use of this and the `DataRaceException` to guarantee for all programs (whether racy or not) that every concurrent execution is sequentially consistent at the byte-code level. This does not restrict the Goldilocks runtime's use as a debugging tool, because, for the Java and C++ memory models, it has been proven[2, 10] that if a program has a racy execution, then it is guaranteed to have at least one execution that is sequentially consistent and racy. Thus, it is sufficient to restrict one's attention to looking for races in sequentially consistent executions only.

3. THE GOLDILOCKS ALGORITHM

In this section, we describe our algorithm for detecting data races in an execution $E = \langle Tid, Act, W, \xrightarrow{po}., \xrightarrow{so}.\rangle$. For simplicity of exposition, we initially do not distinguish between read and write accesses.

The GOLDILOCKS algorithm processes the actions in *Act* one at a time, as a sequence. Before a thread t performs an action α in *Act*, t notifies the GOLDILOCKS algorithm that α is about to occur. The order in which these notifications from different threads are interleaved and processed by GOLDILOCKS is represented mathematically by π, where $\pi(i)$ is the i-th action in the sequence. This linear order, by construction, respects the program order for each thread, and the synchronization total order for each synchronization variable.[b]

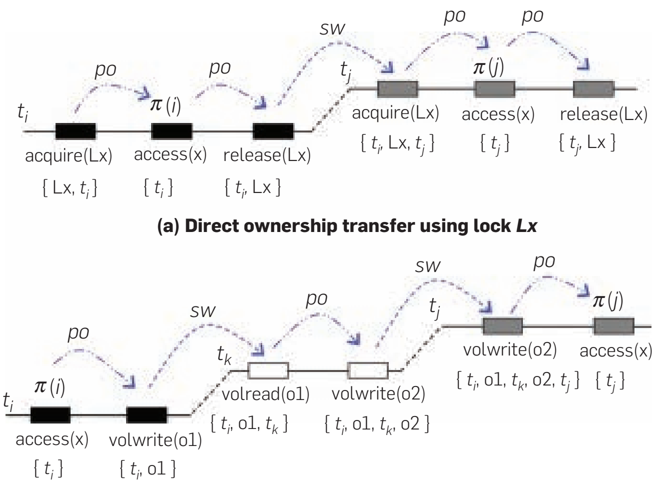
The GOLDILOCKS algorithm maintains for each data variable a "Goldilockset": a map *GLS* such that, for every data variable x, its Goldlilockset is a set $GLS(x) \subseteq Tid \cup Sync$. Roughly speaking, $GLS_i(x)$, the Goldilockset of x immediately before processing action $\pi(i)$, consists of (*i*) the id's of threads that can access x in a race-free way at that point in the execution, and (ii) the synchronization variables on which a thread can perform a sync-sink access in order to gain race-free access to x.

As GOLDILOCKS processes each action $\pi(i)$ from E, it updates the Goldilocksets of variables. Initially, the Goldilockset $GLS(x)$ is empty for all data variables, including static ones. When a new object is created, the Goldilockset for all of its instance fields is initialized to the empty set. After every action, the Goldilockset of every data variable x is potentially updated. For every data variable x, three simple rules specify how $GLS(x)$ is updated after $\pi(i)$ based on whether $\pi(i)$ is (1) a synchronization source, (2) a synchronization sink, or (3) a read or write access to x, as shown in the procedure *ApplyLocksetRules* in Figure 2.

If the action $\pi(i)$ is a synchronization operation on a variable o, we update the lockset $GLS(x)$ for every data variable x in *Data*. If $\pi(i)$ is a sync-source operation, rule 1 adds o to $GLS(x)$ if it contains the id t of the current accessor thread. Intuitively, this represents that a later sync-sink operation by a thread u on synchronization variable o will be sufficient for u to gain race-free access to x. This is formalized by rule 2. If $\pi(i)$ is a sync-sink(o) operation, rule 2 checks whether the synchronization variable o is in $GLS(x)$. If this is the case, then t is added to the Goldilockset.

If the action $\pi(i)$ is an access to a data variable x, rule 3 checks the Goldilockset of the variable $GLS(x)$ to decide whether this access is race free. If $GLS(x)$ is empty, it indicates that x is a fresh variable which has not been accessed so far and any access to x at this point is race free. If $GLS(x)$ is not empty, only threads whose id's are in $GLS(x)$ can perform race-free accesses to x. If the accessing thread's id t is not in $GLS(x)$ then we throw a DataRaceException on x. Otherwise, the access to x is race free and $GLS(x)$ becomes the singleton $\{t\}$, indicating that, without further synchronization operations, only t can access x in a race free manner.

Figure 1. Transferring ownership of *x*, and *GLS*(*x*).



(a) Direct ownership transfer using lock *Lx*

(b) Transitive ownership transfer using volatiles o1 and o2

Figure 1 shows two cases where the ownership of a data variable x is transferred from a thread t_i to another thread t_j, and indicates how the Goldilocksets evolve in each case. Program order ($\xrightarrow{po}$) and synchronizes-with ($\xrightarrow{sw}$) edges between consecutive actions are indicated in the figure. Figure 1a illustrates direct ownership transfer from t_i to t_j. After accessing x, t_i performs a sync-source operation (lock release) on synchronization variable Lx. Later, t_j obtains ownership of x by executing a sync-sink operation (lock acquire) on synchronization variable Lx. Figure 1b illustrates transitive ownership transfer. Threads t_i and t_j do not synchronize on the same synchronization variable. Instead, the synchronization involves a chain of synchronizes-with edges between other threads and synchronization variables. t_i synchronizes with t_k via synchronization variable o1 and, later t_k synchronizes with t_j via synchronization variable o2.

Rules 1 and 2 in Figure 2 require updating the lockset of each data variable. A naive implementation of this algorithm would be too expensive for programs that manipulate large heaps. In Section 4, we present an efficient way of implement our algorithm by representing Goldilocksets implicitly and by applying update rules lazily.

Figure 2. The core lockset update rules.

```
ApplyLocksetRules(π(i)):

1. if π(i) = sync-source(t, o)
       foreach x ∈ Data:
           if t ∈ GLS(x)
               GLS(x) := GLS(x) ∪ {o}

2. if π(i) = sync-sink(t, o)
       foreach x ∈ Data:
           if o ∈ GLS(x)
               GLS(x) := GLS(x) ∪ {t}

3. if π(i) = write(t, x) or π(i) = read(t, x)
       if t ∈ GLS(x) or GLS(x) = ∅
           GLS(x) := {t}
       else
           throw a DataRaceException on x
```

[b] A racy read may appear earlier in π than the write that it sees. If an execution contains a data race between a pair of accesses, GOLDILOCKS declares a race at one of these accesses regardless of which linearization π is picked.

Correctness: The following theorem expresses the fact that the GOLDILOCKS algorithm is both *sound*, i.e., detects all actual races in a given execution, and *precise*, i.e., never reports false alarms. The full proof of the original GOLDILOCKS algorithm for Java can be found in Elmas et al.[5]

THEOREM 1 (CORRECTNESS). *Let* $E = \langle Tid, Act, W, \xrightarrow{po}., \xrightarrow{so}.\rangle$ *be a well-formed execution, x a data variable, and π a linear order on Act as described earlier. Let i < j, and let π(i) and π(j) be two accesses to x performed by threads* t_i *and* t_j*, with no other action π(k) in between (i < k < j) accessing x. Then* $t_j \in GLS_j(x)$*, i.e., the access π(j) is declared to be race free by the* GOLDILOCKS *algorithm iff* $\pi(i) \xrightarrow{hb} \pi(j)$.

3.1. Example: precise data-race detection

In this section, we illustrate on an example the Goldilocks algorithm and how Goldilocksets capture the synchronization mechanism protecting access to a variable at each point in an execution. In this example, earlier lockset algorithms would have erroneously declared a race condition.

Consider the execution shown in Figure 3 in which all actions of `T1` happen first, followed by all actions of `T2` and then of `T3`. This example mimics a scenario in which an object is created and initialized and then made visible globally by `T1`. This `Int` object (referred to as *o* from now on) is a container object for its `data` field (referred to as *x* from now on). The object *o* is referred to by different global variables (`a` and `b`) and local variables (`tmp1, tmp2,` and `tmp3`) at different points in this execution. The contained variable *x* is protected by synchronization on the container object *o*, and during the execution, the lock (`La` or `Lb`) protecting *o* and *x* changes depending on which variable (`a` or `b`) points to *o*. Notice that, `T2` changes the protecting lock of the container object *o* from `La` to `Lb`, without accessing *x*. Figure 3 shows the GOLDILOCKS update rules applied on *GLS*(*x*) for each action and the resulting value of *GLS*(*x*).

Figure 3. Precise data-race detection example.

```
Class Int {  int data; }
Int a, b; // Global variables
```

Execution	Goldilockset update rule applied	*GLS*(*x*)
Thread 1 (`T1`):		
`tmp1 = new Int;`	Initialize lockset	∅
`tmp1.data = 0;`	First access	`{T1}`
`acquire(La);`	`La` ∈ *GLS*(*x*) → *add* `T1` *to GLS*(*x*)	`{T1}`
`a = tmp1;`	No access to *x*	`{T1}`
`release (La);`	`T1` ∈ *GLS*(*x*) → *add* `La` *to GLS*(*x*)	`{T1,La}`
Thread 2 (`T2`):		
`acquire(La);`	`La` ∈ *GLS*(*x*) → *add* `T2` *to GLS*(*x*)	`{T1,La,T2}`
`tmp2 = a;`	No access to *x*	`{T1,La,T2}`
`release(La);`	`T2` ∈ *GLS*(*x*) → *add* `La` *to GLS*(*x*)	`{T1,La,T2}`
`acquire(Lb);`	`Lb` ∈ *GLS*(*x*) → *add* `T2` *to GLS*(*x*)	`{T1,La,T2}`
`b = tmp2;`	No acces s to *x*	`{T1,La,T2}`
`release(Lb);`	`T2` ∈ *GLS*(*x*) → *add* `Lb` *to GLS*(*x*)	`{T1,La,T2,Lb}`
Thread 3 (`T3`):		
`acquire(Lb);`	`Lb` ∈ *GLS*(*x*) → *add* `T3` *to GLS*(*x*)	
`{T1,La,T2,Lb,T3}`		
`b.data = 2;`	`T3` ∈ *GLS*(*x*) → Race-free access	`{T3}`
`tmp3 = b;`	No access to *x*	`{T3}`
`release(Lb);`	`T3` ∈ *GLS*(*x*) → *add* `Lb` *to GLS*(*x*)	`{T3,Lb}`
`tmp3.data = 3;`	`T3` ∈ *GLS*(*x*) → Race-free access	`{T3}`

Observe that the update rules allow a variable's Goldilockset to grow during the execution. This enables them to represent threads transfering ownership using different synchronization variables during the execution. In this example, this ownership transfer takes place without the variable itself being accessed. For example, after `T2` finishes in Figure 2, *GLS*(*x*) has the value {`T1`, `La`, `T2`, `Lb`}, meaning that a thread can access *x* without data race by locking either `La` or `Lb`. Then `T3` makes `Lb` the only protecter lock by acquiring `Lb` and accesses *x*.

3.2. Distinguishing read and write accesses

The basic GOLDILOCKS algorithm in Figure 2 tracks the happens-before relationship between any two accesses to a variable *x*. In order to perform race detection, we must check the happens-before relationship only between conflicting actions, i.e., at least one action in the pair must be a write access. We extend the basic GOLDILOCKS algorithm by keeping track of (i) $GLS^W(x)$, the "write Goldilockset of *x*", and (ii) $GLS^R(t, x)$, the "read Goldilockset of *t* and *x*" for each thread *t*. The update rules in *ApplyLocksetRules* are adapted to maintain these Goldilocksets, but have essentially the same form as the rules in Figure 2. In the extended algorithm, it is sufficient to check happens-before between the current access to *x* and the most recent accesses (in the linear order *π*) to *x*. How this extension is performed for Java can be found in Elmas et al.[6]

3.3. Specializing Goldilocks to the JMM

The JMM requires that *all* synchronization operations be ordered by a total order $\xrightarrow{so}$, whereas in our execution model, a separate total order $\xrightarrow{so}_o$ per synchronization variable is sufficient.

Data Variables and Operations: In Java, every data variable is in the form of (*o*, *d*) where *o* is an object and *d* is a nonvolatile field. The byte-code instructions *x load* and *xstore* access memory to read from and write to fields of objects, respectively (*x* changes depending on the type of the field).

The JMM specifies three synchronization mechanisms: monitors, volatile fields, and fork/join operations.

Monitors: In Java, a monitor per object (denoted by m_o) provides a reentrant lock for each object *o*. Acquiring the lock of an object *o* (acquire(*o*)) corresponds to a sync-sink operation on m_o, while releasing the lock of *o* (release(*o*)) corresponds to a sync-source operation on m_o. Nested acquires and releases of the same lock are treated as no-ops. In the JMM, each release(*o*) synchronizes with the next acquire(*o*) operation in $\xrightarrow{so} m_o$.

Volatile Variables: Each volatile variable is denoted (*o*, *v*) where *o* is an object, and *v* is a volatile field. Each read volread(*o*, *v*) from a volatile variable (*o*, *v*), and each write volwrite(o, v) to (o, v) is implemented by the *xload* and *xstore* byte-code instructions, respectively. While volread(o, v) corresponds to a sync-sink, volwrite(o, v) corresponds to sync-source operation on (o, v). In the JMM, there is a synchronizes-with relationship between each volread(o, v) and the volwrite(o, v) that it sees.

Fork/Join: Creating a new thread with id t (fork(t)) synchronizes with the first action of thread t, denoted start(t). The last action of thread t, denoted end(t) synchronizes with the join operation on t, denoted join(t). For each thead t, fork(t) and end(t) correspond to sync-source operations on a (fictitious) synchronization variable $\bar{t}$, and start(t) and join(t) correspond to sync-sink operations on $\bar{t}$. The JMM guarantees that for each thread t, there exists an order $\xrightarrow{so}_{\bar{t}}$ such that: fork(t) $\xrightarrow{so}_{\bar{t}}$ start(t) $\xrightarrow{so}_{\bar{t}}$ end(t) $\xrightarrow{so}_{\bar{t}}$ join(t).

Handling other Synchronization Mechanisms: Using the lockset update rules above, Goldilocks is able to uniformly handle various approaches to synchronization such as dynamically changing locksets, permanent or temporary thread-locality of objects, container-protected objects, ownership transfer of variable without accessing the variable (as in the example in Section 3.1). Furthermore, Goldilocks can also handle the synchronization idioms in the `java.util.concurrent` package such as semaphores and barriers, since these primitives are built using locks and volatile variables. The happens-before edges induced by the wait/notify(All) construct are computed by simply applying the Goldilockset update rules to the acquire and release operations invoked inside wait.

3.4. Race detection and sequential consistency

The Java and C++ memory models provide the data-race freedom (DRF) property.[2, 10] The DRF property guarantees that if all sequentially consistent executions of a *source* program are race free, then the *compiled* program only exhibits these sequentially consistent executions of the source program, after any compiler and hardware optimizations permitted by the memory model. The Goldilocks algorithm check races by monitoring the executions of the compiled program, and assumes that the compiler and the runtime it is built on (hardware or virtual machine) conform to the language and the memory model specifications. Therefore, if the source program is race free, then any execution of the compiled program corresponds to a sequentially consistent execution of the source program, and no `DataRaceException` is thrown.

If the source program has a race, the Goldilocks runtime still ensures that all executions of the compiled program will run under the sequential consistency semantics, i.e., sequential consistency is guaranteed at the byte-code level. This is accomplished by preventing accesses that will cause a data race and throwing a `DataRaceException` right before that access. However, in the case of a racy program, the JMM permits compiler optimizations that result in executions that are not sequentially consistent behaviors of the original source code. In this case, the JMM and the DRF property are not strong enough to allow the Goldilocks runtime to relate byte-code level executions to executions of the source-level program, which makes debugging hard.

To use Goldilocks for debugging purposes, this difficulty can be remedied by disabling compiler optimizations. For post-deployment use, a stronger memory model[9, 11] that is able to relate each (racy and race-free) execution of the compiled program to a sequentially consistent execution of the source program is needed.

4. IMPLEMENTING GOLDILOCKS

There are two published implementation of the Goldilocks algorithm, both of which monitor the execution at the Java byte-code level. At this level, each variable access or synchronization operation corresponds to a single byte-code instruction, and each byte-code instruction can be associated with a source code line and/or variable.

The first Goldilocks implementation, by the authors of this paper, was carried out in Kaffe,[19] a clean-room implementation of the Java virtual machine (JVM) in C. In Kaffe, we integrated Goldilocks into the interpreting mode of Kaffe's runtime engine. Implementing the algorithm in the JVM enables fast access to internal data structures of the JVM that manage the layout of object in the memory and using the efficient mechanisms that exist in the JVM internally, such as fast mutex locks.

The second implementation of Goldilocks is by Flanagan and Freund and was carried out using the RoadRunner dynamic program analysis tool.[8] In RoadRunner, Goldilocks is implemented in Java and injected by byte-code instrumentation at load-time of the program. This allows the algorithm to benefit from Java compiler optimizations and just-in-time compilation and to be portable to any JVM. Flanagan and Freund showed that this implementation is competitive with ours in Kaffe for most of the common benchmarks.[7]

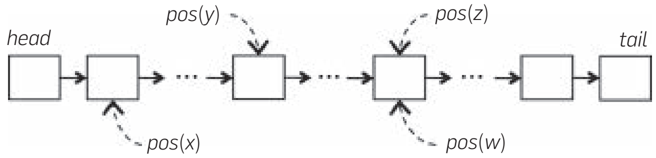
In the following, we present the most important implementation features and optimizations. The implementation is described based on the core algorithm presented in Figure 2. The extension of the implementation that distinguishes read and write accesses can be found in Elmas et al.[6]

4.1. Implicit representation and lazy evaluation of Goldilocksets

For programs with a large number of data variables, representing Goldilocksets explicitly for each data variable and implementing the Goldilocks algorithm as described in Figure 2 may have high memory and computational cost. We avoid the memory cost by representing the Goldilocksets *implicitly* and the computational cost by evaluating Goldilocksets *lazily* as described below.

Instead of keeping a separate Goldilockset $GLS(x)$ for each variable x, we represent $GLS(x)$ implicitly as long as no access to x happens and is computed temporarily when an access happens. At this point, the Goldilockset is a singleton, and we continue to represent it implicitly until the next access. For this, we keep the synchronization events in a single, global linked list called the *synchronization-event list* and represent by its *head* and *tail* pointers in Figure 4. The ordering of these events in the list is consistent with the program order $\xrightarrow{po}_{t}$ for each thread t and the synchronization orders $\xrightarrow{so}_{o}$ for

Figure 4. The synchronization event list.



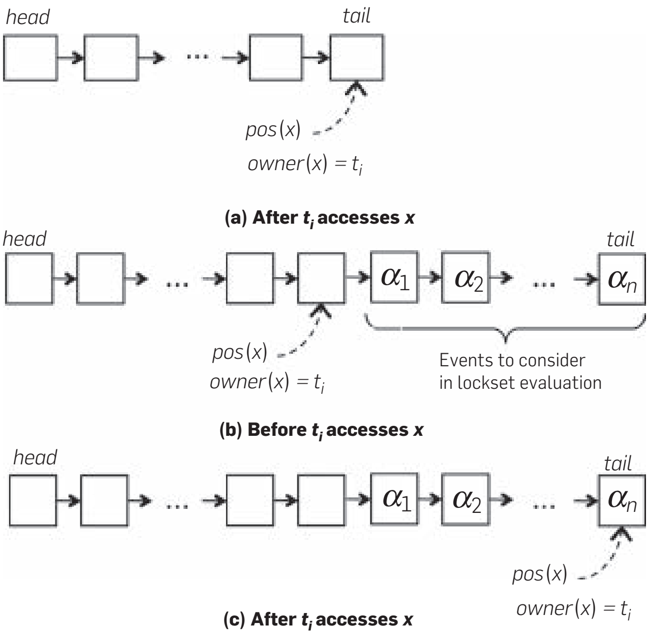
each synchronization variable o.[c] When a thread performs a synchronization action α, it must append a corresponding event to the synchronization-event list atomically with the event. In Kaffe, we make sure this is the case by modifying the implementations of the Java synchronization actions.

In order to represent $GLS(x)$, each variable x in the program is associated with two bits of information regarding the most recent access to x: $owner(x)$ stores the id of the thread that most recently accessed x, and $pos(x)$ points to the last synchronization event in the list that was taken into account when $GLS(x)$ was last computed.

Figure 4 shows four variables pointing to entries in the synchronization-event list. Figure 5 shows how the Goldilockset $GLS(x)$ is computed when x is accessed.

5(a): After each access to x by a thread t_i, $owner(x)$ is set to t_i, and $pos(x)$ is set to point the *tail* of the synchronization event list.
5(b): Right before an access to x by thread t_j, temporarily, we represent $GLS(x)$ explicitly. $GLS(x)$ is initially $\{owner(x)\}$ and is updated by processing the synchronization events between $pos(x)$ (denoted by $\alpha_1, \ldots, \alpha_n$) and *tail* according to the rules 1 and 2 of Figure 2. This process stops either when t_j is added to $GLS(x)$ or the last event (α_n) is processed. In the former case, no race is reported according to the rule 3 of Figure 2. In the latter case, a race is reported since $t_j \notin GLS(x)$ after the evaluation.

Figure 5. Lazy evaluation of Goldilockset *GLS*(*x*).



(a) After t_i accesses x

(b) Before t_j accesses x

(c) After t_j accesses x

5(c): After the check, $owner(x)$ is set to t_j and $pos(x)$ is set to the *tail* of the synchronization event list.

The implementation does not use any extra threads for race detection. The algorithm is performed in a decentralized manner by instrumented threads of the program being checked. For each data variable x, we use a unique lock to make atomic the Goldilockset update and the race-freedom check for each access to x and to serialize all the race-freedom checks for x.

4.2. Performance optimizations

Short-Circuit Checks: A cheap-to-evaluate sufficient condition for a happens-before edge between the last two accesses to a variable can reduce race-detection overhead. We make use of two such conditions, called short-circuit checks, and bypass the traversal of the synchronization event list when these checks succeed. In this case, the final Goldilockset of the variable consists of the id of the thread that accessed it last.

We employ two constant-time short-circuit checks. First, when the last two accesses to a shared variable are performed by the same thread t, the happens-before relationship is guaranteed by the program order of t. This is detected by checking whether $owner(t)$, the last accessor thread, is the same as the thread performing the current access.

In the second short-circuit check, we determine whether the variable x is protected by the same lock during the last two accesses to x. For this, we associate with each variable x a lock $alock(x)$, which is randomly selected among the locks held by the most recent accessor thread. When a thread t accesses x and if $alock(x)$ is held by t, then that access is race free.

Direct Ownership Transfer: A sound but imprecise third optimization is to consider only the subset of synchronization events executed by the current and last accessing thread when examining the portion of the synchronization event list between $pos(x)$ and *tail*. This check is not constant time, but we found that it succeeds often enough to improve GOLDILOCKS overhead.

Garbage Collection: The synchronization events list is periodically garbage-collected when there are entries in the beginning of the list that are not relevant for the Goldilockset computation of any variable. This is the case when an entry in the list is not reachable from $pos(x)$ for any data variable x, and is tracked by maintaining incremental reference counts for each list entry.

Partially Eager Evaluation: Sometimes the synchronization event list gets too long and it is not possible to garbage-collect the event list when variable x is accessed early in an execution but is not used afterwards. We address this problem by "partially eager" Goldilockset evaluation. We move $pos(x)$ forward towards the *tail* to a new position $pos'(x)$, and partially evaluate a Goldilockset $GLS(x)$ of x by processing events (i.e., running *ApplyLocksetRules* on them)

[c] For Java, there is a total order on all synchronization operations, and the entries in the list are in this order.

between $pos(x)$ and $pos'(x)$. During the next access the evaluation of $GLS(x)$ starts from the stored Goldilockset, not from $\{owner(x)\}$.

Sound Static Race Analysis: The runtime overhead of race detection is directly related to the number of data variable accesses checked and synchronization events that occur. To reduce the number of accesses checked at runtime, we use static analysis at compile time to determine accesses that are guaranteed to be race free. While implementing Goldilocks in Kaffe, we worked with two static analysis tools for this purpose: Chord[13] and RccJava.[1]

4.3. Race-detection overhead

At the time of the original Goldilocks work, the vector clock algorithm[12] was the only precise dynamic-race-detection algorithm in the literature. The vector clock algorithm, for an execution with n threads, requires for every thread and synchronization variable a separate vector clock (VC) of size n and performs $O(n)$ operations (merging or comparing two VCs) whenever a synchronization operation or data access happens. In preliminary research, compared to a straightforward implementation of vector clocks, we found Goldilocks overhead to be significantly less.[6]

In Elmas et al.,[6] we measured the overhead of the Goldilocks implementation inside Kaffe on a set of widely used Java benchmarks. This implementation required us to run all programs in interpreted (not just-in-time compiled) mode. We found that, with powerful static analysis tools eliminating much of the monitoring, we were able to obtain a slowdown of within approximately 2 for all benchmarks. Without static elimination of some checks, overheads remained high; some benchmarks experienced slowdowns of over 15. The overhead results with static pre-elimination were encouraging in that they showed precise race detection to be a practical debugging tool, and they indicated that, with further optimizations, post-deployment runtime race detection to support `DataRaceDetection` could be viable.

Later work on FastTrack,[7] a dynamic race detector based on vector clocks, is able to avoid worst-case performance of vector clocks much of the time using optimizations for common cases. Flanagan and Freund[7] compare a number of race-detection algorithms, including just-in-time compiled implementations of FastTrack and Goldilocks in RoadRunner. FastTrack achieves significantly better overheads than both implementations of Goldilocks. The low overheads achieved by FastTrack provide further support that a practical race-aware runtime for deployed programs supporting a `DataRaceException` can be built. It is reported in Flanagan and Freund[7] that additional short-circuit checks similar to ones we discussed above dramatically reduce the runtime of FastTrack. Most of these checks can be incorporated into Goldilocks implementations as well.

5. CONCLUSION

We have presented a race-aware runtime for Java incorporating a novel algorithm, Goldilocks, for precise dynamic race detection. The runtime provides a `DataRaceException`, and thus ensures that executions remain sequentially consistent at the byte-code level. Experiments with Goldilocks have demonstrated that the runtime overhead of supporting a `DataRaceException` can be made reasonable.

Acknowledgments

We would like to thank Hans Boehm, Cormac Flanagan, Steve Freund, and Madan Musuvathi for their critique of this paper. This research was supported by the Software Reliability Research Group at Microsoft Research, Redmond, WA, by the Scientific and Technical Research Council of Turkey (TUBITAK) under grant 104E058, and by the Turkish Academy of Sciences (TUBA).

References

1. Abadi, M, Flanagan, C., Freund, S.N. Types for safe locking: Static race detection for Java. *ACM Trans. Program. Lang. Syst.* (2006).
2. Boehm, H.-J., Adve, S.V. Foundations of the C++ concurrency memory model. In *Proceedings of the 2008 ACM SIGPLAN Conf. on Programming Language Design and Implementation* (PLDI 2008)
3. Cheng, G.-I., Feng, M., Leiserson, C.E., Randall, K.H., Stark, A.F. Detecting data races in Cilk programs that use locks. In *ACM Symposium on Parallel Algorithms and Architectures* (SPAA 1998).
4. Christiaens, M. De Bosschere, K. TRaDe: Data race detection for Java. *Proc. Intl. Conference on Computational Science*. V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, eds. (ICCS 2001).
5. Elmas, T., Qadeer, S., Tasiran, S. Goldilocks: Efficiently computing the happens-before relation using locksets. *Technical Report MSR-TR-2006–163, Microsoft Research* (2006).
6. Elmas, T., Qadeer, S., Tasiran, S. Goldilocks: A race and transaction-aware java runtime. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2007).
7. Flanagan, C., Freund, S.N. FastTrack: Efficient and precise dynamic race detection. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2009).
8. Flanagan, C., Freund, S.N. The RoadRunner dynamic analysis framework for concurrent programs, In *ACM Workshop on Program Analysis for Software Tools and Engineering* (PASTE 2010).
9. Lucia, B., Ceze, L., Strauss, K., Qadeer, S., Boehm H.-J. Conflict exceptions: Simplifying concurrent language semantics with precise hardware exceptions for data-races. In *Proceedings of the 37th International Symposium on Computer Architecture* (ISCA 2010).
10. Manson, J., Pugh, W., Adve, S.V. The Java memory model. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (POPL 2005).
11. Marino, D., Singh, A., Millstein, T., Musuvathi, M., Narayanasamy, S. DRFx: A simple and efficient memory model for concurrent programming languages. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2010).
12. Mattern, F. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms* (1988).
13. Naik, M., Aiken, A., Whaley, J. Effective static race detection for Java. In *Proceedings 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2006).
14. Pozniansky, E., Schuster, A. Multirace: Efficient on-the-fly data race detection in multithreaded C++ programs: Research articles. *Concurr. Comput. Pract. Exp.* (2007).
15. Ronsse, M., Bosschere, K.D. RecPlay: A fully integrated practical record/replay system. *ACM Trans. Comput. Syst.* (1999).
16. Savage, S., Burrows, M., Nelson, G., Sobalvarro, P., Anderson, T. Eraser: A dynamic data race detector for multithreaded programs. *ACM Trans. Comput. Syst.* (1997).
17. Schonberg, E. On-the-fly detection of access anomalies. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 1989).
18. Shavit, N., Touitou, D. Software transactional memory. In *Symposium on Principles of Distributed Computing* (1995).
19. Wilkinson, T. Kaffe: A JIT and interpreting virtual machine to run Java code. http://www.transvirtual.com (1998).
20. Yu, Y., Rodeheffer, T., Chen, W. RaceTrack: Efficient detection of data race conditions via adaptive tracking. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles* (SOSP 2005).

Tayfun Elmas (telmas@ku.edu.tr), Koç University, Istanbul, Turkey.

Shaz Qadeer (qadeer@microsoft.com), Microsoft Research, Redmond, WA.

Serdar Tasiran (stasiran@ku.edu.tr), Koç University, Istanbul, Turkey.

DOI:10.1145/1839676.1839699

FastTrack: Efficient and Precise Dynamic Race Detection

By Cormac Flanagan and Stephen N. Freund

Abstract

Multithreaded programs are notoriously prone to race conditions. Prior work developed precise dynamic race detectors that never report false alarms. However, these checkers employ expensive data structures, such as vector clocks (VCs), that result in significant performance overhead.

This paper exploits the insight that the full generality of VCs is not necessary in most cases. That is, we can replace VCs with an adaptive lightweight representation that, for almost all operations of the target program, requires constant space and supports constant-time operations. Experimental results show that the resulting race detection algorithm is over twice as fast as prior precise race detectors, with no loss of precision.

1. INTRODUCTION

Multithreaded programs are prone to race conditions and other concurrency errors such as deadlocks and violations of expected atomicity or determinism properties. The broad adoption of multicore processors is exacerbating these problems, both by driving the development of multithreaded software and by increasing the interleaving of threads in existing multithreaded systems.

A race condition occurs when two threads concurrently perform memory accesses that *conflict*. Accesses conflict when they read or write the same memory location and at least one of them is a write. In this situation, the order in which the two conflicting accesses are performed can affect the program's subsequent state and behavior, likely with unintended and erroneous consequences.

Race conditions are notoriously problematic because they typically cause problems only on rare interleavings, making them difficult to detect, reproduce, and eliminate. Consequently, much prior work has focused on static and dynamic analysis tools for detecting race conditions.

To maximize test coverage, race detectors use a very broad notion of when two conflicting accesses are considered concurrent. The accesses need not be performed at exactly the same time. Instead, the central requirement is that there is no "synchronization dependence" between the two accesses, such as the dependence between a lock release by one thread and a subsequent lock acquire by a different thread. These various kinds of synchronization dependencies form a partial order over the instructions in the execution trace called the *happens-before relation*.[13] Two memory accesses are then considered to be *concurrent* if they are not ordered by this happens-before relation.

In this paper, we focus on online dynamic race detectors, which generally fall into two categories depending on whether they report false alarms. *Precise* race detectors never produce false alarms. Instead, they compute a precise representation of the happens-before relation for the observed trace and report an error if and only if the observed trace has a race condition. Note that there are typically many possible traces for a particular program, depending on test inputs and scheduling choices. Precise dynamic race detectors do not reason about all possible traces, however, and may not identify races that occur only when other code paths are taken. While full coverage is desirable, it comes at the cost of potential false alarms because of the undecidability of the halting problem. To avoid these false alarms, precise race detectors focus on detecting only race conditions that occur on the observed trace.

Typically, precise detectors represent the happens-before relation with *vector clocks* (VCs),[14] as in the DJIT$^+$ race detector.[16] Vector clocks are expensive to maintain, however, because a VC encodes information about each thread in a system. Thus, if the target program has n threads, each VC requires $O(n)$ storage space and VC operations (such as comparison) require $O(n)$ time. Since a VC must be maintained for each memory location and modified on each access to that location, this $O(n)$ time and space overhead precludes the use of VC-based race detectors in many settings.

A variety of alternative *imprecise* race detectors have been developed, which may provide improved performance (and sometimes better coverage), but which report false alarms on some race-free programs. For example, Eraser's LockSet algorithm[18] enforces a lock-based synchronization discipline and reports an error if no lock is consistently held on each access to a particular memory location. Eraser may report false alarms, however, on programs that use alternative synchronization idioms such as `fork/join` or barrier synchronization. Some LockSet-based race detectors include limited happens-before reasoning to improve precision in such situations.[15, 16, 22]

Other optimizations include using static analyses or dynamic escape analyses[3, 21] or using "accordion" VCs that reduce space overheads for programs with shortlived threads.[5] Alternative approaches record program events for *post-mortem* race identification.[1, 4, 17]

Although these imprecise tools successfully detect race conditions, their potential to generate many false alarms limits their effectiveness. Indeed, it has proven surprisingly difficult and time consuming to identify the real errors among

The original version of this paper was published in the *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2009.

the spurious warnings produced by some tools. Even if a code block looks suspicious, it may still be race-free due to some subtle synchronization discipline that is not (yet) understood by the current code maintainer. Even worse, additional real bugs (e.g., deadlocks or performance problems) could be added while attempting to "fix" a spurious warning produced by these tools. Conversely, real race conditions could be ignored because they appear to be false alarms.

This paper exploits the insight that, while VCs provide a general mechanism for representing the happens-before relation, their full generality is not actually necessary in most cases. The vast majority of data in multithreaded programs is either thread local, lock protected, or read shared. Our FastTrack analysis uses an adaptive representation for the happens-before relation to provide constant-time and constant-space overhead for these common cases, without any loss of precision or correctness.

In more detail, a VC-based race detector such as Djit$^+$ records the time of the most recent write to each variable x by each thread t. By comparison, FastTrack exploits the observation that all writes to x are totally ordered by the happens-before relation, assuming no races on x have been detected so far, and records information only about the *very last* write to x. Specifically, FastTrack records the clock and thread identifier of that write. We refer to this pair of a clock and a thread identifier as an *epoch*.

Read operations on thread-local and lock-protected data are also totally ordered, assuming no races on x have been detected, and FastTrack records only the epoch of the last read from such data. FastTrack adaptively switches from epochs to VCs when necessary (e.g., when data becomes read-shared) in order to guarantee no loss of precision. It also switches from VCs back to lightweight epochs when possible (e.g., when read-shared data is subsequently updated).

Using these representation techniques, FastTrack reduces the analysis overhead of almost all monitored operations from $O(n)$ time, where n is the number of threads in the target program, to $O(1)$ time.

In addition to improving performance, the epoch representation also reduces space overhead. A VC-based race detector requires $O(n)$ space for each memory location of the target program and can quickly exhaust memory resources. By comparison, FastTrack reduces the space overhead for thread-local and lock-protected data from $O(n)$ to $O(1)$.

For comparison purposes, we implemented six different dynamic race detectors: FastTrack plus five other race detectors described in the literature. Experimental results on Java benchmarks, including the Eclipse programming environment, show that FastTrack outperforms the other tools. For example, it provides almost a 10x speedup over a traditional VC-based race detector and a 2.3x speedup over the Djit$^+$ algorithm. It also provides a substantial increase in precision over Eraser, with no loss in performance.

2. PRELIMINARIES

2.1. Multithreaded program traces

We begin with some terminology and definitions regarding multithreaded execution traces. A program consists of a number of concurrently executing threads, each with a thread identifier $t \in Tid$. These threads manipulate variables $x \in Var$ and locks $m \in Lock$. A *trace* α captures an execution of a multithreaded program by listing the sequence of *operations* performed by the various threads. The operations that a thread t can perform include:

- $rd(t, x)$ and $wr(t, x)$, which read and write a value from a variable x
- $acq(t, m)$ and $rel(t, m)$, which acquire and release a lock m
- $fork(t, u)$, which forks a new thread u
- $join(t, u)$, which blocks until thread u terminates

The *happens-before relation* ($<_\alpha$) for a trace α is a partial order over the operations in α that captures control and synchronization dependencies. In particular, the relation $a <_\alpha b$ holds whenever operation a occurs before operation b in α and one of the following conditions applies:

- **Program order**: The two operations are performed by the same thread.
- **Synchronization order**: The two operations acquire or release the same lock.
- **Fork order**: The first operation is $fork(t, u)$ and the second is by thread u.
- **Join order**: The first operation is by thread u and the second is $join(t, u)$.

In addition, the happens-before relation is transitively closed: that is, if $a <_\alpha b$ and $b <_\alpha c$ then $a <_\alpha c$.

If a happens before b, then we also say that b *happens after* a. If two operations in a trace are not related by the happens-before relation, then they are considered *concurrent*. Two memory access *conflict* if they both access (read or write) the same variable, and at least one of the operations is a write. Using this terminology, a trace has a *race condition* if it has two concurrent conflicting accesses.

2.2. Vector clocks and the Djit$^+$ algorithm

Before presenting the FastTrack algorithm, we briefly review the Djit$^+$ online race detection algorithm,[16] which is based on VCs.[14] A VC

$$V : Tid \rightarrow Nat$$

records a clock for each thread in the system. Vector clocks are partially-ordered ($\sqsubseteq$) in a pointwise manner, with an associated join operation ($\sqcup$) and minimal element ($\perp_V$). In addition, the helper function inc_t increments the t-component of a VC:

$$\begin{aligned}
V_1 \quad V_2 \; & \text{iff} \; \forall t.\; V_1(t) \leq V_2(t) \\
V_1 \quad V_2 \; & = \; \lambda t.\; max(V_1(t), V_2(t)) \\
\perp_V \; & = \; \lambda t.\; 0 \\
inc_t(V) \; & = \; \lambda u.\; \text{ if } u = t \text{ then } V(u) + 1 \text{ else } V(u)
\end{aligned}$$

In DJIT$^+$, each thread has its own clock that is incremented at each lock release operation. Each thread t also keeps a VC $\mathbb{C}_t$ such that, for any thread u, the clock entry $\mathbb{C}_t(u)$ records the clock for the last operation of thread u that happens before the current operation of thread t. In addition, the algorithm maintains a VC $\mathbb{L}_m$ for each lock m. These VCs are updated on synchronization operations that impose a happens-before order between operations of different threads. For example, when thread u releases lock m, the DJIT$^+$ algorithm updates $\mathbb{L}_m$ to be $\mathbb{C}_u$. If a thread t subsequently acquires m, the algorithm updates $\mathbb{C}_t$ to be $\mathbb{C}_t \sqcup \mathbb{L}_m$, since subsequent operations of thread t now happen after that release operation.

To identify conflicting accesses, the DJIT$^+$ algorithm keeps two VCs, $\mathbb{R}_x$ and $\mathbb{W}_x$, for each variable x. For any thread t, $\mathbb{R}_x(t)$ and $\mathbb{W}_x(t)$ record the clock of the last read and write to x by thread t. A read from x by thread u is race-free provided it happens after the last write of each thread, that is, $\mathbb{W}_x \sqsubseteq \mathbb{C}_u$. A write to x by thread u is race-free provided that the write happens after all previous accesses to that variable, that is, $\mathbb{W}_x \sqsubseteq \mathbb{C}_u$ and $\mathbb{R}_x \sqsubseteq \mathbb{C}_u$.

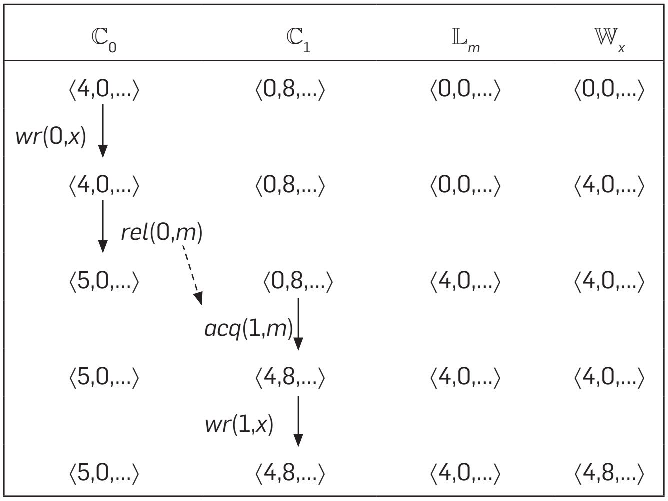
As an example, consider the execution trace fragment shown in Figure 1, where we include the relevant portion of the DJIT$^+$ instrumentation state: the VCs $\mathbb{C}_0$ and $\mathbb{C}_1$ for threads 0 and 1; and the VCs $\mathbb{L}_m$ and $\mathbb{W}_x$ for the last release of lock m and the last write to variable x, respectively. We show two components for each VC, but the target program may of course contain additional threads.[a]

At the write $wr(0, x)$, DJIT$^+$ updates $\mathbb{W}_x$ with current clock of thread 0. At the release $rel(0, m)$, $\mathbb{L}_m$ is updated with $\mathbb{C}_0$. At the acquire $acq(1, m)$, $\mathbb{C}_1$ is joined with $\mathbb{L}_m$, thus capturing the dashed release-acquire happens-before edge shown above. At the second write, DJIT$^+$ compares the VCs:

$$\mathbb{W}_x = \langle 4, 0, \ldots\rangle \sqsubseteq \langle 4, 8, \ldots\rangle = \mathbb{C}_1$$

Since this check passes, the two writes are not concurrent, and no race condition is reported.

Figure 1. Execution trace under DJIT$^+$.



3. THE FastTrack ALGORITHM

A limitation of VC-based race detectors such as DJIT$^+$ is their performance, since each VC requires $O(n)$ space and each VC operation (copying, comparing, joining, etc.) requires $O(n)$ time.

Empirical benchmark data indicates that reads and writes operations account for the vast majority (over 96%) of monitored operations. The key insight behind FastTrack is that the full generality of VCs is not necessary in over 99% of these read and write operations: a more lightweight representation of the happens-before information can be used instead. Only a small fraction of operations performed by the target program necessitate expensive VC operations.

We begin by providing an overview of how our analysis catches each type of race condition on a memory location. A race condition is either: a *write–write race condition* (where a write is concurrent with a later write); a *write–read race condition* (where a write is concurrent with a later read); or a *read–write race condition* (where a read is concurrent with a later write).

Detecting Write–Write Races: We first consider how to efficiently analyze write operations. At the second write operation in the trace in Figure 1, DJIT$^+$ compares the VCs $\mathbb{W}_x \sqsubseteq \mathbb{C}_1$ to determine whether there is a race. A careful inspection reveals, however, that it is not necessary to record the *entire* VC $\langle 4, 0, \ldots\rangle$ from the first write to x. Assuming no races have been detected on x so far, then all writes to x are totally ordered by the happens-before relation, and the *only critical information* that needs to be recorded is the clock (4) and identity (thread 0) of the thread performing the last write. This information (clock 4 of thread 0) is sufficient to determine if a subsequent write to x is in a race with any preceding write.

We refer to a pair of a clock c and a thread t as an *epoch*, denoted $c@t$. Although rather simple, epochs provide the crucial lightweight representation for recording sufficiently-precise aspects of the happens-before relation efficiently. Unlike VCs, an epoch requires only constant space and supports constant-time operations.

An epoch $c@t$ *happens before* a VC V ($c@t \preceq V$) if and only if the clock of the epoch is less than or equal to the corresponding clock in the vector:

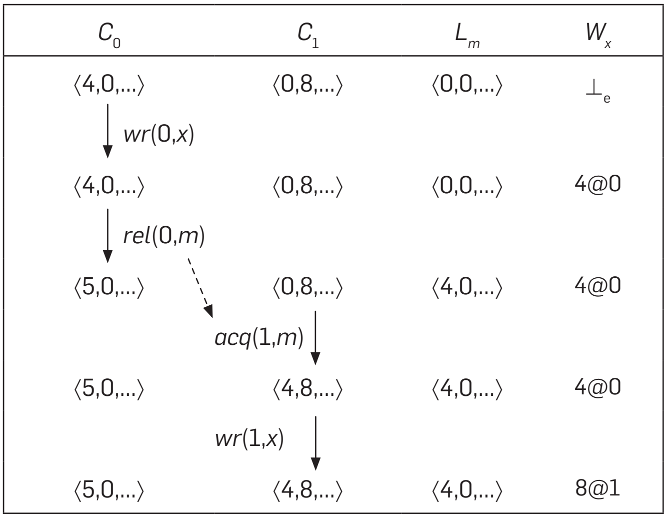
$$c@t \preceq V \quad \text{iff} \quad c \leq V(t)$$

We use $\bot_e$ to denote a minimal epoch 0@0.

Using this optimized representation, FastTrack analyzes the trace from Figure 1 using a more compact instrumentation state that records only a write epoch W_x for variable x, rather than the entire VC $\mathbb{W}_x$, reducing space overhead, as

[a] For clarity, we present a variant of the DJIT$^+$ algorithm where some clocks are one less than in the original formulation.[16] This revised algorithm has the same performance as the original but is slightly simpler and more directly comparable to FastTrack.

Figure 2. Execution trace under FastTrack.



C_0	C_1	L_m	W_x
$\langle 4,0,\ldots\rangle$	$\langle 0,8,\ldots\rangle$	$\langle 0,0,\ldots\rangle$	$\bot_e$
$wr(0,x)$			
$\langle 4,0,\ldots\rangle$	$\langle 0,8,\ldots\rangle$	$\langle 0,0,\ldots\rangle$	4@0
$rel(0,m)$			
$\langle 5,0,\ldots\rangle$	$\langle 0,8,\ldots\rangle$	$\langle 4,0,\ldots\rangle$	4@0
	$acq(1,m)$		
$\langle 5,0,\ldots\rangle$	$\langle 4,8,\ldots\rangle$	$\langle 4,0,\ldots\rangle$	4@0
	$wr(1,x)$		
$\langle 5,0,\ldots\rangle$	$\langle 4,8,\ldots\rangle$	$\langle 4,0,\ldots\rangle$	8@1

shown in Figure 2. (C and L record the same information as $\mathbb{C}$ and $\mathbb{L}$ in Djit$^+$.)

At the first write to x, FastTrack performs an $O(1)$-time epoch write $W_x := 4@0$. FastTrack subsequently ensures that the second write is not concurrent with the preceding write via the $O(1)$-time comparison:

$$W_x = 4@0 \preceq \langle 4, 8, \ldots\rangle = C_1$$

To summarize, epochs reduce the space overhead for detecting write–write conflicts from $O(n)$ to $O(1)$ per allocated memory location, and replaces the $O(n)$-time VC comparison "$\sqsubseteq$" with the $O(1)$-time comparison "$\preceq$".

Detecting Write–Read Races: Detecting write–read races under the new representation is also straightforward. On each read from x with current VC C_t, we check that the read happens after the last write via the same $O(1)$-time comparison $W_x \preceq C_t$.

Detecting Read–Write Races: Detecting read–write race conditions is somewhat more difficult. Unlike write operations, which are totally ordered in race-free programs, reads are not necessarily totally ordered. Thus, a write to a variable x could potentially conflict with the last read of x performed by *any* other thread, not just the last read in the entire trace seen so far. Thus, we may need to record an entire VC R_x, in which $R_x(t)$ records the clock of the last read from x by thread t.

We can avoid keeping a complete VC in many cases, however. Our examination of data access patterns across a variety of multithreaded Java programs indicates that read operations are often totally ordered *in practice*, particularly in the following common situations:

- Thread-local data, where only one thread accesses a variable, and hence these accesses are totally ordered by program-order
- Lock-protected data, where a protecting lock is held on each access to a variable, and hence all access are totally ordered, either by program order (for accesses by the same thread) or by synchronization order (for accesses by different threads)

Reads are typically unordered only when data is *read-shared*, that is, when the data is first initialized by one thread and then shared between multiple threads in a read-only manner.

FastTrack uses an adaptive representation for the read history of each variable that is optimized for the common case of totally-ordered reads, while still retaining the full precision of VCs when necessary.

In particular, if the last read to a variable happens after all preceding reads, then FastTrack records only the epoch of this last read, which is sufficient to precisely detect whether a subsequent write to that variable conflicts with *any* preceding read in the entire program history. Thus, for thread-local and lock-protected data (which do exhibit totally-ordered reads), FastTrack requires only $O(1)$ space for each allocated memory location and only $O(1)$ time per memory access.

In the less common case where reads are not totally ordered, FastTrack stores the entire VC, but still handles read operations in $O(1)$ time. Since such data is typically read-shared, writes to such variables are rare, and their analysis overhead is negligible.

3.1. Analysis details

Based on the above intuition, we now describe the FastTrack algorithm in detail. Our analysis is an online algorithm whose analysis state consists of four components:

- C_t is the current VC of thread t.
- L_m is the VC of the last release of lock m.
- R_x is either the epoch of the last read from x, if all other reads happened-before that read, or else is a VC that records the last read from x by multiple threads.
- W_x is the epoch of the last write to x.

The analysis starts with $C_t = inc_t(\bot_V)$, since the first operations of all threads are not ordered by happens-before. In addition, initially $L_m = \bot_V$ and $R_x = W_x = \bot_e$.

Figure 3 presents the key details of how FastTrack (left column) and Djit$^+$ (right column) handle read and write operations of the target program. For each read or write operation, the relevant rules are applied in the order presented until one matches the current instrumentation state. If an assertion fails, a race condition exists. The figure shows the instruction frequencies observed in the programs described in Section 4, as well as how frequently each rule was applied. For example, 82.3% of all memory and synchronization operations performed by our benchmarks were reads, and rule [FT read same epoch] was used to check 63.4% of those reads. Expensive $O(n)$-time operations are highlighted in grey.

Read Operations: The first four rules provide various alternatives for analyzing a read operation $rd(t, x)$. The first rule

Figure 3. FastTrack race detection algorithm and its comparison to DJIT$^+$.

FastTrack **State:**

```
$C_t$ : $VC$
$L_m$ : $VC$
$W_x$ : $Epoch$
$R_x$ : $Epoch \cup VC$
```

DJIT$^+$ **State:**

```
$\mathbb{C}_t$ : $VC$
$\mathbb{L}_m$ : $VC$
$\mathbb{W}_x$ : $VC$
$\mathbb{R}_x$ : $VC$
```

When Thread t **performs** $rd(t, x)$: 82.3% of all Operations

[FT READ SAME EPOCH] 63.4% OF READS

```
if $R_x = E_t$ then
  skip
endif
```

[DJIT$^+$ READ SAME EPOCH] 78.0% OF READS

```
if $\mathbb{R}_x(t) = \mathbb{C}_t(t)$ then
  skip
endif
```

[FT READ SHARED] 20.8% OF READS

```
if $R_x \in VC$ then
  assert $W_x \preceq C_t$
  $R_x(t) := C_t(t)$
endif
```

[FT READ EXCLUSIVE] 15.7% OF READS

```
if $R_x \in Epoch$ and $R_x \preceq C_t$ then
  assert $W_x \preceq C_t$
  $R_x := E_t$
endif
```

[DJIT$^+$ READ] 22.0% OF READS

```
if $\mathbb{R}_x(t) \neq \mathbb{C}_t(t)$ then
  assert $\mathbb{W}_x \sqsubseteq \mathbb{C}_t$
  $\mathbb{R}_x(t) := \mathbb{C}_t(t)$
endif
```

[FT READ SHARE] 0.1% OF READS

```
if $R_x \in Epoch$ then
  let $c@u = R_x$
  assert $W_x \preceq C_t$
  $R_x := \bot_V[t \mapsto C_t(t), u \mapsto c]$
endif
```

When Thread t **performs** $wr(t, x)$: 14.5% of all Operations

[FT WRITE SAME EPOCH] 71.0% OF WRITES

```
if $W_x = E_t$ then
  skip
endif
```

[DJIT$^+$ WRITE SAME EPOCH] 71.0% OF WRITES

```
if $\mathbb{W}_x(t) = \mathbb{C}_t(t)$ then
  skip
endif
```

[FT WRITE EXCLUSIVE] 28.9% OF WRITES

```
if $R_x \in Epoch$ then
  assert $R_x \preceq C_t$
  assert $W_x \preceq C_t$
  $W_x := E_t$
endif
```

[FT WRITE SHARED] 0.1% OF WRITES

```
if $R_x \in VC$ then
  assert $R_x \sqsubseteq C_t$
  assert $W_x \preceq C_t$
  $W_x := E_t$
  $R_x := \bot_e$
endif
```

[DJIT$^+$ WRITE] 29.0% OF WRITES

```
if $\mathbb{W}_x(t) \neq \mathbb{C}_t(t)$ then
  assert $\mathbb{W}_x \sqsubseteq \mathbb{C}_t$
  assert $\mathbb{R}_x \sqsubseteq \mathbb{C}_t$
  $\mathbb{W}_x(t) := \mathbb{C}_t(t)$
endif
```

[FT READ SAME EPOCH] optimizes the case where x was already read in this epoch. This fast path requires only a single epoch comparison and handles over 60% of all reads. We use E_t to denote the current epoch $c@t$ of thread t, where $c = C_t(t)$ is t's current clock. DJIT$^+$ incorporates a comparable rule [DJIT$^+$ READ SAME EPOCH].

The remaining three read rules all check for write–read conflicts via the fast epoch-VC comparison $W_x \preceq C_t$, and then update R_x appropriately. If R_x is already a VC, then [FT READ SHARED] simply updates the appropriate component of that vector. Note that multiple reads of read-shared data from the same epoch are all covered by this rule. We could extend rule [FT READ SAME EPOCH] to handle same-epoch reads of read-shared data by matching the case that $R_x \in VC$ and $R_x(t) = C_t(t)$. The extended rule would cover 78% of all reads (the same as [DJIT$^+$ READ SAME EPOCH]) but does not improve performance perceptibly.

If the current read happens after the previous read epoch (where that previous read may be either by the same thread or by a different thread, presumably with interleaved synchronization), [FT READ EXCLUSIVE] simply updates R_x with the accessing thread's current epoch. For the more general situation where the current read is concurrent with the previous read, [FT READ SHARE] allocates a VC to record the epochs of *both* reads, since either read could subsequently participate in a read–write race.

Of these three rules, the last rule is the most expensive but is rarely needed (0.1% of reads) and the first three rules provide commonly-executed, constant-time fast paths. In contrast, the corresponding rule [DJIT$^+$ READ] *always* executes an $O(n)$-time VC comparison for these cases.

Write Operations: The next three FASTTRACK rules handle a write operation $wr(t, x)$. Rule [FT WRITE SAME EPOCH] optimizes the case where x was already written in this epoch, which applies to 71.0% of write operations, and DJIT$^+$ incorporates a comparable rule. [FT WRITE EXCLUSIVE] provides a fast path for the 28.9% of writes for which R_x is an epoch, and this rule checks that the write happens after all previous accesses. In the case where R_x is a VC, [FT WRITE SHARED] requires a full (slow) VC comparison, but this rule applies only to a tiny fraction (0.1%) of writes. In contrast, the corresponding DJIT$^+$ rule [DJIT$^+$ WRITE] requires a VC comparison on 29.0% of writes.

Other Operations: Figure 4 shows how FASTTRACK handles synchronization operations. These operations are rare, and the traditional analysis for these operations in terms of expensive VC operations is perfectly adequate. Thus, these FASTTRACK rules are similar to those of DJIT$^+$ and other VC-based analyses.

Example: The execution trace in Figure 5 illustrates how FASTTRACK dynamically adapts the representation for the read history R_x of a variable x. Initially, R_x is $\perp_e$, indicating that x has not yet been read. After the first read operation $rd(1, x)$, R_x becomes the epoch 1@1 recording both the clock and the thread identifier of that read. The second read $rd(0, x)$ at clock 8 is concurrent with the first read, and so FASTTRACK switches to the VC representation ⟨8, 1, ...⟩ for R_x, recording the clocks of the last reads from x by both threads 0 and 1. After the two threads join, the write operation $wr(0, x)$ happens after all reads. Hence, any later operation cannot be in a race with either read without also being in a race on that write operation, and so the rule [FT WRITE SHARED] discards the read history of x by resetting R_x to $\perp_e$, which also switches x back into epoch mode and so

Figure 4. Synchronization, threading operations.

Other: ▮ 3.3% of all Operations

When Thread t performs $acq(t, m)$:

$C_t := C_t \sqcup L_m$

When Thread t performs $rel(t, m)$:

$L_m := C_t$

$C_t := inc_t(C_t)$

When Thread t performs $fork(t, u)$:

$C_u := C_u \sqcup C_t$

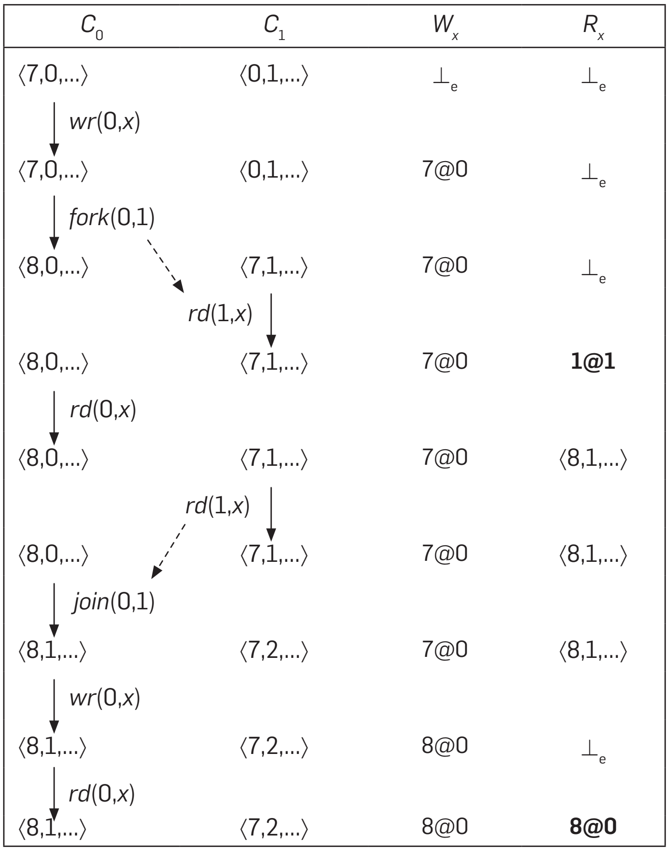
$C_t := inc_t(C_t)$

When Thread t performs $join(t, u)$:

$C_t := C_t \sqcup C_u$

$C_u := inc_u(C_u)$

Figure 5. Adaptive read history representation.



C_0	C_1	W_x	R_x
⟨7,0,...⟩	⟨0,1,...⟩	$\perp_e$	$\perp_e$
$wr(0,x)$			
⟨7,0,...⟩	⟨0,1,...⟩	7@0	$\perp_e$
$fork(0,1)$			
⟨8,0,...⟩	⟨7,1,...⟩	7@0	$\perp_e$
	$rd(1,x)$		
⟨8,0,...⟩	⟨7,1,...⟩	7@0	**1@1**
$rd(0,x)$			
⟨8,0,...⟩	⟨7,1,...⟩	7@0	⟨8,1,...⟩
	$rd(1,x)$		
⟨8,0,...⟩	⟨7,1,...⟩	7@0	⟨8,1,...⟩
$join(0,1)$			
⟨8,1,...⟩	⟨7,2,...⟩	7@0	⟨8,1,...⟩
$wr(0,x)$			
⟨8,1,...⟩	⟨7,2,...⟩	8@0	$\perp_e$
$rd(0,x)$			
⟨8,1,...⟩	⟨7,2,...⟩	8@0	**8@0**

optimizes later accesses to x. The last read in the trace then sets R_x to a nonminimal epoch.

4. EVALUATION

To validate FastTrack, we implemented it as a component of the RoadRunner dynamic analysis framework for multithreaded Java programs.[10] RoadRunner takes as input a compiled Java target program and inserts instrumentation code into the target to generate an event stream of memory and synchronization operations. Back-end checking tools process these events as the target executes. The FastTrack implementation extends the algorithm described so far to handle additional Java primitives, such as `volatile` variables and `wait()`, as outlined previously.[8] Some of the benchmarks contain faulty implementations of barrier synchronization.[9] FastTrack contains a specialized analysis to compensate for these bugs.

We compare FastTrack's precision and performance to six other analyses implemented in the same framework:

- Empty, a trivial checker that performs no analysis and is used to measure the overhead of RoadRunner
- Eraser,[18] an imprecise race detector based on the LockSet algorithm described in Section 1
- Goldilocks, a precise race detector based on an extended notion of LockSets[7]
- BasicVC, a traditional VC-based race detector that maintains a read and a write VC for each memory location and performs at least one VC comparison on every memory access
- Djit$^+$, a high-performance VC-based race detector[16] described in Section 2
- MultiRace, a hybrid LockSet/Djit$^+$ race detector[16]

4.1. Performance and precision

Table 1 lists the size, number of threads, and uninstrumented running times for a variety of benchmark programs drawn from the Java Grande Forum,[12] Standard Performance Evaluation Corporation,[19] and elsewhere.[2,7,11,21] All timing measurements are the average of 10 test runs. Variability across runs was typically less than 10%.

The "Instrumented Time" columns show the running times of each program under each of the tools, reported as the ratio to the uninstrumented running time. Thus, target programs ran 4.1 times slower, on average, under the Empty tool. Most of this overhead is due to communicating all target program operations to the back-end checker.

The variations in slowdowns for different programs are not uncommon for dynamic race condition checkers. Different programs exhibit different memory access and synchronization patterns, some of which impact analysis performance more than others. In addition, instrumentation can impact cache performance, class loading time, and other low-level JVM operations. These differences can sometimes even make an instrumented program run slightly faster than the uninstrumented (as in colt).

The last six columns show the number of warnings produced by each checker. The tools report at most one race for each field of each class, and at most one race for each array access in the program source code. All eight warnings from FastTrack reflect real race conditions. Some of these are benign (as in `tsp`, `mtrt`, and `jbb`) but others can impact program behavior (as in `raytracer` and `hedc`).[15, 20, 21]

Table 1. Benchmark results. Programs marked with '*' are not compute-bound and are excluded from average slowdowns.

				Instrumented Time (slowdown)							Warnings					
Program	Size (loc)	Thread Count	Base Time (s)	Empty	Eraser	MultiRace	Goldilocks	BasicVC	Djit$^+$	FastTrack	Eraser	MultiRace	Goldilocks	BasicVC	Djit$^+$	FastTrack
colt	111,421	11	16.1	0.9	0.9	0.9	1.8	0.9	0.9	0.9	3	0	0	0	0	0
crypt	1,241	7	0.2	7.6	14.7	54.8	77.4	84.4	54.0	14.3	0	0	0	0	0	0
lufact	1,627	4	4.5	2.6	8.1	42.5	–	95.1	36.3	13.5	4	0	–	0	0	0
moldyn	1,402	4	8.5	5.6	9.1	45.0	17.5	111.7	39.6	10.6	0	0	0	0	0	0
montecarlo	3,669	4	5.0	4.2	8.5	32.8	6.3	49.4	30.5	6.4	0	0	0	0	0	0
mtrt	11,317	5	0.5	5.7	6.5	7.1	6.7	8.3	7.1	6.0	1	1	1	1	1	1
raja	12,028	2	0.7	2.8	3.0	3.2	2.7	3.5	3.4	2.8	0	0	0	0	0	0
raytracer	1,970	4	6.8	4.6	6.7	17.9	32.8	250.2	18.1	13.1	1	1	1	1	1	1
sparse	868	4	8.5	5.4	11.3	29.8	64.1	57.5	27.8	14.8	0	0	0	0	0	0
series	967	4	175.1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1	0	0	0	0	0
sor	1,005	4	0.2	4.4	9.1	16.9	63.2	24.6	15.8	9.3	3	0	0	0	0	0
tsp	706	5	0.4	4.4	24.9	8.5	74.2	390.7	8.2	8.9	9	1	1	1	1	1
elevator*	1,447	5	5.0	1.1	1.1	1.1	1.1	1.1	1.1	1.1	0	0	0	0	0	0
philo*	86	6	7.4	1.1	1.0	1.1	7.2	1.1	1.1	1.1	0	0	0	0	0	0
hedc*	24,937	6	5.9	1.1	0.9	1.1	1.1	1.1	1.1	1.1	2	1	0	3	3	3
jbb*	30,491	5	72.9	1.3	1.5	1.6	2.1	1.6	1.6	1.4	3	1	–	2	2	2
Average slowdown/total warnings				4.1	8.6	21.7	31.6	89.8	20.2	8.5	27	5	3	8	8	8

Eraser Comparison: Our reimplementation of Eraser incurs an overhead of 8.7x, which is competitive with similar Eraser implementations built on top of unmodified JVMs.[15] Surprisingly, FastTrack is slightly faster than Eraser on some programs, even though it performs a precise analysis that traditionally has been considered more expensive.

More significantly, Eraser reported many spurious warnings that do not correspond to actual races. Augmenting our Eraser implementation to reason about additional synchronization constructs, such as `fork/join` or `wait/notify` operations,[16, 22] would eliminate some of these spurious warnings, but not all. On `hedc`, Eraser reported a spurious warning but also missed two of the real race conditions reported by FastTrack, due to an (intentional) unsoundness in how the Eraser algorithm reasons about thread-local and read-shared data.[18]

BasicVC and Djit$^+$ Comparison: Djit$^+$ and BasicVC reported exactly the same race conditions as FastTrack. That is, all three checkers provide identical precision. However, FastTrack outperforms the other checkers. It is roughly 10x faster than BasicVC and 2.3x faster than Djit$^+$. These performance improvements are due primarily to the reduction in the allocation and use of VCs. Across all benchmarks, Djit$^+$ allocated more over 790 million VCs, whereas FastTrack allocated only 5.1 million. Djit$^+$ performed over 5.1 billion $O(n)$-time VC operations, while FastTrack performed only 17 million. The memory overhead for storing the extra VCs leads to significant cache performance degradation in some programs, particularly those that randomly access large arrays. These tools are likely to incur even greater overhead when checking programs with larger numbers of threads.

MultiRace Comparison: MultiRace maintains Djit$^+$'s instrumentation state, as well as a lock set for each memory location.[16] The checker updates the lock set for a location on the first access in an epoch, and full VC comparisons are performed only after this lock set becomes empty. This synthesis substantially reduces the number of VC operations, but introduces the overhead of storing and updating lock sets. In addition, the use of Eraser's unsound state machine for thread-local and read-shared data leads to imprecision.

Our reimplementation of the MultiRace algorithm exhibited performance comparable to Djit$^+$. Performance of MultiRace (and, in fact, all of our other checkers) can be improved by adopting a *coarse-grain* analysis in which all fields of an object are represented as a single "logical location" in the instrumentation state.[16, 22]

Goldilocks Comparison: Goldilocks[7] is a precise race detector that does not use VCs to capture the happens-before relation. Instead, it maintains, for each memory location, a set of "synchronization devices" and threads. A thread in that set can safely access the memory location, and a thread can add itself to the set (and possibly remove others) by performing any of the operations described by the synchronization devices in the set.

Goldilocks is a complicated algorithm to optimize, and ideally requires tight integration with the underlying virtual machine and garbage collector, which is not possible under RoadRunner. Because of these difficulties, Goldilocks reimplemented in RoadRunner incurred a slowdown of 31.6x across our benchmarks, but ran out of memory on lufact. Our Goldilocks reimplementation missed three races in `hedc`, due to an unsound performance optimization for handling thread-local data efficiently.[7] We believe some performance improvements are possible, for both Goldilocks and the other tools, by integration into the virtual machine.

4.2. Checking eclipse for race conditions

To validate FastTrack in a more realistic setting, we also applied it to five common operations in the Eclipse development environment.[6] These include launching Eclipse, importing a project, rebuilding small and large workspaces, and starting the debugger. The checking overhead for these operations is as follows:

	Base	Instrumented Time (Slowdown)			
Operation	Time (s)	Empty	Eraser	Djit$^+$	FastTrack
Startup	6.0	13.0	16.0	17.3	16.0
Import	2.5	7.6	14.9	17.1	13.1
Clean Small	2.7	14.1	16.7	24.4	15.2
Clean Large	6.5	17.1	17.9	38.5	15.4
Debug	1.1	1.6	1.7	1.7	1.6

Eraser reported potential races on 960 distinct field and array accesses for these five tests, largely because Eclipse uses many synchronization idioms that Eraser cannot handle, such as `wait()` and `notify()`, semaphores, and readers-writer locks. FastTrack reported 27 distinct warnings, 4 of which were subsequently verified to be potentially destructive.[9] Djit$^+$ reported 28 warnings, which overlapped heavily with those reported by FastTrack, but scheduling differences led to several being missed and several new (benign) races being identified. Although our exploration of Eclipse is far from complete, these preliminary observations are quite promising. FastTrack is able to scale to precisely check large applications with lower run-time and memory overheads than existing tools.

5. CONCLUSION

Race conditions are difficult to find and fix. Precise race detectors avoid the programmer-overhead of identifying and eliminating spurious warnings, which are particularly problematic when using imprecise checkers on large programs with complex synchronization. Our FastTrack analysis is a new precise race detection algorithm that achieves better performance than existing algorithms by tracking less information and dynamically adapting its representation of the happens-before relation based on memory access patterns. We have used FastTrack to identify data races in programs as large as the Eclipse programming environment, and also to improve the performance of other analyses that rely on precise data race information, such as serializability checkers.[8] The FastTrack algorithm and adaptive epoch representation is straightforward to implement and may be useful in other dynamic analyses for multithreaded software.

Acknowledgments

This work was supported in part by NSF Grants 0341179, 0341387, 0644130, and 0707885. We thank Ben Wood for implementing Goldilocks in RoadRunner and for comments on a draft of this paper, and Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran for their assistance with Goldilocks.

References

1. Adve, S.V., Hill, M.D., Miller, B.P., Netzer, R.H.B. Detecting data races on weak memory systems. In *ISCA* (1991), 234–243.
2. CERN. Colt 1.2.0. Available at http://dsd.lbl.gov/~hoschek/colt/ (2007).
3. Choi, J.-D., Lee, K., Loginov, A., O'Callahan, R., Sarkar, V., Sridhara, M. Efficient and precise datarace detection for multithreaded object-oriented programs. In *PLDI* (2002), 258–269.
4. Choi, J.-D., Miller, B.P., Netzer R.H.B. Techniques for debugging parallel programs with flowback analysis. *TOPLAS 13*, 4 (1991), 491–530.
5. Christiaens, M., Bosschere, K.D. TRaDe: Data race detection for Java. In *International Conference on Computational Science* (2001), 761–770.
6. The Eclipse programming environment, version 3.4.0. Available at http://www.eclipse.org, 2009.
7. Elmas, T., Qadeer, S., Tasiran, S. Goldilocks: A race and transaction-aware Java runtime. In *PLDI* (2007), 245–255.
8. Flanagan, C., Freund, S.N. FastTrack: Efficient and precise dynamic race detection. In *PLDI* (2009), 121–133.
9. Flanagan, C., Freund, S.N. Adversarial memory for detecting destructive races. In *PLDI* (2010), 244–254.
10. Flanagan, C., Freund, S.N. The RoadRunner dynamic analysis framework for concurrent programs. In *PASTE* (2010), 1–8.
11. Fleury, E., Sutre, G. Raja, version 0.4.0-pre4. Available at http://raja.sourceforge.net/, 2007.
12. Java Grande Forum. Java Grande benchmark suite. Available at http://www.javagrande.org/, 2008.
13. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (1978), 558–565.
14. Mattern, F. Virtual time and global states of distributed systems. In *Workshop on Parallel and Distributed Algorithms*, 1988.
15. O'Callahan, R., Choi J.-D. Hybrid dynamic data race detection. In *PPOPP* (2003), 167–178.
16. Pozniansky, E., Schuster, A. MultiRace: Efficient on-the-fly data race detection in multithreaded C++ programs. *Concurrency and Computation: Practice and Experience 19*, 3 (2007), 327–340.
17. Ronsse, M., Bosschere, K.D. RecPlay: A fully integrated practical record/replay system. *TCS 17*, 2 (1999), 133–152.
18. Savage, S., Burrows, M., Nelson, G., Sobalvarro, P., Anderson, T.E. Eraser: A dynamic data race detector for multi-threaded programs. *TOCS 15*, 4 (1997), 391–411.
19. Standard Performance Evaluation Corporation. SPEC benchmarks. http://www.spec.org/, 2003.
20. von Praun, C., Gross, T. Object race detection. In *OOPSLA*, 2001, 70–82.
21. von Praun, C., Gross, T. Static conflict analysis for multi-threaded object-oriented programs. In *PLDI* (2003), 115–128.
22. Yu, Y., Rodeheffer, T., Chen, W. RaceTrack: Efficient detection of data race conditions via adaptive tracking. In *SOSP* (2005), 221–234.

Cormac Flanagan, Computer Science Department, University of California at Santa Cruz, Santa Cruz, CA.

Stephen N. Freund, Computer Science Department, Williams College, Williamstown, MA.

Group Term Life Insurance

10- or 20-Year Group Term Life Insurance

Group Disability Income Insurance

Group Accidental Death & Dismemberment Insurance

Group Catastrophic Major Medical Insurance

Group Dental Plan

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh U.S. Consumer, a service of Seabury & Smith, Inc., at **1-800-503-9230** or visit **www.personal-plans.com/promo/acm/49771.**

Administered by:

MARSH

MMC MARSH MERCER KROLL
GUY CARPENTER OLIVER WYMAN

d/b/a in CA Seabury & Smith Insurance Program Management
CA Ins. Lic. #0633005
AR Ins. Lic. #245544

Appalachian State University
Tenure Track Assistant Professor

The Department of Computer Science at Appalachian State University invites applications for a tenure-track faculty position at the rank of assistant professor beginning August 2011. Qualifications for this position include a Ph.D. degree in Computer Science or a closely related field. Responsibilities include teaching undergraduate and graduate computer science courses, an active program of scholarship, pursuit of external funding, and participation in service activities.

The Department of Computer Science at Appalachian State University has 10 tenured faculty members. Approximately 25 students are enrolled in the Master of Science program. The Bachelor of Science in Computer Science program is accredited by the Computing Accreditation Commission of ABET and serves about 220 students.

Appalachian State University is a member institution of the seventeen-campus University of North Carolina. Located in Boone, North Carolina, the university has approximately 17,000 students, primarily in bachelors and masters programs in both liberal arts and applied fields. Additional information about the Department of Computer Science, the university, and the surrounding area can be found at www.appstate.edu.

Applicants must send: 1) statements on teaching, research, and interest/qualifications for this position, 2) a copy of their graduate transcript, 3) a curriculum vita, and 4) names and contact information for three references. All application material should be sent by e-mail to search@cs.appstate.edu in a single PDF file attachment or by mail to the chair of the search committee: Dr. Rahman Tashakkori, Department of Computer Science, 525 Rivers Street, Boone, NC 28608. The initial review of complete applications will begin on December 2, 2010 and will continue until the position is filled.

Appalachian State University is an Affirmative Action/Equal Opportunity Employer. The university has a strong commitment to the principles of diversity and inclusion, and to maintaining working and learning environments that are free of all forms of discrimination.

Individuals with disabilities may request accommodations in the application process by contacting the chair of the search committee. Criminal background checks will be conducted on all finalists invited for on-campus interviews.

College of the Holy Cross
Assistant Professor, Tenure Track

The Department of Mathematics and Computer Science at the College of the Holy Cross invites applications for a full-time tenure-track appointment to begin in August 2011. All research specialties will be considered and breadth of interests within computer science will be viewed favorably. Candidates must commit to, and excel in, undergraduate teaching as well as scholarly achievement. A Ph.D. in computer science or a closely related field is required.

This position carries a 3-2 teaching load with a full-salary one-semester research leave prior to tenure review and generous sabbatical and fellowship leaves for senior faculty. Please send a cover letter describing research and teaching interests, curriculum vitae, statement on teaching, undergraduate and graduate transcripts, and three letters of recommendation to:

Search Committee, reference #001
Department of Mathematics and Computer Science
College of the Holy Cross
Worcester, MA 01610
email: search@mathcs.holycross.edu

Review of applications will begin on December 15, 2010 and continue until the position has been filled.

The College of the Holy Cross is a highly selective Catholic liberal arts college in the Jesuit tradition. It enrolls about 2,700 students and is located in a medium-sized city 45 miles west of Boston. Holy Cross belongs to the Colleges of Worcester Consortium (www.cowc.org) and the New England Higher Education Recruitment Consortium (www.neherc.org). The College is an Equal Employment Opportunity Employer and complies with all Federal and Massachusetts laws concerning equal opportunity and affirmative action in the workplace.

Additional information on the college, the department, and this position available at:
http://mathcs.holycross.edu/positionCS/CSPos11.html

Dartmouth College
Neukom Institute for Computational Science
Department of Computer Science
Assistant Professor

The Neukom Institute for Computational Science and the Department of Computer Science at Dartmouth College invite applications for a tenure-track faculty position at the level of **Assistant Professor** in the Department of Computer Science. We seek candidates in the area of computational biology and bioinformatics whose research focuses on the development and application of new computational methods. Candidates will complement a growing program in computational biology within the Departments of Biology, Computer Science, Engineering Sciences, and Mathematics, as well as the Dartmouth Medical School.

The Neukom Institute for Computational Science (www.dartmouth.edu/~neukom) is an endowed institute whose broad mandate is to inspire and support computational science across the Dartmouth campus. The Institute has considerable financial and computing resources that will be available to the successful candidate. The Department of Computer Science (www.cs.dartmouth.edu) is home to 17 tenured and tenure-track faculty whose research spans computational biology, vision/graphics, machine learning, algorithms, theory, and systems. The department has strong Ph.D. and M.S. programs, outstanding undergraduate majors and minors, and is affiliated with an M.D./Ph.D. program.

Dartmouth is an Ivy League school situated in Hanover, on the Connecticut River, in the Upper Valley region of New Hampshire. It is a beautiful, historic campus, located in a scenic, year-round, outdoor recreational area. Dartmouth hosts an annual film festival; renowned musical and theatrical performers; and convenient public transportation to Boston and New York, as well as local airports.

Applicants are invited to send their CV, research statement, teaching statement, and names of at least four references, one of whom should comment about teaching. All material should be sent to search@cs.dartmouth.edu by December 1st, 2010. All letters of recommendation should be emailed or mailed to search, 6211 Sudikoff Lab, Computer Science Department, Dartmouth College, Hanover, NH 03755 by the recommender themselves.

Direct inquiries may be sent to Professor Hany Farid (farid@cs.dartmouth.edu).

Dartmouth is an equal opportunity/affirmative action employer and encourages applications from women and members of minority groups.

Furman University
Assistant Professor of Computer Science

The Department of Computer Science invites applications for a tenure track position at the Assistant Professor level to begin in the fall of 2011. Candidates must have a Ph.D. in Computer Science or a closely related field. The position requires teaching excellence, effective institutional service, and an ability to work with colleagues across disciplines. An ability to develop a program of scholarly and professional activity involving undergraduates is a priority. Research specialty areas being sought include (but are not limited to) high performance computing, computational science, mathematical modeling, and bioinformatics. Of particular interest are candidates willing to engage in collaborative research that bridges the computational and medical sciences. The position will be initially funded by and is expected to contribute to a major multi-disciplinary and multi-organizational state-wide initiative aimed at biofabrication of tissues and organs.

Furman is a highly selective, independent, top 40 undergraduate liberal arts institution with an enrollment of approximately 2600 students. The university is located in the vibrant and beautiful upstate region of South Carolina, offers generous benefits to fulltime faculty, and subscribes to a problem-solving, project-oriented, experience-based approach to education that is referred to as Engaged Learning. The Department of Computer Science confers the B.S. degree with majors in Computer Science, Information Technology, and Computer Science/Mathematics. The successful candidate will have the opportunity to teach in Furman's First Year Seminar program. Furman University is an equal-opportunity employer. Women and underrepresented minorities are strongly encouraged to apply. For the complete ad, please visit http://cs.furman.edu.

Applicants should submit a curriculum vitae, statement of teaching philosophy, description of research interests, an official copy of most recent transcripts, and have three letters of recommendation sent separately. Please send all materials to Dr. Kevin Treu, Chair, Department of Computer Science, Furman University, 3300 Poinsett Hwy, Greenville, SC 29613. Materials may also be sent in PDF format to kevin.treu@furman.edu. Review of applications will continue until the position is filled.

Harvard University SEAS

Senior Lectureship in Computer Science

The School of Engineering and Applied Sciences (SEAS) at Harvard University expects to appoint a Senior Lecturer in Computer Science, either full-time or part-time, starting in the Fall of 2011. This position will be for five years and may be renewed.

We are particularly interested in candidates who have demonstrated their excellence in and commitment to introductory undergraduate Computer Science instruction. We are seeking applicants whose interests and experience focus on teaching, pedagogy, and classroom and laboratory innovations. While we are considering an internal candidate for this position, we encourage applications from all well qualified individuals, especially from women and minority candidates.

The application deadline is December 1, 2010. A Ph.D. and significant postdoctoral teaching experience are required. Applications including a CV, a statement describing teaching experience and philosophy, and at least three letters of recommendation should be sent to: Computer Science Lectureship Committee, c/o Tristen Hubbard, School of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138. Email: tristen@eecs.harvard.edu.

Harvard is an Equal Opportunity/Affirmative Action employer and encourages applications from women and members of minority groups.

Harvard University SEAS

Tenure-Track Professor in Computer Science

Over the past several years, Harvard's Computer Science faculty has doubled in size, moved into a state-of-the-art teaching and research facility, and made a serious commitment to fostering collaboration with other academic disciplines. The Computer Science program benefits from its outstanding undergraduate and graduate students, an excellent location, significant industrial collaboration, and substantial support from the Harvard School of Engineering and Applied Sciences.

We invite applications for a tenure-track professor within Computer Science, and strongly encourage applications from qualified women and minority candidates. The appointment is expected to begin on July 1, 2011.

We seek outstanding applicants in areas related to machine learning and artificial intelligence. We are particularly interested in applicants whose research examines computational issues raised by very large data sets, broadly construed. Specific areas of interest include but are not limited to statistical machine learning, probabilistic modeling, reinforcement learning and massively parallel processing. Potential application areas of interest include computational science and engineering and the social sciences.

Candidates should have an outstanding research record and a strong commitment to undergraduate teaching and graduate training. Applicants must have completed a Ph.D. by September 1, 2011. Information about Harvard's current faculty, research, and educational programs is available at http://www.seas.harvard.edu/teaching-learning/areas/computer-science.

Candidates should send a curriculum vitae, a list of publications, a statement of research and teaching interests, and up to three representative papers (ideally as a single PDF document) to cs-search@seas.harvard.edu. In addition, candidates should have at least three letters of reference sent to the above address.

Alternatively, material may also be sent via surface mail to CS Search Committee, School of Engineering and Applied Sciences, Harvard University, Maxwell Dworkin 153, 33 Oxford Street, Cambridge, MA 02138.

Applications will be reviewed as they are received. For full consideration, applications should be received by December 1, 2010.

Harvard is an Equal Opportunity/ Affirmative Action Employer. Applications from women and minorities are strongly encouraged.

UNIVERSITY OF DELAWARE

One of the oldest institutions of higher education in this country, the University of Delaware today combines tradition and innovation, offering students a rich heritage along with the latest in instructional and research technology. The University of Delaware is a Land-Grant, Sea-Grant and Space-Grant institution with its main campus in Newark, DE, located halfway between Washington, DC and New York City. Please visit our website at **www.udel.edu**.

Faculty Positions in Electrical and Computer Engineering

The Department of Electrical and Computer Engineering (ECE) at the University of Delaware (UD) invites nominations and applications for tenure-track faculty positions at the Assistant, Associate, and Full Professor ranks.

The ECE Department invites candidates that complement the department's traditional strengths in (1) Computer Engineering, (3) Signal Processing, Communications & Controls, and (3) Nanoelectronics, Electromagnetics, and Photonics. A particular emphasis is placed on candidates aligned with UD's College of Engineering wide cluster searches in: (1) Energy, (2) Information Technologies, (3) Security, and (4) Biomedical Engineering. Exceptional cases outside of these traditional strengths and focused areas may also be considered.

Successful applicants will share our vision to grow the department into a leader in research and educational programs. ECE initiatives are supported by over 40,000 square feet of departmental facilities, including a state-of-the art 7,000 sq ft clean room for nano-fabrication, and fueled with over $10M/year in research expenditures. Applicants should hold a Ph.D. in electrical and/or computer engineering, or closely related field in mathematics, biomedical, computer or physical sciences. Successful candidates are expected to have demonstrated excellence in innovative research and show the potential for high-quality teaching and mentoring. The University of Delaware offers very competitive salary and start-up packages, and has a generous benefit package. The application reviews start November 1st. The search will continue until the positions are filled; however, early application is strongly encouraged.

Applicants should submit a curriculum vitae, a statement of research and teaching interests, and a list of at least four references to www.engr.udel.edu/facultysearch. Questions can be directed to f-search@udel.edu or ECE Faculty Search Committee, 140 Evans Hall, University of Delaware, Newark, DE 19716. Applications will be considered until the position is filled.

The UNIVERSITY OF DELAWARE is an Equal Opportunity Employer which encourages applications from Minority Group Members and Women.

Hologic, Inc.

Software Engineering

Hologic, Inc. is a leading developer, manufacturer and supplier of premium diagnostics, medical imaging systems and surgical products dedicated to serving the healthcare needs of women. Due to continued growth & new products, Hologic has multiple openings in our Danbury, CT and Newark, DE facilities in our Software Engineering department. Medical device experience preferred. Apply to website; www.hologic.com/careers.

Hologic, Inc. is an Equal Opportunity Employer.

Danbury - 1 Sr. Embedded SW Eng. BA in Eng/Comp Sci and 5+ yrs experience. PWM motor drivers/PID controllers. Proficiency with assembly, C and C++ in embedded environment. Knowledge of Freescale, .NET and failsafe experience useful. (IRC #18231)

1 SW Eng. BA in EE/Comp Sci. Excellent C/C#/C++ & object oriented programming skills, DICOM, med imaging & image processing. Use dev tools and all elements of standard dev environment; source code control, compilers, config mgmt. & defect tracking. (IRC #19788)

Be part of this exciting opportunity to produce pioneering research in information systems that will usher in a new era of technological applications. Revolutionize information technology and designs to build a whole new way of life by developing the next generation of technological leaders.

The Singapore University of Technology and Design (SUTD), established in collaboration with the Massachusetts Institute of Technology (MIT), is seeking exceptional faculty members in the area of Information Systems Technology and Design for this new university slated to matriculate its first intake of students in April 2012.

SUTD, the first university in the world with a focus on design accomplished through an integrated mutli-disciplinary curriculum, has a mission to advance knowledge and nurture technically grounded leaders and innovators to serve societal needs. SUTD is charactertized by a breadth of intellectual perspectives (the "university"), a focus on engineering foundations ("technology") and an emphasis on innovation and creativity (design). The University's programmes are based on four pillars leading to separate degree programmes in Architecture and Sustainable Design, Engineering Product Development, Engineering Systems and Design, and Information Systems Technology and Design. Design, as an academic discipline, cuts across the curriculum and will be the framework for novel research and educational programmes.

MIT's multi-faceted collaboration with SUTD includes the development of new courses and curricula, assistance with the early deployment of courses in Singapore, assistance with faculty and student recruiting, mentoring, and career development, and collaborating on a major joint research projects, through a major new international design centre and student exchanges. Many of the newly hired SUTD faculty will spend up to year at MIT in a specially tailored programme for collaboration and professional development.

FACULTY MEMBERS (INFORMATION SYSTEMS TECHNOLOGY AND DESIGN)

The qualifications for the faculty position include: an earned doctorate in Computer Science, Computer Engineering or Information Systems, a strong commitment to teaching at the undergraduate and graduate levels, a demonstrated record of or potential for scholarly research, and excellent communication skills. We invite applications for faculty appointments at all levels, with many opportunities available in particular at the Assistant and Associate Professor levels. Duties include teaching of graduate and undergraduate students, research, supervision of student research, advising undergraduate student projects, and service to SUTD and the community. Faculty will be expected to develop and sustain a strong research programme. Attractive research grant opportunities are also available.

Successful candidates can look forward to internationally competitive remuneration, and assistance for relocation to Singapore.

If you want to be part of the founding faculty with a focus on Information Systems Technology and Design, please apply to SUTD at **www.sutd.edu.sg/careers.htm**

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN
Established in collaboration with MIT

Newark - 2 SW Eng (Embedded). BA in EE/Comp Sci, Masters preferred. 3 yrs software dev using C and C++ in an RTOS/embedded environment. Medical device/ medical imaging or driver dev experience preferred. (IRC #19750)

1 Sr. SW Eng. BA in Comp Eng/Eng. 5 years experience developing C/C#/C++ in Windows environment & object oriented programming skills. DICOM, med imaging & image processing. (IRC #19306)

The Hong Kong University of Science and Technology

Department of Computer Science and Engineering

Faculty Positions

The Department of Computer Science and Engineering is one of the largest departments in the School of Engineering. The Department currently has 40 faculty members recruited from major universities and research institutions around the world, with about 1000 students (including 600 undergraduate and 180 postgraduate students). The medium of instruction is English. More information on the Department can be found at http://www.cse.ust.hk/.

The Department will have at least two tenure-track faculty openings at Assistant Professor/Associate Professor/Professor levels for the 2011-2012 academic year. We are looking for faculty candidates with interests in multidisciplinary research areas related to computational science and engineering such as bioinformatics and financial engineering. Strong candidates in core computer science and engineering research areas will also be considered. Applicants at Assistant Professor level should have an earned PhD degree and demonstrated potential in teaching and research.

Salary is highly competitive and will be commensurate with qualifications and experience. Fringe benefits include medical/dental benefits and annual leave. Housing will also be provided where applicable. For appointment at Assistant Professor/Associate Professor level, initial appointment will normally be on a three-year contract. A gratuity will be payable upon completion of contract.

Applications should be sent through e-mail including a cover letter, curriculum vitae (including the names and contact information of at least three referees), a research statement and a teaching statement (all in PDF format) to csrecruit@cse.ust.hk. Priority will be given to applications received by 28 February 2011. Applicants will be promptly acknowledged through e-mail upon receiving the electronic application material.

(Information provided by applicants will be used for recruitment and other employment-related purposes.)

Indiana University

School of Informatics and Computing

The School of Informatics and Computing at Indiana University, Bloomington, invites applications for two positions beginning in Fall 2011.

Senior Faculty Position in Systems

This position is in systems, broadly encompassing parallel computing and architectures, distributed systems, cyberinfrastructure, and networking. Applicants are expected to have a well-established track record of substantial research contributions to their field, externally funded research, and leadership.

Faculty Position in Complex Networks and Systems

This position is at the junior level but outstanding senior candidates may be considered. Research areas include complex networks, computational biology and epidemiology, artificial life and robotics, computational intelligence, bio-inspired computing, large scale data modeling and simulation, and Web applications, with special interest in modeling the dynamics of complex information networks, social networks and media, and the spread of ideas and disease in human and social systems.

Applicants for either position should have a Ph.D.in Computer Science or another relevant area and a well-established record (senior level) or demonstrable potential for excellence in research and teaching (junior level).

The IU Bloomington School of Informatics and Computing is the first of its kind and among the largest in the country, with a faculty of more than 60 full time members, more than 400 graduate students, and strong undergraduate programs. Degrees offered include M.S. degrees in Computer Science, Bioinformatics, Human Computer Interaction Design, and Security Informatics, and Ph.D. degrees in Computer Science and in Informatics. The School has received public recognition as a "top-ten program to watch" (Computerworld) thanks to its excellence and leadership in academic programs, interdisciplinary research, placement, and outreach. The school offers excellent work conditions, including attractive salaries and research support, and low teaching loads in a setting of strong student growth.

Located in the wooded, rolling hills of southern Indiana, Bloomington is a culturally thriving college town with a moderate cost of living and the amenities for an active lifestyle. IU is renowned for its top-ranked music school, high performance computing and networking facilities, and performing and fine arts.

Applicants should submit a curriculum vitae, a statement of research and teaching, and the names of six references using the recruit link at http://hiring.soic.indiana.edu (preferred) or by mail to the Chair of either the Systems or Complex Networks and Systems Faculty Search Committee, School of Informatics and Computing, 919 E 10th Street, Bloomington, IN 47408. Questions concerning the Systems search may be sent to hiring-systems@informatics.indiana.edu; questions concerning the Complex Networks and Systems search to hiring-cnets@informatics.indiana.edu. To receive full consideration completed applications must be received by December 1, 2010.

Indiana University is an Equal Opportunity/Affirmative Action employer. Applications from women and minorities are strongly encouraged. IU Bloomington is vitally interested in the needs of Dual Career couples.

NANYANG TECHNOLOGICAL UNIVERSITY

NUS National University of Singapore

TEMASEK RESEARCH FELLOWSHIP (TRF

The Nanyang Technological University (NTU) and the National University of Singapore (NUS) invite outstanding young researchers with a PhD Degree in science or technology to apply for the prestigious TRF awards.

The TRF scheme provides selected young researchers an opportunity to conduct and lead research that is a relevant to defence. It offers:

- 3-year research grant, with an option to extend up to a further 3 years,
- possible tenure-track academic appointment with the university at the end of the TRF,
- attractive and competitive remuneration.

Fellows may lead and conduct research, and publish in these areas:

1. Biomimicry
2. Cognitive Sciences
3. Cyber Security
4. Computational Photography
5. Microsystem Technologies

Other fundamental areas of science or technology, where a breakthrough would be of interest to defence and security, will also be considered.

Singapore is a globally connected cosmopolitan city-state with a supportive environment and vibrant research culture. For more information and application procedure, please visit

NTU – http://www3.ntu.edu.sg/trf/
NUS – http://www.nus.edu.sg/dpr/funding/trf.htm

Closing date: 11 January 2011

Shortlisted candidates will be invited to Singapore to present their research plans, meet local researchers and identify potential collaborators in April/May 2011.

Michigan Technological University
Department of Computer Science
Department Chair

Michigan Technological University invites applications and nominations for the position of Chair of the Department of Computer Science. The chair will be expected to build on a strong undergraduate degree program and continue the development of graduate education and research programs. Candidates are expected to have a professional record of accomplishments commensurate with the rank of full professor at Michigan Tech, including a record of high quality publications and external funding. Candidates must also have demonstrated administrative, supervisory, or leadership experience.

The Computer Science Department has 325 undergraduate majors in three BS degree

programs and 50 graduate students in MS and PhD degree programs in Computer Science

and in the Computational Science and Engineering PhD program. The research interests of the 17 faculty include both core areas of computer science and interdisciplinary topics. The Department has close ties to the Department of Electrical and Computer Engineering and offers many courses required by the Computer Engineering, Bioinformatics, and Cheminformatics BS degree programs.

The University has approximately 7,000 students and 400 faculty with educational and research programs that emphasize solving technological problems in all aspects of life. Michigan Tech is located in Michigan's scenic Upper Peninsula and is bounded by Lake Superior and nearby forests. The community offers year-round recreational and cultural opportunities. This environment, combined with a competitive compensation package, provides an excellent quality of life.

In addition to the present search, strategic faculty hiring initiatives with up to ten new positions in "Next Generation Energy Systems" and "Health: Basic Sciences, Technologies, and Medical Informatics" are in their second year. Qualified candidates are encouraged to send a separate application, following the "How to Apply" guidelines at http://www.mtu.edu/sfhi.

Michigan Tech is an ADVANCE institution, one of a limited number of universities in

receipt of NSF funds in support of our commitment to increase diversity and the

participation and advancement of women in STEM.

Applications must include a vita, list of references, and a cover letter that addresses the candidate's professional qualifications and administrative philosophy. Applications received by November 15, 2010, are assured of full consideration. Applications must be submitted by email to CSchairSearch@mtu.edu.

To learn more about this opportunity, please visit: http://www.cs.mtu.edu/CSchairSearch.html or contact:

Prof. Steven Seidel
steve@mtu.edu
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931-1295

Michigan Technological University is an equal opportunity educational institution/equal opportunity employer.

Montana State University
RightNow Technologies Professorships in Computer Science

The Montana State University Computer Science Department is searching for two faculty members at either the Assistant, Associate or Full level, based on experience. Candidates at the Associate or Full level must have established or rising prominence in their field. A three-year start-up package is being provided by RightNow Technologies. Montana State University is a Carnegie Foundation RU/VH research university with an enrollment of approximately 13,000. The website www.cs.montana.edu/faculty-vacancies has information on position requirements and application procedures. ADA/EO/AA/Veterans Preference.

New Mexico State University
Assistant Professor

The Computer Science Department at New Mexico State University invites applications for a tenure-track position at the assistant professor level, with appointment starting in the Fall 2011 semester. We are seeking strong candidates in any areas of Computer Science, although applications with expertise in computer architecture, operating systems, compilers, and computer graphics/animation are particularly encouraged. Applications from women and members of traditionally under-represented groups are strongly encouraged. For the complete announcement and application procedure, please visit http://www.cs.nmsu.edu/˜epontell/job.html. Apply URL: http://www/cs/nmsu.edu/˜cssearch

New Mexico State University is an EEO/AA Employer. All university positions are contingent upon availability of funding. All offers of employment, oral and written, are contingent on the university's verification of credentials and other information required by federal law, state law, and NMSU policies/procedures, and may include the completion of a criminal history check.

New York University
Courant Institute of Mathematical Sciences
Department of Computer Science
Faculty

The department expects to have several regular faculty positions beginning in September 2011 and invites candidates at all levels. We will consider outstanding candidates in any area of computer science, with systems and formal methods/verification being high-priority areas.

Faculty members are expected to be outstanding scholars and to participate in teaching at all levels from undergraduate to doctoral. New appointees will be offered competitive salaries and startup packages, with affordable housing within a short walking distance of the department. New York University is located in Greenwich Village, one of the most attractive residential areas of Manhattan.

The department has 32 regular faculty members and several clinical, research, adjunct, and visiting faculty members. The department's current research interests include algorithms, cryptography and theory; computational biology; distributed computing and networking; graphics, vision and multimedia; machine learning; natural language processing; scientific computing; and verification and programming languages.

Collaborative research with industry is facilitated by geographic proximity to computer science activities at AT&T, Google, IBM, Bell Labs, NEC, and Siemens.

Please apply at https://cs.nyu.edu/webapps/facapp/register

To guarantee full consideration, applications should be submitted no later than December 1, 2010; however, this is not a hard deadline, as all candidates will be considered to the full extent feasible, until all positions are filled. Visiting positions may also be available.

New York University is an equal opportunity/affirmative action employer.

North Carolina State University
Department of Computer Science
Faculty Positions

The Department of Computer Science at NC State University (NCSU) seeks to fill multiple tenure track faculty positions starting August 16, 2011. Exceptional candidates in all areas of Computer Science will be considered, but of particular interest are candidates specializing in Computer and Network Security.

Successful candidates must have a strong commitment to academic and research excellence, and an outstanding research record commensurate with the expectations of a major research university. Required credentials include a doctorate in Computer Science or a related field. While the department expects to hire faculty primarily at the Assistant Professor level, candidates with exceptional research records are encouraged to apply for senior positions. The Department is one of the largest and oldest in the country. It is part of NCSU's College of Engineering, which has recently received significant increases in private and public funding, faculty positions, and facilities that will assist the Department in achieving its goals. The department's research expenditures and recognition are growing steadily. For example, we have one of the largest concentrations in the country of prestigious NSF Early Career Award winners (total of 20).

NCSU is located in Raleigh, the capital of North Carolina, which forms one vertex of the world-famous Research Triangle Park (RTP). RTP is an innovative environment, both as a metropolitan area with one of the most diverse industrial bases in the world, and as a center of excellence promoting technology and science. The Research Triangle area is routinely recognized in nationwide surveys as one of the best places to live in the U.S. We enjoy outstanding public schools, affordable housing, and great weather, all in the proximity to the mountains and the seashore.

Applications will be reviewed as they are received. The positions will remain open until suitable candidates are identified. Applicants are encouraged to apply by December 15, 2010. Applicants should submit the following materials online at http://jobs.ncsu.edu (reference position number 1091) cover letter, curriculum vitae, research statement, teaching statement, and names and complete contact information of four references, including email addresses and phone numbers. Candidates can obtain information about the department and its research programs, as well as more detail about the positions advertised here at http://www.csc.ncsu.edu/. Inquiries may be sent via email to: facultyhire@csc.ncsu.edu.

North Carolina State University is an equal opportunity and affirmative action employer. In addition, NC State University welcomes all persons without regard to sexual orientation. Individuals with disabilities desiring accommodations in the application process should contact the Department of Computer Science at (919) 515-2858.

Oklahoma State University

Assistant Professor
FACULTY SEARCH
University: Oklahoma State University
Department: Computer Science Department
Position: Faculty Position

Applications are invited for two anticipated full-time, tenure-track Assistant Professor positions.

The term of initial appointment will begin in August 2011.

The Oklahoma State University Computer Science Department is seeking applications from qualified candidates with teaching and research experience in any area of Computer Science. A Ph.D. or D.Sc. in Computer Science or a closely related area is required.

The positions being sought are for the Stillwater campus and duties may be assigned to either Stillwater or Tulsa campuses or both. Please send curriculum vita, a statement of teaching and research experience, and names of three references to:

Chair, Faculty Search Committee,
Computer Science Department
219 MSCS Building
Oklahoma State University
Stillwater, OK 74078-1053

Application via e-mail with pdf attachment(s) is preferred.

Send e-mail to
faculty-search2010@cs.okstate.edu .

For full consideration, applications should be received by December 20, 2010, but applications will be accepted until the positions are filled.

These positions are contingent upon available funding.

Oklahoma State University is an Affirmative Action/Equal Opportunity/E-Verify employer committed to diversity. OSU-Stillwater is a tobacco-free campus.

Oklahoma State University is a modern comprehensive land grant university that serves the state, national and international communities by providing its students with exceptional academic experiences, by conducting scholarly research and other creative activities that advance fundamental knowledge, and by disseminating knowledge to the people of Oklahoma and throughout the world.

Rice University,
Computer Science Dept.

Tenure-track Assistant Professor

Rice University's Department of Computer Science seeks applications for a tenure-track assistant professor to start in July 2011. We welcome candidates in all areas of computer science. We have particular interest in candidates with experience in one or more of the areas: programming language theory, formal verification, or security.

The Department also has openings for two joint lecturer/post-doctoral researcher positions and a variety of research positions. Research positions are contingent on external funding.

Applicants must hold a Ph.D. or equivalent in Computer Science or a related discipline, or must complete the Ph.D. by November 1, 2011. Please specify in your application if you are applying for the assistant professor position, for one of the lecturer/postdoc positions, or for a research position.

We will begin evaluating applications on November 15, 2010. Applications submitted after that date may be considered, but we would prefer that you complete your application by November 15.

More details on these positions can be found at the Department's web site http://compsci.rice.edu. Apply URL: http://csfacultyapplications.rice.edu

Rice University is an Equal Opportunity/Affirmative Action Employer.

Saint Vincent College

Tenure-track position

The Computing & Information Science Department at Saint Vincent College invites applications for a tenure track, assistant professor position beginning in August 2011, contingent on funding and staffing within the Department. Applicants should hold a graduate degree (preferably a PhD or ABD) in information technology, information systems, computer science, or a closely-related discipline. Primary duties would include undergraduate teaching and research in the areas of networking, IT, computer security, and databases. Candidates should be prepared to teach introductory CS classes, and the ability to teach some of the following courses is a plus: programming languages, Java, graphics, and game development.

Saint Vincent College is a Catholic, Benedictine liberal arts and science college of about 1700 undergraduate students and 200 graduate students. It is located about forty miles east of Pittsburgh, Pennsylvania in a pleasant suburban/rural environment near the foothills of the Laurel Mountains. Saint Vincent is an equal opportunity employer. Review of applications will begin on December 1, 2010 and continue until the position is filled. To apply, send a letter of application, curriculum vita, research statement, teaching statement, transcripts, and three letters of reference to:

Human Resources Director
Saint Vincent College
300 Fraser Purchase Road
Latrobe, PA 15650-2690
www.stvincent.edu/hr2

Stanford University

Tenured Professor Position in the Communication Dept

The Department of Communication at Stanford University invites applications for a tenured professor position in the Department of Communication. The areas of expertise of applicants can include, but are not limited to, the changing forms of journalism, the economics and regulation of journalism, freedom of expression in the digital age, the changing role of the media in campaigns and elections, and the relation between news programming and informed citizenship. Applicants will be expected to teach at the graduate and undergraduate levels in both academic and pre-professional curricula.

We seek an innovative intellectual leader with an interdisciplinary orientation whose work speaks to both the academic and professional communities. Applicants should have a record of substantial research accomplishments in peer reviewed publications. The successful applicant is expected to eventually assume the directorship of the graduate program in journalism in the department.

Applicants should send curriculum vitae, bibliography, and a brief statement of research interest to: Professor James S. Fishkin, Chair, Department of Communication, McClatchy Hall, Stanford University, Stanford, CA 94305-2050. For full consideration, materials must be received by January 15, 2011. The term of appointment would begin September 1, 2011. 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 and members of minority groups, as well as others who would bring additional dimensions to the university's research and teaching missions.

Swarthmore College

Visiting Assistant Professor

Swarthmore College invites applications for a three-year faculty position in Computer Science, at the rank of Visiting Assistant Professor, beginning September 2011. Specialization is open. Review of applications will begin January 1, 2011, and continue until the position is filled. For information, see http://www.cs.swarthmore.edu/job.

Swarthmore College has a strong commitment to excellence through diversity in education and employment and welcomes applications from candidates with exceptional qualifications, particularly those with demonstrable commitments to a more inclusive society and world.

The University of Michigan - Dearborn

Department of Computer and Information Science
Assistant/Associate/Full Professor

The Department of Computer and Information Science (CIS) at the University of Michigan-Dearborn invites applications for a tenure-track faculty position in any of the following areas: computer and data security, digital forensics, and information assurance. Rank and salary will be commensurate with qualifications and experience. We offer competitive salaries and start-up packages.

Qualified candidates must have, or expect to have, a Ph.D. in CS or a closely related discipline by the time of appointment and will be expected to do scholarly and sponsored research, as well as teaching at both the undergraduate and graduate levels. Candidates at the associate or full professor ranks should already have an established funded research program. The CIS Department offers several BS and MS degrees, and participates in an interdisciplinary Ph.D. program in information systems engineering. The current research areas in the department include computer graphics and geometric modeling, database systems, networking, computer and network security, and software engineering. These areas of research are supported by several established labs.

The University of Michigan-Dearborn is located in the southeastern Michigan area and offers excellent opportunities for faculty collaboration with many industries. We are one of three campuses forming the University of Michigan system and are a comprehensive university with over 8500 students. One of university's strategic visions is to advance the future of manufacturing in a global environment.

The University of Michigan-Dearborn is dedicated to the goal of building a culturally-diverse and pluralistic faculty committed to teaching and working in a multicultural environment, and strongly encourages applications from minorities and women.

A cover letter, curriculum vitae including e-mail address, teaching statement, research statement, and three letters of reference should be sent to,

Dr. William Grosky, Chair
Department of Computer and Information Science
University of Michigan-Dearborn
4901 Evergreen Road
Dearborn, MI 48128-1491

Email: wgrosky@umich.edu
Internet: http://www.cis.umd.umich.edu
Phone: 313.583.6424
Fax: 313.593.4256

The University of Michigan-Dearborn is an equal opportunity/affirmative action employer.

Toyota Technological Institute at Chicago

Computer Science Faculty Positions at All Levels

Toyota Technological Institute at Chicago (TTIC) is a philanthropically endowed degree-granting institute for computer science located on the University of Chicago campus. The Institute is expected to reach a steady-state of 12 traditional faculty (tenure and tenure track), and 12 limited term faculty. Applications are being accepted in all areas, but we are particularly interested in

Theoretical computer science
Speech processing
Machine learning
Computational linguistics
Computer vision
Computational biology
Scientific computing

Positions are available at all ranks, and we have a large number of limited term positions currently available.

For all positions we require a Ph.D. Degree or Ph.D. candidacy, with the degree conferred prior to date of hire. Submit your application electronically at:

http://ttic.uchicago.edu/facapp/

Toyota Technological Institute at Chicago is an Equal Opportunity Employer

University of California, Los Angeles

Computer Science Department

The Computer Science Department of the Henry Samueli School of Engineering and Applied Science at the University of California, Los Angeles, invites applications for tenure-track positions in all areas of Computer Science and Computer Engineering. Applications are also encouraged from distinguished candidates at senior levels. Quality is our key criterion for applicant selection. Applicants should have a strong commitment both to research and teaching and an outstanding record of research for their level of seniority.

The University of California is an Equal Opportunity/Affirmative Action Employer. The department is committed to building a more diverse faculty, staff and student body as it responds to the changing population and educational needs of California and the nation. To apply, please visit http://www.cs.ucla.edu/recruit. Faculty applications received by January 15 will be given full consideration.

UMBC

University of Maryland Baltimore County
An Honors University in Maryland
Information Systems Department

The Information Systems Department at UMBC invites applications for a tenure-track faculty position at the Assistant Professor level in the area of data mining starting August 2011. Outstanding candidates in other areas will also be considered.

Candidates must have an earned PhD in Information Systems or a related field no later than August 2011. Applicants engaged in research in data mining with overlapping interest in artificial intelligence in areas including but not limited to privacy preserving data mining, cybersecurity, spatial and spatio-temporal data mining, and knowledge discovery in healthcare data, are of primary interest. Ideal candidates will be engaged in research that spans two or more of these areas with preference given to those who can collaborate with current faculty. Candidates should have a strong potential for excellence in research, the ability to develop and sustain an externally funded research program, and the ability to contribute to our graduate and undergraduate teaching mission.

The Department offers undergraduate degrees in Information Systems and Business Technology Administration as well as both the MS and PhD in Information Systems. In addition, the Department offers an MS and PhD in Human-Centered Computing. Consistent with the UMBC vision, the Department has excellent technical support and teaching facilities as well as outstanding laboratory space and state of the art technology. UMBC's Technology Center, Research Park, and Center for Entrepreneurship are major indicators of active research and outreach. Further details on our research, academic programs, and faculty can be found at http://www.is.umbc.edu/. Underrepresented groups including women and minorities are especially encouraged to apply.

Applications will not be reviewed until the following materials are received: a cover letter, a one-page statement of teaching interests, a one-page statement of research interests, one or more sample research papers, and a CV. Applicants should also arrange to have three letters of recommendation sent to the department as soon as possible. Electronic submission of materials as PDF documents is preferred. Electronic copies should be sent to bmorris@umbc.edu. Copies can also be sent to: Dr. Andrew Sears, Chair of Faculty Search Committee, Information Systems Department, UMBC, 1000 Hilltop Circle, Baltimore, MD 21250-5398. For inquiries, please contact Barbara Morris at (410) 455-3795 or bmorris@umbc.edu.

Review of applications will begin immediately and will continue until the position is filled. This position is subject to the availability of funds.

UMBC is an Affirmative Action/Equal Opportunity Employer and welcomes applications from minorities, women and individuals with disabilities.

University at Buffalo, The State University of New York

Faculty Position in Computer Science and Engineering

The CSE Department invites excellent candidates in all core areas of Computer science and Engineering, especially Database Systems, Data Mining, Information Retrieval, Machine Learning and Robotics areas, to apply for an opening at the assistant professor level.

The department is affiliated with successful centers devoted to biometrics, bioinformatics, biomedical computing, cognitive science, document analysis and recognition, high performance computing, and information assurance.

Candidates are expected to have a Ph.D. in Computer Science/Engineering or related field by August 2011, with an excellent publication record and potential for developing a strong funded research program.

Applications should be submitted by December 31, 2010 electronically via http://www.ubjobs.buffalo.edu/

The University at Buffalo is an Equal Opportunity Employer/Recruiter.

The University of Alabama

Tenure-Track Faculty Position
Software Engineering Focus
Department of Computer Science

The Department of Computer Science at the University of Alabama invites applications for a new tenure-track Assistant Professor position to begin August 2011. The general area of focus is in software engineering, with a high priority area in model-driven engineering. Candidates must have an earned Ph.D. in Computer Science or a related field, with solid evidence of superior research and scholarly accomplishments, as well as demonstrated excellence in teaching. Applicants who specialize in software engineering are encouraged to apply.

The University of Alabama is considered the Capstone of higher education in Alabama and is also the largest institution in the State. The University was ranked in 2010 by US News and World Report as 34th among US public universities. The University led the nation in 2010 with ten USA Today Academic All-Americans (record for any US university) and ranks 10th in the nation among public universities in enrolling National Merit Scholars.

The Department of Computer Science, housed in the College of Engineering, currently has twenty-three faculty members (16 tenured/tenure track faculty, 6 of whom have interests in software engineering), roughly 200 undergraduates in an ABET accredited B.S. degree program, and approximately 60 M.S. and Ph.D. students. Throughout the 2010-2011 academic year, two postdocs in software engineering will be supported in the Department. A recent Department of Education GAANN award provides extended fellowships to six Ph.D. students who are focusing on software engineering.

The Department and College of Engineering are undergoing a period of extensive growth. The Department is housed in a new (opened August 2009) state-of-the art complex. Over the next three years, the University will complete construction of the science and engineering complex, which is comprised of four buildings focused on the expansion of research in engineering and the sciences. Over 3000 square feet of new research space has been constructed to support the efforts of the software engineering faculty. The software engineering faculty in the Department are PIs or co-PIs on 13 active awards (over $3.25M) across five different funding agencies.

For more information about the Software Engineering Group at UA, please visit http://software.eng.ua.edu

Details regarding the application procedures for this position are available at http://cs.ua.edu. For information about the position, please contact the Search Committee at faculty.search@cs.ua.edu.

Review of applications will begin late-Fall 2010 and will continue until the position is filled. The University of Alabama is an equal opportunity/affirmative action employer. Women and minority applicants are particularly encouraged to apply.

University of Calgary

Department of Computer Science
Assistant Professor Positions

The Department of Computer Science and the University of Calgary seeks outstanding candidates for two tenure track positions at the Assistant Professor level. Applicants from areas of Database Management and Human Computer Interaction/Information Visualization are of particular interest. Details for each position appear at: http://www.cpsc.ucalgary.ca/.

Applicants must possess a doctorate in Computer Science or a related discipline at the time of appointment, and have a strong potential to develop an excellent research record.

The department is one of Canada's leaders as evidenced by our commitment to excellence in research and teaching. It has an expansive graduate program and extensive state-of-the-art computing facilities. Calgary is a multicultural city that is the fastest growing city in Canada. Calgary enjoys a moderate climate located beside the natural beauty of the Rocky Mountains. Further information about the department is available at http://www.cpsc.ucalgary.ca/.

Interested applicants should send a CV, a concise description of their research area and program, a statement of teaching philosophy, and arrange to have at least three reference letters sent to:

Dr. Carey Williamson
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4 or
search@cpsc.ucalgary.ca

The applications will be reviewed beginning November 2010 and continue until the positions are filled.

All qualified candidates are encouraged to apply; however, Canadians and permanent residents will be given priority. The University of Calgary respects, appreciates, and encourages diversity.

University of Pennsylvania

Department of Computer and Information Science
Faculty Position

The University of Pennsylvania invites applicants for tenure-track appointments in computer graphics and animation to start July 1, 2011. Tenured appointments will also be considered.

Faculty duties include teaching undergraduate and graduate students and conducting high-quality research. Teaching duties will be aligned with two programs: the Bachelor of Science and Engineering in Digital Media Design, and the Master of Science and Engineering in Computer Graphics and Game Technology (see http://cg.cis.upenn.edu). Research and teaching will be enhanced by the recently renovated SIG Center for Computer Graphics, which houses the largest motion capture facility in the region, and is also the home of the Center for Human Modeling and Simulation. Successful applicants will find Penn to be a stimulating environment conducive to professional growth.

The University of Pennsylvania is an Ivy League University located near the center of Philadelphia, the 5th largest city in the US. Within walking distance of each other are its Schools of Arts and Sciences, Engineering, Fine Arts, Medicine, the Wharton School, the Annenberg School of Communication, Nursing, and Law. The University campus and the Philadelphia area support a rich diversity of scientific, educational, and cultural opportunities, major technology-driven industries such as pharmaceuticals, finance, and aerospace, as well as attractive urban and suburban residential neighborhoods. Princeton and New York City are within commuting distance.

To apply, please complete the form located on the Faculty Recruitment Web Site at: http://www.cis.upenn.edu/departmental/facultyRecruiting.shtml. Electronic applications are strongly preferred, but hard-copy applications (including the names of at least four references) may alternatively be sent to:

Chair, Faculty Search Committee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389.

Applications should be received by January 15, 2011 to be assured full consideration. Applications will be accepted until the position is filled. Questions can be addressed to faculty-search@cis.upenn.edu. The University of Pennsylvania values diversity and seeks talented students, faculty and staff from diverse backgrounds.

The University of Pennsylvania does not discriminate on the basis of race, sex, sexual orientation, gender identity, religion, color, national or ethnic origin, age, disability, or status as a Vietnam Era Veteran or disabled veteran in the administration of educational policies, programs or activities; admissions policies; scholarship and loan awards; athletic, or other University administered programs or employment. The Penn CIS Faculty is sensitive to "two–body problems" and opportunities in the Philadelphia region.

University of Rochester

Assistant to Full Professor of Computer Science

The UR Department of Computer Science seeks researchers in computer vision and/or machine learning for a tenure-track faculty position beginning in Fall 2011. Outstanding applicants in other areas may be considered. Candidates must have a PhD in computer science or related discipline. Senior candidates should have an extraordinary record of scholarship, leadership, and funding.

The Department of Computer Science is a select research-oriented department, with an unusually collaborative culture and strong ties to cognitive science, linguistics, and electrical and computer engineering. Over the past decade, a third of its PhD graduates have won tenure-track faculty positions, and its alumni include leaders at major research laboratories such as Google, Microsoft, and IBM.

The University of Rochester is a private, Tier I research institution located in western New York State. The University of Rochester consistently ranks among the top 30 institutions, both public and private, in federal funding for research and development. Half of its undergraduates go on to post-graduate or professional education. The university includes the Eastman School of Music, a premiere music conservatory, and the University of Rochester Medical Center, a major medical school, research center, and hospital system. The Rochester area features a wealth of cultural and recreational opportunities, excellent public and private schools, and a low cost of living.

Candidates should apply online at http://www.cs.rochester.edu/recruit after Nov. 1, 2010. Review of applications will begin on Dec. 1, and continue until all interview openings are filled. The University of Rochester has a strong commitment to diversity and actively encourages applications from candidates from groups underrepresented in higher education. The University is an Equal Opportunity Employer.

The University of Washington Bothell

Lecturer or Senior Lecturer

The University of Washington Bothell Computing and Software Systems Program invites applications for a full-time Lecturer or Senior Lecturer position to begin fall 2011. Duties include teaching/mentoring undergraduate and graduate students, including industry internships. Areas of expertise include: security, networking, operating systems, architecture, databases, multimedia software development, computer engineering, and parallel/distributed computing.

The Bothell campus was founded in 1990 as an innovative, interdisciplinary campus within the University of Washington system - one of the premier institutions of higher education in the US. Faculty members have full access to the resources of a major research university, with the culture and close relationships with students of a small liberal arts college.

For additional information, including application procedures, please see our website at http://www.uwb.edu/CSS/. All University faculty engage in teaching, research, and service. The University of Washington, Bothell is an affirmative action, equal opportunity employer.

University of Waterloo

David R. Cheriton School of Computer Science
Tenured and Tenure-Track Faculty Positions

Applications are invited for several positions in computer science: (a) Up to two senior, tenured David R. Cheriton Chairs in Software Systems are open for candidates with outstanding research records in software systems (very broadly defined). Successful applicants will be acknowledged leaders in their fields or have demonstrated the potential to become such leaders. These positions include substantial research support and teaching reduction. (b) One tenured or tenure-track position is open in the area of Health Informatics, including, but not limited to, healthcare IT, medical informatics, and biomedical systems. The successful applicant will help develop a new graduate degree program in health informatics. (c) One other tenured or tenure-track position is available for excellent candidates in any computing area, but highest priority will be given to candidates specializing in systems software (operating systems, distributed systems, networks, etc.) and information systems (e-commerce systems, enterprise resource planning systems, business intelligence, etc.).

Successful applicants who join the University of Waterloo are expected to be leaders in research, have an active graduate-student program, and contribute to the overall development of the School. A Ph.D. in Computer Science, or equivalent, is required, with evidence of excellence in teaching and research. Rank and salary will be commensurate with experience, and appointments are expected to commence during the 2011 calendar year.

With over 70 faculty members, the University of Waterloo's David R. Cheriton School of Computer Science is the largest in Canada. It enjoys an excellent reputation in pure and applied research and houses a diverse research program of international stature. Because of its recognized capabilities, the School attracts exceptionally well-qualified students at both undergraduate and graduate levels. In addition, the University has an enlightened intellectual property policy which vests rights in the inventor: this policy has encouraged the creation of many spin-off companies including iAnywhere Solutions Inc., Maplesoft Inc., Open Text Corp., and Research in Motion. Please see our web site for more information: http://www.cs.uwaterloo.ca.

To submit an application, please register at the submission site: http://www.cs.uwaterloo.ca/faculty-recruiting. Once registered, instructions will be provided regarding how to submit your application. Although applications will be considered as soon as possible after they are complete and as long as positions are available, full consideration is assured for those received by November 30.

The University of Waterloo encourages applications from all qualified individuals, including women, members of visible minorities, native peoples, and persons with disabilities. All qualified candidates are encouraged to apply; however, Canadian citizens and permanent residents will be given priority. Fall 2010.

University of Wisconsin - Platteville

Assistant Professor - Software Engineering

The Department of Computer Science and Software Engineering at the University of Wisconsin-Platteville invites applications for a tenure-track faculty position in Software Engineering starting Fall, 2011. Candidates must have a Ph.D. in Software Engineering, Computer Science, or closely related field.

The primary duty for this position will be teaching courses in an ABET-accredited software engineering program. Candidates must have excellent verbal and written communication skills and a strong commitment to teaching. Commitment to scholarly and professional activities is also required. The candidate must have a demonstrated commitment to or experience with racially diverse populations. Salary will be competitive and commensurate with experience and qualifications. In addition, there are consulting opportunities with local companies.

Applications must be submitted electronically. Send a letter of application, undergraduate & graduate transcripts, a statement illustrating your commitment to fostering campus racial diversity, and a vita including references to stutenbm@uwplatt.edu. Visit our web site at http://www.uwplatt.edu/csse for more information. Review of applications will begin on December 6, 2010 and continue until a suitable candidate is found.

The University of Wisconsin-Platteville, an equal opportunity, affirmative action employer, seeks to build a diverse faculty and staff and encourages applications from women and persons of color. The names of all nominees and applicants who have not requested in writing that their identities be kept confidential, and of all finalists, will be released upon request. Employment requires a criminal background check.

US Air Force Academy

Distinguished Visiting Professor
& Visiting Chair

U.S. AIR FORCE ACADEMY Department of Computer Science is accepting applications for our Coleman-Richardson Chair and Distinguished Visiting Professor positions. See http://www.usafa.edu/df/dfcs/index.cfm or call (719) 333-7474 for details. U.S. Citizenship required.

Valdosta State University

Assistant Professor

Applications are invited for two ten-month tenure-track faculty positions at the rank of Assistant Professor with starting date of August 1, 2011. Responsibilities include teaching at the undergraduate level, scholarly activity, and service to both the department and the university. For a detailed job description and application instructions email Dr. Ashok Kumar, Interim Head, akumar@valdosta.edu.

Applicants must complete a doctorate in Computer Information Systems, Computer Science, or a closely related field by August 2011, have a commitment to undergraduate teaching, and be able to contribute to curriculum development.

Yale University

Senior Faculty

Yale University's Electrical Engineering Department invites applications from qualified individuals for a senior faculty position in either computer systems or signals & systems. Subfields of interest include wireless communications, networking, systems on a chip, embedded systems, and emerging computing methodologies inspired by advances in the biological sciences, quantum computing, and other novel research directions. All candidates should be strongly committed to both teaching and research and should be open to collaborative research. Candidates should have distinguished records of research accomplishments and should be willing and able to take the lead in the shaping of Yale's expanding programs in either computer engineering or signals & systems. Yale University is an Affirmative Action/Equal Opportunity Employer. Yale values diversity among its students, staff, and faculty and strongly welcomes applications from women and under represented minorities. The review process will begin November 1, 2010. Applicants should send a curriculum vitae to:

Chair
Electrical Engineering Search Committee
Yale University
P.O. Box 208267
New Haven, CT 06520-8267

SKIDMORE COLLEGE

The Department of Mathematics and Computer Science at Skidmore College invites applications for a tenure-track position in Computer Science beginning September 2011. Qualifications include a Ph.D. in Computer Science. The appointment will be at the rank of Assistant Professor. Salary will be competitive and commensurate with experience. In addition, startup funds and pre-tenure sabbaticals are available.

A commitment to quality instruction of undergraduates and continuing scholarly achievement is essential. The department has expertise in the theory of computation, algorithms, artificial intelligence, computer vision and graphics; we hope to widen our areas of expertise with this appointment. In addition to strong disciplinary teaching and scholarship, including collaborative research with students, the College encourages interdisciplinary teaching and scholarship via participation in its First-Year Experience program and other interdisciplinary programs such as Neuroscience, Environmental Studies, and Gender Studies.

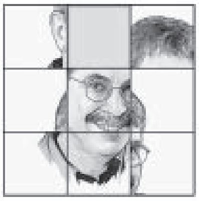
Located in the vibrant community of Saratoga Springs, New York, Skidmore College is a highly selective liberal arts college of 2400 students. The Department of Mathematics and Computer Science consists of eleven faculty members, offers both major and minor programs in Computer Science, and has its own Linux network supported by a dedicated system administrator within the department . For more detailed information, please go to http://cms.skidmore.edu/mcs/.

Review of applications will begin December 15, 2010 and will continue until the position is filled. Applications from members of underrepresented groups are especially encouraged.

To learn more about and apply for this position please visit Skidmore's website at: jobs.skidmore.edu/applicants/Central?quickFind=52518

Skidmore College is committed to being an inclusive campus community and, as an Equal Opportunity Employer, does not discriminate in its hiring or employment practices on the basis of gender, race or ethnicity, color, national origin, religion, age, disability, family, veteran or marital status, sexual orientation, gender identity or expression.

CREATIVE THOUGHT MATTERS.



DOI:10.1145/1839676.1839700

Peter Winkler

Puzzled
Rectangles Galore

Welcome to three new puzzles. Solutions to the first two will be published next month; the third is (as yet) unsolved. In each, the issue is how your intuition matches up with the mathematics.

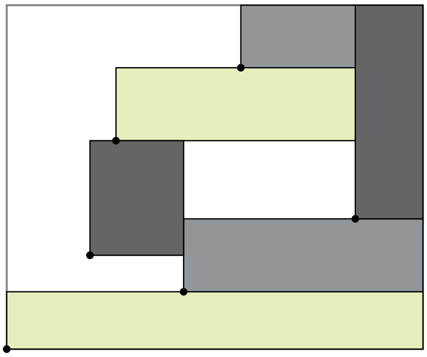
The hero of this column is the simple, ordinary, axis-aligned rectangle. Looking out the window, how many do you see? My view at the moment of Cambridge, MA, easily takes in more than one thousand, mostly windows. Asking new questions about an old figure helps us see it in a new light.

1. A large rectangle in the plane is partitioned into a finite number of smaller rectangles, each with either integer width or integer height; that is, its width, height, or both width and height are whole numbers of units. Now prove that the large rectangle, likewise, has integer width or height (or both).

2. You are in a large rectangular room with mirrored walls. Your mortal enemy, armed with a laser gun, is elsewhere in the room. As your only defense, you may summon a number of graduate students to stand at designated spots in the room, blocking all possible shots by the enemy. How many students do you need?

You may assume for this purpose that you, your enemy, and the students are all slim enough to be considered points (viewed from above), rather than solid figures in 3D space. If, for example, you had continuum many graduate students, you could place them around you in a circle, with the enemy outside. But you can do better…

3. Before you (on the plane) is a large rectangle containing a finite number of distinct dots, one of which is at the rectangle's lower-left-hand corner. Your objective is to pack smaller, disjoint rectangles into the big one (with sides parallel to those of the big one) in such a way that each small rectangle includes one of the dots as its own lower-left-hand corner; see the figure for an example.



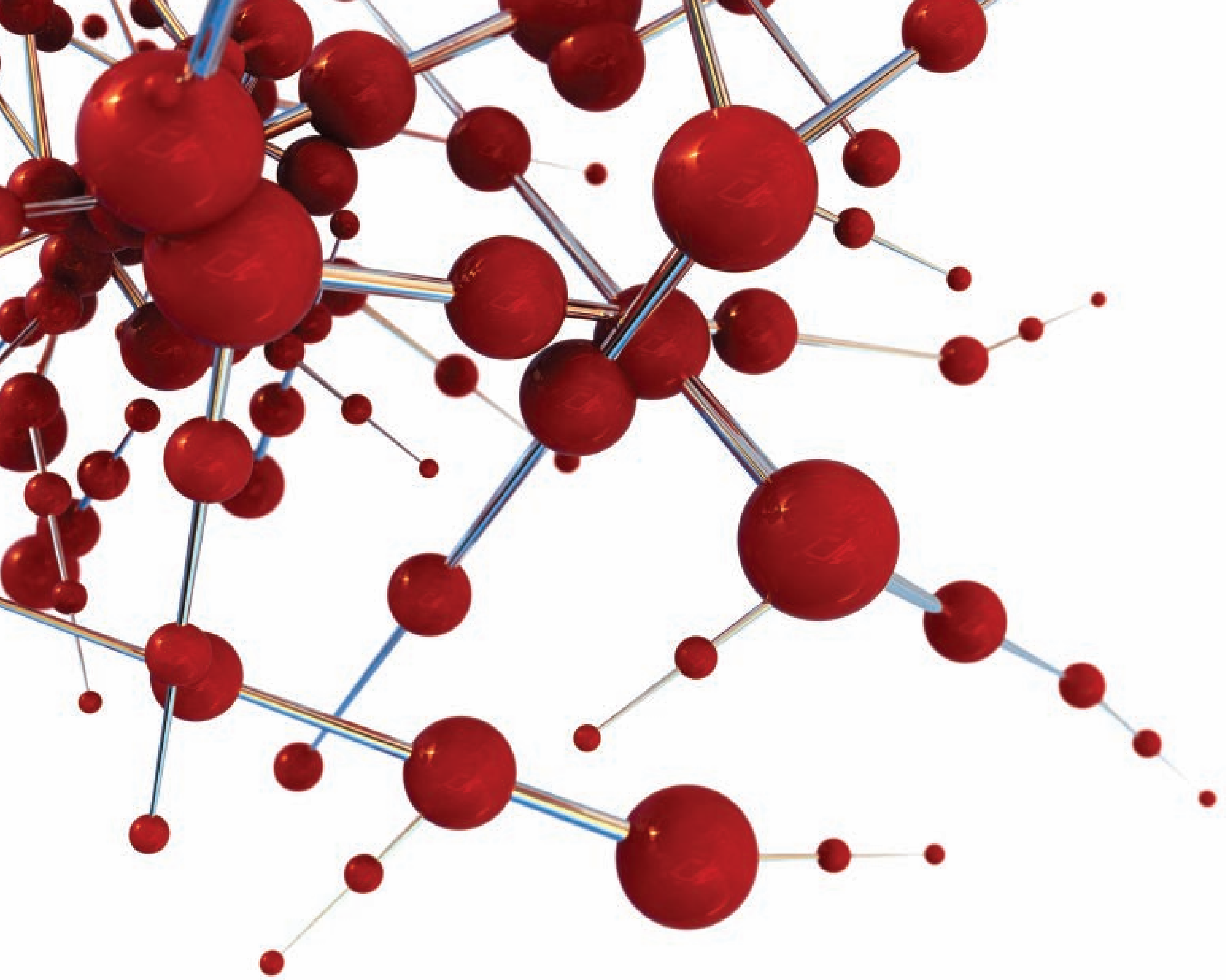
Packing rectangles with six given dots at their lower-left-hand corners.

The conjecture is that there is always a way to choose the small rectangles so they cover at least half the area of the big rectangle. Be the first ever to prove it. A far as I know, no one has succeeded in even showing you can cover any fixed fraction (say, 1/100) of the area of the original rectangle.

Alternatively, if you reject the conjecture, find a counterexample, a way to distribute the dots (be sure to include the lower-left-hand corner of the big rectangle) so there is, provably, no way to do the packing so as to cover half the area of the big rectangle.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.



intel®