

COMMUNICATIONS

CACM.ACM.ORG OF THE

ACM

12/2011 VOL.54 NO.12

Visual Crowd Surveillance Is Like Hydrodynamics

The Legacy of
Steve Jobs

Answer Set
Programming

How Will Astronomy
Archives Survive the
Data Tsunami?

Formal Analysis of
MPI-based Parallel
Programs

Brave NUI World

Association for
Computing Machinery



ADVANCE YOUR CAREER WITH ACM TECH PACKS...

For Serious Computing Professionals.



Searching through technology books, magazines, and websites to find the authoritative information you need takes time.

That's why ACM created "Tech Packs."

- Compiled by **subject matter experts** to help serious computing practitioners, managers, academics, and students understand today's vital computing topics.
- Comprehensive **annotated bibliographies**: from ACM Digital Library articles, conference proceedings, and videos to ACM Learning Center Online Books and Courses to non-ACM books, tutorials, websites, and blogs.
- **Peer-reviewed** for relevance and usability by computing professionals, and **updated** periodically to ensure currency.

Current topics include **Cloud Computing**, **Parallel Computing** and **Enterprise Architecture**. In development are Mobility, Business Intelligence/Data Management, Security, and more.

Suggestions for future Tech Packs? Contact:

Yan Timanovsky
ACM Education Manager
timanovsky@hq.acm.org

Access to ACM resources in Tech Packs is available to ACM members at <http://techpack.acm.org> or <http://learning.acm.org>.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

ACM's Career & Job Center!

Are you looking for your next IT job?

Do you need Career Advice?

Visit ACM's Career & Job Center at:

<http://www.acm.org/careercenter>



The ACM Career & Job Center offers ACM members a host of career-enhancing benefits:

- A highly targeted focus on job opportunities in the computing industry
- Access to hundreds of corporate job postings
- Resume posting keeping you connected to the employment market while letting you maintain full control over your confidential information
- An advanced Job Alert system that notifies you of new opportunities matching your criteria
- Career coaching and guidance from trained experts dedicated to your success
- A content library of the best career articles compiled from hundreds of sources, and much more!



Association for
Computing Machinery

Advancing Computing as a Science & Profession

The ACM Career & Job Center is the perfect place to begin searching for your next employment opportunity!

Visit today at

<http://www.acm.org/careercenter>



Departments

- 5 **Editor's Letter**
Computing for Humans
By Moshe Y. Vardi
-
- 7 **Letters To The Editor**
**To Boost Presentation Quality,
Ask Questions**
-
- 8 **BLOG@CACM**
**Conferences and Video Lectures;
Scientific Educational Games**
John Langford analyzes whether
conferences should offer
video lectures. Judy Robertson
discusses the merits of the Game
Design Through Mentoring
and Collaboration project.
-
- 10 **Nominees for Elections and Report
of the ACM Nominating Committee**
-
- 25 **Calendar**
-
- 104 **Careers**

Last Byte

- 142 **Solutions and Sources**
By Peter Winkler
-
- 144 **Q&A**
Scaling Up
M. Frans Kaashoek talks about
multicore computing, security,
and operating system design.
By Leah Hoffmann

News

- 11 **The Rise of Molecular Machines**
The field of molecular computing
is achieving new levels of control
over biochemical processes and
fostering sophisticated connections
between computer science and
the biological sciences.
By Kirk L. Kroeker
-
- 14 **Brave NUI World**
Natural user interface developments,
such as Microsoft's Kinect, may
indicate the beginning of the end
for the mouse.
By Gregory Goth
-
- 17 **Activism Vs. Slacktivism**
Today's activists are highly plugged
into social media, mobile apps,
and other digital tools. But does
this make a difference where
it matters most?
By Dennis McCafferty
-
- 20 **CSEdWeek Takes Hold**
Groups in more than 130 countries
will participate in Computer Science
Education Week this year.
By Samuel Greengard
-
- 21 **Dennis Ritchie, 1941–2011**
Colleagues recall the creator of C and
codeveloper of Unix, an unassuming
but brilliant man who enjoyed playing
practical jokes on his coworkers.
By Paul Hyman

Viewpoints

- 22 **The Most Ancient Marketing**
By Jaron Lanier
-
- 24 **Life, Death, and the iPad:
Cultural Symbols and Steve Jobs**
By Genevieve Bell
-
- 26 **Technology Strategy and Management**
The Legacy of Steve Jobs
Reflecting on the career and
contributions of the Apple cofounder.
By Michael A. Cusumano
-
- 29 **Emerging Markets**
**On Turbocharged, Heat-Seeking,
Robotic Fishing Poles**
Applying a well-known proverb to
socio-technical transformation.
By Kentaro Toyama
-
- 32 **Kode Vicious**
Debugging on Live Systems
It is more of a social than
a technical problem.
By George V. Neville-Neil
-
- 34 **Broadening Participation**
**Data Trends on Minorities and People
with Disabilities in Computing**
Seeking a comprehensive view of
minority student demographics to
determine what programs and policies
are needed to promote diversity.
By Valerie Taylor and Richard Ladner
-
- 38 **The Profession of IT**
The Grounding Practice
The skill of making and recognizing
grounded claims is essential for
professional practice. Getting
objective data to support your
conclusions is not enough.
By Peter J. Denning
-
- 41 **Viewpoint**
**Doctoral Program Rankings for U.S.
Computing Programs: The National
Research Council Strikes Out**
A proposal for improving doctoral
program ranking strategy.
By Andrew Bernat and Eric Grimson



Practice



52

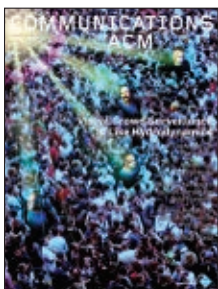
- 44 **Postmortem Debugging in Dynamic Environments**
Many modern dynamic languages lack tools for understanding complex failures.
By David Pacheco

- 52 **How Will Astronomy Archives Survive the Data Tsunami?**
Astronomers are collecting more data than ever. What practices can keep them ahead of the flood?
By G. Bruce Berriman and Steven L. Groom

- 57 **Coding Guidelines: Finding the Art in the Science**
What separates good code from great code?
By Robert Green and Henry Ledgard



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



About the Cover:
This month's cover story (p. 64) investigates the challenges of video surveillance of crowded scenes. The authors propose a framework that treats the interactions of people in a scene like moving particles in a liquid, thus considering techniques often found in the study of hydrodynamics.

Contributed Articles



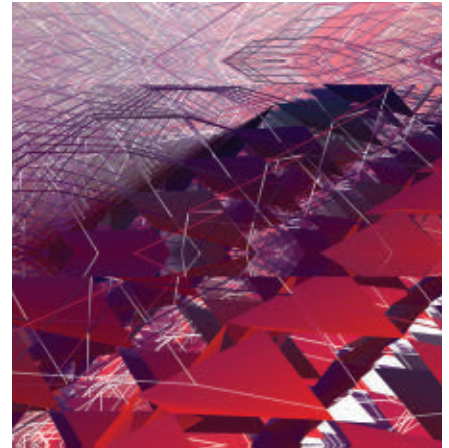
74

- 64 **Visual Crowd Surveillance through a Hydrodynamics Lens**
People in high-density crowds appear to move with the flow of the crowd, like particles in a liquid.
By Brian E. Moore, Saad Ali, Ramin Mehran, and Mubarak Shah

- 74 **License Risks from Ad Hoc Reuse of Code from the Internet**
Software developers' reuse of code from the Internet bears legal and economic risks for their employers.
By Manuel Sojer and Joachim Henkel

- 82 **Formal Analysis of MPI-based Parallel Programs**
The goal is reliable parallel simulations, helping scientists understand nature, from how foams compress to how ribosomes construct proteins.
By Ganesh Gopalakrishnan, Robert M. Kirby, Stephen Siegel, Rajeev Thakur, William Gropp, Ewing Lusk, Bronis R. de Supinski, Martin Schulz, and Greg Bronevetsky

Review Articles



- 92 **Answer Set Programming at a Glance**
The motivation and key concepts behind answer set programming—a promising approach to declarative problem solving.
By Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński

Research Highlights

- 122 **Technical Perspective**
Safety First!
By Xavier Leroy
- 123 **Safe to the Last Instruction: Automated Verification of a Type-Safe Operating System**
By Jean Yang and Chris Hawblitzel
- 132 **Technical Perspective**
Anonymity Is Not Privacy
By Vitaly Shmatikov
- 133 **Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Steganography**
By Lars Backstrom, Cynthia Dwork, and Jon Kleinberg



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

Executive Director and CEO

John White

Deputy Executive Director and COO

Patricia Ryan

Director, Office of Information Systems

Wayne Graves

Director, Office of Financial Services

Russell Harris

Director, Office of SIG Services

Donna Cappel

Director, Office of Publications

Bernard Rous

Director, Office of Group Publishing

Scott E. Delman

ACM COUNCIL

President

Alain Chesnais

Vice-President

Barbara G. Ryder

Secretary/Treasurer

Alexander L. Wolf

Past President

Wendy Hall

Chair, SGB Board

Vicki Hanson

Co-Chairs, Publications Board

Ronald Boisvert and Jack Davidson

Members-at-Large

Vinton G. Cerf; Carlo Ghezzi;

Anthony Joseph; Mathai Joseph;

Kelly Lyons; Mary Lou Soffa; Salil Vadhan

SGB Council Representatives

G. Scott Owens; Andrew Sears;

Douglas Terry

BOARD CHAIRS

Education Board

Andrew McGettrick

Practitioners Board

Stephen Bourne

REGIONAL COUNCIL CHAIRS

ACM Europe Council

Fabrizio Gagliardi

ACM India Council

Anand S. Deshpande, PJ Narayanan

ACM China Council

Jianguang Sun

PUBLICATIONS BOARD

Co-Chairs

Ronald F. Boisvert; Jack Davidson

Board Members

Nikil Dutt; Carol Hutchins;

Joseph A. Konstan; Ee-Peng Lim;

Catherine McGeoch; M. Tamer Ozsu;

Vincent Shen; Mary Lou Soffa

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

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman

publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Jack Rosenberger

Web Editor

David Roman

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Alicia Kubista

Assistant Art Directors

Mia Angelica Balaquiot

Brian Greenberg

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Williams

Columnists

Alok Aggarwal; Phillip G. Armour;

Martin Campbell-Kelly;

Michael Cusumano; Peter J. Denning;

Shane Greenstein; Mark Guzdial;

Peter Harsha; Leah Hoffmann;

Mari Sako; Pamela Samuelson;

Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission

permissions@cacm.acm.org

Calendar items

calendar@cacm.acm.org

Change of address

acmhlp@cacm.acm.org

Letters to the Editor

letters@cacm.acm.org

WEB SITE

<http://cacm.acm.org>

AUTHOR GUIDELINES

<http://cacm.acm.org/guidelines>

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY 10121-0701

T (212) 869-7440

F (212) 869-0481

Director of Media Sales

Jennifer Ruzicka

jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

Association for Computing Machinery (ACM)

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (212) 869-7440; F (212) 869-0481

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi

vic@cacm.acm.org

NEWS

Co-chairs

Marc Najork and Prabhakar Raghavan

Board Members

Hsiao-Wuen Hon; Mei Kobayashi;

William Pulleyblank; Rajeev Rastogi;

Jeannette Wing

VIEWPOINTS

Co-chairs

Susanne E. Hambrusch; John Leslie King;

J Strother Moore

Board Members

P. Anandan; William Aspray; Stefan Bechtold;

Judith Bishop; Stuart I. Feldman;

Peter Freeman; Seymour Goodman;

Shane Greenstein; Mark Guzdial;

Richard Heeks; Rachelle Hollander;

Richard Ladner; Susan Landau;

Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen

PRACTICE

Chair

Stephen Bourne

Board Members

Eric Allman; Charles Beeler; David J. Brown;

Bryan Cantrill; Terry Coatta; Stuart Feldman;

Benjamin Fried; Pat Hanrahan; Marshall Kirk

McKusick; Erik Meijer; George Neville-Nel;

Theo Schlossnagle; Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of *Queue*.

CONTRIBUTED ARTICLES

Co-chairs

Al Aho and Georg Gottlob

Board Members

Robert Austin; Yannis Bakos; Elisa Bertino;

Gilles Brassard; Kim Bruce; Alan Bundy;

Peter Buneman; Andrew Chien;

Peter Druschel; Blake Ives; James Larus;

Igor Markov; Gail C. Murphy; Shree Nayar;

Bernhard Nebel; Lionel M. Ni;

Sriram Rajamani; Marie-Christine Rousset;

Avi Rubin; Krishan Sabnani;

Fred B. Schneider; Abigail Sellen;

Ron Shamir; Marc Snir; Larry Snyder;

Veda Storey; Manuela Veloso; Michael Vitale;

Wolfgang Wahlster; Hannes Werthner;

Andy Chi-Chih Yao

RESEARCH HIGHLIGHTS

Co-chairs

Stuart J. Russell and Gregory Morrisett

Board Members

Martin Abadi; Stuart K. Card; Jon Crowcroft;

Shafi Goldwasser; Monika Henzinger;

Maurice Herlihy; Dan Huttenlocher;

Norm Jouppi; Andrew B. Kahng;

Daphne Koller; Michael Reiter;

Mendel Rosenblum; Ronitt Rubinfeld;

David Salesin; Lawrence K. Saul;

Guy Steele, Jr.; Madhu Sudan;

Gerhard Weikum; Alexander L. Wolf;

Margaret H. Wright

WEB

Co-chairs

James Landay and Greg Linden

Board Members

Gene Golovchinsky; Marti Hearst;

Jason I. Hong; Jeff Johnson; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2011 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

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

Subscriptions

An annual subscription cost is included in ACM member dues of \$99 (\$40 of which is allocated to a subscription to *Communications*); for students, cost is included in \$42 dues (\$20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is \$100.

ACM Media Advertising Policy

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

Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact acmhlp@acm.org.

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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



Association for Computing Machinery



Printed in the U.S.A.



DOI:10.1145/2043174.2043175

Moshe Y. Vardi

Computing for Humans

Gottfried Wilhelm Leibniz (1646–1716) has been called the “Patron Saint of Computing.” Leibniz is most famous for the development—independently

of Isaac Newton—of the infinitesimal calculus. (In fact, it is his mathematical notation we use today, rather than Newton’s.) He was also a prolific inventor of mechanical calculators and developed the binary number system.

Leibniz conceived of a universal mathematical language, *lingua characteristica universalis*, in which all human knowledge can be expressed, and calculational rules, *calculus ratiocinator*, carried out by machines to derive all logical relationships. Leibniz’s goal was nothing short of prophetic: “Once the characteristic numbers are established for most concepts, mankind will then possess a new instrument that will enhance the capabilities of the mind to a far greater extent than optical instruments strengthen the eyes.”

This definition of computing, as an “instrument for the human mind,” captures, I believe, the essence of our field. On one hand, our discipline is a technical one, focusing on hardware, software, and their theoretical foundations. On the other hand, the artifacts we build are meant to enhance the human mind. This duality of our field is witnessed by the two pioneers we lost last October: Steve Jobs and Dennis Ritchie, who passed away within a week of each other.

Dennis MacAlistair Ritchie (September 9, 1941–October 12, 2011) was the techies’ techie, as the creator of the C programming language and the codeveloper of the Unix operating system. The C language paved the way for C++ and Java, while Unix was the basis for many of today’s most widely used

operating systems. Before the development of C and Unix, programming—especially systems programming—was tightly connected to the underlying hardware. C and Unix, in contrast, were highly portable. The citation for the 1983 Turing Award that Richie received together with Ken Thompson refers succinctly to “their development of generic operating systems theory.” There is no computer scientist who is not familiar with C and Unix, but it is unlikely your cousin heard about them, unless she is also a computer scientist. Undoubtedly, however, your cousin is familiar with Steve Jobs.

Steven Paul “Steve” Jobs (February 24, 1955–October 5, 2011) was the founder of Apple, NeXT, and Pixar. His death received a tremendous amount of worldwide news coverage and is addressed by three articles in this issue of *Communications*. It is hard to think of anyone in recent memory whose passing received so much global attention. This level of interest is by itself worthy of observation. As Jaron Lanier makes clear in his essay, “The Most Ancient Marketing,” Jobs was very much *not* an engineer. In fact, the title of one of the many essays published in the wake of Jobs’ death is “Why Jobs Is No Edison.” Yet, it is difficult to point to anyone who had as much impact on computing over the last 30 years as Jobs. In fact, as Genevieve Bell points out in her essay, “Life, Death, and the iPad: Cultural Symbols and Steve Jobs,” his impact goes beyond the world of computing, well into the realm of culture. (For a

discussion of Job’s business strategy, see Michael A. Cusumano’s “Technology Strategy and Management.”

Undoubtedly, Jobs’ uniqueness was his relentless and singular focus on the human side of computing. To start with, the Apple and Apple II were *personal* computers, and the Mac’s claim to fame was its *user* interface. The sequence of products that revolutionized computing over the past 10 years, the iPod, iPhone, and iPad, was unique in its focus on *user experience*. In fact, the very term “user experience” was coined at Apple in the mid-1990s. The success of Apple’s products over the past decade made this term quite fashionable lately.

Yet the user has not always been and is probably still not at the center of our discipline. A quick perusal of ACM’s Special Interest Groups shows that their general focus tends to be quite technical. In fact, one often encounters among computing professionals an attitude that regards the field of human-computer interaction as “soft,” implying it is less worthy than the “harder” technical areas. In my own technical areas, databases and formal methods, I almost never encounter papers that pay attention to usability issues.

The almost simultaneous departure and Jobs and Ritchie should remind us of the fundamental duality of computing. As Leibniz prophesied, computing is “an instrument for the human mind.” Let us keep the human in the center!

Moshe Y. Vardi, EDITOR-IN-CHIEF



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

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

STUDENT MEMBERSHIP:

- ☐ ACM Student Membership: \$19 USD
- ☐ ACM Student Membership plus the ACM Digital Library: \$42 USD
- ☐ ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ☐ 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*. Student membership dues include \$15 toward a subscription to *XRDS*. 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.

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

☐ Professional Member Dues (\$99 or \$198) \$ _____

☐ ACM Digital Library (\$99) \$ _____

☐ Student Member Dues (\$19, \$42, or \$62) \$ _____

Total Amount Due \$ _____

Card #

Expiration date

Signature

To Boost Presentation Quality, Ask Questions

MOSHE Y. VARDI'S Editor's Letter "Are You Talking to Me?" (Sept. 2011) said conference attendees are sometimes unable to follow speakers' presentations and eventually give up trying. So how about if ACM and IEEE would run an experimental conference where session chairs are expected to ask questions during presentations when they themselves lose track or when audience members clearly stop paying attention. Note such an experiment would have to be done without undue disruption and not allowed to reflect on a particular speaker.

The biggest trade-offs would be the extra time presentations might require and the possibility of upsetting overly sensitive speakers. However, they could be addressed experimentally, initially at small, highly technical conferences with flexible break periods and by selecting only expert, personable chairs to manage the sessions.

Robin Williams, San Jose, CA

Author's Response:

I agree that ACM and IEEE conferences should experiment to improve the quality of their talks. Some ideas can be implemented fairly easily, as in, say, asking conference attendees to give anonymous feedback to speakers. However, one must keep in mind that conferences are grassroots operations, and experiments cannot be dictated by association governing bodies. Rather, the effort to improve conference talks must be undertaken by conferences on their own initiative.

Moshe Y. Vardi, Editor-in-Chief

Adopt the End-to-End Principle in Home Networks

To address the user-experience concerns raised in "Advancing the State of Home Networking" by W. Keith Edwards et al. (June 2011), we must first understand why home networks have been so successful despite the very real difficulties cited in the article. In

attempting to do better for users, we might, in fact, do just the opposite. The authors recognized that developers treat networks as opaque infrastructure, which is the fundamental architectural principle that has made the Internet so generative.

Classic telecommunications is the business of providing services like the public switched telephone network, or PSTN. The Internet is a different concept, providing a common infrastructure for all services. Yet the very power of the Internet, which allows us to tunnel through legacy telecom, has also led us to accept the idea that it is just another service, like PSTN.

In the 1990s this was the plan for home computers, too. Working at Microsoft (Jan. 1995), I realized that home networking could be do-it-yourself rather than a service with a monthly bill and restrictions on what we do. I took the approach of removing complexity rather than adding solutions. Windows 98se supported the necessary protocols to "just work." This involved the requirement that the user would not have to buy any service beyond a single IP but share a single IP address. I wanted to use IPv6 so each device would have a first-class presence. But because IPv6 was not available at the time, I used Network Address Translation to share a single IPv4 address.

Rather than make the home network smarter and more cognizant of the particulars of the home, we must honor the end-to-end principle and treat the Internet as infrastructure. Developers would thus be relieved of the impossible burden of having to understand the home environment and its inhabitants. Any number of approaches could coexist.

Today's Internet protocols date from when big computers were immobile and relationships could be defined through fixed IP addresses. To preserve this simplicity, we need stable relationships for our untethered devices. This way, we could address sources of complexity rather than their symptoms.

Bob Frankston, Newton, MA

Fewer Lines of Code for More Results

In Poul-Henning Kamp's article "The Most Expensive One-Byte Mistake" (Sept. 2011), did Ken, Dennis, and Brian indeed choose wrong with NUL-terminated text strings? I say they chose correctly, then and now. The reason C is dying and nobody has used PL/I, Algol, or Pascal for real work for the past 30 years is that C makes it possible to accomplish a lot in a few lines of intuitive code despite requiring little memory or CPU power. Searching and comparing NUL-terminated strings can be accomplished with such short code segments; programmers hardly need a standard library, and code compiles into a few PDP-11 machine instructions. Failing to check untrusted data is fatal in any language.

C allows fast simple code written by competent programmers, and simple code tends to be less buggy and more readable than complex code. For programmers who still want to use address + length strings, such use can be accomplished in just a few lines. There is, of course, the `strlen()` function to measure the string's length and the `fgets()` function to limit how many characters to read into a string from a file.

Sure, copying large strings can run faster with newer hardware if the string lengths are known. This is a trade-off, and programmers can, if desired, use address + length strings in C and even word-align them. For others, there is always "C with Training Wheels," a.k.a. Pascal or Java, if one is in no special hurry for results.

Good programmers write secure code; bad programmers write insecure, buggy code. Good practices are more valuable than "magic" language features. The largest Java application I know is also the buggiest application I know.

Bob Toxen, Atlanta, GA

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

© 2011 ACM 0001-0782/11/12 \$10.00

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



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

DOI:10.1145/2043174.2043178

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

Conferences and Video Lectures; Scientific Educational Games

John Langford analyzes whether conferences should offer video lectures. Judy Robertson discusses the merits of the Game Design Through Mentoring and Collaboration project.



John Langford
"To Video Lecture or Not"

<http://cacm.acm.org/blogs/blog-cacm/100785>
October 29, 2010

For the first time in several years, ICML 2010 did not have VideoLectures.net attending. Luckily, the tutorial on exploration and learning that Alina Beygelzimer and I put together can be viewed since we also presented at KDD 2010, which included video lecture support.

ICML didn't cover the cost of a video lecture, because PASCAL didn't provide a grant for it this year. On the other hand, KDD covered it out of registration costs. The cost of video lectures isn't cheap. For a workshop the baseline quote we have is 270 euros per hour, plus a similar cost for the cameraman's travel and accommodations. This can be reduced substantially by having a volunteer with a camera handle the cameraman duties, uploading the video and slides to be processed for a quoted 216 euros per hour.

YouTube is the most predominant

JOHN LANGFORD

"Video lectures increase the size of the audience and the value to authors by perhaps a factor of two for a cost about $\frac{1}{3}$ of current presentation costs."

free video site with a cost of \$0, but it turns out to be a poor alternative. Fifteen-minute upload limits do not match typical talk lengths. Video lectures also have side-by-side synchronized slides and video that allows quick navigation of the video stream

and acceptable resolution of typical talk slides. Overall, these differences are substantial enough that YouTube is not presently a serious alternative.

So, if we can't avoid paying the cost, is it worthwhile? One way to judge this is by comparing how much authors currently spend traveling to a conference and presenting research vs. the size of the audience. In general, costs vary wildly, but for a typical academic international conference, airfare, hotel, and registration are commonly at least \$1,000 even after scrimping. The size of audiences also varies substantially, but something in the 30–100 range is a typical average. For KDD 2010, the average number of views per presentation is 14.6, but this is misleadingly low, as KDD presentations were just put up. A better number is for KDD 2009, where the average view number is presently 74.2. This number is reasonably representative with ICML 2009 presently at 115.8. We can argue about the relative merits of online vs. in-person viewing, but the order of their value is at least unclear, since in an online system people specifically seek out lectures to view while at the conference itself people are often opportunistic viewers. Valuing these equally, we see that video lectures increase the size of the audience, and the value to authors by perhaps a factor of two for a cost about $\frac{1}{3}$ of current presentation costs.

This conclusion is conservative, because a video lecture is almost surely viewed over more than a year, costs of conference attendance are often high-

er, and the cost in terms of a presenter's time is not accounted for. Overall, video lecture coverage seems quite worthwhile. Since authors also typically are the attendees of a conference, increasing the registration fees to cover the cost of video lectures seems reasonable. A video lecture is simply a new publishing format.

We can hope that the price will drop over time as it's not clear to me that the 216 euros per hour reflects the real costs of VideoLectures.net. Some competition of a similar quality would be the surest way to do that. But in the near future, whether or not a conference has video lecture support substantially impacts its desirability as a place to send papers.

Reader's comment

I share your emphasis on the importance of collecting public video lectures from academic conferences. However, your main reason for not using YouTube reflects a misconception, and you ignore the many significant advantages in features and reach that make YouTube worth a closer look.

1. "Partner channels" do not have the 15-minute upload limit as can be seen with various academic conference channels such as USENIX¹, as well as various open source conference channels. The limit seems to be getting conference organizers to upload content.

2. YouTube provides automatic machine-generated captions and machine translation into ~50 languages, which greatly expands the reach to the hearing impaired, non-English speakers, etc.

3. The captioned text of conference videos on YouTube is indexed by the Google search engine, so a query for an obscure technical concept that is verbally mentioned in a lecture will show up in search results for related queries. This is a huge benefit for the dissemination of knowledge.

In any case, I hope more sites such as VideoLectures.net add captions that can be indexed by search engines in the future so that there is further competition and feature development to make it easier and cheaper in this space.

—Murray Stokely

Reference

1. <http://www.youtube.com/user/USENIXAssociation>

JUDY ROBERTSON

"If we can't convince every child to become a computer scientist, any kind of scientist will have to do."



Judy Robertson
"Game Design Through Mentoring and Collaboration"

<http://cacm.acm.org/blogs/blog-cacm/101956>
November 19, 2010

I was interested to read about a really fantastic National Science Foundation-funded project called Game Design Through Mentoring and Collaboration (GDMC). Taking place at McKinley Tech and George Mason University, the project encourages young people into STEM careers through weekend and summer courses in computer game design. I particularly like two aspects of GDMC. First, the students learn with slightly more experienced peer mentors as well as an instructor. This can be a very effective model because both the mentor and the mentee can learn a lot, and it gives the teacher much-needed assistance in a busy class full of temperamental computers and children. (If you want to know more about different models for effective mentoring, Kafai et al.¹ is a good place to start.) Second, the students also learn about science subjects and integrate their new knowledge into their games, e.g., after input from a Federation of American Scientists biologist, the students' games included accurate information about antibiotics, glial cells, and neurotransmitters. If we can't convince every child to become a computer scientist, any kind of scientist will have to do.

Game design projects are increasingly popular in education, and the evidence is starting to accumulate

about the effectiveness of such schemes. In an article published earlier this year in *Computers and Education*, Vos and colleagues² compared students' motivation and use of strategies for deep learning when they either played a simple memory drag-and-drop game or constructed their own such game. The children enjoyed making games more than playing them, and were more likely to use deep learning strategies while doing so. A notable finding from this study was that the children were less motivated in the play condition than by their normal classroom lessons. This just goes to show that if you're going to spend classroom time on a game, it had better be good or you might as well not bother. Or perhaps a more positive way of looking at that would be to say it takes a high-quality game to beat an enthusiastic teacher.

Game genre and graphical quality are likely to be factors here. The simple 2D board game style application in this study looks rather dull in comparison to the sort of action game you might find gracing the screen of a Wii. It may well be that making an online board game is more fun than playing it only because playing it isn't that exciting to start with. In contrast, the GDMC students learn a wider range of technical skills that enable them to make 3D games with proper physics. I think this is pretty important because, in my experience, kids want to make games that look and feel as good as the games they play at home. After all, they want their friends to be impressed when they play them. So, hats off to the students on GDMC: Your counterparts across the pond in Scotland (see <http://www.adventure-author.org>) salute you! C

References

1. Kafai, Desai, Peppler, Chiu, and Moya, "The multiple roles of mentoring," *The Computer Clubhouse: Constructionism and Creativity in Youth Communities*, Kafai, Peppler, Chapman (Eds.), Teachers College Press, New York, 2009.
2. Vos, van der Meijden, Denessen, "Effects of constructing versus playing an educational game on student motivation and deep learning strategy use," *Computers & Education* 56, 1, January 2011; DOI: 10.1016/j.compedu.2010.08.013.

John Langford is a senior researcher at Yahoo! Research in New York. Judy Robertson is a lecturer at Heriot-Watt University.

© 2011 ACM 0001-0782/11/12 \$10.00

DOI:10.1145/2043174.2043179

Nominees for Elections and Report of the ACM Nominating Committee

In accordance with the Constitution and Bylaws of the ACM, the Nominating Committee hereby submits the following slate of nominees for ACM's officers. In addition to the officers of the ACM, five Members at Large will be elected. The names of the candidates for each office are presented in random order below:

President (7/1/12–6/30/14):

Barbara G. Ryder, Virginia Tech

Vinton G. Cerf, Google

Vice President (7/1/12–6/30/14):

Mathai Joseph, Advisor, Tata Consultancy Services

Alexander L. Wolf, Imperial College London

Secretary/Treasurer (7/1/12–6/30/14):

George V. Neville-Neil, Neville-Neil Consulting

Vicki L. Hanson, University of Dundee

Members at Large (7/1/12–6/30/16):

Radia Perlman, Intel

Ricardo Baeza-Yates, Yahoo! Research, Barcelona/Santiago

Feng Zhao, Microsoft Research, Beijing

Eric Allman, Sendmail Inc.

Mary Lou Soffa, University of Virginia

P.J. Narayanan, IIT-Hyderabad

Eugene Spafford, Purdue University

The Constitution and Bylaws provide that candidates for elected offices of the ACM may also be nominated by petition of one percent of the Members who as of **November 1** are eligible to vote for the nominee. Such petitions must be accompanied by a written declaration that the nominee is willing to stand for election. The number of Member signatures required for the offices of President, Vice President, Secretary/Treasurer, and Members at Large, is 683.

The Bylaws provide that such petitions must reach the Elections Committee before **January 31**. Original petitions for ACM offices are to be submitted to the ACM Elections Committee, c/o Pat Ryan, COO, ACM Headquarters, 2 Penn Plaza, Suite 701, New York, NY 10121, USA, by January 31, 2012. Duplicate copies of the petitions should also be sent to the Chair of the Elections Committee, Gerry Segal, c/o ACM Headquarters. All candidates nominated by petition are reminded of the requirements stated in the Policy and Procedures on Nominations and Elections that a candidate for high office must meet in order to serve with distinction. Copies of this document are available from Rosemary McGuinness, Office of Policy and Administration, ACM Headquarters. Statements and biographical sketches of all candidates will appear in the May 2012 issue of *Communications*.

The Nominating Committee would like to thank all those who helped us with their suggestions and advice.

Dame Wendy Hall, CHAIR

Anand Deshpande, **Ben Fried**, **Cherri Pancake**, and **Robert A. Walker**

ACM Member News

MENTORING WITH MARY FERNÁNDEZ



Before Mary Fernández arrived at Brown University as a freshman in 1981, she had

never seen a computer. Then, an influential Brown professor, Andries van Dam, told her introduction to computer science class that everyone would have one in their home someday. She was astonished.

"He said we'd carry them around in our pockets," Fernández says. "This seemed ludicrous. At that time, computers were still the size of trucks."

Van Dam's teaching captured her imagination to the point where she was "hooked," says Fernández, who switched her major to computer science.

As executive director of distributed computing research for AT&T Labs Research in Florham Park, NJ, Fernández is immersed in cloud infrastructure research, but her other passion is inspiring young women and minority students to pursue a career in computer science. In 1998, Fernández joined MentorNet, which matches mentors and protégés, and became a board member in 2008. She was elected chair earlier this year.

For her professional accomplishments and work with MentorNet, Fernández was recently honored with the Outstanding Technical Achievement Award from HENAAC/Great Minds in STEM.

One of Fernández's key goals now is to raise awareness about opportunities in STEM and computer science in her own ethnic community. "The Latino immigrant population, in particular, often doesn't even know that computer science is an option for them," she says. "Latino parents should know that STEM fields will provide their children with stable careers, excellent starting salaries, and ample room for growth. It is important for them to understand that their child's university education will pay huge dividends in the future."

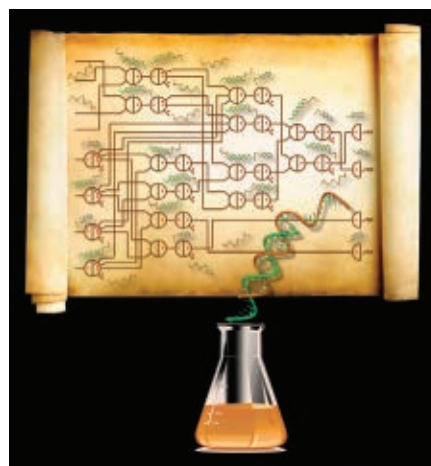
—Dennis McCafferty

The Rise of Molecular Machines

The field of molecular computing is achieving new levels of control over biochemical processes and fostering sophisticated connections between computer science and the biological sciences.

TAKING CUES FROM both speculative fiction and hard science, today's most prolific futurists have envisioned a point in the future when developments in genetics, nanotechnology, and robotics make it possible to sidestep the constraints of human durability and intelligence. Controversial assumptions notwithstanding, even the most optimistic speculation about the future symbiotic convergence of humans and technology is deriving at least some measure of credibility from emerging work in molecular computing. Researchers in this field are achieving new levels of control over biological processes and fostering sophisticated crossovers between computer science and the biological sciences.

In one recent development, scientists in the department of molecular computing at the California Institute of Technology (Caltech) have built what they are calling the most complex biochemical circuit ever created from scratch. These circuits, the Caltech researchers say, will allow scientists to explore the principles of information processing in biological systems and design biochemical pathways with



Wiring diagram specifying a biochemical circuit that consists of 74 different DNA molecules. The circuit, developed at Caltech, demonstrates an approach for implementing arbitrary digital logic in biochemical systems. The lines correspond to single-stranded oligonucleotides, while the nodes correspond to partially double-stranded molecules.

decision-making capabilities. Such circuits, they say, will give biochemists unprecedented control over chemical reactions for biological and chemical engineering and may even lead to the proliferation of molecular-scale biological machines.

Lulu Qian, a senior postdoctoral

scholar in bioengineering at Caltech, and Erik Winfree, a Caltech professor of bioengineering and computer science, computation, and neural systems, used DNA-based components to build the circuit. Instead of depending on electron flows through transistors, the DNA logic gates receive and produce molecules as their signals. The molecular signals travel from one gate to another, connecting the circuit as if the molecules were wires.

With his colleagues Georg Seelig, David Soloveichik, and David Zhang, Winfree first built a biochemical circuit in 2006. In that work, DNA signal molecules connected several DNA logic gates to each other, forming a multilayered circuit consisting of 12 molecules. In the new design, Qian and Winfree made the logic gates from pieces of single- and double-stranded DNA. The two researchers have made several circuits with this approach; the largest, containing 74 different DNA molecules, can compute the square root of any number up to 15 and round the answer down to the nearest integer.

During the calculation process, the custom-built molecules float around the solution and bump into each other, prompting strands with a certain DNA sequence to zip themselves to compatible strands while simultaneously unzipping other strands. The unzipped strands are released back into the solution to continue the cycle until the calculation process is complete. The researchers simply monitor the concentrations of output mole-



Heading to the instrument room at Caltech. In the box are the pipettes and test tubes containing molecules for a biochemical circuit: the 74 tubes each contain one type of DNA logic gate or fuel, and the eight tubes contain input signal strands. To get the DNA circuit running, the 74 gate and fuel molecules must be mixed to form the functional circuit, and a combination of the eight input signal strands must be added. Fluorescence must then be monitored during the next 10 hours to determine the circuit's output.

cules to determine the answer, which takes some 10 hours to compute.

While the logic gates have identical structures and can therefore be standardized, Winfree says the research still faces several significant challenges, such as automating the design and analysis process and finding ways to control the inevitable faulty behavior of the floating molecules. In addition, Winfree says it is not clear how well such molecular systems can be scaled to take on tasks more complex than relatively basic math. According to Winfree, the millions of logic gates that are common in silicon-based computing will not be possible for DNA circuits, at least in the near term.

Despite the evident limitations of DNA circuits, researchers working in this area are already achieving more control over biochemical processes than has ever before been possible. "We're trying to design DNA systems that self-assemble, implement biochemical circuitry, and act as molecular robots," says Winfree, who is modest in his assessment of the science and engineering involved in creating the molecular circuit with Qian. "It took previously demonstrated proofs of principle, simplified them, auto-

mated them, and got them to work at a much larger scale," he says. "Doing so, of course, involved some conceptual advances that might be considered a breakthrough."

To Winfree, the potential applications for such programmable biochemical systems are limitless. Eventually being able to use such systems for diagnosing disease or delivering custom-designed cellular therapies, for example, is a common refrain in discussions about the utility of molecular computing. "You wouldn't say that the purpose of electronic computers is to play video games and keep track of banks' financial records," he says. "The same goes for circuitry at the molecular scale; there are a lot of things you can do with molecules and chemistry."

DNA-based Data Storage

In another molecular-computing project, researchers at the Chinese University of Hong Kong (CUHK) are using DNA for data storage. The project, which recently won the gold medal at the International Genetically Engineered Machine (iGEM) competition at the Massachusetts Institute of Technology, is said to be the first to use the DNA of *E. coli* to store and en-

crypt text and images. The researchers working on the project, headed by Ting Fung Chan, estimate that one gram of bacteria can store up to 650,000 gigabytes, roughly equivalent to 325 hard drives, each with two terabytes of capacity.

Chan, a professor in the school of life sciences and the deputy director of the center for microbial genomics and proteomics at CUHK, explains that the idea for the project initially came from a conversation about *Assassin's Creed*, a video game in which the protagonist has inherited his ancestor's memory through DNA. In the original iGEM competition, Chan and his team encoded a short text message in a single piece of DNA and inserted it into bacterial cells, where the DNA was reshuffled, then retrieved and decoded.

Storing small bits of data, according to Chan, is not the project's ultimate goal. Currently, he and his team are designing a parallel storage system in which the DNA sequence would be cut into segments and stored in multiple cells. "We are pursuing a true massively parallel storage system," Chan says. "Storing a large piece of information, such as a photograph or a dictionary, is impossible to do within a single piece of DNA because of the limits associated with current DNA synthesis technology."

Simply fragmenting the information and inserting it into multiple cells without having an effective method for rebuilding the original file would destroy the data because the order of fragments would be unknown, explains Chan. So he and his team devised a biological system that uses standard storage mechanisms—such as headers and checksums—so each piece of information can be mapped and located for retrieval. The method involves removing DNA from the cells, altering it with enzymes, and returning it to new cells where the DNA sequence will be shuffled. "The idea is to demonstrate that we can, in principle, store any digital information," says Chan.

Chan notes that while he and his team have made progress at encoding pictures in several DNA sequences and storing them in parallel in multiple cells, research in this area has essentially just begun. He says it will

take at least another five to 10 years for the core ideas to develop. "Synthetic biology is an emerging field, and there is special research funding only in some parts of the world," says Chan. "This is multidisciplinary research, and training of a new generation of young researchers is very much needed."

While most research in molecular computing has focused on constructing small modules in which individual genetic units carry out a particular task, Chan says developing a complete machine will require genome-scale engineering that depends not on the individual building blocks, but on a deep understanding of how these individual building blocks interact with each other. To this end, he says, a key challenge to overcome in this area is to gain a much more complete understanding of individual cellular components so they can be accurately modeled as an integrated system.

Despite such challenges, the tools and techniques used by scientists working in molecular computing are beginning to mature, suggesting the possibility that some of these projects may soon form the groundwork for commercial applications. Caltech's Winfree, for his part, predicts the field will gain significant momentum during the next 10 years. "Molecular com-

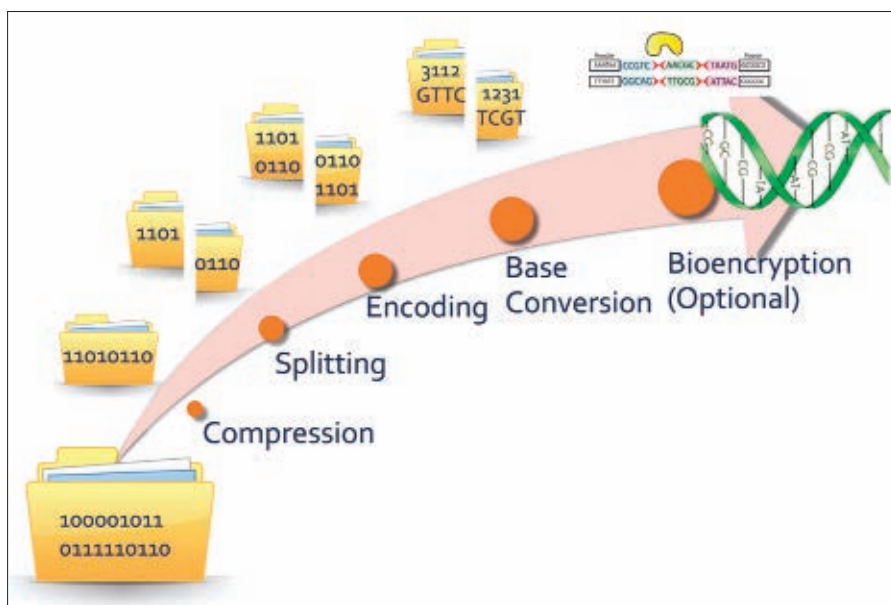
The largest circuit, which contains 74 different DNA molecules, can compute the square root of any number up to 15 and round the answer down to the nearest integer.

puting is beginning to be a real contender for some simple applications," he says, noting that every few years there is a breakthrough that changes what people inside and outside the field think is possible.

By way of example, he cites Paul Rothemund's DNA origami project, the goal of which was to fold DNA into tiny shapes and images. "Before DNA origami, most folks, including me, would not have thought it possible, and afterwards it was routine," says Winfree. "What's next is impossible to predict."

Still, as the pace of innovation in this area of research continues to accelerate, more crossovers between molecular computing and traditional computing will emerge. Many principles from electrical engineering and computer science are already being used to design molecular systems, as was done in the Caltech project, which used the principles of digital abstraction and signal restoration to make the circuits capable of withstanding imperfections, and in the CUHK project, which used traditional storage and cryptography paradigms to encode data in bacteria.

Winfree points out that if the pace of innovation in molecular computing continues, in 30 years scientists will be able to make systems built with 20,000,000 nucleotides, which is larger than *E. coli's* complete genome. "It's hard to imagine what such designed-from-scratch molecular systems will be capable of doing," says Winfree. While it remains to be seen whether such systems will eventually be able to augment human capabilities in a manner consistent with the visions of today's futurists, Chan, like Winfree, remains optimistic about the potential of molecular computing. "I do believe that this is no longer science fiction, as I would have thought when I was a kid," he says. ■



An overview of the biological data storage system developed at the Chinese University of Hong Kong. In the bacteria-based storage system, binary files are compressed and split into data packets. Each packet contains a payload, an address, error-correction code, and an optional encryption marker. A binary-to-quaternary base conversion is performed on the encoded data, followed by substituting the quaternary numbers for the four DNA bases.

Further Reading

Qian, L. and Winfree, E.
Scaling up digital circuit computation
with DNA strand displacement cascades,
Science 32, 6034, June 3, 2011.

Ran, T., Kaplan, S., and Shapiro, E.
Molecular implementation of simple logic
programs, *Nature Nanotechnology* 4, 10,
October 2009.

Rothemund, P.
Folding DNA to create nanoscale shapes
and patterns, *Nature* 440, 7083, March 16,
2006.

Storm, D.
Unhackable data in a box of bacteria: Future of InfoSec? *Computerworld*, January 18, 2011.

Zyga, L.
Biomolecular computer can autonomously
sense multiple signs of disease, PhysOrg.
com, July 6, 2011.

Kirk L. Kroeker works in communications and has written extensively about the impact of emerging technologies.

© 2011 ACM 0001-0782/11/12 \$10.00

Brave NUI World

Natural user interface developments, such as Microsoft's Kinect, may indicate the beginning of the end for the mouse.

COLORADO SPRINGS, CO.-BASED independent software developer Kevin Connolly became a minor YouTube celebrity over the past several months with the demonstration of his natural user interface (NUI) hack of Microsoft's Kinect software development kit (SDK). Connolly demonstrated moving different images on a small bank of screens up and down, in and out, and sorted through three-dimensional image arrays, all through gesture alone. It was, he notes, very similar to the image manipulation featured in Steven Spielberg's 2002 futuristic film, *Minority Report*.

"I'm just some guy," Connolly says. "I made that work in a matter of hours. Imagine what we have the technology to do if one guy in his apartment can do that in a few hours."

Indeed, Microsoft's decision to release the Kinect SDK in June has garnered much attention in the technical and technology trade press and among researchers and enthusiasts. Anoop Gupta, a Microsoft distinguished scientist, says more than 100,000 individuals downloaded the SDK in the first six weeks after its release. However, the terms of the release forbid any commercial use of the SDK, and Connolly says he halted work on his nascent NUI after he got it working to his satisfaction. Yet, the release of the SDK was a signal that low-cost motion and depth-sensing technology may soon herald epochal changes in the way humans and computers interact. The Kinect hardware, for instance, is manufactured by Tel Aviv-based PrimeSense, and lists for about \$200. Researchers in numerous disciplines call such technology for such a price absolutely revolutionary.

NUI Research

John Underkoffler's experience with NUI research goes far beyond hobbyist SDKs. In fact, Underkoffler is



Audrey Penven created this photograph and similar ones by using Kinect's infrared structured light as a source of light.

the man behind the *Minority Report* interface and has commercialized it via Los Angeles-based Oblong Industries, at which he is chief scientist. Underkoffler calls the Microsoft SDK release a "rhetorical event we all love." He says these events, like the 2006 release of Nintendo's Wii, "puts in the foreground for different sets of eyes—the end consumer for the Wii, the home or dorm room hacker for the Kinect SDK—the idea that it isn't going to be mouse and keyboard forever. We've seen the dialogue go from one of doubt or questioning to a kind of acceptance. Everyone now knows it isn't going to be mouse and keyboard forever, but the real question is, What is it going to be?"

Some of the next steps of NUI research are likely to be through gaming applications, which are increasingly dependent upon motion-sensing technology. Recently, four California Insti-

tute of Technology students undertook a study of motion-sensing technologies originating in gaming, and their possible uses in other computationally intensive fields. The students explored the depth camera technology in Kinect, the inertial sensors of the Wii, and electromagnetic sensing technology developed by Sixense Entertainment and Razer. One of the students, Peter Ngo, believes the idea of the NUI as a fully toolless interface is overstated, as does Amir Rubin, CEO of Sixense.

"People don't buy motion control," Rubin says. "They don't buy PCs. They don't buy consoles. They buy the experience being delivered to them. The best input device is something you don't remember is on you, and that is true even for the slogan sold by Microsoft—'You are the controller.' I agree, but if I have to alter the way I use my body in order to interact with Kinect, then it's the worst controller."

Even the most ardent NUI advocates agree with Rubin. Oblong's Underkoffler, for instance, says that writers will likely be well served by the keyboard for the foreseeable future, but that those who design ship hulls or airplane wings would be better served by three-dimensional NUIs. Moreover, he says it is vital not to graft a notion of a new interface design by simply extending two-dimensional GUI concepts onto the prototypes of three-dimensional applications. These applications will need computational capabilities along not just the flat x and y axes—an example might be a wall-sized but still two-dimensional application for designing the very three-dimensional ship's hull he mentioned—but will also need to compute the depth of the z axis. Oblong's g-speak platform, based on work Underkoffler pioneered in the 1990s at the Massachusetts Institute of Technology's Media Lab, computes this spatial environment via networked computers and screens that allow rich three-dimensional interaction. Ultimately, Underkoffler thinks a hybrid UI ecosystem will evolve.

"We're not out to replace the keyboard; let it do what it's best at," he says, "but when it comes to designing airplane wings, you do need a spatial UI. So, it's about situating the right activities in the right interaction zone."

Homebrewed Algorithms

Since the Microsoft SDK precludes commercial use, many early academic and enterprise projects using PrimeSense and/or stripped down Kinect hardware use either homebrewed algorithms or open source drivers and middleware released by consortia such as OpenCV or OpenNI, the natural interface forge formed in November 2010, by PrimeSense and robotics pioneer Willow Garage. OpenNI leverages the PrimeSense depth-sensing technology, which is processed in parallel by PrimeSense's system-on-a-chip processor after receiving coded near-infrared light from its partnered CMOS sensor.

OpenNI supplies a set of APIs to be implemented by the sensor devices, and a set of APIs to be implemented by the middleware components. Thus, OpenNI's API enables applications to be written and ported to operate on

"People don't buy motion control," Amir Rubin says. "They buy the experience being delivered to them. The best input device is something you don't remember is on you."

top of different middleware modules. It also enables middleware developers to write algorithms on top of raw data formats, regardless of which sensor device has produced them, and offers sensor manufacturers the capability to build sensors that power any OpenNI-compliant application.

In fact, one nascent healthcare industry application partially built on open source stacks by a team of surgeons and engineers at Sunnybrook Health Sciences Center in Toronto for the Kinect camera is already drawing attention.

Allowing surgeons access to medical images while not having to touch a controller—and thereby saving them the necessity to re-scrub in order to preserve sterility around the patient—is an early enterprise triumph for the NUI concept. Computer vision specialist Jamie Tremaine says the gesture-based UI he and his colleagues developed has proven exceptionally robust and enables surgeons to view through MRI and CT scan samples that can run from 4,000 to 10,000 slides without ever having to re-scrub.

For such an application, the hand and arm gestures recognized by the Kinect camera are suitable, but Tremaine says "a lot of the work we've done hasn't even been on the technical side as much as creating gestures in the operating room that allow very fine-grained control, but which have to be larger."

Another NUI developer, Evan Lang of Seattle-based UI design firm IdentityMine, says his work with trying to develop Kinect NUIs (on PrimeSense drivers) similar to current GUI commands revealed vexing user issues. In developing a Web button, for instance, Lang says, "I programmed it to recognize a poking gesture, where you move your hand quickly forward and quickly back. When I got some test users to try it out, and said 'Poke it or press it,' everybody had a very different idea of what that actually meant. Some did a kind of poking thing. Other people moved their hand forward but wouldn't move it back, and others, who were very cautious and deliberate about it, the machine wouldn't register as a poke."

Oblong's Underkoffler says problems such as Lang encountered are emblematic of grafting current GUI-based mechanics on an idea that needs something else.

"We believe it's not appropriate to start talking about NUIs until you have a complete solution," Underkoffler says. "If you flash back 30 years, it's like dropping an early prototype of a mouse in everybody's lap and saying, 'We have a new interface.' You don't, because you've just got a new input device. So, really it's a full loop proposition. What's the input modality? What shows up on screen? What's the analogue of the windows and scroll bars and radio buttons? Until that's not only been answered in a way to allow real work to happen, but has become kind of a standard, and more in the cognitive sense, recognizably and pervasively present, then you don't have a new interface."

Sixense's Rubin predicts the next-generation standard UI device would not be a question of which technology is most elegant, but rather, that which meets three criteria: a consumer-friendly price, an intuitive UI design, and ease of software development on top of that device.

"If you can meet the combination of those three," he says, "then you will have the next-generation standard of input devices."

Output Perceptions

Robotics researchers such as Nicholas Roy, associate professor of aeronau-

Allowing surgeons access to medical images while not having to touch a controller—and thereby saving them the need to re-scrub in order to preserve sterility around the patient—is an early enterprise triumph for the NUI concept.

tics at the Massachusetts Institute of Technology, were among the first to adopt the PrimeSense sensor, and it may be their work that shows the longest-range potential for, and a new concept of, what a NUI will be.

Vehicles equipped with the three-dimensional sensors—among them a helicopter Roy and his students programmed—gain what Roy calls a “whole new sense of the human-centered environment,” and are able to sense things such as drop-offs in stairs, table legs, and so on. And now, with more researchers exploiting the extremely cheap sensor technology, it is likely that more UI work will be done exploring how a robot, or any other computer, will “think” its way to interacting with humans.

“A lot of the estimation and planning algorithms my students have developed for the helicopter, we actually reused in the context of interacting with robots,” Roy says. “If the problem is no longer how the vehicle plans to get from Point A to Point B, but the problem is how does the vehicle understand what the human wants in terms of some task, then our research programs have a lot of commonality between those two seemingly very different domains.”

Further Reading

Cohn, G., Morris, D., Patel, S.N., and Tan, D.S.
Your noise is my command: Sensing

gestures using the body as an antenna, *Proceedings of the 2011 Annual Conference on human factors in computing systems*, Vancouver, British Columbia, Canada, May 7–12, 2011.

Gallo, L., Placitelli, A.P., and Ciampi, M.
Controller-free exploration of medical image data: Experiencing the Kinect, *Proceedings of the 24th IEEE International Symposium on Computer-Based Medical Systems*, Los Alamitos, CA, June 27–30, 2011.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D.

RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments, *Proceedings of the International Symposium on Experimental Robotics*, New Delhi, India, Dec. 18–21, 2010.

Shotton, J., et al.

Real-time human pose recognition in parts from single depth images, *Proceedings of the 24th IEEE Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 20–25, 2011.

Underkoffler, J., Ullmer, B., and Ishii, H.
Emancipated pixels: Real-world graphics in the luminous room, *Proceedings of the Special Interest Group on Computer Graphics*, Los Angeles, CA, August 8–13, 1999.

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

© 2011 ACM 0001-0782/11/12 \$10.00

Milestones

Katayanagi Prizes and Other CS Awards

President Obama, ACM, IEEE, Carnegie Mellon University, and Tokyo University of Technology recently honored some of the nation's leading computer scientists.

NATIONAL MEDALS OF SCIENCE, TECHNOLOGY, AND INNOVATION

President Obama honored Richard A. Tapia, director of the Center for Excellence and Equity in Education at Rice University, with the 2010 National Medal of Science for “his pioneering and fundamental contributions in optimization theory and numerical analysis and for his dedication and sustained efforts in fostering diversity and excellence in mathematics and science education.”

The President also presented National Medals of Technology and Innovation, recognizing scientists who have made lasting

contributions to America's competitiveness and quality of life and helped strengthen the nation's technological work force. The recipients were Rakesh Agrawal, Purdue University; B. Jayant Baliga, North Carolina State University; C. Donald Bateman, Honeywell; Yvonne C. Brill, RCA Astro Electronics (retired); and Michael F. Tompsett, TheraManager.

KATAYANAGI PRIZES

Carnegie Mellon University in cooperation with the Tokyo University of Technology presented the Katayanagi Prize for Research Excellence to Barbara Liskov, Institute Professor at the Massachusetts Institute of Technology, for her record of outstanding, sustained achievement. Scott Klemmer, associate professor of computer science at Stanford University, received the Katayanagi

Emerging Leadership Prize for his demonstration of leadership promise.

ACM AND IEEE AWARDS

Susan L. Graham, the Pehong Chen Distinguished Professor of Electrical Engineering and Computer Science Emerita of the University of California Berkeley, received the ACM-IEEE Computer Society Ken Kennedy Award “for contributions to computer programming tools that have significantly advanced software development.” Cleve Moler, chairman of MathWorks, was this year's winner of the IEEE Computer Society Sidney Fernbach Award “for fundamental contributions to linear algebra, mathematical software, and enabling tools for computational science.” Charles Seitz, former president of Myricom, Inc., was awarded the IEEE Computer Society's 2011 Seymour Cray

Computer Engineering Award “for innovations in high-performance message-passing architectures and networks.”

SIGSAC AWARDS

ACM's Special Interest Group on Security, Audit and Control (SIGSAC) presented its top honors to Virgil Gligor, co-director of Carnegie Mellon University's CyLab, and Ravishankar Iyer, director of the Center for Reliable and High-Performance Computing at the University of Illinois. Gligor received the SIGSAC Outstanding Innovation Award for innovations in secure operating systems as well as covert channel analysis, intrusion detection, and secure wireless sensor networks. Iyer received the SIGSAC Outstanding Contributions Award for his fundamental contributions to the assessment and design of secure, dependable computing systems.

Activism Vs. Slacktivism

Today's activists are highly plugged into social media, mobile apps, and other digital tools. But does this make a difference where it matters most?

IF YOU NEED convincing that the state of activism in the digital age is alive and well, look no further than the Web site for the Program on Liberation Technology at Stanford University.

On the program's high-profile email list group, a consumer advocate gives updates about the California Public Utilities Commission's investigation into a proposed merger of AT&T and T-Mobile. Another user promotes a letter-writing campaign to urge federal lawmakers to protect funding for the Directorate for Social, Behavioral and Economic Sciences at the National Science Foundation. And a third offers cautionary advice to fellow organizers: "Don't type anything you wouldn't want snooped on your iPad. Someone has developed software which uses computer vision to do keylogging."

Other postings focus on the Arab spring, environmental sustainability, and a host of other progressive causes, which is understandable since the Stanford program's stated purpose is "to understand how information technology can be used to defend human rights, improve governance, empower

No one disputes that activists' online efforts draw greater attention to a cause, but opinion varies with respect to whether they make a significant, lasting impact.



Engaged or disengaged? A pair of protesters with their smartphones at an anti-Al Khalifa protest last February in London, England.

the poor, promote economic development, and pursue a variety of other social goods."

Of course, there's plenty to find on the right-leaning side of the ideological table. At TeaPartyPatriots.org, for example, you can use a locator to track down events scheduled in your city or state, buy a Tea Party Patriots coloring book, and join a Government Accountability Project group.

The upshot is no matter what your cause is, you can find a great way to connect these days. Activists are making full use of blogs, social media sites, mobile apps, and other tools to promote their message and gain support. Nothing grabs the heartstrings like video, and participants are producing streaming content to take advantage of this. It makes one think of how effective technology could have been through history. Consider how the U.S. founding fathers would have tweeted Paul Revere's famous cry as "Brits R Coming," post real-time video of his nighttime ride on Facebook, and solicit the French and other sympathetic European supporters for financial and

participatory support through Facebook, Kickstarter, and other sites.

Yet, while no one disputes that online initiatives like these draw greater attention to a cause, opinion varies with respect to whether they make a significant, lasting impact. A number of respected thinkers say technology does not really advance activism to achieve its most critical goals: to change the hearts and minds of the public, and effect real change.

On the other side of the debate are activists and other influencers who counter that the impact on hearts and minds cannot be measured. What can be measured are user-traffic numbers generated, e-petition signatures delivered, Facebook "like" counts, and other metrics that convey growing support.

A Contrarian View

The conversation here is essentially positioned as a debate over activism versus slacktivism. The latter term refers to people who are happy to click a "like" button about a cause and may make other nominal, supportive ges-

tures. But they're hardly inspired with the kind of emotional fire that forces a shift in public perception. A telling, supportive anecdote: A popular technique of organizers on all sides of the political spectrum is an online letter-writing campaign in which supporters are encouraged to simply copy and paste from a template form of the letter. Participants aren't asked to come up with their own words. It's not even clear if they read the entire content of the letters they send. Does a simple "copy/paste/send" act constitute activism at its finest?

In one of the more widely discussed articles casting doubt, *New Yorker* contributor Malcolm Gladwell maintains that successful efforts must engage participants by convincing them that they have a great personal stake in the consequences. Traditionally, highly effective movements evolved from within parties built upon "strong tie" personal connections, such as those among classmates and church members. Activism associated with social media, however, is dependent upon "weak tie" relationships, writes Gladwell. Organizers seek involvement from Twitter followers they have never met or Facebook friends with whom they would never otherwise stay in touch, according to Gladwell. These are loose networks, whereas meaningful activism requires strong, robust organizational structure.

Even in the case of the Arab spring—arguably the political movement most enhanced by multiple digital means—those casting doubt upon the influence of technology contend that the events would have mattered little if old-fashioned principles of activism were not applied: effectively planned mass assemblies in which passionate pleas for change were expressed. The fact that the Arab spring demonstrations got YouTube'd, Facebooked, and tweeted is simply a logical progression in the continuing advancement of multimedia, just as broadcasting civil rights demonstrations on TV news during the 1960s at one time seemed novel in its ability to connect a cause with a nationwide audience.

In the end, activism has always been—and will always be—about people. Specifically, people who show up in person. Just witness the pro-



A protester captures the scene at an Occupy Portland rally in Portland, OR last October.

tests over collective-bargaining rights for state union employees in Wisconsin, as the liberal public-policy group MoveOn.org led a solidarity day in which 50,000 supporters turned out in all 49 other state capitals and raised more than \$3 million to support Wisconsin Democrats.

"The Wisconsin protest was old-school organizing, with a digital edge," says Dave Karpf, an assistant professor in communications/information at Rutgers University and a leading researcher on political blogs and Internet-mediated activist organizations. "Angry citizens felt their rights were being trampled, so they showed up and demonstrated. It was the largest extended labor action in a generation, and it was led by labor organizations, fighting for collective bargaining rights."

Similarly, the Tea Party isn't a new social movement either, according to Karpf. It's traditional conservatism that intelligently embraces new-media technologies. "The Tea Party's biggest successes—disrupting health-care town hall meetings, winning Republican primaries—were a boots-on-the-ground affair, with people arriving and causing a ruckus," says Karpf. "Web sites and Twitter were useful in helping activists identify those meetings more easily. But they're basically acting as

much more efficient phone trees."

Some of those downplaying the impact of online activism will even argue that its ability to generate "boots-on-the-ground" user engagement is overstated. Tufts University sociology professor Sarah Sobieraj likens modern efforts as more of an infatuation with technology with little to show for it. For her book, *Soundbitten: The Perils of Media-Centered Political Activism*, Sobieraj researched the methods of more than 50 different groups focused on shaping discourse—including United for Change, Pre-Born Protectors, and the Freedom and Equality League—and concluded that their Internet strategies have done little to influence the public.

Perhaps the greatest irony? As much as these groups enjoy beating up the mainstream media or claim that their use of new media is infinitely more effective than traditional media, these same groups covet coverage from major journalism outlets. "They're very old-media-centric," Sobieraj says. "When they talk about strategies, they're most focused on broadcast TV and even newspapers. If they get mentioned in a *New York Times* or *Boston Globe* feature, that's what they're really after."

Committed to Tech

People both involved with and supportive of online activism concede that they really cannot measure how much technology inspires people to "do something." But they say any kind of attention generated—either by mainstream press or otherwise—increases the opportunity to change minds and instigate action. The Internet has established platform upon platform to present a position in multiple formats. It allows for the exchange of views on a said position. It increases the capability for calls to action and pure organizational logistics. In other words, if the new techniques of activism serve to amplify and even help better organize the old, what is wrong with that?

Besides, technology and activism are a perfect match, says Brie Rogers Lowery, a contributing strategist for FairSay, an eCampaign consultancy. The very founding principle of Web 2.0 itself is based upon the same ideas that fuel efforts toward change. Those

principles include the need to interact, share, and pursue goals.

"Technology offers huge potential to connect," says Rogers Lowery. "An obvious example is Obama's election campaign, which was mobilized primarily online and utilized the full range of new media. But the use of technology in activism extends to all kinds of campaigns, such as the use of SMS in South Africa to report cases of child abuse in remote communities."

Rogers Lowery, who organized a digital activism debate at Oxford University earlier this year, says it is time to move the discussion from the "cyber-skeptic view" that online activism is somehow less legitimate and inferior to older approaches. "Instead," she says, "there's a need to show how 'old' and 'new' activism can work together to serve."

It is not simply a matter of using technology in greater numbers. It is about everyday citizens finding creative ways to exploit it in ways previously not conceived to advance a cause, supporters say. Pachube.com, for example, links activists to data tools that can help establish, manage, and share the quantified basis of their positions.

"We've had a Brooklyn user who built an alert system to help monitor

The Tea Party isn't a new social movement, says Dave Karpf. It's traditional conservatism that intelligently uses new-media technologies.

sewage and other contaminants in an effort to get citizens to keep the harbor cleaner," says Ed Borden, who oversees technology and business development for Pachube. "We have another New Yorker who's collecting data to support his contention of noise pollution created by the Federal Aviation Administration. In Japan, the citizens crowdsourced to come up with radiation data after the Fukushima disaster in March, self-organizing

using Twitter, blogs, and wikis. Data drives activism. The dialogue has reached a deafening point online and everyone has a cause. So it takes hard evidence to turn heads."

Whether those heads remain turned—and join the cause—is subject to continued debate. 

Further Reading

Durbin, P.T.

Philosophy, activism, and computer and information specialists, *Ubiquity*, November 2007.

GetInvolved.ca

Social Media: Politics 2.0—The Power of the Citizen, <http://www.youtube.com/watch?v=1vrczoLm7Es&feature=autoplay&list=PLE8382F8E085EFF12&index=3&playnext=2>, Jan. 21, 2010.

Gladwell, M.

Small change: Why the revolution will not be tweeted, *The New Yorker*, Oct. 4, 2010.

Karpf, D.

Wisconsin and the limits of web power, *The Guardian*, Feb. 25, 2011.

Land, M.B.

Networked Activism, *Harvard Human Rights Journal* 22, 9/10, Sept. 28, 2009.

Dennis McCafferty is a Washington, D.C.-based technology writer.

© 2011 ACM 0001-0782/11/12 \$10.00

Technology

Low-Cost Robots Could Transform Science

A new generation of inexpensive robots could make the machines ubiquitous, opening up robotics to new areas of research, says James McLurkin, assistant professor of computer science and director of the robotics lab at Rice University.

"I wanted to have something the community could use to do research," McLurkin says. "In order for this to have an impact, it has to be low cost."

McLurkin studies multi-robot systems in which swarms of robots work together to perform a task, like searching a building for earthquake survivors. Much of the work in such systems has been done through computer simulations, because building many robots is too expensive. But now McLurkin has built a robot for about \$280, compared to \$2,000 for the previous version. He

is hoping for funding to allow him to sell his R-one machines to researchers and educators at cost.

"I think what he's doing is great," says Rodney Brooks, professor emeritus of robotics at the Massachusetts Institute of Technology, and McLurkin's undergraduate advisor. Brooks thinks cheap robots could have the same effect on his field that moving from expensive mainframes to desktops had in computing. "Every student having a robot, and then being able to get them to work together, will unleash creativity on the physical world in the way that the PC did on the cyberworld."

The robots are inexpensive mainly because the spread of smartphones has driven down the cost of sophisticated electronics.

They contain integrated radio communications, infrared sensors, motors, and an embedded Python interpreter for programming.

McLurkin is interested in physical data structures, using robots as elements in an algorithm. A robot's position in space can be a unit of information that can be manipulated by moving it around or keeping it in a particular orientation. A simple bubble sort algorithm, which sorts a list into the right order, can be rendered physical with robots. "Their position in the world indicates the state of the sort," he explains.

This approach could provide a new way of thinking about the behavior of multi-robot systems. Instead of modeling the individual motions of hundreds or

thousands of robots, a daunting computational task, he hopes a physical algorithm of a handful of robots can act as an accurate representation of a larger group. That in turn will let him write new algorithms so the swarms can perform complex tasks.

Going from a handful to a large number of robots can actually transform a problem, he says. Instead of four robots wandering through a building with laser scanners to make a map, for instance, he could send in hundreds and make a map simply by noting the positions of the machines.

Having many affordable robots will let him test his ideas. "Until you put your robots where your mouth is, you really don't know if you've got something," McLurkin says.

—Neil Savage

CSEdWeek Takes Hold

Groups in more than 130 countries will participate in Computer Science Education Week this year.

IT IS NOTHING short of ironic that in the digital age, instruction about computers and computing is woefully lacking. A 2010 ACM report, *Running on Empty*, found that only nine states in the U.S. count computer science courses as a core academic subject in high school graduation requirements and the total number of courses offered by secondary schools has declined over the last several years. Yet, by 2018, a projected 1.4 million new computing jobs will exist and the current pipeline of graduates will fill only about half of these positions.

"In some schools where AP computer science was once taught, classes have been eliminated," says Debra Richardson, a professor of informatics at University of California, Irvine. "In others, real computer science has never been taught and what is called computing or computer science is just literacy in technology and applications. The difference is whether you understand how to create computing technology or are just able to use it."

As a result, computing scientists, educators, and others are banding together to raise awareness about the impact of computing in today's society. CSEdWeek, which originated in 2009, focuses on how computer science education prepares today's youth for the digital age. The December 4–10 event features programs at businesses, universities, and K–12 schools that are designed to stimulate interest in computing sciences and show the viability of careers in the field.

According to Richardson, who chairs CSEdWeek, the U.S. and other countries are falling further behind the computing curve. From 2005 to 2009, the percentage of U.S. high schools offering classes in computing sciences has fallen from 40% to 27%. In addition, only 17% of those taking advanced placement computing science tests are women and 11% are minorities.



MIT's Leah Buechley engages students with her method of creating cloth printed circuit boards in the form of wallpaper.

The fallout is significant, says Ruthe Farmer, director of the National Center for Women & Information Technology and vice chair for CSEdWeek. Businesses and other institutions consistently lose out on talent as individuals that could find work as computer engineers, designers, and developers stream into other fields. The lack of women and minorities compounds the problem—particularly as organizations focus on designing better products and solutions across a diverse group of consumers.

CSEdWeek aims to address these gaps and social inequities. The organization has asked individuals from around the world to pledge support and develop an educational activity or program in their community. Groups from more than 130 countries, including Brazil, India, and Kenya, are now involved in the initiative.

At the University of California, Berkeley, more than 50 high school students visited the campus in 2010 to learn about robotics, animation, artificial intelligence, game analysis, and other topics. "It's important to get students exposed to computing sciences at a young age," says Joshua Paley, a computer science and mathematics

teacher at Henry M. Gunn High School in Palo Alto, CA. Last year, he led the visit to Berkeley and helped develop a programming contest that attracted nearly 50 students. "It offered a window into computer science puzzles and problems," he explains.

At the University of Puerto Rico, Mayaguez Campus, a group of professors and students focused on the theme "Our Lives Without Computer Science." They created an award-winning video as well as a demonstration, with both hardware and software, of the classic Simon Says game. School children could play with the hardware, edit the software, and learn about how everything interconnects. "It's powerful because they can see themselves as future computer scientists," says Nayda G. Santiago, an associate professor in the school's electrical and computer engineering department.

"Schools must move beyond basic technology literacy curriculum and add courses that warm students up to computer science," concludes Richardson. "The future depends on it." ■

Samuel Greengard is an author and journalist based in West Linn, OR.

© 2011 ACM 0001-0782/11/12 \$10.00

PHOTOGRAPH COURTESY OF LEAH BUECHLEY, MIT

Dennis Ritchie, 1941–2011

Colleagues recall the creator of C and codeveloper Unix, an unassuming but brilliant man who enjoyed playing practical jokes on his coworkers.

OF THE THREE giants in the computer industry who passed away last October, Steve Jobs was easily the most recognizable one. And that is exactly how Dennis Ritchie preferred it.

Even though much of today's digital world is built from tools he created, Ritchie, who authored the C programming language and cocreated Unix with Ken Thompson, never sought the spotlight.

Brian Kernighan, who worked at Bell Labs alongside Ritchie and Thompson for more than 30 years and is now a computer science professor at Princeton University, observes, "Jobs was very out in public, which was one of his strengths. Dennis was a private person and didn't do any self-salesmanship. But the work Jobs did at NeXT and Apple built on what Dennis did because all those programs are fundamentally written in C or derivatives like C++ and Java. Life would be very different without the work Dennis did singlehandedly in just a few months."

C might be Ritchie's crowning achievement as it is regarded as one of the world's two most influential programming languages (the other is Fortran). C, of course, is not a very large language, mainly because the DEC PDP-11 minicomputer Ritchie ran it on was technologically constrained, so there wasn't much room to get fancy, which, Kernighan notes, was fine given Ritchie's minimalistic approach.

"Dennis and Ken worked together on Unix," says Kernighan, for which the duo received the ACM A.M. Turing Award in 1983. "He always said Ken did most of the work with just some of his assistance, but that's characteristically modest on Dennis' part."

Last May, Bell Labs hosted a ceremony in Murray Hill, NJ, in honor of Ritchie and Thompson who had won the Japan Prize. One speaker was



Ken Thompson (left) and Dennis Ritchie received the National Medal of Technology in 1999 from President Clinton.

Douglas McIlroy, an adjunct professor of computer science at Dartmouth College, who had been a manager at Bell Labs and knew Ritchie for nearly 50 years ever since Ritchie's first summer job there in 1962.

"Dennis was a fixture at meetings of the Usenix users group," McIlroy noted. "Crowds networking in the corridors would break to pack his talks. Every newcomer wanted to see and hear the man behind the system. Old hands came to listen to the master perhaps even more eagerly. If you read one of his papers, you'll see why. Dennis combines a perfect control of the technical matter with a polished but easy writing style, and an unerring sense of how much to say. That felicity is also on display on his home page, which offers engaging pieces about his work."

But not everything on his Bell Labs home page relates to work. A practical joker, Ritchie also details "Labscam," an elaborate prank that he and colleague Rob Pike pulled on their boss, Nobel prize winner Arno Penzias, in

1989 with the help of magicians Penn and Teller. [See the prank at <http://www.youtube.com/watch?v=fxMKuv0A6z4>.]

His sense of humor also shows in his work. "In perhaps the trickiest part of the Unix code," notes McIlroy, "where a couple of instructions play with hardware registers as if by magic, there is a comment by Dennis that says: 'You are not expected to understand this.' That's been published over and over again on T-shirts."

Ritchie was 70 when he was found dead in his Berkeley Heights, NJ, home. He had been in frail health in recent years after treatment for prostate cancer and heart disease.

"Dennis was thoughtful, he was totally approachable," says McIlroy, "but I think he will best be remembered as an extremely talented, bright guy who created something we absolutely all use—and he never really sought credit for it."

Paul Hyman is a science and technology writer based in Great Neck, NY.

© 2011 ACM 0001-0782/11/12 \$10.00

The Most Ancient Marketing

BEFORE APPLE, STEVE JOBS famously went to India with his college friend Dan Kottke. While I never had occasion to talk to Jobs about it, I did hear many a tale from Kottke, and I have a theory I wish I had a chance to try out on Jobs.

Jobs loved the Beatles and referred to them fairly often, so I'll use some Beatles references. When John Lennon was a boy, he once recalled seeing Elvis in a movie and suddenly thought to himself, "I want that job!" The theory is that Jobs saw gurus in India, focal points of love and respect, surrounded by devotees, and he similarly thought to himself, "I want that job!"

This observation is not meant as a criticism, and certainly not as an insult. It simply provides an explanatory framework for what made Jobs a unique figure.

For instance, he liberally used the guru's tactic of treating certain devotees badly from time to time as a way of making them more devoted. I heard members of the original Macintosh team confess that they succumbed. They were tangibly stunned by it, repeatedly. They recognized it happening in real time, and yet they consented. Jobs would scold and humiliate people and somehow elicit an ever more intense determination to attempt to win his approval, or more precisely, his pleasure.

Jobs imported the marketing techniques of India's gurus to the business of computation.

The process is described in an essay by Alan Watts on how to be a guru that was well known around the time Apple was first taking off. The successful guru is neither universally nor arbitrarily scornful to followers, but there should be enough randomness to keep them guessing and off guard. When praise comes, it should be utterly piercing and luminous, so as to make the recipient feel as though they've never known love before that moment.

Apple's relationship with its customers often followed a similar course. There would be a pandemic of bleating about a problem, such as a phone that lost calls when touched a certain way, and somehow the strife seemed to further cement customer devotion instead of driving them away. What other tech company has experienced such a thing? Jobs imported the marketing techniques of India's gurus to the business of computation.

Another way in which Jobs emulated the practices of gurus is in the psychology of pseudo-asceticism.

Consider the way he used physical spaces. Jobs always created personal and work spaces that were spare like an ashram, but it is the white Apple store interior that most recalls the ashram. White conveys purity, a holy place beyond reproach. At the same time, the white space must be highly structured and formal. There must be a tangible aura of discipline and adherence to the master's plan.

The glass exteriors and staircases of elite Apple stores go further. They are temples, and I imagine they might someday be repurposed for use along those lines. (Maybe, some decades from now, our home 3D printers will just pop out the latest gadgets, leaving stores empty.)

There is yet another Beatles reference to bring up: It was Yoko Ono who first painted a New York City artist's loft white. Conceptual avant-garde art invites people to project whatever they will project into it, and yet the artist offering a white space, or the silence of John Cage's "4'33'" still becomes well known. This is the template followed by Apple marketing.

A dual message is conveyed. The white void is empty, awaiting you and almost anything you project into it. The exception is the surrounding institution—the business—which is not



something to be projected away.

While that setup might seem to only benefit the establishment offering the white space, there is actually a benefit to the visitor who projects what they will into it. It's like a good parent or lover who will listen endlessly without complaint but also sets boundaries. Narcissism can then be indulged without the terror of being out of touch or out of control. This formula is a magnet for human longings.

It's all about you, iThis and iThat, but we will hold you, so you won't screw yourself up. Of course, that's not really a possible bargain. To the degree you buy into the ashram, you do give up a certain degree of yourself. Maybe that's not a bad thing. It's like how Apple customers experience culture in general through the lens of Apple curation whenever they use a tablet. Maybe it's the right mix for some people. But one ought to be aware.

It's tempting to ridicule this aspect of Job's legacy, but everything people do is infused with some degree of duplicity. This is doubly true of marketing.

Putting the duplicity up front might be best. Back to the Beatles:

Lennon's "Sexy Sadie" ridiculed the guru shtick, while McCartney's "Fool on the Hill" praised it, and they were singing about the same guru. These two songs could well be applied to the appeal of Apple under Jobs. Yes, he manipulated people and was often not a nice guy, and yet he also did either elicit or anticipate the passions of his devotees, over and over. (No one can say what the mix of eliciting versus anticipating really was.)

Jobs didn't just use pseudo-asceticism for marketing. He wielded purist fanaticism so as to have power in the world of nerds. This is how it came to be that Jobs is so often remembered as an "inventor," though he rarely was one. His genius was not technical, but he was a genius at manipulating technical minds.

An example is Jobs' obsession with engineering beautiful fonts into personal computers. While plenty of people wanted this (Don Knuth comes to mind), it wasn't easy to make such a luxury into a high-priority item in the engineering culture that drove early PC companies. But Jobs often mentioned his pride at having done it.

It is perhaps surprising that so few figures in tech companies have been able to push engineers around enough to enforce principles of elegance and simplicity, as understood by non-engineers. Apple's commercial success has created a better atmosphere for such things in all the companies. But how did Jobs do it in the first place?

My impression, based on a number of interactions I witnessed over many years, is that Jobs traded one form of obsessive, principled nerdiness against another. It was useless for a typical designer or marketing person to plead with engineers during the early years of personal computers. Engineers had airtight criteria and data, and that trumped mere opinions and intuitions. But Jobs didn't plead. He declared even more rigid and exacting criteria.

Jobs won the arms race of control freakery. He remains the only figure in a non-engineering role I have ever seen win this race against engineers outright. ■

Jaron Lanier is Partner Architect, Microsoft Research, and Innovator in Residence, USC Annenberg School.

Copyright held by author.

Life, Death, and the iPad: Cultural Symbols and Steve Jobs

IN 1997, STEVE JOBS rejoined the Apple he had left years earlier. For the next 14 years, he led that company to create technology that found a global audience. Indeed, the period of his leadership coincided with astonishing changes in the profile of technology users. In 1998, more than 75% of the world's Internet users were in the U.S., today it is less than 15%. The complexion of the Web—its users, their desires, their languages, points of entry and experiences—has subtly and not so subtly changed over that period. All these new online participants bring with them potentially different conceptual models of information, knowledge, and knowledge systems with profound consequences for the ideological basis of the Net. These new participants also operate within different regulatory and legislative regimes, which will bring markedly different ideas about how to shape what happens online. And in this same time period, the number and kind of digital devices in people's lives has grown and changed. Devices have proliferated with ensembles and debris collecting in the bottom of backpacks, on the dashboards of dusty trucks, and in drawers, cabinets, and baskets.

Many of those devices owe their contours, if not their direct production, to Apple and Steve Jobs. In the days that followed Steve Jobs' death, he was frequently compared to Henry Ford and Thomas Edison, inventors both and men who helped shape the American landscape. But Jobs was an inventor who came of age in a very different world, one where it was possi-



In traditional Chinese culture, people burn paper offerings for their ancestors. In a funerary goods store in Singapore, Genevieve Bell bought the last two paper iPads—complete with paper travel case and finger smudges—one of the most popular items for the hereafter.

ble to make products that would find their place in living rooms from Bakersfield to Beijing and many points in between. Furthermore, Jobs was much clearer than Ford or Edison that he was creating experiences, not technologies or products. He, and Apple, were creating a new symbolic register in which we all might participate, even if we all didn't purchase.

As an anthropologist, I am always interested in this notion of symbols, and I have collected all manner of things along the way. My fondest collection, by far, is that of paper offerings from Chinese funerary goods stores. In traditional Chinese culture, people burn paper offerings for gods, ghosts, and ancestors. There are paper objects for

religious occasions, for festivals, for ceremonial events—part of both public worship and private devotion. In this cosmology, or world view, fire transforms all these paper objects into real things in the other world. At funerals, and during Qingming—a yearly festival at which ancestors and family are honored—you burn paper money, paper gold nuggets, paper clothes, paper cars, paper cigarettes, paper beer, paper pork buns, paper false teeth; and even a range of everyday household items. In Shanghai in the 1930s, wealthy families burned full-sized paper copies of Rolls Royce and Bentley cars to ensure their ancestors had appropriate transportation. Family obligations, in this world view, do not end with death. Instead, as

a good child, you take care of your family and see to their needs in perpetuity. Today, in many Chinese communities around the world, a range of newer information and communication technologies have been added to the pyre of paper goods.

And over the years, in funerary goods stores in Penang, Ipoh, Kuala Lumpur, Singapore, Hong Kong, and even New York City, I have found all manner of paper technologies—a desktop PC with a Windows operating system, USB ports, and a mouse; a flat panel LCD TV screen with a remote control and HDMI outputs; game consoles with all the buttons and hints of small blinking lights; and a branded mobile phone, prepaid phone cards, a charger and a carrying case. The products are always subtly rebranded—Nokia, Panasonic—and the logos are tweaked, but they are recognizable technology.

On my first field trip years ago, as I bought my first paper Nokia candy-bar phone, I had to ask the question: “Who are the ancestors calling, are they calling you?” This question made one woman laugh out loud. “No,” she said emphatically, “don’t be foolish; they are calling each other.” At least one other family I spoke with told me of upgrading their ancestors’ mobile phones every year during Qingming—burning them a newer model or a better brand—and of providing them with more prepaid minutes. Keeping up and keeping connected are pressures that can, apparently, transcend death. I have learned to see these stores, and the things they sell, as a proxy for what is desirable in the world of the living. I sometimes think you can visit these stores and know all you need to know about people’s daily preoccupations and their aspirations.

I was in Malaysia and Singapore doing fieldwork in the run-up to Qingming this year and the stores were full again of this year’s aspirations and desires. I happened into one such store in Singapore near Toa Payoh with its famous beef noodles. I poked around the store—paper Hermès suitcases, paper automatic foot-massage machines, paper Mercedes cars with capped driver, full paper suitcases of paper clothes, and paper piles and piles of money were also on display. And then tucked in a corner, I spied the first paper Apple product I have ever seen—a paper iPad

Queuing up to buy a paper iPad for your dead relatives was a quintessential expression of filial piety, an act of love and care.

complete with paper leather traveling case and the smudges of fingerprints on the paper screen, the logo tweaked to have two bites out of the apple. There were only two and by the usual economies of the store, they weren’t cheap. More expensive than the phones and the foot-massage machine, but I bought them both. Chatting with the shopkeeper, I asked: “Why so few iPads?” He laughed and said “Well, they are iPads—they are scarce.” Retelling this story to colleagues in Penang several days later, revealed a similar shortage. Laughingly, Penang residents said: “That’s nothing. Here the newspapers carried the addresses of the stores that had iPads and iPhones and people actually queued to get them, and they sold out. We even posted on Facebook looking for extras.” The dead it appears also like Jonathan Ive’s sleek lines and Steve Jobs’ extraordinary vision.

It was clear in that moment that Apple products had crossed the line between mere technologies and beloved object, they had become something you would not live without, or indeed be dead without. They were as valuable a purchase for your ancestors as for you. And queuing up to buy a paper iPad for your dead relatives was a quintessential expression of filial piety, an act of love and care. Those paper iPads and iPhones that were burned this year remind us that Apple under Steve Jobs’ leadership wasn’t just making technology; it was making experiences, and those experiences transcend death. 

Genevieve Bell is director of Interaction and Experience Research for Intel Labs.

Copyright held by author.

Calendar of Events

December 17–20

International Conference on Wireless Technologies for Humanitarian Relief, Amritapuri, India, Contact: R. Chidambaram, Email: ammasvcoffice@gmail.com

December 18–20

Seventh Asia Information Retrieval Societies Conference, Dubai, UAE, Contact: Oroumchian Farhad, Email: farhadOroumchian@uowdubai.ac.ae

December 18–21

International Conference on Wireless Technologies for Humanitarian Relief, Amritapuri, India, Contact: R. Chidambaram, Email: ammasvcoffice@gmail.com

January 3–7

Fourth International Conference on Communication Systems and Networks, Bangalore, India, Contact: Don Towsley, Email: towsley@cs.umass.edu

January 6–8

Innovations in Theoretical Computer Science, Cambridge, MA, Sponsored: SIGACT, Contact: April Mosques, Email: mosques@hq.acm.org

January 10–12

Annual Conference of ACM SIGART on Artificial Intelligence Innovation, San Francisco, CA, Contact: Qiang Yang, Email: qyang@cse.ust.hk, Phone: 852-235-88768

January 17–19

ACM-SIAM Symposium on Discrete Algorithms, Kyoto, Japan, Contact: David S. Johnson, Email: dsj@research.att.com, Phone: 908-582-4742

January 18–20

ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Taipei, Taiwan, Sponsored: SIGDA, Contact: Charlie Chung Ping Chen, Email: cchen@cc.ee.ntu.edu.tw



Technology Strategy and Management

The Legacy of Steve Jobs

Reflecting on the career and contributions of the Apple cofounder.

MUCH HAS BEEN written about Steve Jobs since his announcement in August 2011 that he was stepping down as CEO of Apple and his death less than two months later in October. In the past, I have been disappointed that Apple did not pursue a more “open” strategy for the Macintosh (1984) as well as early versions of the iPod (2001), iTunes (2003), and the iPhone (2007) (see “The Puzzle of Apple,” *Communications*, Sept. 2008). I have noted that Apple did become a better platform leader, gradually, and in May 2010 topped Microsoft to become the world’s most valuable technology company (see “The Resurgence of Apple,” *Communications*, Oct. 2010). Jobs probably did not care much about what professors write or what other companies do; he always followed a unique path in life and in business. Nonetheless, anyone who cares about technology and innovation, or the type of entrepreneurship that Americans should be most proud of, should take the time to reflect on the career and contributions of Steve Jobs.

Products, Not Just Platforms

The point I made about Apple in the past was simple: In platform markets (those defined by a core technology and complementary innovations, driven by “network effects”—see “The Evolution of Platform Thinking,” *Communications*, Jan. 2010), the best platform will usually win—even if



there is a better product that we might benefit more from as consumers.^a We saw this with the Macintosh computer, which was far superior to the DOS-Windows PCs that won the mass market. Dominant platforms need to be sufficiently open and modular technologically as well as priced right for the mass market but also attractive for other companies to adopt as foundations to produce their own complementary products and services. These outside innovations tend to make the platform increasingly valuable to us-

^a See Annabelle Gawer and Michael A. Cusumano, *Platform Leadership* (Free Press, 2002) as well as Michael A. Cusumano, *Staying Power* (Oxford, 2010), among other publications.

ers and often determine who wins or loses a platform battle.

The problem with many platforms, though, is that they involve design compromises; they need to accommodate the needs of many users and partners, as well as maintain continuity with the past, which constrains innovation. The Macintosh was a breakthrough product, pioneering new ground with its graphical user interface, mouse, language and graphics processing capabilities, among other innovations. Yet it was expensive, was incompatible with DOS, had relatively few business applications, and failed to become adopted by the mass market. The NeXT workstation computer, which Steve Jobs introduced in 1988, was an even more expensive marvel of hardware and software design; it attracted even fewer customers.

Today, Windows running on Intel-compatible chips remains the most common software platform for personal computers (though cellphones far outsell PCs and have become the dominant mode of computing). But Microsoft has introduced only incremental innovations, following the path set by the Macintosh more than 25 years ago. And Android-based smartphones and tablets, which rely on Google’s “free” and “open” operating system, follow the lead of the iPhone and the iPad. My point is that Microsoft, Intel, and Google have taken the usual route to platform leadership, with inexpensive or free products, relatively open

interfaces, and extensive efforts to cultivate a broad ecosystem of partners. But Jobs and Apple have shown us another path to platform leadership, and not just for a niche product segment: Design breakthrough products that set new standards for form, function, and aesthetics; market them creatively and aggressively, with some modest reductions in price over time; open them up gradually as industrywide platforms, and let the chips fall where they may. This strategy will be very hard to duplicate without a Steve Jobs at the helm. But it is more of a win-win scenario for the innovator (still lots of money to be made) and the user (we all end up with better products, not just platforms).

Raising the Bar

But beating Microsoft or Google at their own platform game is not what seems to have motivated Steve Jobs. He appears to have cared most about the impact that technology and innovation, delivered in their most cultivated forms, can have on people's lives. For example, he is famous for a quip about Microsoft back in the mid-1990s that the company "has no taste" and did not bring the best of human culture into its products. Jobs cited the example of proportionally spaced fonts in the Macintosh, an idea he got from looking at beautiful books and the history of printing, and which Windows later copied.^b

From the beginning, Jobs wanted Apple to create computers that would be as elegant and simple to use as a typewriter or even a toaster. Now, looking back, we can see that every product Jobs championed, whether or not it succeeded commercially, set new standards for aesthetics as well as utility, such as in ease-of-use or handling graphics and multimedia. What stands out most to me are the ultra-simple, intuitive user interfaces of the Macintosh (GUI plus mouse, albeit invented earlier at the Stanford Research Institute and Xerox PARC) and then the iPod's clickwheel and the iPhone and iPad touchscreens.

By contrast, Bill Gates and Microsoft focused on software operating systems that led to cheap and powerful PCs as well as lots of applications but were

Beating Microsoft or Google at their own platform game is not what seems to have motivated Steve Jobs.

relatively clumsy and difficult to use for the average consumer. Today's PCs, digital media players, smartphones, and tablets based on Windows or even Android are as good as they are *only* because of how much Steve Jobs and Apple raised the bar—for everyone.

Charisma and Leadership

In the 1996 PBS documentary, "Triumph of the Nerds," Larry Tesler, who used to work at Apple, discussed how Steve Jobs was able to inspire people to surpass what even they believed they could accomplish. He would never settle for anything less than someone's absolutely best effort, and then some. That is how Jobs raised the bar for the Macintosh project—whose competition was the character-based IBM PC and compatibles—and many products since then, most recently the iPad. Tesler recalled: "When I wasn't sure what the word charisma meant, I met Steve Jobs and then I knew."^c

Let's be sure to give adequate credit to Apple cofounder Steve Wozniak for Apple's early products as well as to Jonathan Ive for being chief designer of the iPod, iPhone, iPad, and several hit Macintosh desktop computers and laptops. And to Scott Forstall, who headed iOS software development. New CEO Tim Cook, formerly the COO, has also been a highly effective leader of sales and operations since Jobs recruited him from Compaq Computer in 1998. But it has taken extraordinary charisma and leadership skills to bring so many diverse personalities together and channel their considerable talents so productively. This does not happen often or by chance.

It's in the Details

When Jobs and Wozniak cofounded Apple in 1976, they believed, along with Bill Gates and Paul Allen, who cofounded Microsoft in 1975, that the world would one day be full of personal computers. These entrepreneurs had something else in common: They all had the skills, in varying degrees, to build the products they dreamed of. Jobs needed Wozniak's technical wizardry to shrink down the number of chips and construct the internals of Apple's early computers. Gates and Allen were preeminent software programmers, especially Gates. But Jobs stands far above his peers for the degree to which he combined extremely astute technological vision with an ability to dive into the smallest details of his products, including hardware, software, industrial design, and marketing.

We have heard about the care Jobs took to design the Apple II case, with consumer electronics as his model, and how he tackled the many challenges posed by the Macintosh, ranging from reducing the price tag to the time it took to boot up. The users that bought these and other Apple products quickly came to love them—truly *love* their elegant look and feel in a way that seems unparalleled in any other competitor's products.

Until recently, Jobs continued to be deeply involved in the iPhone and iPad designs, both of which have the same look and feel about them that we first marveled at in the original Macintosh. I should not have been surprised to learn from the recent reporting that Jobs is listed as a coinventor of 313 patents, beginning with personal computer cases but extending to internal PC electronics and designs for laptops, multimedia devices (the iPod), smartphones (the iPhone), operating systems (NeXT, iOS), keyboards, mice, and Apple TV.^d

I am also left with the thought that great entrepreneurs do not really see the future as much as they *create* the future they envision. Steve Jobs knew how to build and sell game-changing products, down to the smallest details.

d Shan Carter, "Steve Jobs' Patents," *The New York Times* (Aug. 25, 2011); <http://www.nytimes.com/interactive/2011/08/24/technology/steve-jobs-patents.html?emc=eta1>

b Public Broadcasting System, "Triumph of the Nerds," The Television Program Transcript, Part III; <http://www.pbs.org/nerds/part3.html>.

c Public Broadcasting System, "Triumph of the Nerds," The Television Program Transcript, Part III; <http://www.pbs.org/nerds/part3.html>.

Computers and Consumer Electronics

As Steve Jobs moved forward in his career, he also brought related but formerly distinct technologies—and businesses—together. In fact, he felt compelled to shed the historic “Apple Computer” name in 2007 in favor of “Apple, Inc.” to reflect the broader set of aspirations that he and the company had adopted. It is instructive again to compare Jobs and Apple with Gates and Microsoft. Gates’ main entrepreneurial legacy has been to create a mass-market software products company that continues to “print money” and exploit those remarkable gross margins of packaged software (see “The Legacy of Bill Gates,” *Communications*, Jan. 2009). Indeed, Apple has yet to match Microsoft in profitability. But Microsoft is now a slow-growth gold mine, struggling to make money beyond the personal computer industry and the Windows and Office franchises.

By contrast, Apple has fully integrated computers with consumer electronics, including telephony and the mobile Internet (apps, music, video, text). Many firms played a role in merging computers with personal digital assistants, digital media players, and cellphones. But no one has pushed this convergence as far and as elegantly as Steve Jobs, especially as seen in the iPhone and the iPad. It was also important that Apple combined hardware and software in one company—something Microsoft could never do except in video games because it does not make hardware. And, with iTunes, Jobs solved an extremely vexing problem for the industry and for consumers: how to price digital content in the form of music, video clips, movies, and TV programs. This innovation in digital services is no less profound than Steve Jobs’ innovations in consumer products.

The Master Strategist

Early observers of Jobs and Apple, including myself, underestimated his ability to master the business side of technology. Clearly, over time, Jobs got better at this—much better—perhaps as the world caught up to what he was trying to do. Recall that he resigned under pressure from Apple in 1985, after the Macintosh failed to catch on in the marketplace. He started NeXT Comput-

Early observers of Jobs and Apple underestimated his ability to master the business side of technology.

er the same year, but this company also failed. Jobs recovered by selling NeXT to Apple in 1996, a brilliant stroke because he not only rejoined Apple but was able to use NeXT’s novel software technology to replace the aging Macintosh operating system. Meanwhile, Jobs’ love of graphics, movies, and computers led him to found Pixar in 1986 by acquiring the computer animation department from Lucas Films (which he sold to Disney in 2006 for a cool \$7.4 billion).

Jobs made other moves that showed he could put aside personal pride or biases to do what was necessary to save and grow the company. Two incidents stand out. First, when he rejoined Apple in 1996, the firm was practically bankrupt, with only a few months of cash left. But Jobs got a \$150 million investment from archrival Microsoft as well as a commitment from Bill Gates that Microsoft would continue to produce Office for the Mac. This agreement was critical to maintain the Macintosh business, then the only real source of revenue for Apple. Second, in 2005, Jobs abandoned his 20-year commitment to the Motorola microprocessor and adopted archrival Intel’s technology. This move helped bridge the growing cost-performance gap with Windows PCs, and enabled the Macintosh to continue as a second platform that was also much more interoperable with the Windows world. As I look back at this history—disappointment with the original Macintosh, failure at NeXT, success with Pixar, awkward but highly useful alliances with Microsoft and Intel, the two companies Apple customers loved to hate—one must conclude that, in addition to his other extraordinary talents, Steve Jobs truly was a master strategist, second to no one.

The Legacy

And so now Apple grows not at the slow pace of the personal computer business, like Microsoft, or of Internet advertising, like Google, but at the breathtaking speed of exciting new global markets led by smartphones, tablets, app stores, cloud services (iCloud), and digital content distribution (music, video, and TV, as well as books, magazines, and newspapers—what we used to call “print”). Looking at rates of growth in sales and profits, or increases in market value, Apple has left Microsoft and Google, as well as IBM, Hewlett-Packard, Dell, Nokia, Intel, Cisco, EMC, Sony, AOL, and many other prominent technology firms, in the dust. Jobs and Apple defied conventional strategy Microsoft-style and have shown that more substantive innovation—coming up with consumer products that truly do seem new or almost new *to the world*, and not just to the company—can be exciting and fun, as well as enormously profitable.

Apple should do well for several years to come, as Tim Cook’s team executes on the vision and product portfolio that Jobs has left behind. But most observers worry about what will happen when Apple exhausts the ideas still on the drawing board. Jobs will not be around to champion yet another product that changes the world and fills up yet another sales pipeline.

The end had to come, of course; no one lives forever and no company is immune to competition. We were somewhat prepared: Jobs was seriously ill for several years and had a limited role in Apple for the past couple of years. But words are inadequate to express what Steve Jobs meant to Apple and to the world of technology, innovation, and high-tech entrepreneurship American-style. Surely, as an entrepreneur and innovator, he represented the very best that the U.S., or any country, has to offer. Our thoughts must be with the people closest to him, beginning with his family and intimate friends. They will miss Steve Jobs the most, but they are not alone. □

Michael A. Cusumano (cusumano@mit.edu) is a professor at the MIT Sloan School of Management and School of Engineering and author of *Staying Power: Six Enduring Principles for Managing Strategy and Innovation in an Unpredictable World* (Oxford University Press, 2010).

Copyright held by author.

Emerging Markets On Turbocharged, Heat-Seeking, Robotic Fishing Poles

Applying a well-known proverb to socio-technical transformation.

IN THE BASEMENT of an office building in Bangalore, India, a housekeeper sat at a PC and painstakingly typed search terms into a browser. The PC was part of an early experiment at Microsoft Research India, which I co-founded in 2005. In the experiment, we were interested in what lower-income adults would do with an Internet-connected PC, if they had unrestricted access to one.

We were part of a larger movement called “information and communication technologies for development” (ICT4D), and at the time interest focused on what PCs and the Internet could do for international development. Digital technologies had transformed the lives of wealthy, educated people in developed countries. Could they help solve the challenges of poverty in the developing world? Proponents argued, for example, that telemedicine would revolutionize health care, that distance learning would close educational gaps, and that village telecenters would double rural incomes in even the poorest countries.

ICT4D has been gaining momentum since the late 1990s: On the one hand are technologists and entrepreneurs looking for ways to contribute to society beyond novel toys for rich folks; on the other hand, there is the international development community hoping to learn from the economic success



OLPC delivered by boat as part of OLPC Mexico Nayarit.

of the technology sector. The trend has only grown with the advent of the mobile phone, the numbers of which—over five billion accounts worldwide—comfortably exceed the total adult population of the planet.

The dominant model of ICT4D is to seek to apply technology innovations for the benefit of very low-income communities. Among the best known examples are One Laptop Per Child (OLPC), initially announced as a specially designed \$100 laptop that would fill the hole left by absent or under-

trained teachers in developing-country education; and M-PESA, a mobile payment system widely used in Kenya that allows users to send money via SMS text messages and a nationwide network of agents. Related projects have been featured previously in *Communications*.^{1,5}

I have conducted or supervised approximately 50 research projects in ICT4D, but while a few projects demonstrated meaningful impact and continue to do so in some form, the vast majority ended as temporary pilot projects with learning outcomes but



An M-PESA stand in Kenya.

little long-term effect. My experiences taught me a single, simple lesson: Technology is an amplifier of human intent and capacity, and only an amplifier.⁴ So, in well-meaning, capable hands, technology can work wonders; but absent good intentions or capability to use and support it fully, technology often ends up having zero or negative impact. Technology never guarantees net positive impact.

Technology and Societal Change

To be clear, it is not that technology cannot play a role in positive change. For example, M-PESA increases incomes in some rural areas, as urban migrants send money home with greater frequency. This kind of evidence has enamored the international development community to mobile-payment systems as a way to provide financial services to people who are “unbanked.”

But potential does not always translate to actuality. It is not at all clear that the net effect of systems like M-PESA will be positive overall, especially when one considers that they are two-way pipes between the pockets of poor, less educated people, and powerful corporations with savvy, well-funded marketing departments.

We have seen cycles of hype and disappointment before: In the 1960s, the television was hailed as a revolutionary technology that would replace the need for schools altogether. Today, it is better understood as a means by which millions of people watch reality TV.

This realization is ironic and disheartening for the technologist interested in social causes. Technology is supposed to be a means to scale the ingenuity of a few inventors for the benefit of many. We all grew up inspired by stories of Thomas Edison and Jonas Salk. Yet, with information and communication technologies, it is exactly those communities that most lack information-processing skills, a strong foundation of knowledge, and connections to influential social networks—and that are therefore poor—that are also the least interested or least able to make productive use of the technology.

Returns on Investment

Over the years, I have received many inquiries from computer scientists and engineers who say, “I’ve achieved comfort and security in my own life, but I’d now like to apply my skills for the less privileged people in the world.” Most of them then follow up with the question, “How can I apply my technical skills to the challenges of impoverished people?”

Poverty, though, cannot be engineered away, any more than a failing business’s problems can be. The deeper challenge lies with people and institutions, not technology. Perhaps sensing this issue, a few people ask a broader question: “What is the best way that someone like me can contribute to the lives of the less privileged?”

There is a well-known proverb, “If you give someone a fish, they’ll eat for

a day; if you teach them how to fish, they’ll eat for a lifetime.” The main point is obvious enough: Yes, it is great to give someone food when they are starving, but doing so is a short-term, stopgap measure. What is really worthwhile is to teach them how to grow (or catch) their own food, so that they can independently help themselves.

The saying, however, also packs several layers of additional insight that are particularly relevant for the technologist interested in international development. First, it is interesting to note what the saying does *not* say: It doesn’t say, “If you give someone a fish, they’ll eat for a day; if you give them a turbo-charged, heat-seeking, robotic fishing pole, they’ll eat for a lifetime.” That’s because, while such technology might result in more than one day of fish, it still leaves the person no better able to fend for themselves. Now, they need to adapt, maintain, and upgrade a technology—which is likely at least as great a challenge as fishing. The history of international development is full of rusting tractors, broken medical equipment, and increasingly, defunct PCs and mobile apps that worked until well-meaning specialists left when the funding dried up.

Second, if we draw an analogy between fish and technology, the saying suggests that simply providing technology (or selling it at low cost) to poor people is just another kind of charity: great as a stopgap measure, but not a long-term solution. A narrow interpretation of the analogy suggests we should instead teach people how to create technology themselves.

The larger point is that there is a world of difference between consumption of goods and production of goods—whether it be of fish, technology, or anything else. The ability to consume what you want is more a result of productive capacity than vice versa. ICT4D discourse tends to conflate the two, believing that any association with technology is good, but it is those who produce, not consume, technology that are best protected against poverty in the long term. If you had to give up one or the other, which would you rather do without...? All of the electronic devices you currently own (which will break or become obsolete within a few years), or all of your education, professional ex-

perience, leadership skills, and social contacts (which will serve you for the rest of your life, and propagate to people you raise, teach, or mentor)?

Finally, the saying implicitly recommends teaching over giving. The most meaningful contribution is to help another person grow, in knowledge, in new skills, and in forward-looking attitudes. Imagine a strange utopia in which technology feeds, heals, and generates income for the poor, so that the appearance of poverty itself is eliminated, but people remain unable to take care of themselves absent the technology. Is *that* the outcome we're seeking?

Real-World Applications

Reality, of course, is more complex than the black-and-white alternatives I have articulated in this column. Rarely are real-life choices constrained to two options of pure giving or pure teaching. In any case, we could not teach millions of non-literate people how to become world-class software engineers overnight, even if we wanted. And, just to do productive work often requires consumption of technology.

Nevertheless, the deeper wisdom of the fish proverb remains. Wherever possible, it is more meaningful, and more sustaining, to support the growth of productive capacity within people, than to simply supply technologies for them to consume.

For international development, that means that our skills as engineers, computer scientists, managers, and leaders are better applied to teaching and mentorship than for technological innovation on behalf of poor populations. The greatest contributions we can make are not displays of our own brilliance and heroism, but helping people to help themselves.

What would this mean in practice? One example was set by Patrick Awuah, who left a successful career as a program manager in the U.S. to establish a ground-breaking new private college in his home country of Ghana. Still less than 10 years old, Ashesi University just inaugurated a new campus for over 400 students in business administration, computer science, and management information systems, and it has won awards for raising the bar for tertiary educa-

tion in West Africa. Many of its graduates now write code for Ghanaian corporations or run start-up companies, thereby supplying the engine of growth for the country.

Or, consider Trish Dziko. After 15 years as a developer, designer, and manager, she founded the Technology Access Foundation (TAF), which runs educational programs that focus on science, technology, engineering, and mathematics for students of color in the Greater Seattle area. (Sometimes, the developing world is in your own backyard.) TAF provides children of low-income households hands-on exposure to robotics, chemistry experiments, and other experiences that are all too often cut from public schools. Then, through supplementary programs like internships and interview training, they prepare students for a strong future. Students who might otherwise fall through the cracks are nurtured through to college and beyond.

We do not have to be as bold as Awuah or Dziko; individuals who are less bold can also make a difference. The reason I know their stories is because I took personal leave from my job to teach calculus at Ashesi in its first year, and I am now considering how best I can volunteer time with TAF. Good organizations often need experienced employees, volunteers, board members, and mentors.

Teaching and mentorship, of course, must be tailored to the individual, and for many people in the developing world, we may have to start with the basics. Budding entrepreneurs might benefit from management advice and introductions to investors, but for illiterate children, we would need to start with simple reading skills. In between, there are rural teenagers who would benefit from exposure to careers in engineering, college students who could use a course on interviewing skills, and inexperienced computer programmers who would benefit from a good code review.

And, that brings us back to the Bangalore basement mentioned earlier. At the lab, we quickly found that free access to the Internet was most often used for entertainment. Understandably after a long day of work, the staff would search for the latest Tamil movies and watch them on YouTube.

Aishwarya Ratan, one of the researchers in my group then, was unsatisfied with this outcome. Though she acknowledged the value the staff got out of watching free movies, she felt that true development ought somehow to contribute to the staff's capabilities (along the lines argued by Nobel economist Amartya Sen³). So, she decided it was important to provide more than just the technology, and she ran a computer literacy course that taught the staff the basics of word processing, spreadsheets, and some educational software.²

For some members of the staff, this was all the encouragement they needed. One of the building's security guards began using the PC in the basement to practice data-entry skills that he learned in an outside evening class. One day, he came in and told us he was moving on. He had been offered a job in computer data entry. Though the job involved an initial cut in pay, his future prospects were much brighter, as he had effectively crossed over from a blue-collar job to a white-collar profession. He told us proudly, "Today I can stand up in front of my father and friends and say that I am no more a watchman, but I am doing a computer job." What allowed this transformation was less the technology in the basement, but a solid secondary-school education and the inspiration, instruction, and encouragement he received from Ratan and his data-entry teachers.

In short, it was the fishing lessons, not the fish, that made all the difference. ■

References

1. Dias, M.B., and Brewer, E. How computer science serves the developing world. *Commun. ACM* 52, 6 (June 2009), 74–80; <http://doi.acm.org/10.1145/1516046.1516064>.
2. Ratan, A. et al. Kelsa+: Digital literacy for low-income office workers. In *Proceedings of the Third International Conference on Information and Communication Technologies and Development (ICTD '09)*. IEEE Press, Piscataway, NJ, 150–162.
3. Sen, A. *Development as Freedom*. Oxford University Press, 2000.
4. Toyama, K. Can technology end poverty? *Boston Review* 35, 6 (Nov./Dec. 2010).
5. Underwood, S. Challenging poverty. *Commun. ACM* 51, 8 (Aug. 2008), 15–17; <http://doi.acm.org/10.1145/1378704.1378710>.

Kentaro Toyama (kentaro_toyama@hotmail.com) is a researcher in the School of Information at the University of California, Berkeley, and the former assistant managing director of Microsoft Research India.

Copyright held by author.



Code Vicious Debugging on Live Systems

It is more of a social than a technical problem.

Dear KV,

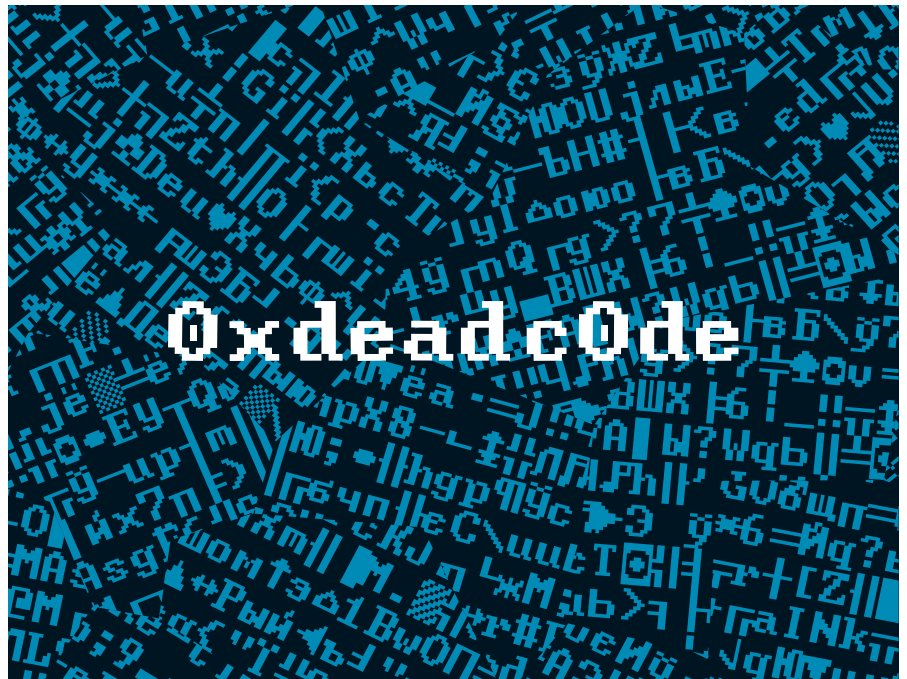
I have been trying to debug a problem on a system at work, but the control freaks that run our production systems don't want to give me access to the systems on which the bug always occurs. I have not been able to reproduce the problem in the test environment on my desktop, but every day the bug happens on several production systems. I am at the point of thinking about getting a key logger so I can steal the passwords necessary to get onto the production systems and finally see the problem "in the wild." I have never worked for such a bunch of fascists in my entire career.

Locked Down and Out

Dear Locked,

First of all, while most companies are inherently nondemocratic, few of them are fascist. Fascism went out of style sometime around 1945 and really hasn't made a comeback since. Secondly, I do sympathize—no one should be prevented from fixing a bug simply because of lack of access to the appropriate systems.

What many programmers and technical people fail to comprehend is that, as a colleague recently put it, "access implies responsibility." This is why the sudo program has the warning, stolen from the Spider-Man comics: "With great power comes great responsibility."



Debugging a program or a system can, and often does, have negative side effects, either by slowing down the system or changing the results of some calculation in an unintended fashion. The people who run your production systems are right to be wary of letting any random programmer loose in their domain. If you break something, it is likely to come down on their heads, and they will have to fix it while you stand there glumly repeating, "Well, it wasn't supposed to do *that*!"

Your best bet is to try setting up

a production system outside of the production environment first, as a test machine. I am surprised by how many companies work without such staging machines, going directly from the developers' desktops to their production environments. If the bug won't happen without real workloads, then it is time to get a machine in the production environment sufficiently isolated so that it can be given a workload without destroying the machines that are doing productive work.

By now you might have noticed that this advice is less technical and more about social engineering. Programmers must be willing to work with the people who have to keep systems up 24 hours a day, 7 days a week, if they want to be trusted enough to be able to debug live or near-live systems.

Two final thoughts: using a keyboard logger is not a way to gain trust, and telling someone in a public column that you're thinking about it is as dumb as tweeting your murder plans.

KV

Dear KV,

A program I have just been handed at work keeps crashing, and each time I look at it in the debugger and examine various bits of memory I see the pattern `0xdead00de` in different parts of allocated memory. Is this a joke? Do you think that my coworkers are hazing me?

OxDead Tired of this Code

Dear OxDead,

It is common practice for programmers to set memory to an easily recognizable value when they are trying to debug memory-smash bugs. You might think they would clear all the bytes in the buffer to be `0x00`, but that does not help if some piece of code is writing NULL bytes all over your buffers. Using a known pattern such as `0xdead00de` makes it easier to find these problems in a debugger. As you have seen, you print a buffer and you see the pattern. If instead you saw, say, `0xde00c0de`, you would know that someone had written a NULL byte in the middle of your memory. Maybe you wanted that, maybe you didn't, but now, at least, you can clearly see it. For extra cleverness points you can set a watchpoint—if it is supported by your hardware—which stops the program if some variable or part of memory does *not* equal `0xdead00de`. I tend to set buffers I am debugging to be all `0x69`, because if I see that number, then I know it is my own personal bit of work.

For programmers who deal with network packets, a known pattern has another advantage. Most people write code on systems based on the Intel

It is common practice for programmers to set memory to an easily recognizable value when they are trying to debug memory-smash bugs.

x86 architecture, which is known in network parlance as a little-endian system. A little-endian system stores the most significant byte of a multibyte word last. Network protocols are big endian, which is the opposite of how x86 processors store data in memory. All network programmers know the C macros `htonl()`, `ntohl()`, `htons()`, and `ntohs()`, which do the proper swapping of host-to-network endianness and back. A good way to debug a network protocol is to transmit data such as `0xdead00de` in the packets and then make sure it does not look like `0xdec0adde` when it arrives in your program's memory. Using this trick makes it easier to figure out where you might have left out a byte-swapping macro.

So, much as I would like to think your coworkers are hazing you, it is far more likely they are trying to be helpful.

KV

Q Related articles on queue.acm.org

Massively Multiplayer Middleware

Michi Henning

<http://queue.acm.org/detail.cfm?id=971591>

Cybercrime 2.0: When the Cloud Turns Dark

Niels Provos, Moheeb Abu Rajab,

and Panayiotis Mavrommatis

<http://queue.acm.org/detail.cfm?id=1517412>

Enhanced Debugging with Traces

Peter Phillips

<http://queue.acm.org/detail.cfm?id=1753170>

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

Copyright held by author.



ACM's *interactions* magazine explores critical relationships between experiences, people, and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of the interaction design. Our readers represent a growing community of practice that is of increasing and vital global importance.

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



Broadening Participation

Data Trends on Minorities and People with Disabilities in Computing

Seeking a comprehensive view of minority student demographics to determine what programs and policies are needed to promote diversity.

INCREASING DIVERSITY IN computing is very important for multiple reasons. First, there is the issue of the work force. According to the U.S. Census, Blacks and Hispanics were approximately 12% and 16% of the U.S. residents in 2010, respectively. According to the 2008 Census Bureau projections, Hispanics, African-Americans, and Native Americans/Alaska Natives are projected to account for 47% of the U.S. population by 2050. Second, there is the issue of having diverse perspectives involved in the design of products thereby having more robust end products on the market. Lastly, there is the issue of inclusion—that the field be representative of society.

Given the importance of increasing diversity, it follows that trends about the demographics of students in the computing field are necessary to determine what programs and policies are needed to promote diversity. To this end, we present different sources for data on minorities and discuss the importance of having multiple sources to get a comprehensive view. In addition, we begin a discussion about what the data indicates with respect to minorities and the difficulties in the data collection process for people with disabilities. In particular, the focus is on Blacks/African Americans, Hispanics, Native Americans, and

Examining multiple data sources can help find gaps in some data sources and help validate data in other data sources.

people with disabilities. The graphs shown in the accompanying figures were developed by the Center for Minorities and People with Disabilities in IT (CMD-IT).^a

For this column, the focus is on two major data sources:

► Computing Research Association (CRA) Taulbee reports (<http://www.cra.org/resources/taulbee/>) for computer science only. (Data source used for minorities at CRA-affiliated universities, which are primarily Ph.D.-granting institutions.)

► WebCASPAR (<https://webcaspar.nsf.gov/>), using IPEDS/NCES—the Integrated Postsecondary Education

Data System (IPEDS)—which is a survey conducted by the U.S. Department of Education's National Center for Education Statistics (NCES), to obtain data for race and ethnicity. The WebCASPAR database provides easy access to a large body of statistical data resources for science and engineering at U.S. academic institutions. The focus, however, is on the field of computer science. (Data source used for minorities at over 1,000 institutions, including community colleges, for-profit institutions, undergraduate institutions, and Ph.D.-granting institutions.)

The different data sources have different sets of U.S. institutions for which data is obtained. Examining multiple data sources can help find gaps in some data sources and help validate data in other data sources. The union of the data sources helps give a picture of the demographics of the broad computing community. In particular, it is important to include non-Ph.D.-granting institutions, community colleges, for-profit institutions, as well as Ph.D.-granting institutions. For example, in fall 2006, there were approximately 11.2 million students enrolled in four-year institutions and approximately 6.5 million students enrolled in two-year institutions.^b It is important to consider all degree levels: associate's, bachelor's,

^a Center for Minorities and People with Disabilities in IT, CMD-IT (pronounced "command it"); <http://www.cmd-it.org>.

^b *Digest of Education Statistics 2008*, Table 194.

master's, and doctorates because they represent stages in the pipeline. Further, it is important to have the data broken down by gender and ethnicity to allow analysis of trends related to minority women. It is recognized that surveys regarding ethnicity and gender are usually based upon self-identification, for which people may select the option to not provide the information. The survey results, however, provide the best data available for understanding trends.

Associate's Degrees

The primary data source for the associate's degree is WebCASPAR. With respect to number of institutions, for 2009, WebCASPAR included 1,065 institutions for the associate degree. The CRA Taulbee data does not report on the number of associate degrees; data is given for bachelor's, master's, and doctorate degrees. Table 1 provides the number of degrees awarded to students from the different ethnic groups in addition to the total number of degrees awarded for the past five years. With respect to associate's degrees, the data indicates the number of degrees awarded is along the same order of magnitude as the bachelor's degree for minorities. This trend, however, does not occur when considering the total number of degrees at the different levels; the total number of bachelor's degrees far outnumbers the total number of associate's degrees. This high participation of minorities at the associate's degree level indicates the importance of encouraging students at the community colleges to complete the bachelor's degree. Hence, while significant recruiting for minorities occurs at the high school level, significant effort needs to be devoted to recruiting minorities at the associate's degree level.

Table 2 identifies the number of minority women for the associate's degree. The data indicates the number of minority women for all three ethnic group numbers increased from 2000 to 2005, but then decreased by 2009. This trend, however, is consistent with the trend for total number of associate's degrees.

Bachelor's Degree

The WebCASPAR and Taulbee data sources have vastly different numbers of institutions for the bachelor's degree. For example, for 2009, WebCAS-

Table 1. Number of associate's and bachelor's degrees awarded.

Year	Associate's Degree				Bachelor's Degree			
	Blacks	Hispanics	Native Amer.	Total No. Degrees	Blacks	Hispanics	Native Amer.	Total No. Degrees
2005	5,119	3,888	352	36,140	5,815	3,529	281	54,588
2006	4,617	3,261	325	31,170	5,275	3,351	274	48,000
2007	3,988	2,980	291	27,680	4,588	2,970	249	42,596
2008	4,171	2,897	298	28,327	4,011	2,923	221	38,922
2009	4,316	2,995	293	30,050	3,868	2,999	213	38,496

Source: WebCASPAR; <https://webcaspar.nsf.gov/>

Table 2. Number of minority women for the associate's degree.

Year	Black Women	Hispanic Women	Native Amer. Women	Total No. Degrees
2000	1,711	1,097	153	23,576
2005	2,239	1,159	156	36,140
2009	1,567	675	107	30,050

Source: WebCASPAR; <https://webcaspar.nsf.gov/>

Table 3. Number of minority women for the bachelor's degree.

Year	Black Women	Hispanic Women	Native Amer. Women	Total No. Degrees
2000	1,698	693	59	37,519
2005	2,383	930	86	54,588
2009	1,330	591	60	38,496

Source: WebCASPAR; <https://webcaspar.nsf.gov/>

PAR included at least 1,283 institutions for the bachelor's degree, 442 institutions for master's degree, and 97 institutions for the doctorate degree. In particular, the institution counts correspond to those that reported awarding at least one degree at the given level. By contrast, for the 2009–2010 academic year the CRA Taulbee data is based upon completed surveys from 150 CS programs. It is noted that CRA is focused on research, for which the

primary membership of CRA entails Ph.D.-granting institutions.

Figure 1 compares the data from WebCASPAR with that obtained from CRA Taulbee for the bachelor's degree. We consider the computer science bachelor's degree only because for the WebCASPAR classification variable of "Academic Discipline, Detailed (standardized)" computer science is given, but not computer engineering. The data indicates a significant difference in the trends reported by the two data sources regarding minorities. For WebCASPAR, the percentage for Blacks is in the 10%–11% range, in contrast to 3%–4% as indicated with Taulbee. In the case of Hispanics, there is some difference in the percentages, with WebCASPAR indicating percentages in the range of 5%–8% with Taulbee indicating percentages in the range of 3%–6%. For the case of American Indian/Alaska Native, the percentages are less than 1% for both data sources. Further, it is noted that the Taulbee data indicates a

It is recognized that surveys regarding ethnicity and gender are usually based on self-identification.

recent decline in the percentage of Hispanic bachelor's degrees in contrast to the WebCASPAP data, which indicates a recent increase in the percentage of Hispanic bachelor's degrees. Hence, the data indicates a large number of minorities at the bachelor's level are not at the Ph.D.-granting institutions.

With respect to the number of minority women at the bachelor's degree level indicated in Table 3, we see similar trends as that given with the associate's degree. The numbers increased from 2000–2005 and then decreased from 2005–2009. Similarly, the total number of degrees had a similar trend. It is noted that the number of minority women at the bachelor's level is very small in comparison to the total number of degrees. Hence, significant effort is needed to increase the number of minority women.

Master's Degrees

Figure 2 provides data on the percentages of the different ethnic groups awarded master's degrees for Taulbee and WebCASPAP. While the number of WebCASPAP institutions is much smaller for the master's degrees than the bachelor's degrees, there is still a significant difference between the percentages for the Black and Hispanic groups for WebCASPAP versus Taulbee. For Blacks, the WebCASPAP source indicates percentages in the range of 4%–5.5% in contrast to the Taulbee source, which indicates percentages in the range of 1%–2%. Both sources indicate a slight decline in the percentages when going from 2008–2009. For Hispanics, the WebCASPAP range is between 2%–3% in contrast to the Taulbee range, which is between 1%–2%. Both data sources provide similar trends. For the case of American Indian/Alaska Native, the percentages from both sources are consistently small, less than 1%. With respect to the percentage of minority women at the master's degree level, the percentage of women from the three groups remains approximately flat in the range of 2% for Black women, 0.8% for Hispanic women, and 0% for Native American women from 2000–2009.

Doctorate Degrees

In Figure 3, which focuses on the doctorate degrees, the numbers are very small from both data sources as the

Figure 1. WebCASPAP and Taulbee data for percentage of bachelor's degrees awarded to minorities.

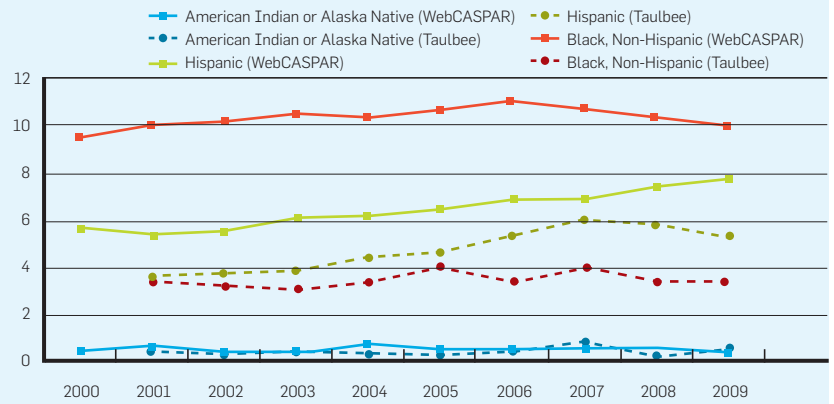


Figure 2. WebCASPAP and Taulbee data for percentage of master's degrees awarded to minorities.

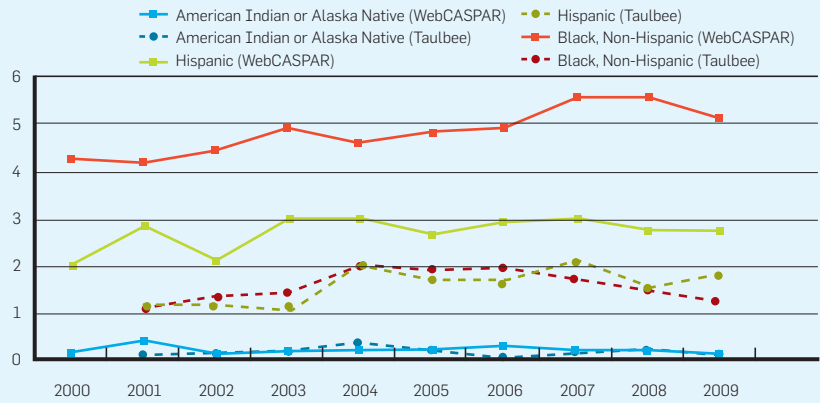
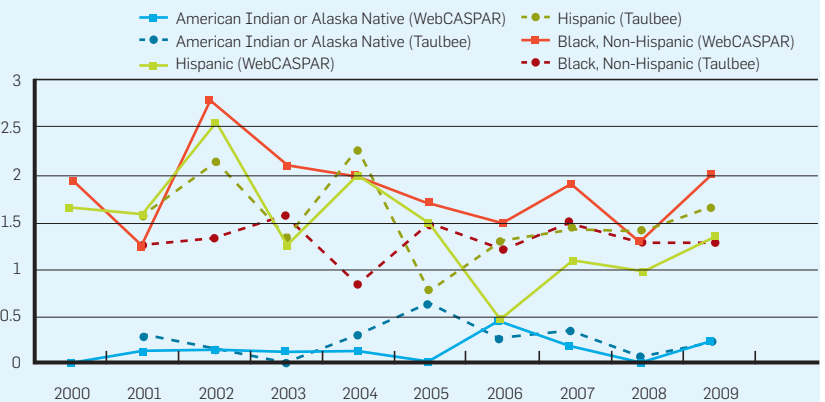


Figure 3. WebCASPAP and Taulbee data for percentage of doctorate degrees awarded to minorities.



maximum percentage is only 2.80% in 2002. Because the focus is on Ph.D.-granting institutions, the data from the two sources are fairly close. From

the NCES data source, the percentage of minority women at this level has remained flat in the range of 0.7% for Black women, 0.3% for Hispanic wom-

en, and 0% for Native American women from 2000–2009.

People with Disabilities

People with disabilities are an important group to consider because they are underrepresented in science and engineering fields, and there are a number of programs to increase their number in computing fields. The data collection process, however, is very difficult for a number of reasons. First, institutions differ in how they count students with disabilities. The counts can be based on services or accommodations provided, self-reporting to the disability support service office, verification of disabilities, or external/proxy report to the disability support service office. Further, institutions differ in how they maintain records of students with disabilities. Some institutions include data on students with disabilities in the general student record system from which degree data is reported to the Department of Education. Further, it is noted that one cannot consider the pipeline with people with disabilities, as a person can become or be recognized as disabled at any point in one's life. For example, a person with the degenerative syndrome retinitis pigmentosa is not born blind, but will become blind gradually over time. Moreover, a student may not be recognized with a learning disability until problems arise when the student attends college.

We are very fortunate that Joan Burrelli, retired Senior Science Resources Analyst from the NSF National Center for Science and Engineering Statistics was able to provide us with data for this column from the National Center for Education Statistics, 2008 National Postsecondary Student Aid Study (available through their Data Analysis System and the National Science Foundation) and the 2008 Survey of Earned Doctorates. These reports indicate that in terms of enrollment between the years 2004–2008, 12% of undergraduate IT majors and 8% of graduate IT students had a disability (where IT refers to computer science, information science and systems, and computer engineering).

By contrast, only 0.7% (63 Ph.D.'s) received a doctorate in computer science in the same period. The 12% at the undergraduate level is considered

What are the factors influencing the institution choice for minorities majoring in computer science?

close to the percentage of all students who have a disability attending college. The low percentage reported in the Survey of Earned Doctorates may have two contributing factors. First, data from the 2008 National Postsecondary Student Aid Study indicates students with disabilities are significantly less likely to persist to obtain a bachelor's degree than those without a disability, with about 40% persistence for those with a disability compared with 60% persistence for those without a disability. Second, the data in the Survey of Earned Doctorates is self-reported, and there may be some reluctance for a person with a disability who has achieved such a high level to report their disability.

Summary

The data presented in this column demonstrates the importance of using multiple sources with respect to obtaining the data about minorities and people with disabilities in computer science. To understand the broad trends about minorities in computer science, one must consider two-year institutions, for-profit institutions, non-research institutions, as well as Ph.D.-granting institutions. For example, the number of minority students receiving associate's degrees is in the same range as the number receiving bachelor's degrees in computer science.

It is good to find the trends for Blacks and Hispanics at the bachelor's and master's levels are not as bleak as portrayed in the Taulbee data. The percentage of Blacks earning bachelor's degrees at the 1,283 WebCASPER institutions is about 10%, which approaches the approximately 12% representation in the general population (2010 census). The percentage of Hispanics earning bachelor's degrees at the WebCASPER institutions is about 7.8% compared to

approximately 16% representation in the general population (2010 census). By contrast, the percentages reported in the most recent Taulbee survey are 3.4% (Blacks) and 5.3% (Hispanics). These large differences reflect the different sets of institutions represented in the two data sources. However, at the doctoral level, the two sources of data show more similar percentages for all groups, Blacks, Hispanics, and Native Americans. This reflects the similarity between 97 WebCASPAR institutions and the 150 Taulbee institutions.

Further, the data indicates minorities at the bachelor's level are not at the Ph.D.-granting institutions. The number of minorities receiving associate's degrees is approximately the same as the number receiving bachelor's degrees; this is not the case when comparing total number of degrees at the two degree levels. These trends raise a number of questions. First, what are the factors influencing the institution choice for minorities majoring in computer science? Second, how much recruiting for minorities at the bachelor's degree level is targeted to community colleges? Lastly, for the Ph.D.-granting institutions, how much recruiting for minorities for the Ph.D. is done at non-Ph.D.-granting institutions? These are all important questions, whose answers could lead to actions that improve the number of undergraduate and graduate minority students at Ph.D.-granting institutions.

Finally, our comparison of the Taulbee and WebCASPER data and the results from the recent TauRUs (Taulbee for the Rest of Us) survey¹ indicate a need for a more comprehensive annual report of the demographics of computing students than is currently being done. Decision makers at all levels need better data about minority and disabled students on which to base their decisions. 

Reference

1. Goldwasser, M. TauRUs: A Taulbee Survey for the Rest of Us. *ACM Inroads* 2, 2 (2011), 38–42.

Valerie E. Taylor (taylor@cse.tamu.edu) is the Royce E. Wisenbaker Professor in the Department of Computer Science and Engineering at Texas A&M University and Executive Director of CMD-IT.

Richard Ladner (ladner@cs.washington.edu) is the Boeing Professor in Computer Science and Engineering at the University of Washington, Seattle, WA.

Copyright held by author.



The Profession of IT The Grounding Practice

The skill of making and recognizing grounded claims is essential for professional practice. Getting objective data to support your conclusions is not enough.

IN MY WORK, I constantly have to assess whether claims are valid. This applies not only to my own claims, but to the claims of others that I am considering as evidence to support my claims or to launch actions.

The problem of validating claims seems to be growing in recent years. Google searches yield many exaggerated claims that are not useful as evidence. Many apparently independent news items all derive from a single source, such as a press release, whose accuracy cannot be verified. Even the crowdsourced Wikipedia can be untrustworthy. How do we recognize or generate valid claims in this environment?

Some Web services already offer help with the quality of evidence. Reputation.com, a for-profit, locates derogatory information about its clients and tries to neutralize it or cut off the sources. Snopes.com investigates urban myths and other hot “memes” and rates them according to whether they can be independently verified. Truth-Seal.org vets and guarantees claims, and pays bounties to those who successfully refute them. Idoscience.org helps kids doing science experiments obtain data to sustain or refute their science claims.

For our daily work we need not Web services, but practices that enable us to generate valid claims and recognize when others’ claims are valid. One commonly recommended practice toward this goal is to “base decisions on data”—meaning per-



form experiments to back up your hypotheses before asking others to act on them. Another is the agile developer mantra to “fail fast and often”—meaning organize your project so that you only move forward with components that pass quick field tests. As useful as these practices are, they do not directly address the formation of valid and compelling claims. Bad decisions based on (insufficient) data, and failures that teach us nothing, are all too common.

Let us examine the deeper structure of valid claims. We will see a practice called “grounding claims” that consistently produces them.

The Deep Structure of Many Professional Claims

A *claim* is a statement that asserts something is true. A *grounded claim* is

a claim accompanied by sufficient, relevant supporting evidence.

If you reflect for a moment on claims you have heard, you will see that most claims are actually subjective. They are hypotheses, judgments, evaluations, or opinions that something is true. That is why supporting evidence is so important. Good evidence makes the claim credible to listeners and makes it easy for them to accept. The evidence can be either facts or opinions, or a mixture of the two:

- *Objective evidence* consists of facts. Facts are statements generally accepted as true. Facts can be independently re-verified or possibly falsified.

- *Subjective evidence* consists of opinions. Opinions are evaluations, judgments, or assessments. Whether we accept an opinion as supportive of a claim depends on how much we trust the opinion maker.

Evidence is sufficient if it deals with all the objections listeners are likely to have. Evidence is relevant if it supports the claim and omitting it would weaken the claim. In the next sections, I will give examples of objective grounding from science and subjective grounding from team-member selection.

The preceding structural description is not sufficient to guarantee that listeners will actually accept a claim. Various other factors influence listeners, including:

- *Plausibility*—does the claim make sense?
- *Balance*—does the evidence deal with competing or opposing claims?
- *Commitment*—does the speaker

defend the claim and deal with its consequences?

Sometimes, even the speaker's comportment will affect listeners' willingness to accept your claim.

Often unconsciously, we rely on these distinctions in our daily work. As professionals, we size up a client's problem and claim whether or not we can help. As managers, we evaluate alternative means to get projects done and claim the least expensive or fastest one. As leaders, we try to mobilize people to take care of a concern by claiming a path of acceptable risk. Everywhere we turn, we are making or hearing claims, and we base our actions on them.

Note that there are many other forms of argument and rhetoric than the type being considering here. In the sciences and professions, we want to persuade based on evidence. That is the sole focus of this column.

Objective Grounding

Some professions such as science, engineering, and medicine have strong traditions of grounding their claims. In science someone who makes a new claim (hypothesis) is expected to support the claim with data, logic, and other evidence that will allow others to accept the claim. The peer review process for publication tries to evaluate whether claims are well grounded, and seeks to reject papers whose claims are not.

In these professions a claim evolves from the status of hypothesis to fact over a period of time. Initially, a hypothesis will have few followers. Over time, it will gain allies as others test and confirm it for themselves. Eventually, when it is universally accepted and no one can find contrary evidence, the hypothesis will be accepted as a fact by the community. Even so, scientific facts are subject to refutation later if new evidence turns up, for example, new data from more precise instruments. This is why science sociologist Bruno Latour says that science is a process of constructing facts.⁴

In its investigation of the space shuttle Challenger disaster on a cold morning in January 1986, the Rogers Commission debated without resolution the hypothesis that O-ring failure was the cause. Physicist Richard Feynman confirmed it dramatically with a

Everywhere we turn, we are making or hearing claims, and we base our actions on them.

public experiment showing that the O-ring material became hard and brittle in a glass of ice water. His simple demonstration instantly pushed the claim over a threshold of credibility. The commission concluded that although the data was available it had not been presented in a compelling way to NASA managers determined to launch. Although the potential for a well-grounded claim was there, the NASA managers did not "hear" the engineers' actual claims as well grounded. The commission also concluded the NASA managers were not open that morning to any claim that launching was too dangerous.

In the early 1900s, U.S. Navy Lieutenant William Sims observed that British ships whose gunners used hand cranks to dynamically adjust the angle of cannons had much higher hit rates during battle.¹ He measured the British hit rate around 10% and the U.S. rate less than 1%. He advocated to the U.S. Navy that continuous-aim gunnery would turn more battles to U.S. wins. Navy officials ignored Sims's initial technical reports. He wrote more reports, offering more data; they continued to ignore him. He became very critical of their attitude. They saw him as an egotistical crank. Eventually he decided his career was tanked and wrote a complaint directly to his Commander-in-Chief, President Theodore Roosevelt. Lucky for him, Roosevelt thought his claim had merit and brought him to Washington to oversee Naval Target Practice. This got his innovation adopted and, in the end, won him great respect and honor. But Roosevelt's intervention was a stroke of luck. Most officers who buck their chains of command so flagrantly are dismissed.

What was Sims's problem? He made a claim and backed it up with a lot of objective evidence. Historians tell us the reason leadership rejected his claim was they believed his proposal would disrupt the "ship society." The gunner corps was elite and specially trained. Sims advocated a capability that would allow any sailor to be a good gunner. Sims's mistake was to assume hit rate was the main criterion of importance to the Navy leadership. They had other standards around the social impact of the new technology. Had Sims included arguments about how the technology would enhance the ship society, he might have gotten a different response.

These examples illustrate a key point about grounded claims. It is wise to learn all the criteria important to the listeners and provide relevant and sufficient evidence for each criterion. Otherwise, the grounding offered will not be compelling.

Subjective Grounding

As a manager or leader, you hire or select people to be on your team. You are very interested in their competence because without it your team cannot perform. When you interview people for a place on your team, you have to evaluate their competence claims. These claims can seldom be objectively grounded, but they can be subjectively grounded.

To be competent means to be able to perform standard actions in a community without supervision and without causing breakdowns for customers. Communities develop criteria for assessing competence and awards for recognizing outstanding examples. In assessing a competence claim, it is very important to learn what the community members say about the prospect in terms of performance tests, recognitions, and testimonials.

The first thing to notice about a competence claim is that it enters your awareness with the status of a hypothesis, and evolves to an acceptable statement as you consider the evidence in light of your acceptance criteria. Unlike a scientific hypothesis, this claim cannot evolve to the status of a fact. The reason is that your "data" is actually the opinions of others, who may not agree on the interpretation of what they have witnessed. Since your

Anatomy of a grounded claim.

	Distinction	Explanation
Structure of a supported claim	Claim	Statement that something holds; the something is usually a hypothesis, evaluation, judgment, or opinion.
	Fact	Statement that something is verifiably true; can be independently re-verified, and possibly falsified.
	Opinion	A judgment, assessment, or estimate of something.
	Grounding	Series of statements relevant to and supportive of the claim. For objective grounding, all supporting statements are facts. For subjective grounding some supporting statements may be opinions by trusted speakers.
Acceptance criteria of the claim and its support	Domain	Community, discourse, discussion, situation, event to which the claim applies.
	Purpose	The point of making the claim. What are the concerns? Who cares?
	Relevance	Are supporting statements connected to the domain and purpose of the claim?
	Sufficiency	Are supporting statements sufficient in number to support the claim?
	Frame	Is the claim credible to the community to which it is offered?
	Balance	Does the speaker deal adequately with opposing arguments?
	Commitment	Is speaker committed to defending the claim and taking care of its consequences? Is the speaker's purpose genuine?
	Comportment	Does the speaker display confidence? Authenticity? Centeredness?

data are all facts, the claim cannot be objectively grounded. However, it can be *subjectively grounded*—meaning that you are willing to accept both the supporting facts and opinions and bring the candidate onto your team.

In addition to community opinions, you will want to evaluate whether the prospect's expertise matches what you need for your team and whether their expertise meets all your requirements. For example, you may not want a competent Web programmer when your team needs a game programmer, or you may want someone who is competent both at programming and team management. You may have other criteria to evaluate as well.

Once the person joins your team, you are hardly done in evaluating their claims. For example, in daily work, the person will claim they can deliver a result by a certain time. Can you acceptably ground that their claim is within their competence, that they are sincere, and that they are reliable?

Truthiness

In 2005 Stephen Colbert, the political satirist, proposed the new word

"truthiness" to mean the presentation of claims supported only by gut feelings or emotional beliefs. Truthy claims are nothing more than ungrounded assessments. Nevertheless, they can gain allies.

Negative political advertisements, which are common during elections, are effective because so few listeners are skeptical; they are willing to accept claims without evidence. If they actually checked for supporting evidence, they would find the claims unsupportable, and reject them. Rick Hayes-Roth has recently devoted an entire book to this problem and to means for combating it.^{2,3}

It is quite easy to confuse a truthy claim with a truth because the claim is often worded in a manner such as "I claim that..." or "I assert that..." or even "It is true that..." Do not let the choice of words obscure the basic distinctions of claims and grounding. Get into the general habit of noticing whether a statement is a claim, an opinion, or a fact. Then you are on solid ground when it is time to decide whether to act on the claim or not. Make acceptance of a claim a conscious choice.

Conclusion

The grounding practice is the skill of making claims that move people to action. Your goal as a speaker is to give your listeners plenty of well-crafted support for accepting your claim. Your goal as a listener is to decide whether you accept the claim based on the supporting evidence.

The accompanying table summarizes the distinctions for grounding claims. A grounded claim consists of the claim statement and a set of supporting statements. The acceptance criteria can vary according to the listeners and their standards, but will include one or more of the eight criteria listed.

When gathering data, therefore, you need to fully understand the listener's concerns and interests. You want to supply relevant data—directly supporting the claim and the purpose for which you make the claim—and omit irrelevant data no matter how interesting they may be to you. You want to have a sufficient amount of data to be convincing.

The grounding practice is also helpful for assessing background assumptions to see whether they are well grounded in the current environment. For example, is my self-assessment that "I'm not good at management," or my group's assumption that "There's no way to cut the red tape," grounded? If an important assumption is not grounded, we would want to stop accepting and acting on it, and replace it with a new, grounded assumption.

Grounding claims is an essential professional skill for practitioners, managers, and leaders. The skill supports many decisions, and will help us design trustworthy systems. **C**

References

1. Denning, P.J. and Dunham, R. *The Innovator's Way: Essential Practices for Successful Innovation*. MIT Press (2010).
2. Hayes-Roth, R. (interviewed by Peter Denning). Honesty is the best policy. *ACM Ubiquity* (July 2011); <http://ubiquity.acm.org/article.cfm?id=2002437>. DOI: 10.1145/2002436.2002437.
3. Hayes-Roth, R. *Truthiness Fever*. Booklocker.com, 2011.
4. Latour, B. *Science in Action*. Harvard University Press, 1987.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

Copyright held by author.

Viewpoint

Doctoral Program Rankings for U.S. Computing Programs: The National Research Council Strikes Out

A proposal for improving doctoral program ranking strategy.

WHY DO WE care about rankings of graduate programs? Beyond the ability to cheer “We’re Number One!” there are very practical reasons. For example, resource allocation is often based on using rankings as synonyms for quality indicators. An institution recently decided it would become a “top 25 institution” by ensuring that each of its graduate programs was ranked within the top 25% of all the graduate programs in the corresponding fields. And it was going to accomplish this by simply eliminating any program that was not—mission accomplished! Besides resource allocation, prospective graduate students and faculty candidates look to rankings when deciding where to apply, so the rankings for U.S. institutions considered in this Viewpoint are of considerable interest both within the U.S. and internationally. Funders look at rankings when considering ability to perform the proposed research. Alumni look to rankings when making donation decisions. Despite all their acknowledged warts, rankings *do* matter.

In principle, generating rankings is straightforward mathematically:

1. Develop a list of the n most important metrics (amount of external



- research funding, number of quality publications, impact of publications, awards, entering student GRE scores, placement success of graduates, plus any other factor deemed important by the community).

2. Plot the value of these n metrics for each institution in an n -dimensional space in which the axes are the metrics.

3. Develop a mapping to turn this n -

dimensional space into a one-dimensional ordered set of integers. (That this cannot really be done in a principled or defensible manner is one of the fundamental problems with rankings.)

Of course the practical difficulties are enormous. Among them:

- What metrics should you be including?
- How do you get (accurate) values for these metrics?
- The mapping is going to require weighting of these metrics and how do you determine these weights? Effectively how should one weigh publication counts vs. citation counts vs. external grants vs. faculty awards vs. entering student GRE scores vs. any other factors?

So there are ample reasons why rankings based upon a transparent comprehensive analysis are not done frequently. Nevertheless, the U.S. National Research Council (NRC), through its Committee on an Assessment of Research Doctorate Programs, bravely tackled this thorny problem for U.S. institutions. (The NRC is the operating arm of the U.S. National Academies of Science and Engineering and the Institute of Medicine, honorific academies with a mission to improve government decision making and public policy, increase public educa-

tion and understanding, and promote the acquisition and dissemination of knowledge in matters involving science, engineering, technology, and health. In many respects, the academies and NRC represent the “gold standard” of technical policy advice in the U.S. Because of the prestige of the academies and the NRC, their methodologies and reports have considerable international impact as well.)

The NRC last ranked doctoral programs in the mid-1990s and these rankings are clearly out of date. Further, the earlier rankings depended heavily on “reputation” as determined by respondents and this is often an inexact and lagging indicator. This time around the NRC sought to focus on a purely quantitative approach.

In this Viewpoint we describe how this process has played out for computing. While these comments clearly apply directly only to the NRC rankings effort, they are relevant to other similar efforts.

The NRC Ranking Process

The specifics of the NRC process were the following. The NRC developed a single set of metrics for all 62 disciplines being analyzed, covering disciplines in science, engineering, humanities, social sciences, and others. It then collected the data for these metrics via questionnaires administered to institutions, programs, faculty, and Ph.D. students plus submitted faculty CVs. Determining the weights was done via two related approaches: ask a set of participants how much various metrics mattered in their perception of department rankings, and a linear regression of a set of rankings vs. these metrics. Because these two approaches yielded substantively different results, the NRC established two sets of rankings—Sur-

vey and Regression rankings—and reported these probabilistically. Specifically, they ran a set of samples using weights derived from these acquired distributions, and then reported the range of rankings corresponding to a 90th percentile, meaning that with 95% probability, an institution’s rank would lie within the designated range. In other words, as an example, the NRC states that with 95% probability Georgia Tech ranks somewhere between 14th and 57th using the Survey weights and somewhere between 7th and 28th using the Regression weights.

The first issue is that this range, arising out of the probabilistic analysis, is difficult to reconcile. What does a rank between 14th and 57th mean? How does one reconcile differences between the two ranking systems—between the Survey weights which measure what respondents claim is important and the Regression weights which measure these claims against departmental reputations? Of how much value is a range if a 95th percentile span is being used?

Even if the rankings were not as impactful as in prior NRC studies, a rigorous data collection process could have yielded valuable data, which departments could use to assess their standing relative to peers. Unfortunately, there were a number of issues with the quality of the data:

- Data collection took place in 2006 but the ultimate release of data and rankings was in 2010 (with corrections well into 2011). For some metrics small changes might have large consequences, for example, given our low numbers of female faculty the addition of a single woman would result in a large percentage impact on the diversity metric or the departure/arrival of a single highly productive faculty member would similarly have a large impact on the scholarly productivity metric.

- The metrics to be used are not discipline specific—exactly the same information was collected for physics, for English, for computing, and for every other discipline. But we know that publication practices in particular vary significantly across disciplines: the humanities rely heavily on book publications; the computing fields rely heavily on conferences. However, the NRC decided that the metric to use for scholarly publication was going to be jour-

nals (possibly primarily because they had low-cost access to the ISI database of citations for journal publications). We know that this will not provide accurate results for computer science, as it misses almost all of the conference publications (and corresponding citation data), and the specific choice of this database also means that many journal publications are missed.

- The descriptions of data to be provided were often ambiguous, leading different institutions to respond differently. Thus the data being compared was often not measuring the same parameters across departments—this was especially true when gathering lists of faculty to be included in the data gathering, a factor that has impact on many of the data categories since parameters were often measured per faculty member.

- Measurements of scholarly quality are not equivalent to measurements of scholarly quantity, that is, the most impactful publication is not necessarily the one with the most citations nor is the most impactful professor necessarily the one with the most publications. There is considerable literature on resolving this issue; for example, by measuring publication quality via the impact factor of the journal.

- The NRC did not measure scholarly productivity other than publications and grants. For example, software artifacts and patents were not considered.

- The NRC did not get CVs from all faculty so they simply scaled results by the number of faculty in a given department. This approach is easily gamed by having only the most productive faculty provide CVs.

- The list of faculty awards was seriously incomplete—computer science was not even listed as a distinct category. The ACM A.M. Turing Award was not considered “Highly Prestigious”; no awards from organizations other than ACM and IEEE were included; and many other gaps in awards were apparent.

- The NRC chose to invent data when they could not obtain it, for example, for entering student GRE scores the NRC used the national average for these scores when an institution did not collect or provide them.

The second issue noted here has gained the most attention from our

**Despite all
their acknowledged
warts, rankings
do matter.**

community. CRA and ACM provided testimony to the NRC in 2002 when the study was just beginning, pointing out the importance of conferences to our field. Unfortunately, this advice was simply ignored by the NRC, a fact we did not discover until February 2010. We immediately notified the NRC, urging it to include conference publications, both for measuring publication productivity and for measuring citation impact. The NRC ultimately agreed to do so after extensive discussion at various levels. CRA worked with its member societies to provide a list of quality conferences; due to the tight deadline we know that this list is not 100% complete or accurate. The NRC took this list and then searched all vitae provided by CS faculty (which we also know to be incomplete) to generate conference publication counts. Since citations for conference publications were not available via the ISI database used by the NRC, citation data was not used at all for computer science as alternatives were not acceptable to the NRC. Based upon the NRC's analysis, a typical department had one conference publication per faculty member per year. In our view, this is not credible. Further, the NRC claims that more computing publications appear in journals than in conferences, which is very difficult to reconcile with what we see in practice.

Similarly, CRA worked with its member societies to put together lists of the awards that should be included and to correctly categorize them as "Highly Prestigious" or "Prestigious." This is not a trivial process; for example, does one include the many SIG awards? Again, the deadline to provide the list was tight and we are unable to verify that our list was applied. Thus, it is not clear that the NRC even now has a meaningful method for measuring faculty awards.

Just as troubling is that various member departments have not been able to verify the data that the NRC presents. That is, using the same vita and publication and awards listings, they simply cannot reproduce the numbers that the NRC provides for their departments. The NRC process used temporary workers trained by the NRC staff. Perhaps they were unable to

There are ample reasons why rankings based upon a transparent comprehensive analysis are not done frequently.

deal with the multiple possible titles of publications—*Commun. ACM* = *CACM* = *Communications*—self-reported by faculty on their CVs. The conference publication numbers do not provide much confidence that they were.

One might suggest that the central problem is that computer science is unusual in its practices, and that our field is simply an outlier. This does not appear to be the case. The Council of the American Sociological Association recently passed a resolution condemning the NRC rankings and saying that they should not be used for program evaluation. Input from colleagues suggests that other fields, such as aeronautics/astronautics and chemical engineering are uncomfortable with the NRC process, for many of the reasons we have raised in this Viewpoint.

So we have a situation in which incorrect data are provided for invalid metrics and rankings are calculated using weights that are not readily understood. It would be easy to dismiss the entire process except that institutions are using the results to make programmatic decisions including closing programs. At a recent symposium, many university administrators expressed considerable support for continuing the data collection effort, and generating rankings if it can be accomplished in a meaningful way.

Conclusion

So how should the process work? Here are our suggestions:

- ▶ Work with the relevant societies in order to generate metrics that matter to their constituents.

- ▶ Realize that reputation does matter and include it in the metrics. There is an interesting feedback loop between rankings and reputation, of course. But this also means reputation has some validity as a measure of rank, so incorporate it.

- ▶ Explore making the rankings subdiscipline-dependent. It is clear that different departments have different strengths. Thus, enabling a finer-grained assessment would allow a department with strength in a subfield, but perhaps not the same across-the-board strength, to gain appropriate visibility. This may be particularly valuable for students deciding where to apply.

- ▶ Use data mining to generate scholarly productivity data to replace commercially collected citation data that is incomplete and expensive.

- ▶ Have institutions collect the remaining data under clear guidelines.

- ▶ Provide a time period during which departments can correct errors in the data collected. The NRC did allow institutions to correct some errors of fact, but the allowable corrections did not include publication counts and other information. And the NRC apparently refused to remove data it invented, such as substituting national GRE average scores for institutions that do not record such information.

- ▶ Provide sample weights but allow individuals to develop their own weights and apply them to the collected data so that they can generate rankings of interest to them. We realize this does not satisfy the desire for single overarching rankings. However, it does provide a tool of potential value for individual departments seeking to compare themselves against peers.

We do not claim that this strategy will eliminate all of the many issues with rankings, but it will provide a consistent set of fundamental data that administrators, faculty, students and others can use to understand departmental strengths and weaknesses in a way that matters to them. □

Andrew Bernat (abernat@cra.org) is the executive director of the Computing Research Association in Washington, D.C.

Eric Grimson (welg@csail.mit.edu) is a professor of computer science and engineering at Massachusetts Institute of Technology in Cambridge, MA.

Copyright held by author.

Many modern dynamic languages lack tools for understanding complex failures.

BY DAVID PACHECO

Postmortem Debugging in Dynamic Environments

DESPITE THE BEST efforts of software engineers to produce high-quality software, inevitably some bugs escape even the most rigorous testing process and are first encountered by end users. When this happens, such failures must be understood quickly, the underlying bugs fixed, and deployments patched to avoid another user (or the same one) running into the same problem again. As far back as 1951, the dawn of modern computing, Stanley Gill⁶ wrote that “some attention has, therefore, been given to the problem of dealing with mistakes after the program has been tried and found to fail.” Gill went on to describe the first use of “the post-mortem technique” in software, whereby the running program was modified to record important system state as it ran so that the programmer could later understand what happened and why the software failed.

Since then, postmortem debugging technology has been developed and used in many different systems, including all major consumer and enterprise operating systems, as well as the native execution environments on those systems. These environments make up much of today’s core infrastructure, from the operating systems that underlie every application to core services such as Domain Name System (DNS), and thus form the building blocks of nearly all larger systems. To achieve the high levels of reliability expected from such software, these systems are designed to restore service quickly after each failure while preserving enough information that the failure itself can later be completely understood.

While such software was historically written in C and other native environments, core infrastructure is increasingly being developed in dynamic languages, from Java over the past two decades to server-side JavaScript over the past 18 months. Dynamic languages are attractive for many reasons, not least of which is that they often accelerate the development of complex software.

Conspicuously absent from many of these environments, however, are facilities for even basic postmortem debugging, which makes understanding production failures extremely difficult. Dynamic languages must bridge this gap and provide rich tools for understanding failures in deployed systems in order to match the reliability demanded from their growing role in the bedrock of software systems.

To understand the real potential for sophisticated postmortem analysis tools, we first review the state of debugging today and the role of postmortem analysis tools in other environments. We then examine the unique challenges around building such tools for dynamic environments and the state of such tools today.

Debugging in the Large

To understand the unique value of postmortem debugging, it is worth



ILLUSTRATION BY GARY NEILL

examining the alternative. Both native and dynamic environments today provide facilities for *in situ* debugging, or debugging faulty programs while they're still running. This typically involves attaching a separate debugger program to the faulty program and then directing execution of the faulty program interactively, instruction by instruction or using breakpoints. The user thus stops the program at various points to inspect program state in or-

der to figure out where it goes wrong. Often this process is repeated to test successive theories about the problem.

This technique can be very effective for bugs that are readily reproducible, but it has several drawbacks. First, the act of stopping a program often changes its behavior. Bugs resulting from unexpected interactions between parallel operations (such as race conditions) can be especially challenging to analyze this way because the timing of

various events is significantly affected by the debugger itself. More importantly, *in situ* debugging is often untenable on production systems: many debuggers rely on an unoptimized debug build that's too slow to run in production; engineers often do not have access to the systems where the program is running (as in the case of most mobile and desktop applications and many enterprise systems); and the requisite debugging tools are often not

available on those systems anyway.

Even for cases where engineers can access the buggy software with the tools they need, pausing the program in the debugger usually represents an unacceptable disruption of production service and an unacceptable risk that a fat-fingered debugger command might cause the program to crash. Administrators often cannot take the risk of downtime in order to understand a failure that caused a previous outage. More importantly, they should not have to. Even in 1951 Gill cited the “extravagant waste of machine time involved” in concluding that “single-[step] operation is a useful facility for the maintenance engineer, but the programmer can only regard it as a last resort.”

The most crippling problem with *in situ* debugging is it can only be used to understand reproducible problems. Many production issues are either very rare or involve complex interactions of many systems, which are often very difficult to replicate in a development environment. The rarity of such issues does not make them unimportant: quite the contrary, an operating system crash that happens only once a week can be extremely costly in terms of downtime, but any bug that can be made to occur only once a week is very difficult to debug live. Similarly, a fatal error that occurs once a week in an application used by thousands of people may result in many users hitting the bug each day, but engineers cannot attach a debugger on every user’s system.

So-called `printf` debugging is a

common technique for dealing with the reproducibility issue. In this approach, engineers modify the software to log bits of relevant program state at key points in the code. This causes data to be collected without human intervention so it can be examined after a problem occurs to understand what happened. By automating the data collection, this technique usually results in significantly less impact to production service because when the program crashes, the system can immediately restart it without waiting for an engineer to log in and debug the problem interactively.

Extracting enough information about fatal failures from a log file is often very difficult, however, and frequently it is necessary to run through several iterations of inserting additional logging, deploying the modified program, and examining the output. This, too, is untenable for production systems since ad hoc code changes are often impractical (in the case of desktop and mobile applications) or prohibited by change control policies (and common sense).

The solution is to build a facility that captures *all* program state when the program crashes. In 1980 Douglas R. McGregor and Jon R. Malone⁹ of the University of Strathclyde in Glasgow observed that with this approach “there is virtually no runtime overhead in either space or speed” and “no extra trace routines are necessary,” but the facility “remains effective when a program has passed into production

use.” Most importantly, after the system saves all the program state, it can restart the program immediately to restore service quickly. With such systems in place, even rare bugs can often be root-caused and fixed based on the first occurrence, whether in development, test, or production. This enables software vendors to fix bugs before too many users encounter them.

To summarize, in order to root-cause failures that occur anywhere from development to production, a postmortem debugging facility must satisfy several constraints:

- ▶ Application software must not require modifications that cannot be used in production in order to support postmortem debugging, such as unoptimized code or additional debug data that would significantly impact performance (or affect correctness at all).

- ▶ The facility must be always on: It must not require an administrator to attach a debugger or otherwise enable postmortem support before the problem occurs.

- ▶ The facility must be fully automatic: It should detect the crash, save program state, and then immediately allow the system to restart the failed component to restore service as quickly as possible.

- ▶ The dump (saved state) must be comprehensive: a stack trace, while probably the single most valuable piece of information, very often does not provide sufficient information to root-cause a problem from a single occurrence. Usually engineers want both global state and each thread’s state (including stack trace and each stack frame’s arguments and variables). Of course, there’s a wide range of possible results in this dimension; the “constraint” (such as it is) is that the facility must provide enough information to be useful for nontrivial problems. The more information that can be included in the dump, the more likely engineers will be able to identify the root cause based on just one occurrence.

- ▶ The dump must be transferable to other systems for analysis. This allows engineers to analyze the data using whatever tools they need in a familiar environment and obviates the need for engineers to access production systems in many cases.

Figure 1. A simple MDB example.

```
$ mdb core
Loading modules: [ ld.so.1 ]
> ::status
debugging core file of example1 (32-bit) from solaron
file: /export/home/dap/tmp/example1
initial argv: ./example1
threading model: native threads
status: process terminated by SIGSEGV (Segmentation Fault), addr=10

> ::walk thread | ::findstack -v
stack pointer for thread 1: 8047b98
[ 08047b98 func+0x20() ]
  08047bbc main+0x21(1, 8047bdc, 8047be4)
  08047bd0 _start+0x80(1, 8047cc4, 0, 8047ccf, 8047cdc, 8047ced)

> func+0x20::dis
...
func+0x20:          movl     $0x0, (%eax)
...
```

Postmortem Debugging in Native Environments

To understand the potential value of postmortem debugging in dynamic languages, it is also helpful to examine those areas where postmortem analysis techniques are well developed and widely used. The best examples are operating systems and their native execution environments. Historically this software has comprised core infrastructure; failures at this level of the stack are often very costly either because the systems themselves are necessary for business-critical functions (as in the case of operating systems on which business-critical software is running) or because they are relied upon by business-critical systems upstack (as in the case of infrastructure services such as DNS).

Most modern operating systems can be configured so that when they crash, they immediately save a “crash dump” of all of their state and then reboot. Likewise, these systems can be configured so that when a user application crashes, the operating system saves a “core dump” of all program state to a file and then restarts the application. In most cases, these mechanisms allow the operating system or user application to return to service quickly while preserving enough information to root-cause the failure later.

As an example, let’s look at core dumps under Illumos, an open source Solaris-based system. Take the following broken program:

```
1 int
2 main(int argc, char *argv[])
3 {
4     func();
5     return (0);
6 }
7
8 int
9 func(void)
10 {
11     int ii;
12     int *ptrs[100];
13
14     for (ii = -1; ii < 100; ii++)
15         *(ptrs[ii]) = 0;
16
17     return (0);
18 }
```

This simple program has a fatal flaw:

Figure 2. Analyzing thread stacks.

```
> ::stacks -m zfs
THREAD      STATE      SOBJ      COUNT
fffff0007c0fc60 SLEEP      CV        2
    swtch+0x147
    cv_wait+0x61
    txg_thread_wait+0x5f
    txg_quiesce_thread+0x94
    thread_start+8

fffff0007f51c60 FREE      <NONE>      1
    cpu_decay+0x2f
    bitset_atomic_del+0x38
    apic_setspl+0x5c
    do_splx+0x50
    disp_lock_exit+0x55
    cv_signal+0x96
    taskq_dispatch+0x351
    zio_taskq_dispatch+0x6b
    zio_interrupt+0x1a
    vdev_disk_io_intr+0x6b
    biodone+0x84
    dadk_iodone+0xe7
    dadk_pktcb+0xc6
    ata_disk_complete+0x119
    ata_hba_complete+0x38
    ghd_doneq_process+0xb3
    0x16
    dispatch_softint+0x3f

fffff0007b25c60 SLEEP      CV        1
    swtch+0x147
    cv_timedwait+0xba
    arc_reclaim_thread+0x17b
    thread_start+8

fffff0007b2bc60 SLEEP      CV        1
    swtch+0x147
    cv_timedwait+0xba
    l2arc_feed_thread+0xa5
    thread_start+8

fffff0009b95c60 SLEEP      CV        1
    swtch+0x147
    cv_timedwait+0xba
    txg_thread_wait+0x7b
    txg_sync_thread+0x114
    thread_start+8

fffff01e26d08e0 SLEEP      CV        1
    swtch+0x147
    cv_wait+0x61
    txg_wait_synced+0x7f
    spa_sync_allpools+0x76
    zfs_sync+0xce
    vfs_sync+0x9c
    syssync+0xb
    sys_syscall32+0x101

fffff0007c15c60 SLEEP      CV        1
    swtch+0x147
    cv_wait+0x61
    zio_wait+0x5d
    dsl_pool_sync+0xe1
    spa_sync+0x32a
    txg_sync_thread+0x265
    thread_start+8
```

while trying to clear each item in the `ptrs` array at lines 14–15, it clears an extra element before the array (where `ii = -1`). When running this program, you see:

```
$ gcc -o example1 example1.c
$ ./example1
Segmentation Fault (core dumped)
```

and the system generates a file called `core`. The Illumos modular debugger (MDB) shown in Figure 1 can help in examining this file.

MDB's syntax may seem arcane to new users, but this example is rather basic. First the `::status` command produces a summary of what happened: the process was terminated as a result of a segmentation fault attempting to access memory address `0x10`. Next the `::walk thread | ::find-stack -v` command is used to examine thread stacks (in this case, just one), and it shows that the program died in function `func` at offset `0x20` in the program text. Then the file dumps out this instruction to see that the process died on the store of `0` into the address contained in register `%eax`.

While this example is admittedly contrived, it illustrates the basic method of postmortem debugging. Note that unlike *in situ* debugging, this method scales well with the complexity of the program being debugged. If instead of one thread in one process there were thousands of threads across dozens of components (as in the case of an operating system), a comprehensive dump would include information about all of them. The next challenge would be making sense of so much information, but root-causing the bug is at least tractable because all the information is available.

In such situations, the next step is to build custom tools for extracting, analyzing, and summarizing specific component state. A comprehensive postmortem facility enables engineers to build such tools. For example, `gdb` supports user-defined macros. These macros can be distributed with the source code so that all developers can use them both *in situ* (by attaching `gdb` to a running process) and postmortem (by opening a core file with `gdb`). The Python interpreter, for example, provides such macros, allowing both inter-

preter and native module developers to pick apart the C representations of Python-level objects.

MDB takes this idea to the next level: it was designed specifically around building custom tools for understanding specific components of the system both *in situ* and postmortem. On Illumos systems, the kernel ships with MDB modules that provide more than 1,000 commands to iterate and inspect various components of the kernel. Among the most frequently used is the `::stacks` command, which iterates all kernel threads, optionally filters them based on the presence of a particular kernel module or function in the stack trace, and then dumps out a list of unique thread stacks sorted by frequency. Figure 2 offers an example from a system doing some light I/O.

This invocation collapsed the complexity of more than 600 threads on this system to only about seven unique thread stacks that are related to the ZFS file system. You can quickly see the state of the threads in each group (e.g., sleeping on a condition variable) and examine a representative thread for more information. Dozens of other operating-system components deliver their own MDB commands for inspecting specific component state, including the networking stack, the NFS server, `DTrace`, and ZFS.

Some of these higher-level analysis tools are quite sophisticated. For example, the `::typegraph` command³ analyzes an entire production crash dump (without debug data) and constructs a graph of object references and their types. With this graph, users can query the type of an arbitrary memory object. This is useful for understanding memory corruption issues, where the main problem is identifying which component overwrote a particular block of memory. Knowing the type of the corrupting object narrows the investigation from the entire kernel to the component responsible for that type.

Such tools are by no means limited to production environments. On most systems, it is possible to generate a core dump from *running* processes too, which make core-dump analysis attractive during development as well. When testers or other engineers file bugs on application crashes, it is often easier to have them include a

core dump than to try to guess what steps they took that led to the crash and then reproduce the problem from those steps. Examining the core dump is also the only way to be sure the problem you found is the same one the bug reporter encountered.

Higher-level dump analysis tools can be built explicitly for development as well. `Libumem`, a drop-in replacement for `malloc(3c)` and friends, provides (among other features) an MDB module for iterating and inspecting objects related to the allocator. Combined with an optional feature to record stack traces for each allocator operation, the `::findleaks` MDB command can be used to identify various types of memory leaks very quickly without having added any explicit support for this in the application itself. The `::findleaks` command literally prints out a list of leaked objects and the stack trace from which each one was allocated—pointing directly to the location of each leak. `Libumem` is based on the kernel memory allocator, which provides many of the same facilities for the kernel.²

Postmortem Debugging in Dynamic Environments

While operating-system and native environments have highly developed facilities for handling crashes, saving dumps, and analyzing them postmortem, the problem of postmortem analysis (and software observability more generally) is far from solved in the realm of dynamic environments such as Java, Python, and JavaScript. In the past postmortem analysis was arguably less critical for these languages because crashes in these environments are less significant: most end-user applications save work frequently anyway, and the operating system or browser will often restart the application after a crash. These crashes still represent disruptions to the user experience, however, and postmortem debugging is the only hope of understanding such failures.

More importantly, dynamic languages such as `Node.js` are exploding in popularity as building blocks for larger distributed systems, where what might seem like a minor crash can cause cascading failures up the stack. As a result, just as with operating systems and core services, fully understanding each failure is essential to achieving the levels

of reliability expected of such foundational software.

Providing a postmortem facility for dynamic environments, however, is not easy. While native programs can leverage operating-system support for core dumps, dynamic languages must present postmortem state using the same higher-level abstractions with which their developers are familiar. A postmortem environment for C programs can simply present a list of global symbols, pointers to thread stacks, and all of a process's virtual memory (all of which the operating system has to maintain anyway), but a similar facility for Java must augment (or replace) these with analogous Java abstractions. When Java programs crash, Java developers want to look at Java thread stacks, local variables, and objects, not (necessarily) the threads, variables, and raw memory used by the Java virtual machine (JVM) implementation. Also, because programs in dynamic languages run inside an interpreter or VM, when the user program "crashes," the interpreter or VM itself does not crash. For example, when a Python program uses an undefined variable (the C equivalent of a NULL pointer), the interpreter detects this condition and gracefully exits. Therefore, to support postmortem debugging, the interpreter would need to trigger the core-dump facility explicitly, not rely on the operating system to detect the crash.

In some cases, presenting useful postmortem state requires formalizing abstractions that do not exist explicitly in the language at all. JavaScript presents a particularly interesting challenge in this regard. In addition to the usual global state and stack details, JavaScript maintains a pending event queue, as well as a collection of events that may happen later—both of which exist only as functions with associated context that will be invoked at some later time by the runtime. For example, a Web browser might have many outstanding asynchronous HTTP requests. For each one, there is a function with associated context that may not be reachable from the global scope, and so would not be included in a simple dump of all global state and thread state. Nevertheless, understanding which of these requests are outstanding and what state is associ-



The most crippling problem with *in situ* debugging is it can only be used to understand reproducible problems.



ated with them may very well be critical to understanding a fatal failure.

This problem is even more acute with Node.js on the server, which is frequently used to manage thousands of concurrent connections to many different types of components. A single Node program might have hundreds of outstanding HTTP requests, each one waiting on a database query to complete. The program may crash while processing one of the database query results because it encountered an invalid database state resulting from one of the other outstanding queries. Such problems beg for postmortem debugging because each instance is seen relatively rarely; they are essentially impossible to understand from just a stack trace, but they can often be identified from the first occurrence, given enough information from the time of the crash. The challenge is presenting information about outstanding asynchronous events (that is, callbacks that will be invoked at some future time) in a meaningful way to JavaScript developers, who generally do not have direct access to the event queue or the collection of outstanding events; these abstractions are implicit in the underlying APIs, so exposing this requires first figuring out how to express these abstractions.

Finally, user-facing applications have the additional problem of transferring postmortem state from the user's computer to developers who can root-cause the bug (while preserving user privacy). As Eric Schrock¹¹ details, this problem remains largely unsolved for one of the most significant dynamic environments today: the JavaScript Web application. There is no browser-based facility for automatically uploading postmortem program state back to the server.

Despite these difficulties, some dynamic environments do provide postmortem facilities. For example, the Oracle Java HotSpot VM supports extracting Java-level state from JVM native core dumps. When the JVM crashes, or when a core file is manually created using operating system tools such as `gcore(1)`, you can use the `jdb(1)` tool to examine the state of the Java program (rather than the JVM itself) when the core file was generated. The core file can also be processed by a tool called `jmap(1)` to create a Java heap dump that can in turn be analyzed us-

ing several different programs.

Such facilities are only a start, however: setting up an application to trigger a core dump on crash in the first place is nontrivial. Additionally, these facilities are very specific to the HotSpot VM. There's an active Java Community Specification proposal for a common API to access debugging information, but at the time of this writing this project is stalled pending clarity about Oracle's commitment to the project.⁸

While the Java facility has several important limitations, many other dynamic environments do not appear to have postmortem facilities at all—at least not any that meet the constraints just described.

Python¹⁰ and Ruby⁴ each has a facility called a postmortem debugger, but these refer to starting a program under a debugger and having the program break into an interactive debugger session when the program crashes. This is not suitable for production for several reasons, not least of which is that it is not fully automatic. As described earlier, it is not tenable to interrupt production service while an engineer logs in to diagnose a problem interactively.

Erlang⁵ provides a rich crash-dump facility for the Erlang runtime itself. It works much like a native crash dump in that on failure it saves a comprehensive state dump to a file and then exits, allowing the operating system to see the program has exited and restart it immediately. The crash dump file can then be analyzed later.

The bash shell¹ is interesting because its deployment model is so different even from other dynamic environments. Bash provides a mechanism called `xtrace` for producing a comprehensive trace file describing nearly every expression that the shell evaluates as part of execution. This is very useful for understanding shell script failures but can produce a lot of output even for simple scripts. The output grows unbounded as the program runs, which would normally make it untenable for production use in servers or applications, but since most bash scripts have very finite lifetimes, this mechanism is an effective postmortem facility as long as the output can be reasonably stored and managed (that is, automatically deleted after successful runs).

JavaScript, unlike many of the



There is no browser-based facility for automatically uploading postmortem program state back to the server.



above languages, is widely deployed under several completely different runtime environments such as Mozilla's SpiderMonkey, Google's V8 (used in both Chrome and Node.js) and the WebKit JavaScript engine. Although JavaScript *in situ* debugging facilities have improved substantially in recent years in the form of improved browser support for runtime program inspection, there remains no widely used postmortem facility for JavaScript.

A Primitive Postmortem Facility for Node.js

Despite the lack of JavaScript language support, we have developed a crude but effective postmortem debugging facility for use in Joyent's Node.js production deployments. Recall that Node typically runs on a server rather than a Web browser and is commonly used to implement services that scale to hundreds or thousands of network connections. We use the following primitives provided by Node and the underlying V8 virtual machine to construct a simple implementation:

- ▶ An `uncaughtException` event, which allows a program to register a function to be invoked when the program throws an exception that bubbles all the way to the top level (that would normally cause the program to crash).
- ▶ Built-in mechanisms for serializing/deserializing simple JavaScript objects as a text string (`JSON.stringify()` and `JSON.parse()`).
- ▶ Synchronous functions for writing to files.

The first challenge is actually identifying which state to dump. JavaScript provides a way to introspect global state, but Node.js programs that declare variables do not use global state *per se*. What looks like the top-level scope is actually contained inside a function scope, and function scopes cannot be introspected. To work around this, programs using our postmortem facility must explicitly register debugging state ahead of time. While this solution is deeply unsatisfying because it is always difficult to know ahead of time what information would be useful to have when debugging, it has proved effective in practice because each of our programs essentially just instantiates a singleton object representing the program itself and then registers that with the post-

mortem facility. Most relevant program state is referenced by this pseudo-global object in one way or another.

The next challenge is serializing circular objects. `JSON.stringify()` does not support this for obvious reasons, so our implementation avoids this issue by pruning all circular references before serializing the debug object. While this makes it harder to find information in the dump, we know that at least one copy of every object will be present somewhere in the dump.

Given all this, the implementation is straightforward: on the `uncaughtException` event, we prune circular references from the debug state, serialize it using the built-in `JSON.stringify()` routine, and save the result to disk in a file called `core`. To analyze the core file, we use a tool that reads `core` using `JSON.parse()` and presents the serialized state for engineers to examine. The implementation is open source and available on GitHub.⁷

In addition to the implementation challenges just described, this approach has several significant limitations. First, it can save only state that programmers can register ahead of time, but as already discussed, there is a great deal of other important state inside a JavaScript program such as function arguments in the call stack and the contexts associated with pending and future events, none of which is reachable from the global scope.

Second, since the entire point of this system is to capture program state in the event of a crash, it must be highly reliable. This implementation is robust to most runtime failures, but it still requires additional memory first to execute the dump code and to serialize the program state. The additional memory could easily be as large as the whole heap, which makes it untenable for failures resulting from memory pressure—a common cause of failures in dynamic environments.

Third, because the implementation removes circular references before serializing the program data, the resulting dump is more difficult to browse, and the facility cannot support dumps that are not intended for postmortem analysis (such as live dumps).

Despite these deficiencies, this implementation has proved quite effective because it meets the require-

ments set forth earlier: it is always-on in production, fully automatic, the result is transferable to other systems for analysis, and it is comprehensive enough to solve complex problems. To address many of the scope, robustness, and richness problems described here, however, and to provide such a facility for all users of a language, the postmortem facility must be incorporated into the VM itself. Such an implementation would work similarly in principle, but it could include absolutely all program state, be made to work reliably in the face of failure of the program itself, stream the output to avoid using much additional memory, and use a format that preserves the underlying memory structures to ease understanding of the dump. Most importantly, including tools for postmortem analysis out of the box would go a long way toward the adoption of postmortem techniques in these environments.

Conclusion

Postmortem debugging facilities have long enabled operating-system engineers and native-application developers to understand complex software failures from the first occurrence in deployed systems. Such facilities form the backbone of the support process for enterprise systems and are essential for software components at the core of a complex software environment. Even simple platforms for recording postmortem state enable engineers to develop sophisticated analysis tools that help them to quickly root-cause many types of problems.

Meanwhile, modern dynamic languages are growing in popularity because they so effectively facilitate rapid development. Environments such as Node.js also promote programming models that scale well, particularly in the face of latency bubbles. This is becoming increasingly important in today's real-time systems.

Postmortem debugging for dynamic environments is still in its infancy. Most such environments, even those considered mature, do not provide any facility for recording postmortem state, let alone tools for higher-level analysis of such failures. Those tools that do exist are not first-class tools in their respective environments and so are not widely used. As dynamic languages grow in

popularity for building critical software components, this gap is becoming increasingly important. Languages that ignore the problems associated with debugging production systems will increasingly be relegated to solving simpler, well-confined, well-understood problems, while those that provide rich tools for understanding failure postmortem will form the basis of the next generation of software bedrock.

Acknowledgments

Many thanks to Bryan Cantrill, Peter Memishian, and Theo Schlossnagle for reviewing earlier drafts of this article and to Adam Cath, Ryan Dahl, Robert Mustacchi, and many others for helpful discussions on this topic. ☐

Related articles on queue.acm.org

Erlang for Concurrent Programming

Jim Larson

<http://queue.acm.org/detail.cfm?id=1454463>

Orchestrating an Automated Test Lab

Michael Donat

<http://queue.acm.org/detail.cfm?id=1046946>

Scripting Web Services Prototypes

Christopher Vincent

<http://queue.acm.org/detail.cfm?id=640158>

References

1. Bash Reference Manual (2009); <https://www.gnu.org/s/bash/manual/bash.html>.
2. Bonwick, J. The slab allocator: An object-caching kernel memory allocator. *Usenix Summer 1994 Technical Conference*.
3. Cantrill, B.M. Postmortem object type identification. In *Proceedings of the 5th International Workshop on Automated and Algorithmic Debugging*. (2003)
4. Debugging with ruby-debug. 2011; http://bashdb.sourceforge.net/ruby-debug.html#Post_002dMortem-Debugging.
5. Erlang Runtime System Application User's Guide, version 5.8.4. 2011. How to interpret the Erlang crash dumps; http://www.erlang.org/doc/apps/erts/crash_dump.html.
6. Gill, S. The diagnosis of mistakes in programmes on the EDSAC. In *Proceedings of the Royal Society A 206* (1951), 538–554.
7. GitHub Project. 2011; <https://github.com/joyent/node-panic>
8. Incubator Wiki. March 2011 Board reports. Kato Project; <http://wiki.apache.org/incubator/March2011>.
9. McGregor, D.R., Malone, J.R. Stabdump—A dump interpreter program to assist debugging. *Software Practice and Experience* 10, 4 (1980), 329–332.
10. Python Standard Library. Python v2.7.2 documentation pdb—the Python debugger; 2011; <http://docs.python.org/library/pdb.html>.
11. Schrock, E. Debugging AJAX in production. *ACM Queue* 7, 1 (2009); <http://queue.acm.org/detail.cfm?id=1515745>.

David Pacheco is an engineer at Joyent where he leads the design and implementation of Cloud Analytics, a real-time Node.js/DTrace-based system for visualizing server and application performance in the cloud. Previously a member of the Sun Microsystems Fishworks team, he worked on several features of the Sun Storage 7000 appliances.

© 2011 ACM 0001-0782/11/12 \$10.00

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

Astronomers are collecting more data than ever. What practices can keep them ahead of the flood?

BY G. BRUCE BERRIMAN AND STEVEN L. GROOM

How Will Astronomy Archives Survive the Data Tsunami?

ASTRONOMY IS ALREADY awash with data: currently 1PB (petabyte) of public data is electronically accessible, and this volume is growing at 0.5PB per year. The availability of this data has already transformed research in astronomy, and the Space Telescope Science Institute (STScI) now reports that more papers are published with archived data sets than with newly acquired data.¹⁸

This growth in data size and anticipated usage will accelerate in the coming few years as new projects

such as the Large Synoptic Survey Telescope (LSST), Atacama Large Millimeter Array (ALMA), and Square Kilometer Array (SKA) move into operation. These new projects will use much larger arrays of telescopes and detectors or much higher data acquisition rates than are now used. Projections indicate that by 2020, more than 60PB of archived data will be accessible to astronomers.¹⁰

The data tsunami is already affecting the performance of astronomy archives and data centers. One example is the NASA Infrared Processing and Analysis Center (IPAC) Infrared Science Archive (IRSA), which archives and serves data sets from NASA's infrared missions. It is going through a period of exceptional growth in its science holdings, as shown in Figure 1, because it is assuming responsibility for the curation of data sets released by the Spitzer Space Telescope and Wide-field Infrared Survey Explorer (WISE) mission.

The volume of these two data sets alone exceeds the total volume of the 35-plus missions and projects already archived. The availability of the data, together with rapid growth in program-based queries, has driven up usage of the archive, as shown by the annual growth in downloaded data volume and queries in Figure 2. Usage is expected to accelerate as new data sets are released through the archive, yet the response times to queries have already suffered, primarily because of a growth in requests for large volumes of data.

The degradation in performance cannot be corrected simply by adding infrastructure as usage increases, as is common in commercial enterprises, because astronomy archives generally operate on limited budgets that are fixed for several years. Without intervention, the current data-access and computing model used in astronomy, in which data downloaded from archives is analyzed on local machines, will break down rapidly. The very scale of data sets such as those just described will transform the design and operation of archives as places



that not only make data accessible to users, but also support in situ processing of these data with the end users' software: network bandwidth limitations prevent transfer of data on this scale, and users' desktops in any case generally lack the power to process PB-scale data.

Moreover, data discovery, access, and processing are likely to be distributed across several archives, given that the maximum science return will involve federation of data from several archives, usually over a broad wavelength range, and in some cases will involve confrontation with large and complex simulations. Managing the impact of PB-scale data sets on archives and the community was recognized as an important infrastructure issue in the report of the 2010 Decadal Survey of Astronomy and

Astrophysics,⁶ commissioned by the National Academy of Sciences to recommend national priorities in astronomy for the coming decade.

Figure 3 illustrates the impact of the growth of archive holdings. As holdings grow, so does the demand for data, for more sophisticated types of queries, and for new areas of support, such as analysis of massive new data sets to understand how astronomical objects vary with time, described in the 2010 Decadal Survey as the "last frontier in astronomy." Thus, growth in holdings drives up storage costs, as well as compute and database costs, and the archive must bear all of these costs. Given that archives are likely to operate on shoe-string budgets for the foreseeable future, the rest of this article looks at

strategies and techniques for managing the data tsunami.

At the Innovations in Data-intensive Astronomy workshop earlier this year (Green Bank, WV, May 2011¹⁵) participants recognized that the problems of managing and serving massive data sets will require a community effort and partnerships with national cyber-infrastructure programs. The solutions will require rigorous investigation of emerging technologies and innovative approaches to discovering and serving, especially as archives are likely to continue to operate on limited budgets. How can archives develop new and efficient ways of discovering data? When should, for example, an archive adopt technologies such as graphical processing units (GPUs) or cloud computing? What kinds of

Figure 1. Growth in the scientific data holdings of IRSA, projected to 2014. The graphic calls out the dramatic impact of the Spitzer and WISE missions on the volume of the archive's science data holdings.

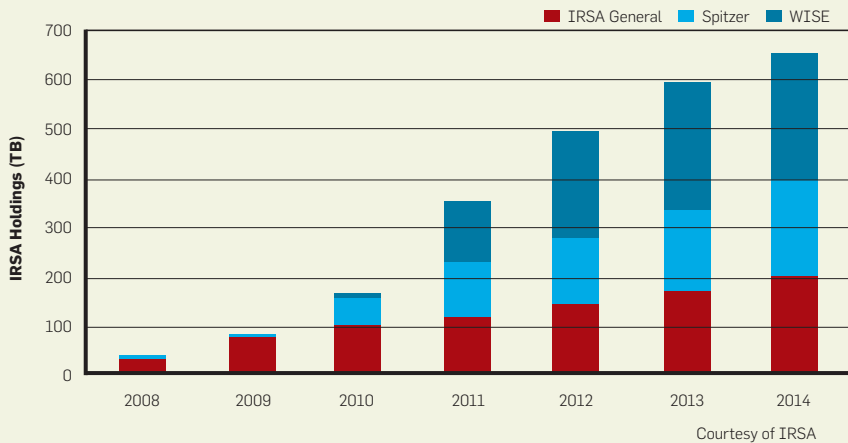
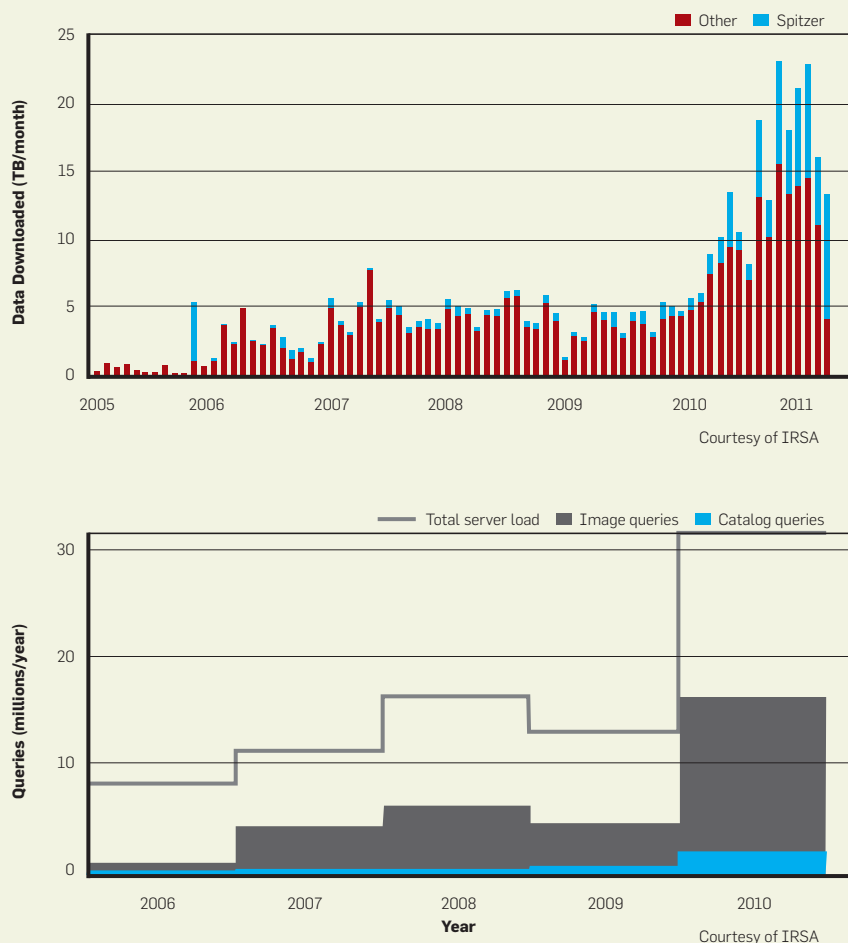


Figure 2. Growth in usage of IRSA from 2005 until the beginning of 2011. WISE data was not available until spring 2011.



technologies are needed to manage distribution of data time, compute-intensive data-access jobs, and end-user processing jobs?

This article emphasizes those issues we believe must be addressed by archives to support their end users in the coming decade, as well as those issues that affect end users in their interactions with archives.

Innovations in Serving and Discovering Data

The discipline of astronomy needs new data-discovery techniques that respond to the anticipated growth in the size of data sets and that support efficient discovery of large data sets across distributed archives. These techniques must aim to offer data discovery and access across PB-sized data sets (for example, discovering images over many wavelengths over a large swath of the sky such as the Galactic Plane) while preventing excessive loads on servers.

The Virtual Astronomical Observatory (VAO),¹⁹ part of a worldwide effort to offer seamless international astronomical data-discovery services, is exploring such techniques. It is developing an R-tree-based indexing scheme that supports fast, scalable access to massive databases of astronomical sources and imaging data sets.⁹ (R-trees are tree data structures used for indexing multidimensional information. They are commonly used to index database records and thereby speed up access times.)

In the current implementation, the indices are stored outside the database, in memory-mapped files that reside on a dedicated Linux cluster. It offers speedups of up to 1,000 times over database table scans and has been implemented on databases containing two billion records and TB-scale image sets. It is already in operation in the Spitzer Space Telescope Heritage Archive and the VAO Image and Catalog Discovery Service. Expanding techniques such as this to PB-scale data is an important next step.

Such custom solutions may prove more useful than adapting an expensive geographical information system (GIS) to astronomy. These systems are necessarily more complex than are needed in astronomy, where the celestial sphere is by definition a perfect sphere and the footprints on the sky of

instruments and data sets are generally simple geometric shapes.

Investigations of Emerging Technologies

A growing number of investigators are taking part in a concerted and rigorous effort to understand how archives and data centers can take advantage of new technologies to reduce computational and financial costs.

Benjamin Barsdell et al.¹ and Christopher Fluke et al.⁷ have investigated the applicability of GPUs to astronomy. Developed to accelerate the output of an image on a display device, GPUs consist of many floating-point processors. These authors point out that speed ups of more than 100 times promised by manufacturers strictly apply to graphics-like applications; GPUs support single-precision calculations rather than the double precisions often needed in astronomy; and their performance is often limited by data transfer to and from the GPUs. The two studies cited here indicate applications that submit to “brute-force parallelization” will give the best performance with minimum development effort; they show that code profiling will likely help optimization and provide a first list of the types of astronomical applications that may benefit from running on GPUs. These applications include fixed-resolution mesh simulations, as well as machine-learning and volume-rendering packages.

Others are investigating how to exploit cloud computing for astronomy. Applications best suited for commercial clouds are those that are processing and memory intensive, which take advantage of the relatively low cost of processing *under current fee structures*.² Applications that are I/O intensive, which in astronomy often involve processing large quantities of image data, are, however, uneconomical to run because of the high cost of data transfer and storage. They require high-throughput networks and parallel file systems to achieve best performance.

Under current fee structures, renting mass storage space on the Amazon cloud is more expensive than purchasing it. Neither option offers a solution to the fundamental business problem that storage costs scale with volume, while funding does not. Any use of com-

mercial clouds should be made after a thorough cost-benefit study. It may be that commercial clouds are best suited for short-term tasks, such as regression testing of applications and handling excessive server load, or to one-time bulk-processing tasks, as well as supporting end-user processing.

Implementing and managing new technologies always have a business cost, of course. Shane Canon⁴ and others have provided a realistic assessment of the business impact of cloud computing. Studies such as these are needed for all emerging technologies.

Despite the high costs often associated with clouds, the virtualization technologies used in commercial clouds may prove valuable when used within a data center. Indeed, the Canadian Astronomy Data Center (CADC) is moving its entire operation to an academic cloud called Canadian Advanced Network for Astronomical Research (CANFAR), “an operational system for the delivery, processing, storage, analysis, and distribution of very large astronomical datasets. The goal of CANFAR is to support large Canadian astronomy projects.”¹¹ To our knowledge, this is the first astronomy archive that has migrated to cloud technologies.⁸ It can be considered a first model of the archive of the future, and consequently the community

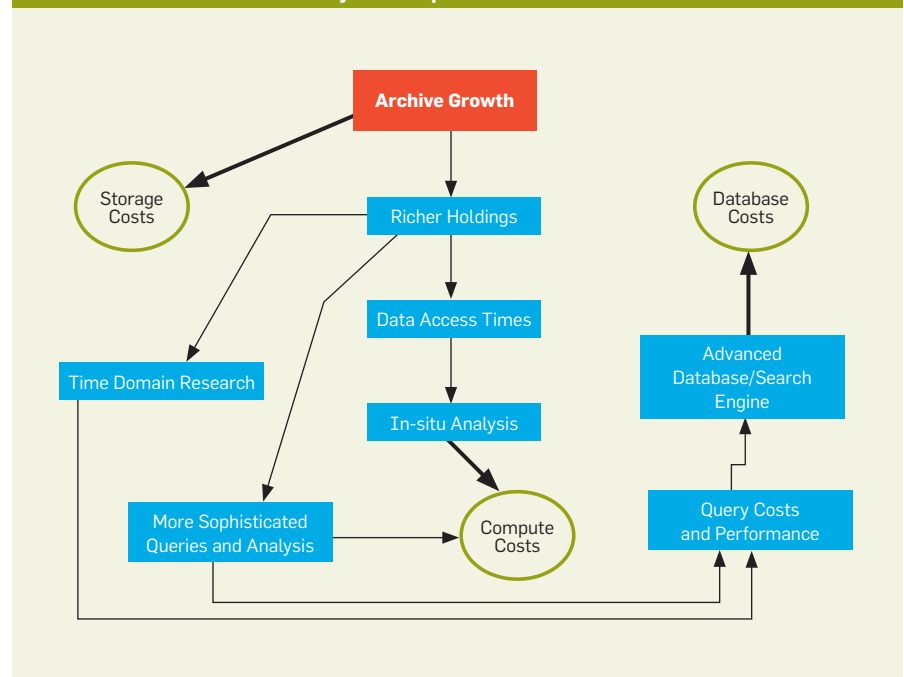
should monitor its performance.

The SKA has rejected the use of commercial cloud platforms. Instead, after a successful prototyping experiment, it proposes a design based on the open source Nereus V Cloud²⁰ computing technology, selected because of its Java codebase and security features. The prototype test bed used 200 clients at the University of Western Australia, Curtin University, and iVEC, with two servers deployed through management at a NereusCloud domain. The clients include Mac Minis and Linux-based desktop machines. When complete, “theskynet,” as it has been called, would provide open access to the SKA data sets for professionals and citizen scientists alike.¹² The design offers a cheaper and much greener alternative to earlier designs based exclusively on a centrally based GPU cluster.

Compute Infrastructure

Astronomy needs to engage and partner with national cyber infrastructure initiatives. Much of the infrastructure to optimize task scheduling and workflow performance and to support distributed processing of data is driven by the needs of science applications. Indeed, the IT community has adopted the Montage image mosaic engine³ to develop infrastructure (for example, task schedulers in distributed environ-

Figure 3. Schematic representation of how growth in data holdings drives up demands on the archive's services and thereby drives up the archive's costs.



ments and workflow optimization techniques). These efforts have not, however, been formally organized, and future efforts may well benefit from such.

Cultural changes. There is at present no effective means of disseminating the latest IT knowledge to the astronomical community. Information is scattered across numerous journals and conference proceedings. To rectify this, we propose an interactive online journal dedicated to information technology in astronomy or even physical sciences as a whole.

Even more important is the need to change the reward system in astronomy to offer recognition for computational work. This would help retain quality people in the field.

Finally, astronomers must engage the computer science community to develop science-driven infrastructure. The SciDB database,¹⁶ a PB-scale next-generation database optimized for science applications, is an excellent example of such collaboration.

Educational Changes. An archive model that includes processing of data on servers local to the data will have profound implications for end users, who generally lack the skills not only to manage and maintain software, but also to develop software that is environment agnostic and scalable to large data sets. Zeeya Merali¹⁴ and Igor Chilingarian and Ivan Zolotukhin⁵ have made compelling cases that self-teaching of software development is the root cause of this phenomenon. Chilingarian and Zolotukhin in particular present some telling examples of clumsy and inefficient design in astronomy.

One solution would be to make software engineering a mandatory part of graduate education, with a demonstration of competency as part of the formal requirements for graduation. Just as classes in instrumentation prepare students for a career in which they design experiments to obtain new data, so instruction in computer science prepares them for massive data-mining and processing tasks. Software has become, in effect, a scientific instrument.

The software engineering curriculum should include the principles of software requirements, design, and maintenance (version control, documentation, basics of design for ade-

quate testing); how a computer works and what limits its performance; at least one low-level language and one scripting language, development of portable code, parallel-processing techniques, principles of databases, and how to use high-performance platforms such as clouds, clusters, and grids. Teaching high-performance computing techniques is particularly important, as the load on servers needs to be kept under control. Such a curriculum would position astronomers to develop their own scalable code and to work with computer scientists in supporting next-generation applications.

Curricula designers can take advantage of existing teaching methods. Software Carpentry¹⁷ is an open source project that provides online classes in the basics of software engineering and encourages contributions from its user community. Frank Löffler et al.¹³ described a graduate class in high-performance computing at Louisiana State University in which they used the TeraGrid to instruct students in high-performance computing techniques that they could then use in day-to-day research. Students were given hands-on experience at running simulation codes on the TeraGrid, including codes to model black holes, predict the effects of hurricanes, and optimize oil and gas production from underground reservoirs.

Conclusion

The field of astronomy is starting to generate more data than can be managed, served, and processed by current techniques. This article has outlined practices for developing next-generation tools and techniques for surviving this data tsunami, including rigorous evaluation of new technologies, partnerships between astronomers and computer scientists, and training of scientists in high-end software engineering skills. 

Related articles on queue.acm.org

Why Your Data Won't Mix

Alon Halevy

<http://queue.acm.org/detail.cfm?id=1103836>

If You Have Too Much Data, then "Good Enough" Is Good Enough

Pat Helland

<http://queue.acm.org/detail.cfm?id=1988603>

Information Extraction: Distilling Structured Data from Unstructured Text

Andrew McCallum

<http://queue.acm.org/detail.cfm?id=1105679>

References

1. Barsdell, B.R., Barnes, D.G. and Fluke, C. J. Analysing astronomy algorithms for graphics processing units and beyond. *Monthly Notices of the Royal Astronomical Society* 408, 3 (2010), 1936–1944.
2. Berriman, G.B., Deelman, E., Juve, G., Regelson, M. and Plavchan, P. The application of cloud computing to astronomy: A study of cost and performance. Accepted for publication in *Proceedings of the e-Science in Astronomy Conference* (Brisbane, AU, 2011).
3. Berriman, G.B., Good, J., Deelman, E. and Alexov, A. Ten years of software sustainability at the Infrared Processing and Analysis Center. *Phil. Trans. R. Soc.* 369 (2011), 3384–3397.
4. Canon, S. Debunking some common misconceptions of science in the cloud. Presented at ScienceCloud2011: 2nd Workshop on Scientific Cloud Computing (San Jose, CA); <http://datasys.cs.it.edu/events/ScienceCloud2011/>.
5. Chilingarian, I. and Zolotukhin, I. The true bottleneck of modern scientific computing in astronomy. *Astronomical Data Analysis Software and Systems XX*. I. Evans et al., Eds. *ASP Conference Series* 442, 471 (2010).
6. Committee for a Decadal Survey of Astronomy and Astrophysics, National Research Council of the National Academy of Sciences. *New Worlds, New Horizons in Astronomy and Astrophysics*, 2010.
7. Fluke, C.J., Barnes, D.G., Barsdell, B.R., Hassan, A.H. Astrophysical supercomputing with GPUs: critical decisions for early adopters. *Publications of the Astronomical Society of Australia* 28, 15 (2011).
8. Gaudet, S. et al. CANFAR: The Canadian Advanced Network for Astronomical Research. *Software and Cyber Infrastructure for Astronomy*. N. Radziwill and A. Bridger, Eds. SPIE 7740, 1I (2010).
9. Good, J. Private communication, 2011.
10. Hanisch, R.J. Data discovery, access, and management with the virtual observatory. 2011, Paper presented at Innovations in Data-intensive Astronomy, Green Bank, WV, 2011; http://www.nrao.edu/meetings/bigdata/presentations/May5/1-Hanisch/Hanisch_VAO_Green_Bank.ppt.
11. Hemsoth, N. Canada explores new frontiers in astroinformatics. *HPC in the Cloud*; http://www.hpcinthecloud.com/hpccloud/2011-01-17/canada_explores_new_frontiers_in_astroinformatics.html.
12. Hutchinson, J. SKA bid looks to SkyNet for computing, 2011; http://www.cio.com.au/article/387097/exclusive_ska_bid_looks_skyenet_computing/.
13. Löffler, F., Allen, G., Bengner, W., Hutanu, A., Jha, S. and Schnetter, E. Using the TeraGrid to teach scientific computing. TeraGrid '11: Extreme Digital Discovery Conference (Salt Lake City, UT, July 18–21, 2011); <https://www.teragrid.org/web/tg11/home>.
14. Merali, Z. Why scientific programming does not compute. *Nature* 467 (2010) 775–777.
15. National Radio Astronomy Observatory. Innovations in Data-intensive Astronomy Workshop (Green Bank, WV, May 3–5, 2011); <http://www.nrao.edu/meetings/bigdata/index.shtml>.
16. SciDB Open Source Data Management and Analytics Software System. 2011; <http://scidb.org>.
17. Software Carpentry; <http://software-carpentry.org/>.
18. Space Telescope Science Institute. Hubble Space Telescope Publication Statistics 1991–2010 (2011); <http://archive.stsci.edu/hst/bibliography/pubstat.html>.
19. Virtual Astronomical Observatory; <http://us-vao.org>.
20. Nereus overview. http://www-nereus.physics.ox.ac.uk/about_overview.html

G. Bruce Berriman is a senior scientist at Infrared Processing and Analysis Center (IPAC). He is the program manager for the Virtual Astronomical Observatory and project manager for the W. M. Keck Observatory Archive and was formerly the manager of the NASA/IPAC Infrared Science Archive.

Steven L. Groom is a systems engineer at IPAC and is manager of the NASA/IPAC Infrared Science Archive. He has worked with mass storage, parallel processing, and data archiving in the space sciences, as well as commercial applications.

© 2011 ACM 0001-0782/11/12 \$10.00



What separates good code from great code?

BY ROBERT GREEN AND HENRY LEDGARD

Coding Guidelines: Finding the Art in the Science

COMPUTER SCIENCE IS both a science and an art. Its scientific aspects range from the theory of computation and algorithmic studies to code design and program architecture. Yet, when it comes time for implementation, there is a combination of artistic flare, nuanced style, and technical prowess that separates good code from great code.

Like art, code is simultaneously subjective and non-subjective. The non-subjective aspects of coding include “hard” ideas that must be followed to create good code: design patterns, project structures, the use of common libraries, and so on. Although these concepts lay the foundation for developing high-quality, maintainable code, it is the nuances of a programmer’s technique and tools—alignment, naming, use of white space, use of context, syntax highlighting, and IDE choice—that truly make code clear, maintainable, and understandable, while also giving

code the ability to clearly communicate intent, function, and usage.

This separation between good and great code occurs because every person has an affinity for his or her own particular coding style based on his or her own good (or bad) habits and preferences. Anyone can write code within a design pattern or using certain “hard” techniques, but it takes a great programmer to fill in the details of the code in way that is clear, concise, and understandable. This is important because just as every person may draw a unique meaning or experience from a single piece of artwork, every devel-

oper or reader of code may infer different meanings from the code depending on naming and other conventions, despite the architecture and design of the code.

From another angle, programming may also be seen as a form of “encryption.” In various ways the programmer devises a solution to a problem and then encrypts the solution in terms of a program and its support files. Months or years later, when a change is called for, a new programmer must decrypt the solution. This is usually not an enviable task, which can mainly be blamed on a failure of clear communication during the initial “encryption” of the project. Decrypting information is simple when the necessary key is present. So, too, is understanding old code when special attention has been paid to what the code itself communicates.

To address this issue, some works

have defined a single coding standard for an entire programming language,⁷ while others have acquiesced to accepting naming conventions as long as they are consistent.⁶ Beautiful code has been defined in general terms as readable, focused, testable, and elegant.¹ The more extreme case is the invention of an entire programming language built around a concrete set of ideals, such as Ruby or Python. Ruby emphasizes brevity, simplicity, flexibility, and balance.⁴ The principles behind Python are clear in the *Zen of Python*,⁵ where the focus lies on beauty, simplicity, readability, and reliability.

Our approach to this issue has been to develop a system of coding guidelines (available online³). While these guidelines come from an educational environment, they are designed to be useful to practitioners as well. The guidelines are based on

a few broad principles that capture some fundamental principles of communication and elevate the notion of coding conventions to a higher level. The use of these conventions will also improve the sustainability of a code base. This article looks at these underlying principles.

One area not considered here is the use of syntax highlighting or IDEs. While either one may make code more readable (because of syntax highlighting or code folding, among others) and easier to manage (for example, quickly looking up or refactoring functions and/or variables), our guidelines have been developed to be IDE and color neutral. They are meant to reflect foundational principles that are important when writing code in any setting. Also, while IDEs can help improve readability and understanding in some ways, the features found in these tools are not standard (consider the different features found in Visual Studio, Eclipse, and VIM, for example). Likewise, syntax highlighting varies greatly among environments and may easily be changed to match personal preference. The goal of the following principles is to build a foundation for good programming that is independent of the programming IDE.

Consider a Program as a “Table”

In a recent *ACM Queue* article, Poul-Henning Kamp² makes the fascinating point that much of the style of programming languages stems from the ASCII character set and typewriter-based terminals. Programming languages make no use of the graphical properties and options of modern devices. While code must be written with the clarity of good English grammar, it is not English text. Instead it is more like math and tables.

This is a far-reaching principle. First, it speaks directly to the use of fonts. Do not use a variable-width (proportional) font for program code, as code is not text. Fixed-width fonts (for example, Courier and Data Gothic) look appealing and allow easy alignment of code. Proportional (variable-width) fonts prevent proper alignment, and even more importantly, do not “look like” code.

While one should continue to think

Figure 1. Use of vertical alignment to show symmetry.

```
char c1;
c1 = getChoice();
switch(c1){
    case 'q': case 'Q': quit();           break;
    case 'e': case 'E': enterPerson(content); break;
    case 'd': case 'D': delPerson(content); break;
    case 's': case 'S': sortByName();     break;
    case 'l': case 'L': showAll();        break;
    case 'f': case 'F': searchByName(content); break;
    case default:      System.out.println("---Invalid Command!!\n");
}
```

Figure 2. Example of cluttered presentation.

```
private JFrame mainFrame = new JFrame("Wind Power Calculator");
private JTextArea windVel = new JTextArea(VEL, 2, TEXT_WIDTH);
private JLabel velTag = new JLabel("Wind Velocity");
private JTextArea sweptArea = new JTextArea(SWEPT_AREA, 2, TEXT_WIDTH);
private JLabel sweptAreaTag = new JLabel("Swept Area");
private JTextArea genSize = new JTextArea(GEN_SIZE, 2, TEXT_WIDTH);
private JButton calculatePower = new JButton("Calculate Power");
```

Figure 3. Revision of code in Figure 2 showing tabular structure.

```
private JFrame    mainFrame    = new JFrame    ("Wind Power Calculator");
private JTextArea windVel      = new JTextArea (VEL, 2, TEXT_WIDTH);
private JLabel    velTag       = new JLabel    ("Wind Velocity");
private JTextArea sweptArea    = new JTextArea (SWEPT_AREA, 2, TEXT_WIDTH);
private JLabel    sweptAreaTag = new JLabel    ("Swept Area");
private JTextArea genSize      = new JTextArea (GEN_SIZE, 2, TEXT_WIDTH);
private JButton   calculatePower = new JButton  ("Calculate Power");
```

of a program as a sequence of actions or as an algorithm at a high level, each section of code should also be thought of as a presentation of a chart, table, or menu. In figures 1, 2, and 3 notice the use of vertical alignment to show symmetry. This is a powerful method of communication.

In the case when a long line of code spills into multiple lines, we suggest breaking and realigning the code.^a For example, instead of

```
participant newEntry = new participant
(id, name, address1, address2, city,
state, zip, phone, email);
```

use

```
participant newEntry = new participant
(id, name, address1, address2, |
    city, state, zip, phone, email);
```

or

```
participant newEntry = new participant
(id, name, address1, address2, city,|
    state, zip, phone, email);
```

Let Simple English be Your Guide

A programmer creates a name for something with full knowledge of its use, and often many names make sense when one knows what the name represents. Thus, the programmer has this problem: *creating a name based on a concept*. The true challenge, however, is precisely the opposite: *inferring the concept based on the name*! This is the problem that the program reader has.

Consider the simple name `sputn` taken from the common C++ header file `<iostream.h>`. An inexperienced or unfamiliar programmer may suddenly be mentally barraged with a bout of questions such as: Is it an integer? A pointer? An array or a structure? A method or a variable? Does *sp* stand for saved pointer? Is *sput* an operation to be done *n* times? Do you pronounce it *sputn* or *s-putn* or *sput-n* or *s-put-n*?

We advocate basing names on conventional English usage—in particular, simple, informal, abbreviated English usage. Consider the following

more specific guidelines:

- Variables and classes should be nouns or noun phrases;
- Class names are like collective nouns;
- Variable names are like proper nouns;
- Procedure names should be verbs

or verb phrases;

- Methods used to return a value should be nouns or noun phrases;
- Booleans should be adjectives;
- For compound names, retain conventional English syntax; and
- Try to make names pronounceable.

Figure 4. Examples of basing names on conventional English usage.

Variables		Class Names	
Not the Right Noun	Better	Not the Right Noun	Better
Round	Wheel	Accounting	BankAccount
LoopTimes	NumLoops	SetPoint	Point
Valid	InputStatus	NodeNetworking	SocketInfo
Starting	Source		
Ending	Destination		
Rows	NumRows		

Problematic	Preferable
Person personInfo;	PersonInfo P1, P2;
Socket socketDesc;	SocketDescription socket;
Frame TopFrameSection;	Frame TopFrame;
Message = EmergencyAlertLabels[i]	AlertText = EmergencyLabel[i]

Not the Right Verb	More Readable
NameSet	SetName
Modified	Modify
Withdrawal	Withdraw
Right	MoveRight

Incorrect Function Names	More Readable
numFiles = countFiles(directory);	numFiles = fileCount(directory);
A = computeArea(parcel);	A = Area(parcel);
x = getImagePos(i).x;	x = Image(i).xCoord;

Incorrect Boolean Vars	Grammatically Better
Fill	Full
Terminate	Terminated
Real	isReal
Edit	IsEditable
Waits	Waiting
License	hasLicense

Grammatically Incorrect	Better
IdVehicle	VehicleID
NoSectors	NumSectors
FormEnable();	EnableForm();

Unpronounceable	Pronounceable
Tbl	Table
GenYmDhMs	GenerateTime
Cntr	Counter
Nbr	Num

^a Given the limited spacing here, the | denotes a line break.

Some examples of this broad principle are shown in Figure 4.

There is an interesting but small issue when considering examples such as:

```
numFiles = countFiles(directory);
```

While `countFiles` is a good name, it is not an optimal name since it is a verb. Verbs should be reserved for procedure calls that have an effect on variables. For functions that have no side effects on variables, use a noun or noun phrase. One does not usually say

```
y = computeSine(x);
or
milesDriven =
computeDistance(location1, location2);
```

but rather

```
y = sine(x);
or
milesDriven = Distance(location1, location2);
```

We suggest that

```
numFiles = fileCount(directory);
```

is a slight improvement. More importantly, this enforces the general rule that verbs denote procedures, and nouns or adjectives denote functions.

Rely on Context to Simplify Code

All other things being equal, shorter programs are always better. As an example, local variables that are used as index variables may be named `i`, `j`, `k`, and so on. An array index used on every line of a loop need not be named any more elaborately than `i`. Using index or `elementNumber` obscures the details of the computation through excessive description. A variable that is rarely used may deserve a long name: for example, `MaxPhysicalAddr`. When variable names are long, especially if there are many of them, it quickly becomes difficult to see what's going on. A variable name can often be shortened by relying on the context in which it is used. For example, the variable `Store` in a stack implementation rather than `StackStore`.

Major variables (objects) that are used frequently should be especially short, as seen in the examples in Fig-

ure 5. For major variables that are used throughout the program, a single letter may encourage program clarity.

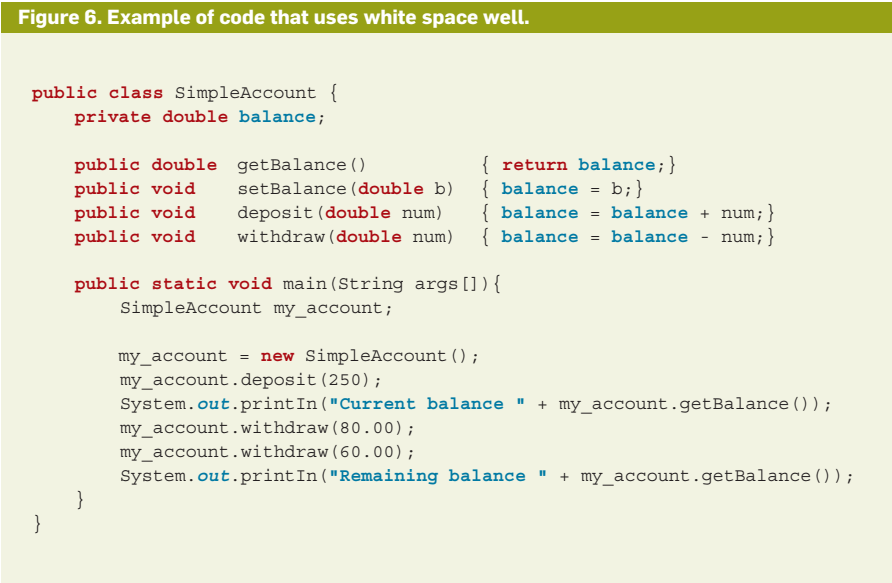
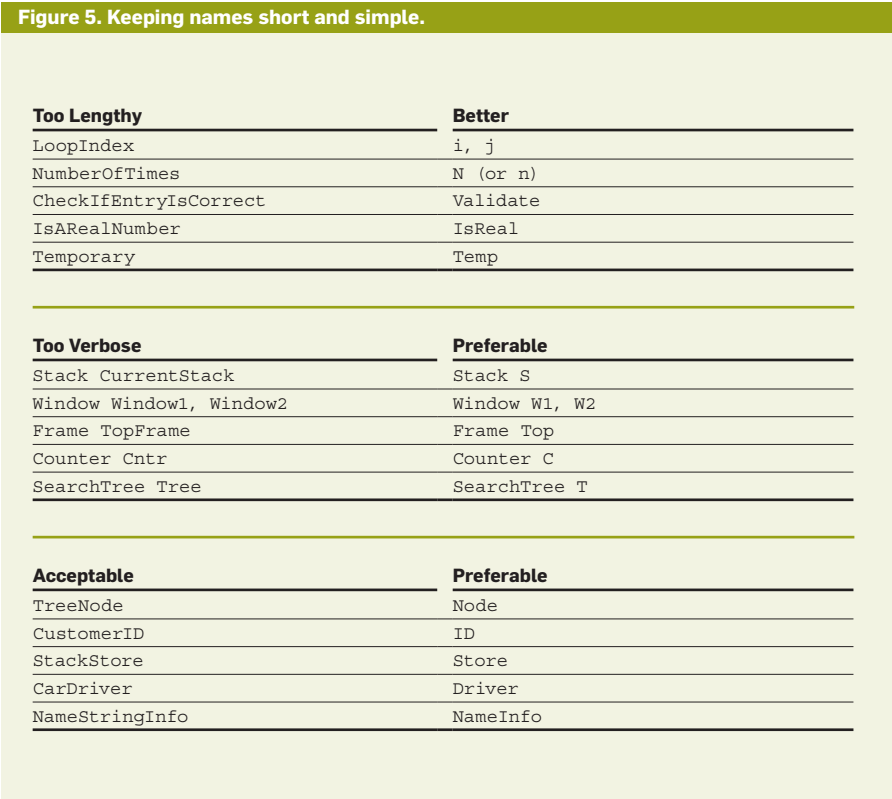
Use White Space to Show Structure

While written and spoken communication may reach a high level of clarity, it is often left wanting of meaning if not accompanied by the personal touch of nonverbal cues and tendencies. An individual's body language helps clarify the spoken word. In a similar sense, the programmer relies

on white space—what is not said directly—in the code to communicate logic, intent, and understanding.

An example is the use of blank lines between conceptually different sections of code. Blank lines should improve readability as they separate logically different segments of the code and thus provide the literary equivalent of a section break. Appropriate places to use blank lines include:

- When changing from preprocess-



sor directives to code;

- ▶ Around class and structure declarations;
- ▶ Around a function definition of some length;
- ▶ Around a group of logically connected statements of some length; and
- ▶ Between declarations and the executable statements that follow.

Consider the code listing in Figure 6. Individual blank spaces should also be used to show the logical structure within a single statement. Strategic blank spaces within a line simplify the parsing done by the human reader. At a minimum, blank spaces should be included after the commas in argument lists and around the assignment operator “=” and the redirection operators “<<” and “>>”.

On the other hand, blank spaces should not be used for unary operators such as unary minus (-), address of (&), indirection (*), member access (.), increment (++), and decrement (--).

Also, if it makes sense, put two to three statements on one line. This practice has the effect of simplifying the code, but it must be used with discretion and only where it is sensible to do so.

Let Decision Structures Speak for Themselves

The case statement used in Figure 1 brings up a general point: very simple decision statement structures can be tersely presented, showing the alternative code simply, and, if possible, without braces, as in the example in Figure 7.

It is not uncommon for simple conditions to be mutually exclusive, creating a kind of generalized case statement. This, as is common practice, can be printed as a chain, as in Figure 8.

Of course, it may be that the structures are truly nested, and then one must use either nested spacing or functions to indicate the alternatives. Again, the general point is to let the structure drive the layout, not the syntax of the programming language.

In the brace wars, we do not take a strong stand on the various preferences shown in Figure 9, but we do feel strongly that the indent is vital,

as it is the indent that shows the structure.

Focus on the Code, Not the Comments

The ability to communicate clearly is an issue that is faced in all facets of the human experience. Programmers must achieve a level of clarity, continuity, and beauty when writing code. This means focusing on the code and its clarity, balance, and symmetry, not on its length or comments. While this concept does not advocate the removal of comments or negate their use and importance in appropriate

situations, it does suggest that programmers must use comments wisely and judiciously. The focus should be on developing code that, for the most part, clearly communicates intent and functionality. This practice will automatically reduce the need for many comments.

Discussion

Although the guidelines presented here are used in an educational setting, they also have merit in industrial environments. Students who are educated using these guidelines will most likely use them (or some variant) as

Figure 7. Decision statement structure, tersely presented.

```
if(Card != null)    display.setText(Card.getText());
else                display.setText("No More Cards.");
```

Figure 8. Case statement presented as a chain.

```
if (result >= 90)
    cout << "Grade of A!";
else if (result >= 80)
    cout << "Grade of B";
else if (result >= 70)
    cout << "Sorry, grade of C";
else
    cout << "Not very good";
```

Figure 9. Examples of K&R, ANSI, and Whitesmiths coding styles.

<pre>if (expression) { statements }</pre>	<pre>if (expression) { statements }</pre>	<pre>if (expression) { statements }</pre>
---	---	---

Figure 10. Example of a systems-programming coding style.

```
//Unix Style
void tokenizeStr(string str, vector<string>& result, const string& delim = " ") {
    int pos = 0;
    string strtok;
    for(;;) {
        pos = str.find(delim);
        if(pos == (int)string::npos) {
            result.push_back(str);
            break;
        }
        strtok = str.substr(0, pos);
        result.push_back(strtok);
        str = str.substr(pos+1);
    }
}
```

they enter industry. To demonstrate this, we have developed an example that applies these guidelines to two very different styles. The first is the Unix style. It is terse, often making use of vowel deletion, and is often found in realistic applications such as operating-system code. This is not to imply that all or most system programmers use this style, only that it is not unusual. Figure 10 shows a small example of this style.

We call the second style the textbook style, as illustrated in Figure 11. Again, this in no way means to imply that all or most textbooks use this style, only that the style in the example is not unusual. In this style the focus is on learning. This means that there is frequent commenting, and the code is well spread out. For the purposes of learning and understanding the details of a language, this style can be excellent. From a practical perspective or for any program of some scale, this style does not work well as it can be overwhelming to use or to read. Moreover, this style makes it difficult to see the overall design, as if one is stuck under the trees and cannot see the forest around.

Figure 12 is a rework of the function in figures 10 and 11, using the guidelines discussed here to make a smooth transition between academic and practical code. This figure shows a balance of both styles, relying more directly on the code itself to communicate intent and functionality clearly. Compared with the textbook style, the resultant code is shorter and more compact while still clearly communicating meaning, intent, and functionality. When compared with the Unix style, the code is slightly longer, but the meaning, intent, and functionality are clearer than the original code.

Figure 13 illustrates the guidelines presented here in another setting. This is a function taken from a complex program (10,000 lines) related to power-system reliability and energy use regarding PHEVs (plug-in hybrid electric vehicles). The program makes numerous calculations related to the effect that such vehicles will have on the current power grid and the effect on generation and transmission systems. This program attempts to evaluate the reliability of

power systems by developing a model for reliability evaluation using a Monte Carlo simulation.

While the previous examples show the merit of the guidelines present-

ed here, one argument against such guidelines is that making changes to keep a certain coding style intact is time consuming, particularly when a version-control system is used. In the

Figure 11. Example of a textbook coding style.

```
// TEXTBOOK STYLE
void tokenizeString(string myString, vector<string>& listOfTokens,
    const string& tokenDelimiter = " ")
{
    // Precondition: myString is not null
    //
    // Parses myString into a list of tokens using the given delimiter.
    // If no specific delimiter is given, uses the space as a delimiter
    //
    // Postcondition: listOfTokens contains the individual tokens as values

    int index = 0;
    string nextToken;
    boolean loop = true;

    // Obtain tokens and store in vector
    while(loop)
    {
        index = myString.find(delimiter);
        if(index == (int)string::npos)
        {
            // end of string found
            tokenList.push_back(myString);
            loop = false;
        }
        else
        {
            // Append nextToken to vector
            nextToken = myString.substr(0, index);
            tokenList.push_back(nextToken);
            myString = myString.substr(index + 1);
        }
    }
}
```

Figure 12. Example of a coding style using the guidelines presented here.

```
// OUR STYLE
void tokenizeString(string S, vector<string>& tokenList,
    const string& delimiter = " ") {
    // Given a string S, compute the list of its tokens.

    int position;
    string token;
    boolean moreTokens;

    moreTokens = true;
    while(moreTokens){
        position = S.find(delimiter);
        if(position == (int)string::npos){
            tokenList.push_back(S);
            moreTokens = false;
        }else{
            token = S.substr(0, position);
            tokenList.push_back(token);
            S = S.substr(position + 1);
        }
    }
}
```

face of a time-sensitive project or a project that most likely will not be updated or maintained in the future, the effort may not be worthwhile. Typical cases include class projects, a Ph.D.

thesis, or a temporary application.

If, however, the codebase in question has a long lifespan or will be updated and maintained by others (for example, an operating system, server,

interactive Web site, or other useful application), then almost any changes to improve readability are important, and the time should be taken to ensure the readability and maintainability of the code. This should be a matter of pride, as well as an essential function of one's job. 

Figure 13. Realistic and complex example of code following the guidelines presented here.

```
void loadDataFile(double& pLoad,      double& qLoad,
                 int&  numBuses, int&  numTransLines, string systemName,
                 vector<Generator>& gens, vector<Line>& transLines, vector<Bus>& buses){
    // This function loads the various system parameters from the power system data file.
    // The power system data is encoded as a csv file,

    ifstream      systemData;
    string         dataLine;
    vector<string> dataItem;
    int           numGens;

    systemData.open("../Data/" + systemName).c_str());
    if (systemData.is_open()) {
        systemData >> numGens;
        systemData >> pLoad;
        systemData >> qLoad;
        systemData >> numBuses;
        systemData >> numTransLines;

        numGens = 0;

        //Clear Vectors
        gens.clear(); transLines.clear(); buses.clear();

        // Set Generators
        for(int i = 0; i<numGens; i++){
            systemData >> dataLine;
            Utils::tokenizeString(dataLine, dataItem, ",");

            gens.push_back(Generator(
                atof(dataItem[3].c_str()), atof(dataItem[4].c_str()),
                atof(dataItem[5].c_str()), atof(dataItem[6].c_str()),
                atof(dataItem[7].c_str()), atoi(dataItem[0].c_str())
            );

            gens[i].setIndex(i);
            dataItem.clear();
        }

        // Set transmission lines
        for(int i = 0; i<numTransLines; i++){
            systemData >> dataLine;
            Utils::tokenizeString(dataLine, dataItem, ",");

            transLines.push_back(Line(
                atoi(dataItem[0].c_str()),  atoi(dataItem[1].c_str()),
                atoi(dataItem[2].c_str()),  atof(dataItem[3].c_str()),
                atof(dataItem[4].c_str()),  atof(dataItem[5].c_str()),
                atof(dataItem[6].c_str()),  atof(dataItem[7].c_str()),
                atof(dataItem[8].c_str()),  atof(dataItem[9].c_str()),
                atof(dataItem[10].c_str()),  atof(dataItem[11].c_str()),
                atof(dataItem[12].c_str()),  atof(dataItem[13].c_str())
            );
            dataItem.clear();
        }

        // Set bus loadings
        for(int i=0; i<numBuses; i++){
            systemData >> dataLine;
            Utils::tokenizeString(dataLine, dataItem, ",");
            buses.push_back(Bus(
                atoi(dataItem[0].c_str()),  atoi(dataItem[1].c_str()),
                atoi(dataItem[6].c_str()),  atoi(dataItem[10].c_str()),
                atof(dataItem[2].c_str()),  atof(dataItem[3].c_str()),
                atof(dataItem[4].c_str()),  atof(dataItem[5].c_str()),
                atof(dataItem[6].c_str()),  atof(dataItem[7].c_str()),
                atof(dataItem[12].c_str()),  atof(dataItem[11].c_str()),
                atof(dataItem[9].c_str())
            );
            dataItem.clear();
        }
        systemData.close();
    }
}
```

Related articles on queue.acm.org

Beautiful Code Exists, if You Know Where to Look

George Neville-Neil

<http://queue.acm.org/detail.cfm?id=1454458>

Software Development with Code Maps

Robert DeLine, Gina Venolia, and Kael Rowan

<http://queue.acm.org/detail.cfm?id=1831329>

Reading, Writing, and Code

Diomidis Spinellis

<http://queue.acm.org/detail.cfm?id=957782>

References

1. Heusser, M. Beautiful code. *Dr. Dobbs's* (Aug. 2005); <http://www.ddj.com/184407802>.
2. Kamp, P-H. Sir, please step away from the ASR-33! *ACM Queue* 8, 10 (2010); <http://queue.acm.org/detail.cfm?id=1871406>.
3. Ledgard, H. Professional coding guidelines. 2011 Unpublished report, University of Toledo; http://www.eng.utoledo.edu/eecs/faculty_web/hledgard/softe/upload/.
4. Molina, M. What makes code beautiful. *Ruby Hoedown*, 2007.
5. Peters, T. *The Zen of Python*. PEP (Python Enhancement Proposals). Aug. 20, 2004; <http://www.python.org/dev/peps/pep-0020/>.
6. Reed, D. Sometimes style really does matter. *J. Computing Sciences in Colleges* 25, 5 (2010), 180-187.
7. Sun Developer Network. Code conventions for the Java programming language, 1999; <http://java.sun.com/docs/codeconv/>.

Acknowledgments

The authors would like to thank David Marcus and Poul-Henning Kemp for their insightful comments while completing this work, as well as the software engineering students who have contributed to these guidelines over the years.

Robert Green is pursuing his Ph.D. at the University of Toledo. He has multiple years of experience developing software across a variety of industries. His research interests include biologically inspired computing, high-performance computing, and alternative energy.

Henry Ledgard was a member of the design team that created the programming language ADA, a language he believes was a creative, sound design. He is the author of several books on programming, and is a professor at the University of Toledo. His research interests include principles of language design, human engineering and effective ways to teach computer science.

DOI:10.1145/2043174.2043192

People in high-density crowds appear to move with the flow of the crowd, like particles in a liquid.

BY BRIAN E. MOORE, SAAD ALI, RAMIN MEHRAN, AND MUBARAK SHAH

Visual Crowd Surveillance through a Hydrodynamics Lens

VIDEO CAMERAS MONITORING the activity of people in public settings are commonplace in cities worldwide. At large events, where crowds of hundreds or thousands gather, such monitoring is important for safety and security purposes but is also extremely (technically) challenging. Human operators are generally employed for the task, but even the most vigilant humans miss important information that could ultimately contribute to unfavorable consequences.

Major research efforts are under way to develop systems that cue security personnel to individuals or events of interest in crowded scenes. Essential are methods by which information can be extracted from video data in order to recognize crowd behaviors, track

individuals in crowds, and detect abnormal events.

This article explores cutting-edge techniques we have used in real-world scenarios to provide solutions to such problems.^{1,2,17} We developed them based on the notion that people in crowds behave, in ways, like particles in fluids. Hence, we treat crowds as collections of mutually interacting particles.

Typically, the motion of a high-density crowd appears to behave like a liquid, and interaction forces tend to dominate the motion of the people. This is in contrast to crowd motion appearing in states like gases, where interactions between people are few but random motions of individuals tend to dominate the behavior. With all this in mind, we contemplate visual crowd surveillance using ideas and techniques based in hydrodynamics. Hence, we say “fluid” and “liquid” interchangeably, distinguishing our approach from aerodynamics, which considers fluids in gaseous states.

Our hydrodynamics point of view is well suited for analyzing high-density crowds,^{9,12} with surveillance the primary concern. Though the number of people will never reach the astronomical numbers of particles in fluids, we pursue tasks in crowd analysis using a similar concept of scale. Ranging from the macroscopic view of all particles to the microscopic view of individual

» key insights

- **Computer algorithms extract information from digital videos of people in crowds as a way to automatically track individuals, detect abnormal behavior, and segment characteristic patterns of flow in crowds.**
- **Individuals in dense crowds, like particles in a fluid, are restricted in their motion by neighboring individuals, reflecting a kind of interdependence that is pivotal for solution development.**
- **The tools of computational and applied mathematics are indispensable for visual analysis of crowds; pixel information is translated into particle trajectories used to understand crowd flow on length scales ranging from the macroscopic to the microscopic.**



Figure 1. (a) New York City Marathon; (b) political rally in Los Angeles; (c) pilgrims circling Kabba in Mecca.

particles, we address the problems of segmentation, abnormal-behavior detection, and tracking.

Techniques devised by other researchers have been used to consider similar problems. Here, we briefly review research in the area of visual crowd surveillance, referring the reader to detailed articles on tracking²³ and crowd-behavior analysis.²⁴

For problems involving crowd segmentation, Chan and Vasconcelos⁸ used dynamic texture-based representations of scenes to determine how regions differ, proposing a method⁷ for counting pedestrians in high-density crowds. Sand et al.²⁰ implemented a particle-based framework for the purpose of estimating the motion in a scene but did not use it for interpretation of significant segments.

For problems involving behavior analysis, methodologies are available for understanding crowd behavior. The first, advanced by Marques et al.¹⁶ and Tu et al.,²² perceived a crowd as an assembly of individuals, using segmentation or tracking algorithms to understand their behavior. The other, promoted by Andrade et al.,³ viewed a crowd as an organism, such that its behavior is studied and accepted on a global level. Reisman et al.¹⁹ proposed that crowd behavior is recognized by modeling the scene, giving a description of important features within it. Kratz et al.¹⁵ detected anomalies as statistical deviations from the ordinary motion patterns in space-time volumes to characterize the scene.

With regard to tracking in crowded scenes, one of the first important methods was devised by Zhao et al.²⁵ using ellipsoids to model the human shape and color differences to mark appearances. Another framework, by Brostow et al.,⁵ assumed that points appearing to move together are prob-

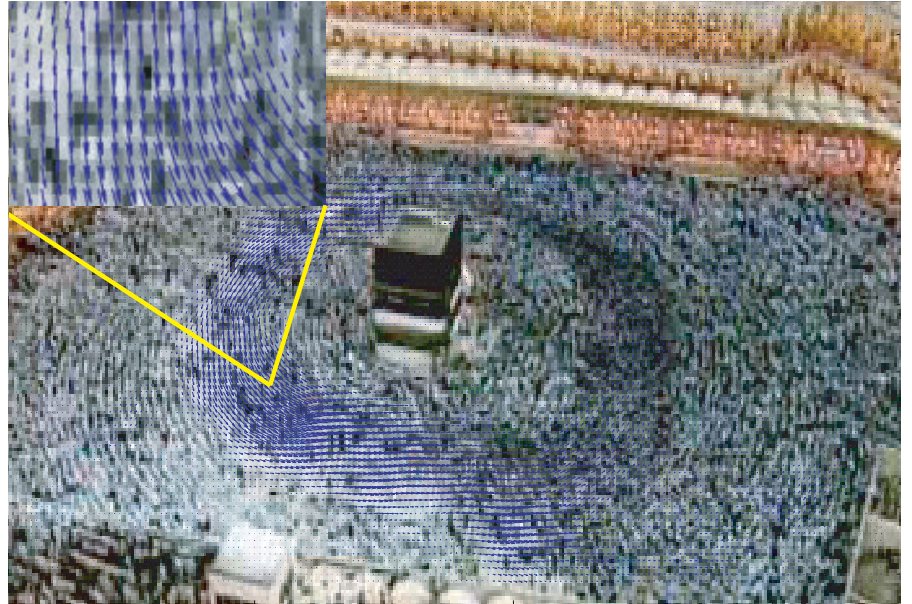


Figure 2. Flow field of a frame in the Kabba video.

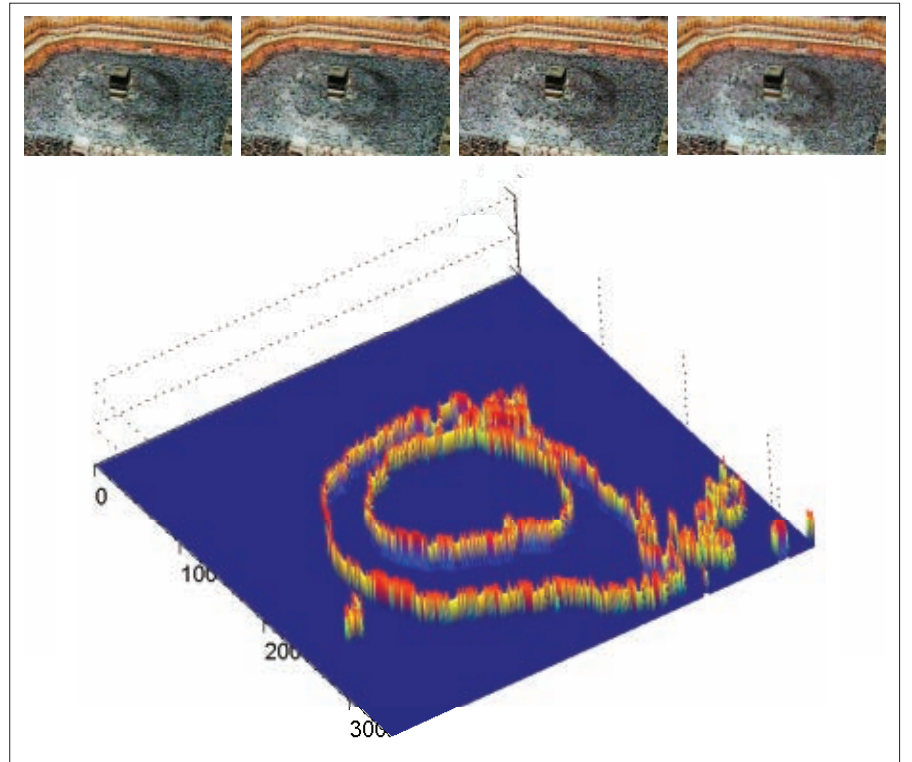


Figure 3. Four frames from the video sequence of pilgrims circling Kabba and the FTLE field.

ably part of the same object, tracking individuals based on the probability that points could be clustered together. More recently, Pellegrini et al.¹⁸ expanded a social-force model to take into account destinations and desired directions of individuals, making it

suitable for tracking individuals in crowded scenes.

Particles and People

Random actions, relationships between energy and density, and a gas/liquid/solid-state demeanor are all

characteristics of particles in a fluid and of people in a crowd. Most important, the motions of particles/people are determined by the external forces exerted on them; for example, both particles and people are affected by boundary forces (such as walls) and feel the forces of neighboring particles/people. One difference is that people are, to some extent, able to determine their own destiny, so the crowd may be viewed as a “thinking fluid,”¹¹ but there are still probabilistic similarities to particle motion regardless of this difference.

When scientists consider hydrodynamics, they often use different scales, depending on the questions being addressed.¹⁴ At the microscopic, one may examine the position or velocity of a particular particle among many. On another, the macroscopic is used to scrutinize the nature of enormous collections of particles (such as a tree branch moving in water). Between them is the mesoscopic scale, which is used to analyze the interaction of “small” collections of particles, giving characteristic information (such as temperature and average density).

Considering the behavior of people in a crowd, we take a similar approach, depending on the questions we want to answer. We focus on three generally recognized key problems in visual crowd surveillance—crowd segmentation, behavior analysis, and tracking—corresponding to the scales. To be clear, some situations might necessitate tracking a particular person in a crowd, requiring a microscopic point of view. Others might call for descriptive information on when the behavior of a crowd is abnormal, meaning it is neither necessary nor feasible to track every individual in a crowd but important to understand how groups of individuals interact, for which we employ a mesoscopic point of view. Still, a macroscopic point of view is more appropriate for segmenting global patterns of flow.

Here, it is pertinent to discuss the types of scenes and spatio-temporal range of crowd behaviors that can be handled through an understanding of the hydrodynamics point of view. To begin, hydrodynamics-based techniques require that a crowd be viewed from above, thereby minimizing artifacts resulting from independent

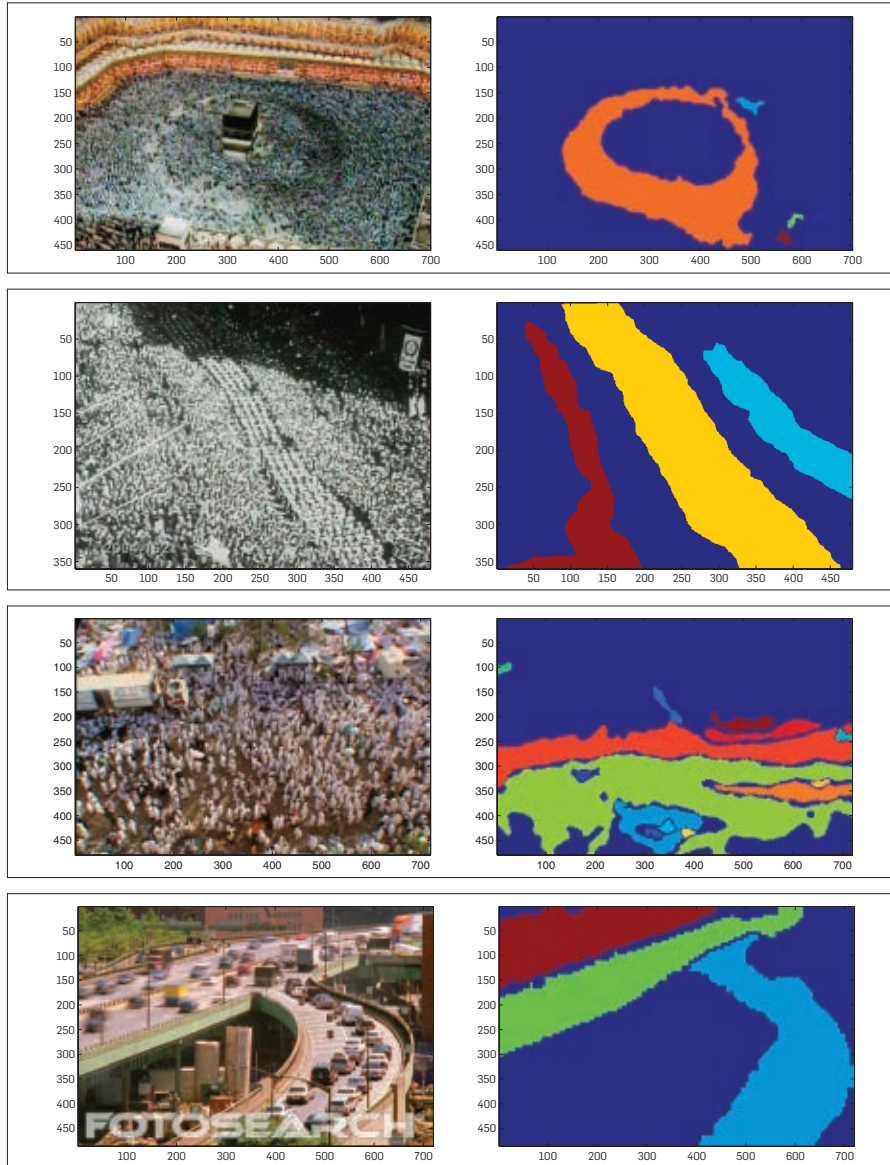


Figure 4. Video scene (left) and corresponding segmentation (right).

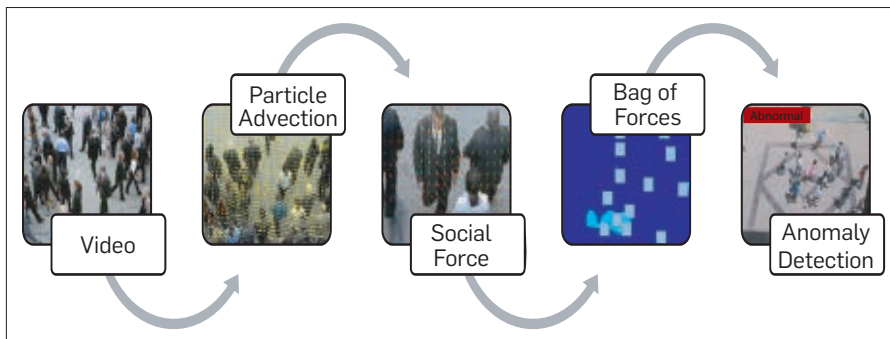


Figure 5. Our approach for detecting abnormal behavior in crowd videos.

movement of multiple body parts. Side views of the scene are least preferable within the particle-based framework; Figure 1 includes examples of such scenes and camera setup. Our algorithms next allow each pixel to represent a particle, with a minimum requirement on the spatial scale of at least one pixel per person. If two or more people are matched to a single particle, the methods may encounter problems, but allowing as many particles per person as the scene dictates is certainly permissible.

Another noteworthy requirement is that video scenes exhibit a dominant trend typical of high-density crowds, where the movement of individuals is restricted by other individuals, obligating the group to move as a whole, like a fluid. However, the dominant trend is key to the analysis, while the density of the crowd is allowed to vary. Since crowd behavior is naturally dynamic, and the flow (trend) of a crowd can change with time, any video-based analysis of crowd motion should take a sliding-window approach. It performs the analysis over a particular sequence of frames (a window in time), then “slides” the window to another sequence of frames to repeat the analysis. The size of the window may be adaptable or fixed but depends on the level of activity in the scene. Methods explored here follow this principal.

Macroscopic scale (crowd segmentation). The macroscopic scale suggests a focus on global crowd behavior, requiring a comprehensive point of view; Figure 1 includes examples of the types of scenes we consider, with thousands of people in view. In such settings, we are primarily interested in the overall movement of the crowd, meaning we are able to find segments of common motion within it.¹

A key ingredient for our solution to the problem of segmentation is called “particle advection” and used in each of our three problem synopses. The approach itself mimics a common mathematical formulation of fluid mechanics, or Lagrangian specification, characterized by following particular particles as they move with the flow.⁴ The first step in applying this idea to a video sequence is to compute the optical flow, or apparent visual motion of objects in the scene



Random actions, relationships between energy and density, and a gas/liquid/solid-state demeanor are all characteristics of particles in fluids and of people in a crowd.



(see Figure 2). Every pixel has position $\mathbf{x} = (x, y)$, and the optical flow provides velocities (u, v) at each position, so objects are related to their velocities by the system of equations

$$\frac{dx}{dt} = u(x, y, t), \frac{dy}{dt} = v(x, y, t)$$

Particle advection is performed by overlaying the scene with a grid of particles that serves as the initial conditions for this system of equations; particles are then transported to new coordinates in subsequent frames using a time-stepping technique for integrating the system of equations, as in

$$\begin{aligned} x(t+1) &= x(t) + u(x(t), y(t), t), \\ y(t+1) &= y(t) + v(x(t), y(t), t) \end{aligned}$$

Thus, the flow of the crowd in the scene is given by particle trajectories.

Important to note is that errors and noise in optical flow are averaged out to some extent as a result of time integration performed to determine particle trajectories. Thus, the particular method used to produce optical flow is not crucial for the three problems we consider and has been verified experimentally. Temporal scale for analysis is determined by the integration time t . In practice, t should depend on the rate of change of the flow field, with a higher rate of change of flow field resulting in smaller time scales and vice versa. In our experiments, we fixed $t =$ two seconds or 60 frames for all scenarios.

Particle advection produces a flow map, a function $\phi_{t_0}^t(\mathbf{x}_0) = \mathbf{x}(t; t_0; \mathbf{x}_0)$ relating the position $\mathbf{x}$ of a particle at time t_0 to its original position $\mathbf{x}_0$ at the initial time t_0 . That is, the flow map fully describes the trajectory of each particle, which does not necessarily correspond to a person in a crowd but to a small region in the scene exhibiting a collective pattern of motion. In sections with coherent motion, the flow maps show qualitatively similar behavior, but trajectories experiencing different behavior are from sections with different coherent motion. These qualitative differences define flow segments. Our primary mathematical tool for finding these qualitative differences is called a “finite time Lyapunov exponent,” or FTLE, we use to define Lagrangian coherent structures.²¹ The FTLE is essen-

tially a number that reflects how two neighboring particles separate from one another over time and is computed using the maximum eigenvalue λ_{max} of the Cauchy-Green deformation tensor Δ , obtained from the Jacobian matrix for the flow map, $D\phi_{t_0}^t(\mathbf{x})$. More precisely, the largest FTLE with integration time t is

$$\sigma = \frac{1}{t} \ln \sqrt{\lambda_{max}(\Delta)}$$

where

$$\Delta = D\phi_{t_0}^{t_0+t}(\mathbf{x})^T D\phi_{t_0}^{t_0+t}(\mathbf{x})$$

Computing the FTLE at every point produces the FTLE field, a scalar field that immediately exposes any regions in the scene with differing flow by finding particle trajectories that start close together but end far apart. In practice, the particle advection approach allows implementation of the algorithm in both forward and backward time, meaning the flow segments are the same regardless of which direction the flow is moving.

Combining the FTLE fields for both forward and backward motion yields vivid results (see Figure 3). We use a watershed algorithm to segment the FTLE field, making it find the exact number of flow segments. This process is repeated by moving the sliding temporal window to obtain segments for subsequent time steps.

The end result is a net segmentation showing each region exhibiting a single clearly defined characteristic flow pattern. Such a result is not possible through segmentation based solely on optical flow, because optical flow captures only motion between two frames. On the other hand, particle advection motion in several frames is integrated over time and nicely captured by the scalar FTLE field. Figure 4 includes several results in which the motion in crowded pedestrian and traffic scenes is properly segmented through our method; each row shows a frame from a different video sequence, along with subsequent segmentation. Regions of different colors signify qualitative changes in the flow,

and dark blue represents areas with no coherent flow. A clear example is the traffic scene in the last row, with dark blue representing regions outside the lanes, and red, green and light blue representing movement in each direction.

Mesoscopic scale (behavior detection). Beyond a global understanding of pedestrian flow in crowds, detection of abnormal events or behavior is important, generally for the sake of public safety. We use the local interactions of multiple people to identify regular patterns of motion, in addition to any anomalies.¹⁷ A fundamental component of our approach (see Figure 5) in this setting is a social force (fluids-based mathematical) model for describing pedestrian movement, as pioneered by Helbing and Molnar¹⁰ almost 20 years ago.

The central idea hinges on Newton's second law of motion—force equals mass times acceleration, or $F = ma$. In it, each individual in the scene reacts to forces that produce motion. These forces can be deconstructed into two parts: the personal-desire force (individuals striving to get to their desired destinations) and the interaction force (exerted on individuals by other individuals or things in the scene). Thus, pedestrian i changes velocity according to

$$a = \frac{dv_i}{dt} = F_p + F_{int}$$

where F_p and F_{int} refer to personal and interaction forces, respectively. In a given scene, since individuals are all relatively the same size, the masses are assumed to be one. Quantifying these forces (see Figure 6a for an example) allows our method to establish the ongoing behavior in the crowd, enabling detection of any behavior out of the ordinary (Figure 6b).

Note that in very dense crowds, pedestrians follow group velocity and goals,¹² but as density decreases, personal interest plays a greater role in pedestrian motion. Hence, at the mesoscopic scale, our algorithm may use scenes with mid-to-high crowd density, provided the interaction force is not negligible, meaning behavior is still fluid-like.

The algorithm itself starts with particle advection, followed by computation of the forces. Each person in

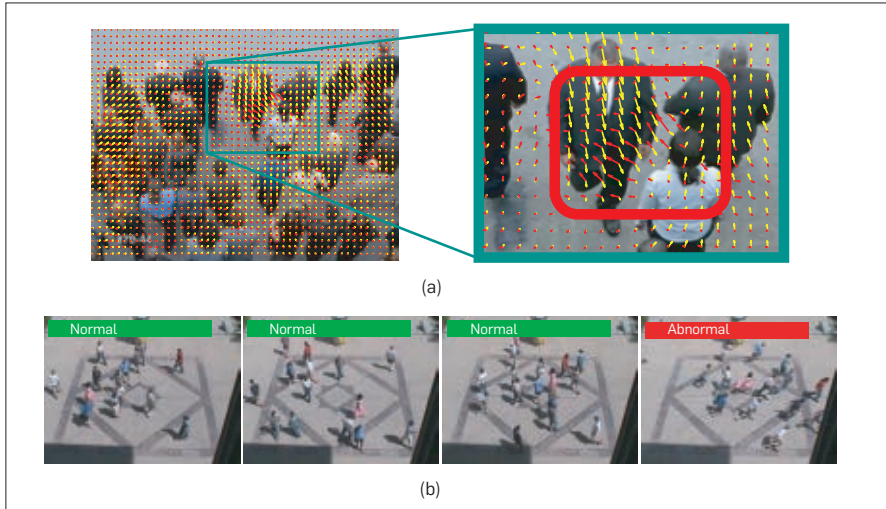


Figure 6. (a) Optical flow (yellow) and computed interaction vectors (red) for pedestrians with opposing directions; (b) frames of a sequence where the observed behavior suddenly becomes abnormal (people running in panic) in the last frame.

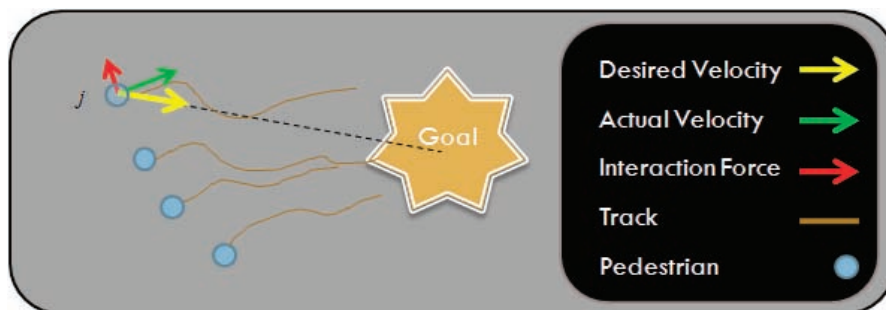


Figure 7. Scheme for computing interaction force.

the crowd has a desired direction and speed, but individual direction and speed is limited by the surrounding pedestrians. The actual velocity v_i of a particle in the (x_i, y_i) coordinate is obtained from the spatial-temporal average of optical flow. On the other hand, the desired velocity v_i^p is given by the optical flow for that particle. Hence, the personal-desire force is

$$F_p = \frac{1}{\tau} (v_i^p - v_i)$$

where τ is a relaxation parameter. Thus, the interaction force (see Figure 7) is given as

$$F_{int} = \frac{dv_i}{dt} - F_p$$

These forces together yield a sufficient description of the motion in the scene based on the acting forces.

Specification of the forces determining the motion in the scene provides understanding of synergy between interacting particles but does not, by itself, secure evidence of changes in behavior; for example, normal interaction forces on a stock-market trading floor may differ drastically from those of pedestrians on the street. Using this technique to detect and localize any changes in behavior, the computer must first learn the “normal” behavior for the scene, for which our algorithm takes a bag-of-words approach. (In the same way a document can be considered a bag of words, a video can be considered a bag, or collection, of spatial-temporal cuboids, for which the interaction force is computed.) The idea is for the algorithm to use a training set of videos and match the interaction forces with given dynamics. A video in question can then be compared with those from the training set, and changes from the regular behavior in the scene are easily identified by the computer.

To improve the fidelity of the results, optical flow is smoothed by a Gaussian filter, where the standard deviation of the Gaussian distribution is empirically set to half the width of the typical person in the crowd. This smoothing compensates for the inaccuracies of optical flow in textureless regions. Moreover, using a bag of video words for several frames could also reduce the effects of inaccurate instantaneous optical flow.

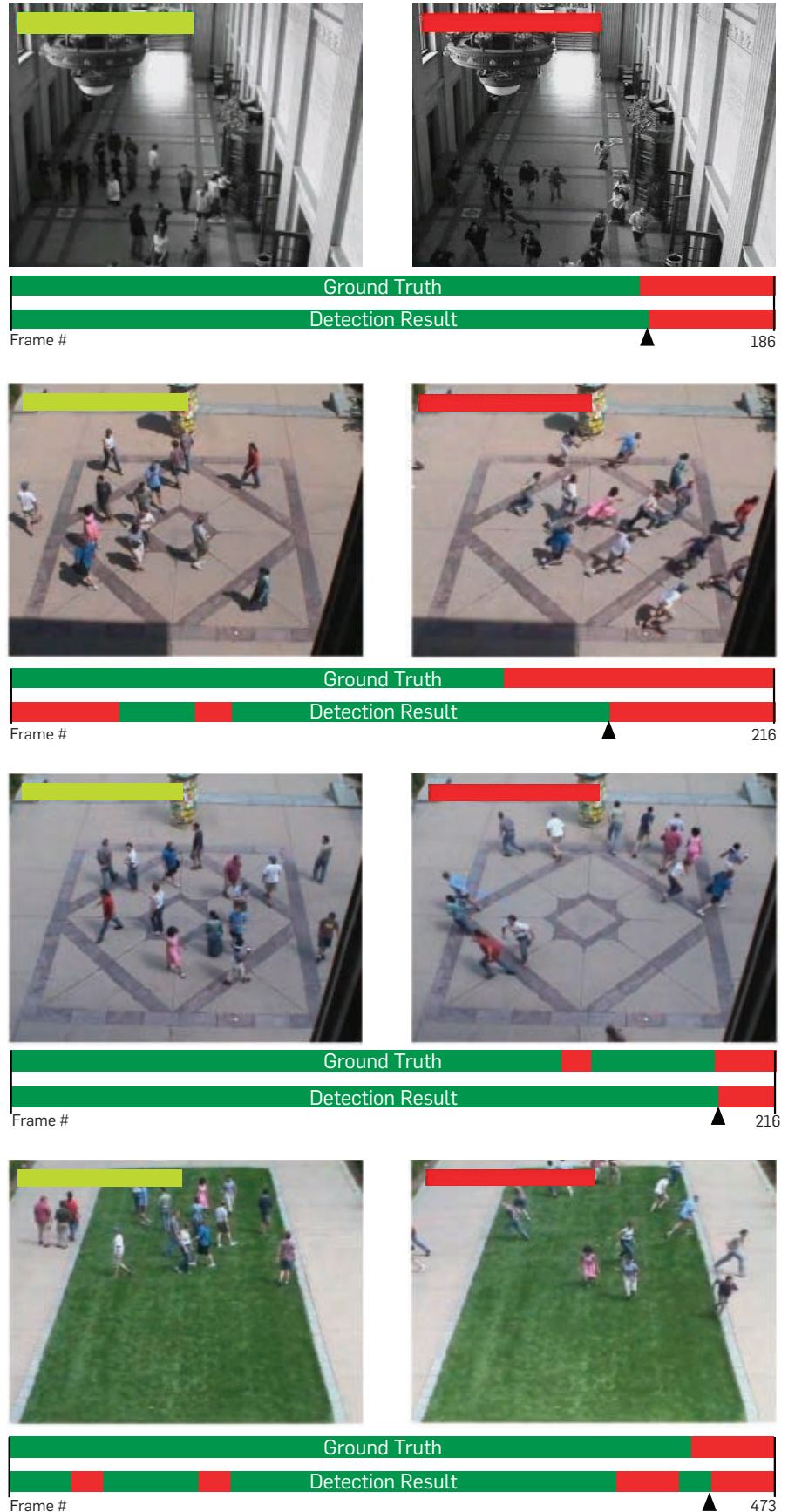


Figure 8. Frames from different sequences, showing (left) normal behavior (green) and (right) abnormal escape panic (red), comparing ground truth to abnormal behavior detection.

Sample results of the algorithm are in Figure 8; the videos for these experiments are from the University of Minnesota and show walking pedestrians as the normal behavior. At the end of each video the pedestrians suddenly run in all directions to escape the scene. The figure shows detection of abnormal behavior by our method (indicated as black triangles) compared with the ground truth. In most cases, panic detection occurs immediately following the change in behavior. The receiver operating characteristic curves in Figure 9 show a clear advantage of our method over simply using the optical flow to detect abnormal behaviors.

Microscopic scale (individual track-

ing). At the “atomic” level, a surveillance analyst is interested in automatically following a person in a high-density crowd, a very challenging problem, as the object our algorithm is tracking is subject to occlusion, and other nearby objects may lead the tracker away from the original object. Figure 10 shows tracking results using our method in which individuals are correctly tracked in four video sequences involving hundreds of people; each image shows the tracks overlaying a single frame of the video.

Inspired by research on evacuation dynamics,^{6,13} our method uses a scene-structure-based force model that likens pedestrians to particles, such that the forces acting on them determine

their direction and velocity. The algorithm computes the probability that a particular particle will move from one position to another, building on floor fields that provide information about the scene.² To make this clear, we make three assumptions about the flow influencing the individual’s behavior: First, the person has a goal (place to get to and clear direction how to get there) and, in the absence of obstacles, will go there directly; this is the effect of what is called the “static floor field.” Second, the person avoids permanent fixtures (such as trash cans and walls) and virtual barriers (such as opposing crowd flow) as a consequence of what is called the “boundary floor field.” And third, the person can move toward the goal only as the flow of the crowd allows; this motion and direction is the influence from the dynamic floor field.

A basic assumption on the static floor field, based on the observation that directions of motion in high-density crowds have dominant trends, is that crowd behavior remains constant during tracking. However, the static floor field can be updated periodically to respond to changes in the dominant trends. To respond to any instantaneous change in crowd flow, the model uses the dynamic floor, which is representative of instantaneous crowd behavior in the vicinity of the target. The main limitation of the floor-field tracking model is the inability to handle locations with no dominant trend (such as a crowded museum) and locations with more than one dominant trend (such as pedestrian crossings).

We begin our description of the method with the inference that people in crowds are constantly avoiding collisions. Hence, the boundary floor field is repulsive and computed easily through particle advection and the FTLE field, as described earlier in terms of segmentation of crowd flow. The edges of the computed segments give the boundaries of the flow, leading to the resulting boundary floor field (see Figure 11).

Computation of the static floor field (Figure 11d) is performed only once for a given video using a small subset of all video frames. The first step provides a representation of the instantaneous changes in motion, or “point flow,” achieved by calculating

Figure 9. ROC curves for detection of abnormal behaviors in the University of Minnesota data set; the area under the social force curve (red) is 0.96, and the area under the optical flow curve (blue) is 0.84.

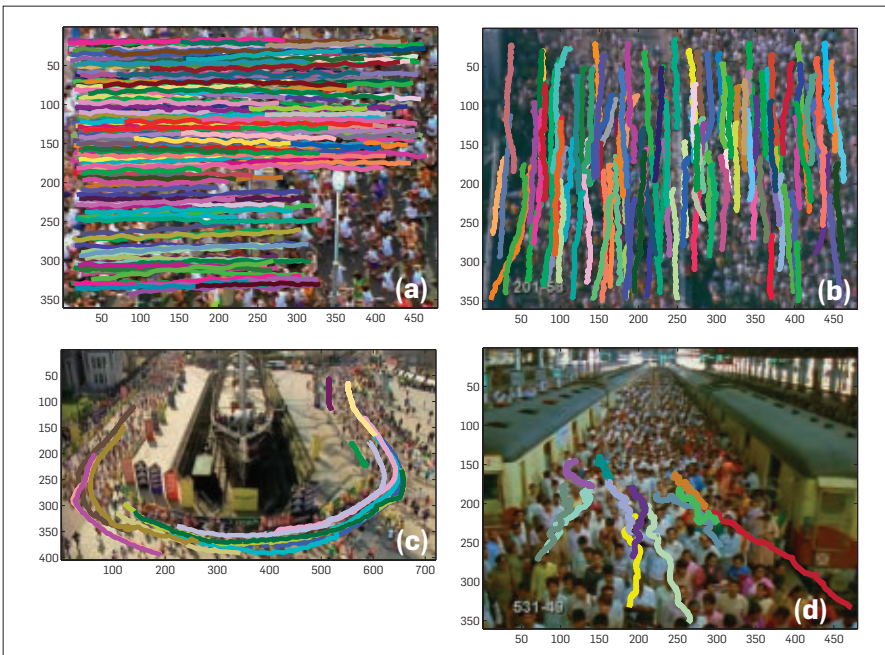


Figure 10. Tracking individuals using our method in (a–c) marathon scenes and (d) a crowded train station.

the average optical flow for each location over the entire subset of frames. Our algorithm can then place a grid of particles over the scene and determine the preferred direction of each particle based on the motion of neighboring point-flow vectors. If the influence is great enough to move the particle to the next cell, then the algorithm continues the process until the velocities are not significant enough to move it to the next position (see Figure 12). This process is used by the algorithm to find the sinks, defined computationally as the points where particle motion ceases to exist. In terms of crowd behavior, sinks are the desired goals or locations of the individuals in the crowd (such as preferred exits and frequently visited areas dominated by the flow of the crowd). The sinks, as well as the shortest distance needed to reach them, produce the static floor field.

Computing the dynamic floor field means discovering the behavior of the crowd around the individual. To do this, the algorithm uses the optical flow for a subset of video frames and performs particle advection. If a particle changes its position between frames, then the value of interaction between those frames is increased by one, and zero interaction is assumed at the first frame in this sequence. The individual's interactions in a local neighborhood are thus captured for that interval of time (see Figure 13).

To bring the three floor fields together for the purpose of tracking, the algorithm divides image space into cells, so each cell is occupied by one particle. The probability that a particle at cell i will move to neighboring cell j is then computed and combined with appearance information to complete the tracking. This method depends on computation of the influences from the static, dynamic, and boundary floor fields, denoted S_{ij} , B_{ij} , and D_{ij} , respectively, with each needed for accurately modeling the interaction of individuals and their preferred direction of motion. Described precisely, the probability that a particle will move from i to j is

$$p_{ij} = C e^{k_d D_{ij}} e^{k_s S_{ij}} e^{k_b B_{ij}} R_{ij}$$

where k_d , k_s , and k_b are the coupling

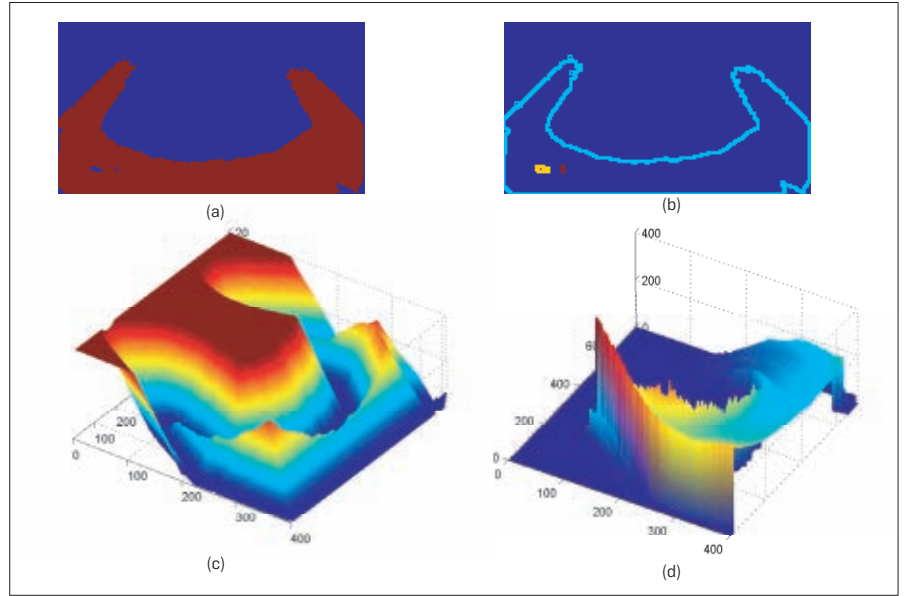


Figure 11. For the marathon sequence in Figure 10c: (a) crowd-flow segmentation obtained through particle advection; (b) the corresponding edge map; (c) the boundary floor field; and (d) the static floor field.



Figure 12. (a) The sink-seeking process. Red arrows signify the point flow influenced by neighboring points; the yellow curve is the sink path. (b) The sliding window used to find sinks; the solid circle is the point under consideration; hollow circles inside the box are neighboring points and outside the box non-neighboring points.

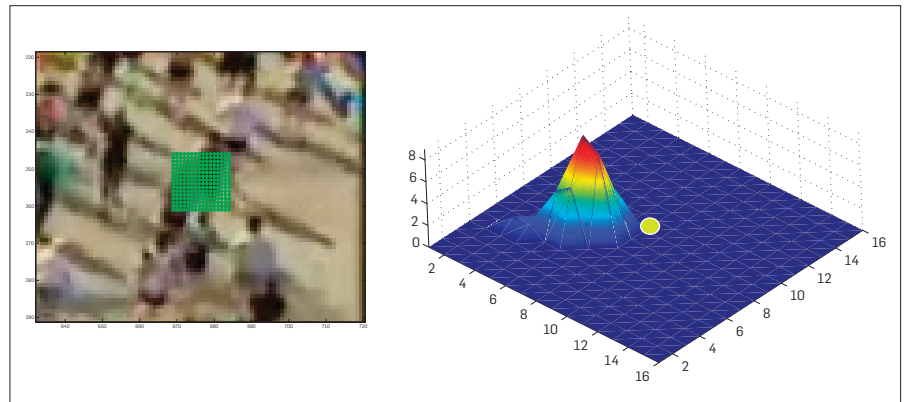


Figure 13. (left) Region for computing dynamic floor field, where green chip is a target individual; (right) dynamic floor field reflects strong relationship between the yellow cell and neighboring cells at the peak.

strength of the object to the respective field, C is a normalization constant, and R_{ij} is a similarity measure for the initial and updated appearance templates.

Experimentally, a target individual is selected by a surveillance analyst

by computing the gray-scale appearance template for a rectangular region, called a chip, surrounding the individual, with average chip size 14×22 pixels. The algorithm computes the position of the target at the next time instant according to the probable loca-

tion, as determined by the computed floor fields; the appearance similarity is then computed by normalized cross correlation, and the appearance template is automatically updated. Figure 14 charts results for 50 tracks in a video of a marathon, showing objects are tracked correctly in most cases.

Some tracking methods (though not ours) depend mainly on appearance information, but in crowded scenes appearance is not enough, as neighboring objects may have similar appearance. Figure 15a shows the appearance similarity surface for a marathon scene; the surface is relatively flat, so which runner is being tracked from

frame to frame is uncertain. However, by combining the dynamic and static floor fields (Figures 15b and c) with the appearance surface our method obtains a surface (Figure 15d) providing the best match for the tracked individual. Figure 16 also shows that, when using all three floor fields, the tracking error is consistently low, but if using only one floor field, the error increases, often significantly.

Conclusion

We have devised methods for segmenting motion, detecting abnormal behavior, and tracking individuals in video scenes of high-density crowds.

Our underlying supposition is that people in crowds appear to move according to the flow, like particles in a liquid. Hence, we gaze through a hydrodynamics lens to analyze video scenes in various scenarios on three different length scales. Each of our methods relies to some extent on the optical flow and associated particle advection adapted from the Lagrangian approach to fluid dynamics.

Our experimental results have been excellent, and we expect the underlying hydrodynamics theme can be taken further to solve other problems in visual surveillance of high-density crowds. Ultimately, we envision the ability to predict potentially hazardous situations in crowded scenes, though it is work for the future. Training a computer to decipher and understand crowd behavior from a video sequence is extremely challenging; aside from having to sort through a plethora of digital information, there are also questions specific to each of the three problems—segmenting motion, detecting abnormal behavior, tracking individuals—discussed here.

For crowd segmentation, our method makes use of flow maps corresponding to each particle, computing maximal Lyapunov exponents to reveal segments of coherent motion in the scene.¹ Our method performs well for steady flows, with no changes in geometry, but segmentation for unsteady flows is an open problem with several challenges. Coherent flow segments in crowds can change quickly, and to capture such changes, an algorithm must distinguish changes within segments from changes in segment boundaries. One location in a scene may also exhibit alternating collective patterns of motion, meaning several segmentations are needed to describe different modes within a single region. In addition, modeling abstract human behaviors that help define segments (such as courteous acts, social agreement, and individual intention) is difficult. Moreover, scenes can grow more complex, as moving/cluttered back/foregrounds are important in segmentations more discriminating than ours.

For detecting abnormal behavior, our method approximates the interaction forces in the crowd to build a model for the motion, detecting anomalies as deviations from the norm.¹⁷

Figure 14. Computed track lengths vs. ground truth for a marathon sequence.

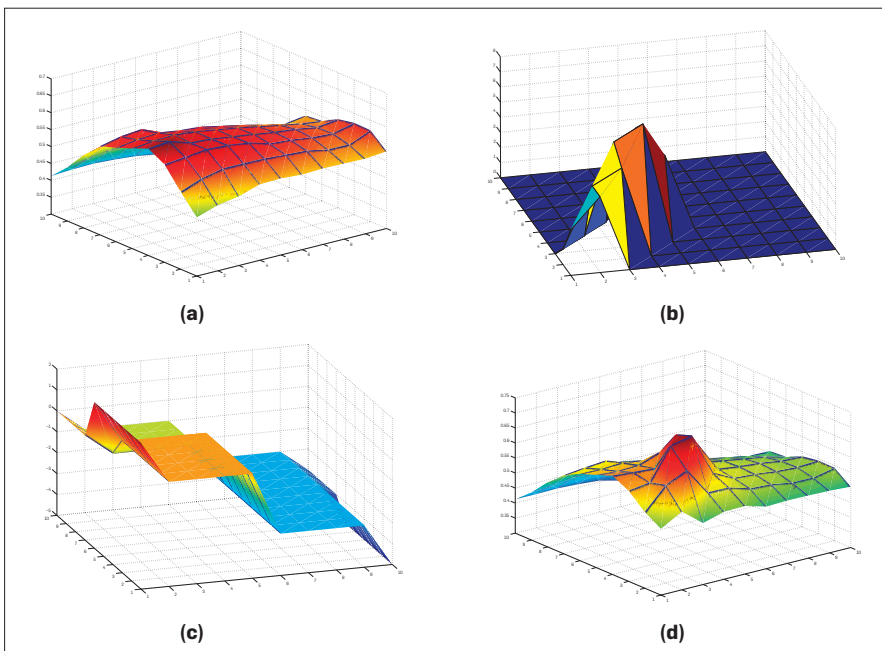
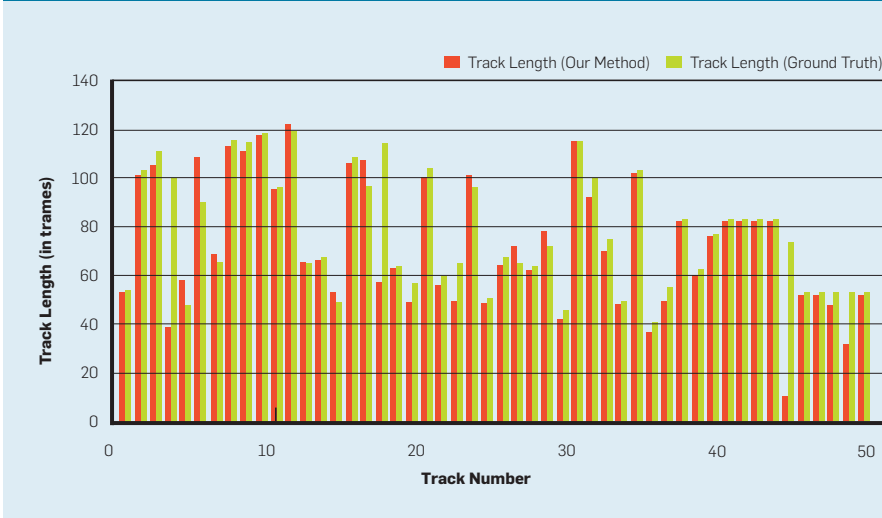
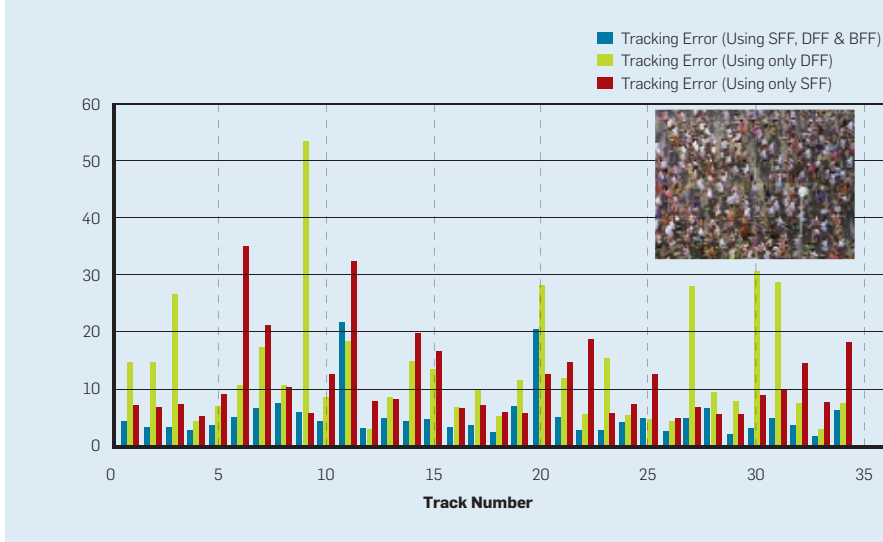


Figure 15. For tracking a runner in a marathon sequence: (a) appearance similarity surface; (b) dynamic floor field; (c) static floor field; and (d) final decision surface.

Figure 16. Average tracking error for each object in a marathon sequence using only dynamic (green), only static (maroon), and all three (blue) floor fields.



This approach works well for detecting global changes in regular motion, but detecting smaller (more local) events is more difficult. Our method is also, by design, good at measuring the forces individuals exert on one another but is unable to recognize specific behaviors and distinguish the acceptable from the unacceptable. This limitation stems from an enormous variety in the behaviors observed in crowded scenes, along with the difficulty of distinguishing certain activities from other activities. Some behaviors are easily defined (such as bottlenecks or lanes), but formulating clearly defined behaviors for general crowd motion is difficult, as is categorizing unsteady flows, with the flow constantly changing.

For tracking in high-density crowds, our method exploits the influences of boundaries, neighboring pedestrians, and desired direction, along with appearance information, to identify the position of a target in subsequent frames.² Our algorithm produces excellent results for extremely crowded scenes, where the tracked individual is highly influenced by the flow of the crowd, but tracking in crowds that are less dense, allowing pedestrians to move against the flow, still involves many research problems; for example, crowd dynamics involve psychological aspects (such as preferences and habits) that influence individual behavior, thereby increasing scene complexity. Aside from the thoughts and intent of individuals, the constant interaction

among them makes it difficult to distinguish one from another. In addition, occlusions result in loss of observation of a target object, while the object's appearance (such as shape and color) varies, not only from one setting to the next, but also as a given setting evolves.

Acknowledgment

This article summarizes and incorporates three earlier publications: Ali et al.,¹ Ali and Shah,² and Mehran et al.¹⁷ The research is partially supported by the U.S. Army Research Office, part of the U.S. Army Research Laboratory, under grant number W911NF-09-1-0255 and by the U.S. Department of Defense.

References

1. Ali, S. and Shah, M. A Lagrangian particle dynamics approach for crowd flow segmentation and stability analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Minneapolis, June 18–23, 2007), 1–6.
2. Ali, S. and Shah, M. Floor fields for tracking in high-density crowd scenes. In *Proceedings of the 10th European Conference on Computer Vision* (Marseille, France, Oct. 12–18), Springer, 2008.
3. Andrade, E.L., Blunsden, S., and Fisher, R.B. Modeling crowd scenes for event detection. In *Proceedings of the 18th International Conference of Pattern Recognition* (Hong Kong, Aug. 20–24, 2006).
4. Bennett, A. *Lagrangian Fluid Dynamics*. Cambridge University Press, New York, 2006.
5. Brostow, G. and Cipolla, R. Unsupervised Bayesian detection of independent motion in crowds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (New York, June 17–22, 2006).
6. Burstedde, C., Klauck, K., Schadschneider, A., and Zittartz, J. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications* 295, 3–4 (June 2001), 507–525.
7. Chan, A.B. and Vasconcelos, N. Bayesian poisson regression for crowd counting. In *Proceedings of 12th IEEE International Conference on Computer Vision* (Kyoto, Sept. 27–Oct. 4, 2009), 545–551.

8. Chan, A.B. and Vasconcelos, N. Mixtures of dynamic textures. In *Proceedings of the 10th IEEE International Conference on Computer Vision* (Beijing, Oct. 17–20, 2005), 641–647.
9. Helbing, D. Traffic and related self-driven many-particle systems. *Review of Modern Physics* 73, 4 (Dec. 2001), 1067–1141.
10. Helbing, D. and Molnar, P. Social force model for pedestrian dynamics. *Physical Review E* 51, 5 (May 1995), 4282–4286.
11. Hughes, R.L. The flow of human crowds. *Annual Review of Fluid Mechanics* 3 (2003), 169–182.
12. Hughes, R.L. A continuum theory for the flow of pedestrians. *Transportation Research (Part B: Methodological)* 36, 6 (July 2002), 507–535.
13. Kirchner, A. and Schadschneider, A. Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. *Physica A: Statistical Mechanics and its Applications* 312, 1–2 (Sept. 2002), 260–276.
14. Kotelenetz, P. *Stochastic Ordinary and Stochastic Differential Equations*. Springer, New York, 2008.
15. Kratz, L. and Nishino, K. Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Miami, June 20–26, 2009), 1446–1453.
16. Marques, J.S., Jorge, P.M., Abrantes, A.J., and Lemos, J.M. Tracking groups of pedestrians in video sequences. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Workshop* (2003), 101.
17. Mehran, R., Oyama, A., and Shah, M. Abnormal behavior detection using social force model. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Miami, June 20–26, 2009), 935–942.
18. Pellegrini, S., Ess, A., Schindler, K., and van Gool, L. You'll never walk alone: Modeling social behavior for multi-target tracking. In *Proceedings of the 12th IEEE International Conference on Computer Vision* (Kyoto, Sept. 27–Oct. 4, 2009).
19. Reisman, P., Mano, O., Avidan, S., and Shashua, A. Crowd detection in video sequences. In *Proceedings of the IEEE Intelligent Vehicles Symposium* (Parma, Italy, June 14–17, 2004), 66–71.
20. Sand, P. and Teller, S. Particle video: Long-range motion estimation using point trajectories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (New York, June 17–22, 2006), 2195–2202.
21. Shadden, S.C., Lekien, F., and Marsden, J.E. Definition and properties of Lagrangian coherent structures from finite time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena* 212, 3–4 (Dec. 2005), 271–304.
22. Tu, P., Sebastian, T., Doretto, G., Krahnstoeber, N., Rittscher, J., and Yu, T. Unified crowd segmentation. In *Proceedings of the 10th European Conference on Computer Vision* (Marseille, Oct. 12–18, 2008), 691–704.
23. Yilmaz, A., Javed, O., and Shah, M., Object tracking: A survey. *ACM Computing Surveys* 38, 4 (2006), 131–1345.
24. Zhan, B., Monokosso, D.N., Remagnino, P., Velastin, S.A., and Xu, L. Crowd analysis: A survey. *Machine Vision and Applications* 19, 5–6 (2008), 345–357.
25. Zhao, T. and Nevatia, R. Tracking multiple humans in a crowded environment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Washington, D.C., June 27–July 2, 2004), II-406–II-413.

Brian E. Moore (bmoore@math.ucf.edu) is an assistant professor of mathematics in the Department of Mathematics at the University of Central Florida, Orlando, FL.

Saad Ali (saad.ali@sri.com) is a computer scientist in the Vision Technologies department at SRI International Sarnoff, Princeton, NJ.

Ramin Mehran (ramin@cs.ucf.edu) is a Ph.D. student in the Computer Vision Lab at the University of Central Florida, Orlando, FL.

Mubarak Shah (shah@cs.ucf.edu) is Agere Chair Professor of Computer Science and founding director of the Computer Vision Lab at the University of Central Florida, Orlando, FL.

© 2011 ACM 0001-0782/11/12 \$10.00

DOI:10.1145/2043174.2043193

Software developers' reuse of code from the Internet bears legal and economic risks for their employers.

BY MANUEL SOJER AND JOACHIM HENKEL

License Risks from Ad Hoc Reuse of Code from the Internet

REUSING EXISTING SOFTWARE artifacts when developing new software is an attractive way to reduce development costs and time to market while improving software quality.⁴ Code is the artifact most commonly reused in software development.¹⁶ Researchers have identified such reuse in commercial software development as a new facet of software reuse.^{13,22} Here, “Internet code” means code in the form of components (such as a library encapsulating required functionality) and snippets (such as containing a synchronization block) that can be downloaded from the Internet for free and without individual agreement with the originator; an important instance of such code is publicly available open source software (OSS). Internet code generally

includes permission to be reused in commercial software development,¹⁴ making it highly attractive for firms.² Therefore, some firms systematically reuse it by including identification, evaluation, and integration of suitable code in their development processes.¹⁸ Alternatively, Internet code can also be reused in ad hoc fashion, as described in Umarji et al.,²³ with individual professional developers, on their own and typically without telling anybody, searching the Internet for existing code as a shortcut in their work, downloading and integrating it into the software they develop.^a

Despite its general suitability for reuse in commercial software, Internet code is rarely in the public domain and usually under licenses that demand compliance with specific conditions as a prerequisite for reuse.⁸ These conditions vary widely and may, for example, demand attribution of the original creators of the reused code. More critical for firms are the obligations demanded by the GNU General Public License (GPL)^b

a Places to search for code include OSS repositories (such as SourceForge.net), code search engines (such as Koders.com), and code bases of related OSS projects; for a detailed overview and quantitative analyses, see Sojer.²⁰

b The GPL is a family of licenses, including versions 1, 2, and 3; since all versions share the “copyleft” obligation, we refer to the whole family as “the GPL” throughout this article.

» key insights

- Professional software developers reuse code freely available on the Internet (such as open source code) in their commercial projects in ad hoc fashion.
- Such code often comes with license obligations; noncompliance can mean legal and economic risk, but developers are often not sufficiently knowledgeable in these matters.
- Firms should establish clear policies regarding reuse, leveraging reliable information resources on the Internet and complementing them with internal training, lobby universities to include the topic in their curricula, and acknowledge the interdisciplinary nature of the issue.

tatively investigated license issues from OSS code reuse through the analysis of code bases and software distributions.

To complement this work, we employed quantitative data obtained from a global survey we conducted in 2009 involving 869 professional software developers to explore ad hoc reuse of Internet code, with a special focus on license issues. Our findings should provide firms with a starting point for assessing their exposure to license risks from their developers' ad hoc reuse of Internet code and devising measures to avoid potential related liabilities.

Survey

We developed the questionnaire following our literature review and 20 interviews with industry experts.^d Before conducting the survey, we enlisted four academic peers and 113 software de-

velopers to pre-test the questionnaire. We chose a survey-based research approach over an analysis measuring the share of reused Internet code in commercial software code bases. While this setup did not allow us to calculate a precise percentage of reuse of Internet code in commercial software development, it did allow us to include more professional software developers. Moreover, if deviations between developers' actual and survey-reported reuse would arise, they would be unlikely to be systematic and thus should not affect the results of our multivariate analyses.

Since we were among the first to investigate ad hoc reuse of Internet code by individual professional software developers, we opted not to use a limited sample of developers from a single firm but rather a broad and heterogeneous group of professional software developers active in Internet newsgroups as our survey popu-

lation.^e We extracted a total of 93,541 unique email addresses from more than one million messages posted over the previous three years in 528 newsgroups dealing with software development.^f After cleaning the addresses, we selected a random sample of 14,000 addresses and invited the newsgroup participants via email messages to take our online survey. We received 1,133 fully filled-in responses, yielding a response rate of 9.9% (consistent with other Internet surveys).^g Of them, 869 responses were submitted by current or former professional software developers who are the focus of the analyses discussed in the following sections.

The vast majority (98%) of the 869 professional software developers we surveyed was male, with average age 35.6, living in Europe (53%), North America (28%), Asia (12%), and South America (4%); 56% had previously contributed to OSS. At the time of the survey, in 2009, 79% of the developers were employed as professional software developers; the others had been working as professional developers but had quit before 2009.^h On average, survey participants had 9.7 years of work experience as professional developers in 2009, most as programmers (51%), others as software architects (28%) and project managers (4%); 23% were employed as freelancers in 2009, and the others worked on permanent contracts.

Also at the time of the survey, 54% of the developers worked for firms for which software development was the main business, with 68% developing software for external customers, the rest for internal use in their firms. Among the 68% writing software for external customers, 62% were creating off-the-shelf software for multi-

^e Potential limitations of our approach are discussed later in the section on threats to validity.

^f The 528 newsgroups included all main and high-traffic groups (such as comp.lang.c++ and comp.lang.java.programmer).

^g To calculate response rate, we adjusted the number of invitations we sent to potential survey participants by the number of email messages that did not reach their designated recipients.

^h In the following sections, we report the characteristics of the last software development activities of developers who quit creating software.

Figure 1. Extent of ad hoc reuse of Internet code, 2009.

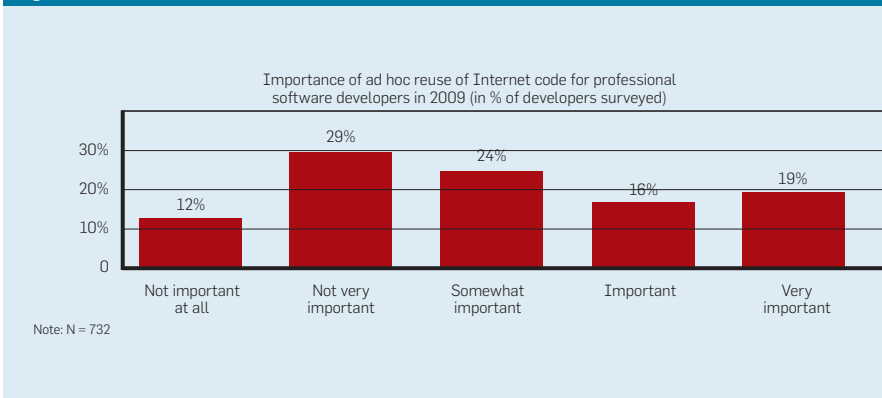
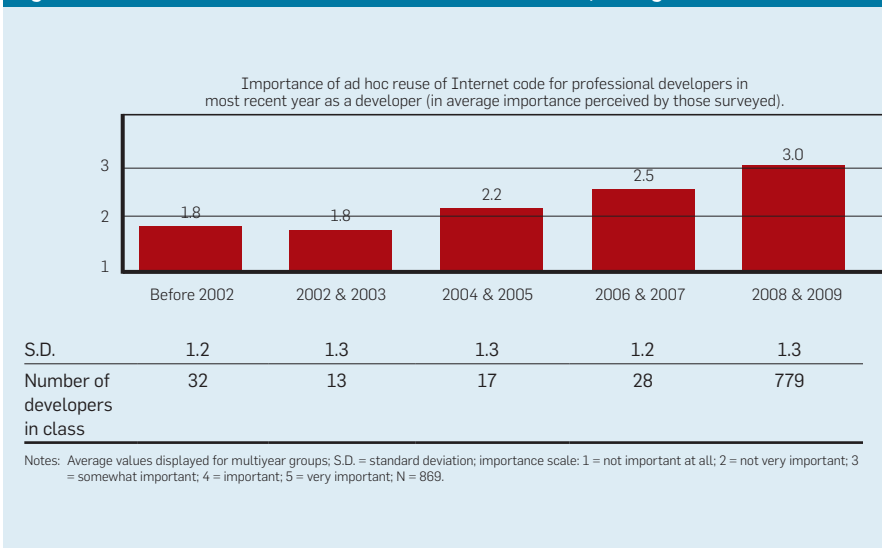


Figure 2. Evolution of extent of ad hoc reuse of Internet code, through 2009.



ple customers, and the rest developed custom software. These distinctions are important because the license risks resulting from reuse of Internet code are typically more severe for software developed for multiple external customers.

Extent of Code Reuse

To quantitatively assess the extent of ad hoc reuse of Internet code in commercial software development, we asked survey participants to indicate how important reusing Internet code (components and snippets) in an ad hoc fashion was for their work.

Outlining the perceptions of professional software developers active in 2009, Figure 1 reflects that ad hoc reuse was an essential part of the work of many professional developers. More than half of those we surveyed (59%) considered ad hoc reuse of Internet code at least “somewhat important” for their work, while only 12% apparently did not reuse any Internet code in ad hoc fashion. This finding contrasts with the prevailing assumption of many firms that their code base does not or only to a small, controlled degree contain Internet code.¹⁵

In addition to analyzing the extent of ad hoc reuse of Internet code, we also investigated the historic development of such reuse. Figure 2 includes the perceptions of professional software developers who quit creating software before 2009. Since we asked survey participants about their last year as active developers, their responses are informative about the respective year. Our survey data shows that starting with 2004 the importance of ad hoc reuse of Internet code for professional software developers had increased, rising from a mean importance value of 1.8 (“not very important”) in 2002 and 2003 to 3.0 (“somewhat important”) in 2008 and 2009.

A possible interpretation is that before 2004, code available from the Internet might have only rarely been suited for reuse in commercial software development because it was not mature enough and covered only a few functional areas. However, resulting from the strong recent growth of OSS,³ both the quality and the fields for which code exists should have increased strongly, thus making Internet

code reuse much more attractive to professional developers.

Determinants of Code Reuse

To understand which factors most influence the importance professional

software developers attribute to ad hoc reuse of Internet code we conducted an exploratory regression analysis with the data collected in our survey. The model (see Table 1) employs an ordered logistic regression¹⁰ and the

Table 1. Multivariate analysis of the importance of ad hoc reuse of Internet code.¹³

	Ordered Logistic Regression	
	Coef.	Std. Err.
License risk level of developer's work	0.111	0.085
Developer has never received any form of training or information on Internet code reuse (dummy)	-0.258	0.167
Developer's self-assessed knowledge about Internet code licenses	0.442***	0.099
Developer's objectively assessed knowledge about Internet code licenses	-0.032	0.057
Developer has OSS experience (dummy)	0.391***	0.143
Experience as professional software developer (in years)	0.017*	0.009
Last year as professional software developer	0.197***	0.043
Software development role (dummies, reference group: architect)		
Project manager	0.155	0.358
Programmer	-0.356**	0.149
Analyst	-0.943	0.969
Tester	-1.176	0.717
Database developer	-0.751**	0.350
Other	-0.281	0.241
Primary programming language (dummies, reference group: Ruby)		
Python	-0.284	0.276
Perl	-0.861**	0.435
Java	-1.015***	0.268
PHP	-1.533***	0.381
C	-1.550***	0.333
C++	-1.808***	0.269
Visual Basic	-2.001***	0.516
C#	-1.957***	0.315
Other	-1.842***	0.258
Developer lives in...(dummies, reference group: Europe)		
...North America	0.016	0.164
...South America	0.727**	0.337
...Asia or rest of world	-0.206	0.210
Developer is working as a freelancer (dummy)	0.041	0.163
Education (dummies, reference group: engineering)		
Computer science or related subject	-0.223	0.158
Mathematics or physics	-0.300	0.251
Business administration	-0.222	0.421
Other subject	0.147	0.258
Developer is working on embedded software projects (dummy)	-0.159	0.185
Developers' self-assessed software development skill level	-0.048	0.087
Observations	807	
Pseudo R ²	0.09	
Wald test	X ² (32)=208.94, p<0.0001	
Cuts	393.318, 395.001, 396.161, 397.153	

* significant at 10%, ** significant at 5%, *** significant at 1%

Notes: Significant coefficients are in bold type; reported standard errors are robust standard errors.

Descriptive statistics and the correlation table of the explanatory variables used are available from the authors on request.

perceived importance of ad hoc reuse for the individual work of professional developers measured on a five-point scale as a dependent variable. As independent variables we included multiple characteristics of professional developers, some as dummy variables. Regression coefficients are not standardized, such that the range or standard deviation of a variable must be taken into account when assessing the variable's effect on the importance professional developers attribute to ad hoc reuse in their work.

First, the model results point out that developers' ad hoc reuse seemed to be independent of the "license risk level"; that is, developers creating software to be sold to multiple exter-

nal customers did not deem ad hoc reuse as less important than developers working on custom software or software for internal firm use. A possible interpretation is that developers, in deciding to reuse Internet code, did not acknowledge the real possibility of negative legal and economic consequences their employers might face due to license violations. However, we can also think of two alternative explanations: One could assume less reusable code was available for internal use or custom software due to its tailored nature; and one could also imagine that while not considering ad hoc reuse less important, professional developers were still more careful when reusing such code in development projects for multiple external customers.

Moreover, developers who never had any training or information on reusing Internet code and thus should be more likely to create license issues

did not differ significantly^j in their view of the importance of ad hoc reuse of Internet code from developers who were trained or had received such information. Also, while developers who self-assessed their knowledge about Internet code licenses better also deemed ad hoc reuse of Internet code reuse more important, this relationship does not hold for an objective assessment of developer proficiency regarding licenses for the code.^k If we (plausibly) assume that the results of our objective assessment are more informative about developers' license-related knowledge than their self-assessment, we can also assume that developers, at least as of 2009, on average did not correctly account for their own knowledge about licenses for Internet code when considering ad hoc reuse of Internet code.

The model also indicates that developers who had been active in OSS projects and those with longer experience as professional developers considered ad hoc reuse significantly more important.^l A plausible interpretation of this finding, consistent with Sojer and Henkel,²¹ is that for OSS-savvy developers, the costs of searching, evaluating, and understanding Internet code should be lower than for developers with less OSS experience. Likewise, more senior developers should face lower costs for reuse due to their typically larger personal networks and reuse experience. The

i We set "license risk level" to 1 if developers were working on internal projects, to 2 if they were working on external projects for only one customer, and to 3 if they were working on projects for multiple external customers.

j Throughout this article, "significant" means "statistically significant."

k This objective assessment of developer knowledge of the obligations of reusing Internet code is based on a five-question quiz in our survey. We developed the quiz following 20 interviews with experts in the reuse of Internet code. The quiz covers five typical scenarios in which professional software developers may violate license obligations when reusing Internet code. One might conjecture that the insignificance of the objectively assessed knowledge is caused by the fact that it is correlated with the self-assessed knowledge. However, this insignificance persists when the self-assessed knowledge level is dropped from the list of explanatory variables.

l Note regression coefficients are not standardized. Since "experience as professional software developer" is measured in years, ranging from 0.5 to 45, its effect is comparable in size to the dummy variable "developer has OSS experience." The coefficient of the latter variable is much larger (0.39 vs. 0.017), though its range is much smaller (1 vs. 44.5).

Figure 3. Sources for learning about reuse of Internet code, 2009.

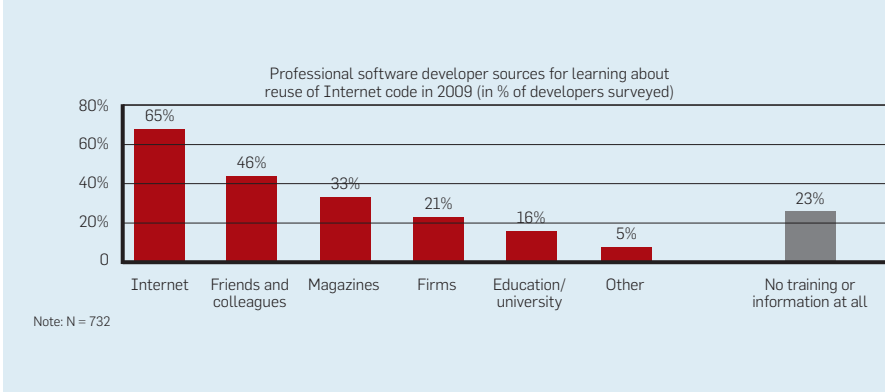


Table 2. Software developer familiarity with license obligations concerning Internet code, 2009.

	Not familiar at all	Not very familiar	Somewhat familiar	Familiar	Very familiar
Share of developers self-assessing their familiarity with Internet code license obligations in the respective groups	2%	3%	29%	50%	16%
Developers' average score in quiz on license obligations from Internet code reuse (max. score attainable: 5, average score across all groups: 2.54)	0.88	1.50	2.08	2.74	3.11

Note: N=732.

multivariate model also supports the result outlined in Figure 2, showing the perceived importance of ad hoc reuse of Internet code grew significantly from 2004 to 2009.

Moreover, the developers we surveyed had different views of the importance of ad hoc reuse of Internet code depending on their development role. Programmers and database developers attributed significantly less importance to it than the architects we defined as a reference group. For all other roles, the difference with the “architects” was insignificant at a 10% level. The finding that architects deemed ad hoc reuse significantly more important than programmers is startling since architects should be concerned with systematic rather than ad hoc reuse. However, architects, especially in smaller and mid-size firms, might also take on programmer responsibilities and leverage their greater architectural latitude to reuse Internet code in an ad hoc fashion. The architecture of a piece of software influences how easy it should be to reuse external code.⁵ Shaping architecture, architects might have more control over reusing Internet code than programmers for whom the architecture of the software they develop is often exogenous. Moreover, greater architectural latitude could also allow developers to integrate Internet code in such a way as to avoid license violations,⁹ assuming developers are aware of the relevant issues in the first place. Supporting this line of thought, our survey found that architects are significantly more knowledgeable regarding licensing topics than other developers, including programmers. Architects should still be able to reuse Internet code properly, while programmers would have to choose between reusing the code in a way that violates the code’s license obligations and not reusing it at all.

The main programming language developers were using influenced how they viewed ad hoc reuse in their work. For example, developers relying mainly on Ruby or Python found ad hoc reuse most important, followed by those working with Perl, Java, PHP, and other such languages. Developers using more traditional programming languages (such as C and C++), less

common ones (such as Visual Basic and C#), and various others formed the last group viewing code reuse as least important.

While one could conjecture that diverse legal systems (such as common law vs. civil law), cultural variations, and the availability of Internet code in local language lead to different views of the importance of ad hoc reuse in different geographies, our survey did not find substantial support for such reasoning; Asian, European, and North American developers did not differ significantly in how they perceived the importance of ad hoc reuse; only South American developers deemed such reuse significantly more important. However, since only 33 South American developers participated in the survey, this finding may not be representative.

Finally, our survey did not find significant differences in professional developers’ perception of the importance of ad hoc reuse based on their education and skills and whether they develop embedded or traditional software or were employed, at the time of the survey, in time-limited contracts (such as freelancers) or as permanent employees.

Developer Knowledge and Risks for Firms

How well are professional software developers prepared to deal with the licenses and obligations associated with ad hoc reuse of Internet code?

It seems reasonable to assume that professional developers who are more aware of the particularities of Internet code (such as its licenses) are less likely to ignore license obligations. Thus, we first investigated whether professional software developers had received training or information on reuse at the time of the survey and the sources of such training and information (see Figure 3).

Two rather informal channels—the Internet (65%) and friends and colleagues (46%)—were developers’ reported main sources of information about Internet code licenses and their particularities. Comparatively unimportant were firms (21%) and educational institutions, including universities (16%). Meanwhile, 23% of the developers we surveyed had not received any form of training or infor-

mation on the reuse of Internet code. Overall, these findings suggest that conveying knowledge about reusing Internet code and potential license risks was not high on the agenda of firms and universities, at least until 2009.

Given the high number of developers surveyed who reported never having received training or information on the reuse of Internet code or who relied on information from unofficial channels (such as the Internet and friends), we were compelled to investigate their knowledge of licenses for such code. When self-assessing their knowledge, two-thirds of surveyed developers reported being “familiar” or “very familiar” with nearly all obligations in Internet code licenses (see Table 2). Contrasting this self-assessment with the results of our five-question quiz about license obligations resulting from the reuse of Internet code (discussed earlier) suggests developers overestimated their knowledge. Even those who viewed themselves as “very familiar” with license obligations on average failed on two questions in our quiz, obtaining a mean score of 3.11 out of a maximum of 5.^m Moreover, while positive and statistically significant ($p < 0.001$), the correlation between self-assessment and quiz score in the survey was weak, with a correlation coefficient of 0.345.

We also sought to identify the factors that influence developers’ objectively assessed knowledge about Internet code licenses and their obligations. The exploratory Tobit¹⁰ regression model (see Table 3) uses developers’ scores in the survey’s license quiz as the dependent variable. The results underscore that developers with OSS experience were significantly more knowledgeable about Internet code licenses than other developers. Furthermore, most forms of training and information about reusing Internet

^m We pre-tested the quiz questions to make sure, as much as possible, they were of comparable difficulty and relevance. However, there was some variation among them, and it is possible that respondents who described themselves as “very familiar” with the obligations of licensing Internet code (and who failed on average to answer 1.89 questions out of 5 in the survey), often struggled with questions on license issues that appear less frequently and are thus less critical for firms.

code (from firms, friends, colleagues, magazines, and other sources) did not exert significant influence on developer knowledge. Developers who had received training or information in educational institutions were significantly less proficient than other developers. Only information acquired from the Internet had a significant positive effect on developer knowledge.

Along with these factors, the developers from Asia and North America seemed to know less about Internet code licenses than their European and South American counterparts in 2009. Regarding educational backgrounds, developers with academic degrees in computer science and engineering were more proficient regarding Internet code licenses than other developers.

In the situation described earlier in which ad hoc reuse of Internet code

seemed prevalent while also exposing firms to risks, it would seem reasonable for firms to introduce explicit policies providing guardrails to developers considering reuse of Internet code.

However, only about one-third of the developers we surveyed worked in firms with policies regulating such reuse. More detailed analysis of this matter emphasizes that firms with more than 5,000 employees were 31% more likely to have such policies, while there was no significant difference among smaller firms of various sizes.ⁿ Moreover, firms for which software development was the main business had a 19% greater probability of having such policies, while firm age had no consis-

n These findings result from exploratory logistic regression analyses and resulting marginal effects not covered here; full regression tables are available from the authors.

tently significant effect on whether or not a firm had such policies.

Of the developers working in firms with policies regarding Internet code reuse, nearly one-quarter reported not to have read them. Programmers were less likely to have read policies than architects; also, developers unhappy with their jobs were significantly less likely to have read their employers' policies.^o Additionally, developers who were not involved in development projects for multiple external customers were significantly less likely to have read the policies.

As a consequence of the overall situation regarding the ad hoc reuse of Internet code described here, it is not surprising that our survey found that 21% of the developers creating software in 2009 had at least once not checked thoroughly for Internet code license obligations when reusing snippets; 16% did the same when reusing components; and 14% ignored license obligations they were aware of when reusing snippets.

Threats to Validity

Given the multiple variables in our regression models, the size of our sample, and significance levels reported, our results should reflect statistical validity. However, the threats to internal, construct, and external validity of this work should be addressed in future research.

In terms of internal validity, the explanatory and control variables in our models should ensure no omitted variable biases influence our survey results. However, since our questionnaires were completed anonymously by developers identified through email addresses, we cannot be sure of the accuracy and truthfulness of the answers to our questions.

Regarding construct validity, the main dependent variable of our research is the perceived importance of ad hoc reuse of Internet code for developers' individual work. While this variable is a suitable proxy for the extent to which professional software developers practice ad hoc reuse, fu-

o These findings result from exploratory logistic regression analyses and resulting marginal effects not covered here; full regression tables are available from the authors.

Table 3. Multivariate analysis of software developer knowledge concerning Internet code licenses.¹⁷

	Tobit Regression	
	Coef.	Std. Err.
Developer has OSS experience (dummy)	0.835***	0.098
Developer has received training or information on Internet code from...(dummies)		
...firms	0.124	0.120
...educational institutions	-0.243*	0.126
...friends and colleagues	0.080	0.112
...Internet	0.390***	0.122
...magazines	0.089	0.112
...other sources	-0.091	0.213
Developer lives in...(dummies, reference group: Europe)		
...North America	-0.238**	0.117
...South America	-0.119	0.222
...Asia or rest of world	-0.297**	0.142
Education (dummies, reference group: computer science or related subject)		
Engineering	0.073	0.124
Mathematics or physics	-0.320*	0.170
Business administration	-0.751**	0.354
Other subject	-0.385**	0.184
Experience as professional software developer (in years)	0.002	0.007
Constant	1.890***	0.141
Observations	869	
Pseudo R ²	0.04	
F test	F(15, 854)=8.62, p<0.0001	
σ	1.376	

* significant at 10%, ** significant at 5%, *** significant at 1%

Notes: Significant coefficients are in bold type; reported standard errors are robust standard errors. Descriptive statistics and the correlation matrix of the explanatory variables used are available from the authors on request.

ture research might want to take more direct measures to check the robustness of our findings and conclusions. Moreover, despite our extensive pretest with more than 100 developers, it might be possible that some survey participants misunderstood the meaning of some of our survey questions.

Addressing external validity, there is still the risk that our survey population of 869 developers active in Internet newsgroups is not representative of professional developers in general. Since this research is among the first to quantitatively investigate ad hoc reuse of Internet code by individual developers, we deliberately chose developers from newsgroups to ensure broad heterogeneity in our sample. Moreover, the comparison of the demographics of our sample with that of other recent studies among professional developers (such as Alexy¹) gives us confidence in the representativeness of our sample. Still, it would be worthwhile to repeat our study in a more homogeneous single-firm setting.

Conclusion

Our analyses of ad hoc reuse of Internet code in commercial software development suggest its importance has increased over time; in 2009 over 50% of the developers we surveyed deemed ad hoc reuse at least “somewhat important” for their own work. This result differs from the prevailing assumption of many firms that their code base does not or only to a small and controlled degree contains Internet code.¹⁵

Addressing the knowledge of professional developers about Internet code licenses and their legal obligations, we found about one-quarter of them had never received any form of training or information on the topic. Only a small fraction had received training or information from firms or from educational institutions. Moreover, many existing forms of training and information were apparently not effective.

As a consequence of this lack of useful training and information, many developers, at least in 2009, lacked detailed knowledge about their obligations potentially resulting from the reuse of Internet code. Despite this, only a minority of firms

had deployed policies addressing reuse of Internet code in 2009. Consequently, a considerable share of developers—14%–21% of our sample, depending on scenario—had at some point either not checked thoroughly for license obligations or even knowingly ignored them when reusing Internet code in the past.

Firms must recognize and acknowledge the existence of Internet code in their own code bases. Given our findings, they should further consider that some of the Internet code reused in their software might also violate license obligations.

Our study offers multiple levers for firms to mitigate the economic and legal risk from ad hoc reuse of such code. First, the topic itself must be positioned more prominently on their agendas. Firms should actively make developers aware of the potential license issues resulting from their reuse of code. They should leverage reliable information resources on the Internet, complementing them with mandatory internal training and other practical information. Second, they should lobby universities and other educational institutions to include the topic in their curricula. Third, they should establish easy-to-understand policies providing guidance as to how to deal with Internet code. Moreover, they need to ensure that developers are aware of these policies and actually read and understand them. Finally, they need to recognize the interdisciplinary nature of license risks from reuse of Internet code relating to developers and engineers, as well as to lawyers.

They should thus facilitate communication between developers and legal experts such that clearance for specific instances of the reuse of Internet code can be obtained quickly. Otherwise, developers would have to choose between practicing reuse on their own or abandoning it altogether, an option that would ignore a valuable source of efficiency and quality gains. C

References

1. Alexy, O. *Free Revealing: How Firms Can Profit from Being Open*. Gabler, Wiesbaden, 2009.
2. Chen, W., Li, J., Ma, J., Conradi, R., Ji, J., and Liu, C. An empirical study on software development with open source components in the Chinese software industry. *Software Process Improvement and Practice* 13, 1 (Jan. 2008), 89–100.
3. Deshpande, A. and Riehle, D. The total growth of open

source. In *Proceedings of the Fourth International Conference on Open Source Systems* (Milan, Italy, Sept. 7–10). Springer, Boston, 2008, 197–209.

4. Frakes, W.B. and Kang, K. Software reuse research: Status and future. *IEEE Transactions of Software Engineering* 31, 7 (July 2005), 529–536.
5. Gartlan, D., Allen, R., and Ockerbloom, J. Architectural mismatch: Why reuse is still so hard. *IEEE Software* 26, 4 (July/Aug. 2009), 66–69.
6. German, D.M., Di Penta, M., and Davies, J. Understanding and auditing the licensing of open source software distributions. In *Proceedings of the 18th IEEE International Conference in Program Comprehension* (Braga, Portugal, June 30–July 2). IEEE Press, Los Alamitos, CA, 2010, 84–93.
7. German, D.M., Di Penta, M., Guéhéneuc, Y.-G., and Antoniol, G. Code siblings: Technical and legal implications of copying code between applications. In *Proceedings of the Sixth IEEE International Workshop on Mining Software Repositories* (Vancouver, Canada, May 16–17). IEEE Press, Los Alamitos, CA, 2009, 81–90.
8. German, D.M. and Gonzalez-Barahona, J.M. An empirical study of the reuse of software licensed under the GNU general public license. In *Proceedings of the Fifth International Conference on Open Source Systems* (Skövde, Sweden, June 3–6). Springer, Boston, 2009, 185–198.
9. German, D.M. and Hassan, A.E. License integration patterns: Dealing with license mismatches in component-based development. In *Proceedings of the 31st IEEE International Conference on Software Engineering* (Vancouver, Canada, May 16–24). IEEE Press, Los Alamitos, CA, 2009, 188–198.
10. Greene, W.H. *Econometric Analysis*. Prentice Hall, Upper Saddle River, NJ, 2007.
11. Lerner, J. and Tirole, J. The scope of open source licensing. *The Journal of Law, Economics, and Organization* 21, 1 (Apr. 2005), 20–56.
12. Levi, S.D. and Woodard, A. Open source software: How to use it and control it in the corporate environment. *Computer & Internet Lawyer* 21, 8 (Aug. 2004), 8–13.
13. Li, J., Conradi, R., Bunse, C., Torchiano, M., Slingstad, O.P.N., and Morisio, M. Development with off-the-shelf components: 10 facts. *IEEE Software* 26, 2 (Mar. 2009), 80–87.
14. Madanmohan, T.R. and De, R. Open source reuse in commercial firms. *IEEE Software* 21, 6 (Nov. 2004), 62–69.
15. McGhee, D.D. Free and open source software licenses: Benefits, risks, and steps toward ensuring compliance. *Intellectual Property & Technology Law Journal* 19, 11 (Nov. 2007), 5–9.
16. Morisio, M., Ezran, M., and Tully, C. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering* 28, 4 (Apr. 2002), 340–357.
17. Murray, G.F. Categorization of open source licenses: More than just semantics. *Computer & Internet Lawyer* 26, 1 (Jan. 2009), 1–11.
18. Norris, J.S. Mission-critical development with open source software: Lessons learned. *IEEE Software* 21, 1 (Jan. 2004), 42–49.
19. Rosen, L. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice-Hall, Englewood Cliffs, NJ, 2004.
20. Sojer, M. *Reusing Open Source Code*. Gabler, Wiesbaden, 2010.
21. Sojer, M. and Henkel, J. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems* 11, 12 (Dec. 2010), 868–901.
22. Spinellis, D. and Szyperski, C. How is open source affecting software development? *IEEE Software* 21, 1 (Jan. 2004), 28–33.
23. Umarji, M., Sim, S.E., and Lopes, C. Archetypal Internet-scale source code searching. In *Open Source Development, Communities and Quality*, B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi, Eds. Springer, Boston, 2008, 257–263.

Manuel Sojer (sojer@wi.tum.de) is a management consultant at Bain & Company, Munich, Germany. This article is part of his Ph.D. work completed in 2010 at Technische Universität München, Munich, Germany.

Joachim Henkel (henkel@wi.tum.de) is a professor of technology and innovation management at Technische Universität München, Munich, Germany.

DOI:10.1145/2043174.2043194

The goal is reliable parallel simulations, helping scientists understand nature, from how foams compress to how ribosomes construct proteins.

BY GANESH GOPALAKRISHNAN, ROBERT M. KIRBY, STEPHEN SIEGEL, RAJEEV THAKUR, WILLIAM GROPP, EWING LUSK, BRONIS R. DE SUPINSKI, MARTIN SCHULZ, AND GREG BRONEVETSKY

Formal Analysis of MPI-based Parallel Programs

MOST PARALLEL COMPUTING applications in high-performance computing use the Message Passing Interface (MPI) API. Given the fundamental importance of parallel computing to science and engineering research, application correctness is paramount. MPI was originally developed around 1993 by the MPI Forum, a group of vendors, parallel programming researchers, and computational scientists. However, the document defining the standard is not issued by an official standards organization but has become a de facto standard

through near-universal adoption. MPI development continues, with MPI-2.2 released in 2009 and MPI 3.0 expected in 2012. The standard is published on the Web²¹ and as a book, along with several other books based on it; see, for example, Gropp et al.¹³ and Pacheco.²⁶ Implementations are available in open source from MPICH2²² and Open MPI²⁵ from software vendors and from every vendor of HPC systems. MPI is widely cited; Google Scholar recently returned 39,600 hits for the term “+MPI +Message Passing Interface.”

MPI is designed to support highly scalable computing applications using more than 100,000 cores on, say, the IBM Blue Gene/P (see Figure 1) and Cray XT5. Many MPI programs represent dozens, if not hundreds, of person-years of development, including calibration for accuracy and performance tuning. Scientists and engineers worldwide use MPI in thousands of applications, including in investigations of alternate-energy sources and in weather simulation. For HPC computing, MPI is by far the dominant programming model; most (at some centers, all) applications running on supercomputers use MPI. Many application developers for exascale systems¹⁵ regard support for MPI as a requirement.

Still, the MPI debugging methods available to these developers are typically wasteful and ultimately unreliable. Existing MPI testing tools seldom provide coverage guarantees, examin-

» key insights

- Addressing the challenges of distributed systems, debugging necessitates collaboration between HPC and formal verification.
- Along with HPC, distributed computing based on communication libraries is going mainstream in the commodity world, two communities that must look to learn and benefit from one another.
- Catastrophic disruption of programmer productivity can be avoided through formal verification tools that handle problems of scale, enhance coverage by avoiding redundant searches, and decrease false-alarm rates through more precise analysis.



Figure 1. The Intrepid Blue Gene/P Open Science machine at Argonne National Laboratory, with 163,840 cores and 557TFlops/sec (peak).

ing essentially equivalent execution sequences, thus reducing testing efficiency. These methods fare even worse at large problem scales. Consider the costs of HPC bugs. A high-end HPC center costs hundreds of millions of dollars to commission, and the machines become obsolete within six years; in many centers, the annual electricity bill can run more than \$3 million, and research teams apply for computer time through competitive proposals, spending years planning their experiments. In addition to these costs, one must add the costs to society of relying on potentially defective software to inform decisions involving issues of great public importance (such as climate change).

Formal methods can play an important role in debugging and verifying MPI applications. Here, we describe existing techniques, including their pros and cons, and why they have value beyond MPI, addressing the general needs of future concurrency application developers who will inevitably use low-level concurrency APIs.

Historically, parallel systems

have used either message passing or shared memory for communication. Compared to other message-passing systems noted for their parsimony, MPI supports a large number of cohesively engineered features essential for designing large-scale simulations; for example, MPI-2.2²¹ specifies more than 300 functions, though most developers use only a few dozen in any given application.

MPI programs consist of one or more threads of execution with private memories (called “MPI processes”) and communicate through message exchanges. The two most common are point-to-point messages (such as sends and receives) and collective operations (such as broadcasts and reductions). MPI also supports non-blocking operations that help overlap computation and communication and persistent operations that make repeated sends/receives efficient. In addition, MPI allows processes and communication spaces to be structured using topologies and communicators. MPI’s derived datatypes further enhance the portability and efficiency

of MPI codes by enabling the architect to communicate noncontiguous data with a single MPI function call. MPI also supports a limited form of shared-memory communication based on one-sided communication. A majority of MPI programs are still written using the “two-sided” (message-passing-oriented) constructs we focus on through the rest of the article. Finally, MPI-IO addresses portable access to high-performance input/output systems.

MPI applications and libraries are written predominantly in C, C++, and/or Fortran. Languages that use garbage collection or managed runtimes (such as Java and C#) are rarely used in HPC; preexisting libraries, compiler support, and memory locality management drive these choices. Memory is a precious resource in large-scale systems; a rule of thumb is an application cannot afford to consume more than one byte per FLOP. Computer memory is expensive and increases cluster energy consumption. Even when developing traditional shared-memory applications, system architects must work with low amounts of cache-coherent memory

per core and manage data locality, something done routinely by MPI programmers. Computer scientists are also realizing that future uses of MPI will be in conjunction with shared-memory libraries (such as Pthreads⁷) to reduce message-copy proliferation. While some MPI applications are written from scratch, many are built atop user libraries, including ParMETIS¹⁶ for parallel hypergraph partitioning, ScaLAPACK⁵ for high-performance linear algebra, and PETSc³ for solving partial differential equations.

MPI processes execute in disjoint address spaces, interacting through communication commands involving deterministic, nondeterministic, collective, and non-blocking modes. Existing (shared-memory concurrent program) debugging techniques do not directly carry over to MPI, where operations typically match and complete out-of-program order according to an MPI-specific *matches-before* order.^{30,33} The overall behavior of an MPI program is also heavily influenced by how specific MPI library implementations take advantage of the latitude provided by the MPI standard.

An MPI program bug is often introduced when modeling the problem and approximating the numerical methods or while coding, including whole classes of floating-point challenges.¹¹ While lower-level bugs (such as deadlocks and data races) are serious concerns, detecting them requires specialized techniques of the kind described here. Since many MPI programs are poorly parameterized, it is

not easy for HPC developers to down-scale a program to a smaller instance and locate the bug. For these reasons, HPC developers need a variety of verification methods, each narrowly focused on subsets of correctness issues and making specific trade-offs. Our main focus here is formal analysis methods for smaller-scale MPI applications and semiformal analysis methods for the very large scale. For detecting MPI bugs in practice, formal analysis tools must be coupled with runtime instrumentation methods found in tools like Umpire,³² Marmot,¹⁹ and MUST,¹⁴ though much more research is needed in tool integration.

Dynamic analysis. MPI provides many nondeterministic constructs that free the runtime system to choose the most efficient way to carry out an operation but also mean a program can exhibit multiple behaviors when run on the same input, posing verification challenges; an example is a communication race arising from a “wildcard” receive, an operation that does not specify the source process of the message to be received, leaving the decision to the runtime system. Many subtle program defects are revealed only for a specific sequence of choices. Though random testing might happen on one such sequence, it is hardly a reliable approach.

In contrast, dynamic verification approaches control the exact choices made by the MPI runtime, using this control to methodically explore a carefully constructed subset of behaviors. For each such behavior, a number of

properties may be verified, including absence of deadlocks, assertion violations, incompatible data payloads between senders and receivers, and MPI resource leaks. Using a formal model of the MPI semantics, a dynamic verifier can conclude that if no violations occur on the subset of executions, then there can be no violation on an execution. If even this reduced subset cannot be explored exhaustively, the developer can specify precise coverage criteria and obtain a lesser (but still quantifiable) degree of assurance. This approach was originally demonstrated in the VeriSoft¹⁰ tool and has the advantage of not requiring modifications to the program source code, compiler, or libraries.

Full-scale debugging. Traditional “step-by-step” debugging techniques are untenable for traces involving millions of threads. Later, in an expanded description of full-scale debugging, we describe a new debugging approach called Stack Trace Analysis that analyzes an execution trace and partitions the threads into equivalence classes based on their behavior. Experience on real large-scale systems shows that only a small number of classes typically emerge, and the information provided can help a developer isolate defects. While this approach is not comparable to the others covered here, in that the focus is on the analysis of one trace rather than reasoning about all executions, it provides a clear advantage in terms of scalability.

Symbolic analysis. The techniques discussed earlier are only as good as the set of inputs chosen during analysis. Defects revealed for very specific input or parameter values may be difficult to discover with these techniques alone. Symbolic execution¹⁸ is a well-known technique for identifying defects, described later in the section on symbolic analysis of MPI, including how it is applied to MPI programs. The TASS toolkit²⁷ uses symbolic execution and state-enumeration techniques to verify properties of MPI programs, not only for all possible behaviors of the runtime system, but for all possible inputs as well. It can even be used to establish that two versions of a program are functionally equivalent, at least within specified bounds. On the other hand, implementing the symbolic ex-



Component of Blue Gene/P supercomputer at Argonne National Laboratory, Argonne, IL.

ecution technique requires sophisticated theorem-proving technology and a symbolic interpreter for all program constructs and library functions; for this reason, TASS supports only C and a subset of MPI. Moreover, it generally does not scale beyond a relatively small number of processes, though, as we show, defects that usually appear only in large configurations can often be detected in much smaller configurations through symbolic execution.

Static analysis. Compilers use static analyses to verify a variety of simple safety properties of sequential programs, working on a formal structure that abstractly represents some aspect of the program (such as a control-flow graph, or CFG). Extending these techniques to verify concurrency properties of MPI programs (such as deadlock freedom) requires new abstractions and techniques. Later, in the section on static analysis of MPI, we outline a new analysis framework targeting this problem that introduces the notion of a parallel CFG. The framework has the advantage that the pCFG is independent of the number of processes, essentially making it infinitely scalable. However, because automating these analyses is so difficult they may require user-provided program annotation to guide them.

Dynamic Verification of MPI

Here, we explore two dynamic analysis approaches: The first, implemented by the tool ISP^{31,35} (see Figure 2), delivers a formal coverage guarantee with respect to deadlocks and local safety assertions³⁰; ISP has been demonstrated on MPI applications of up to 15,000 lines of code. Running on modern laptop computers, ISP can verify such applications for up to 32 MPI processes on mostly deterministic MPI programs.

ISP's scheduler, as outlined in the figure, exerts centralized control over every MPI action. It limits ISP scalability to at most a few dozen MPI processes and does not help programmers encountering difficulty at higher ends of the scale where user applications and library codes often use different algorithms. What if a designer has optimized an HPC computation to work efficiently on 1,000 processors and suddenly finds an inexplicable bug? Traditional HPC debugging support is

severely lacking in terms of ensuring coverage goals. To address this limitation, some of the authors have built a tool called Distributed Analyzer of MPI, or DAMPI,³⁴ which uses a distributed scheduler while still ensuring nondeterminism coverage. DAMPI scales demonstrably far more than ISP.

Dynamic verification using ISP. For programs with nondeterministic MPI calls, simply modulating the absolute times at which MPI calls are issued (such as by inserting nondeterministic sleep durations, as performed by stress-testing tools) is ineffective because most often it does not alter the way racing MPI sends match with MPI nondeterministic receives deep inside the MPI runtime. Also, such delays slow the entire testing process unnecessarily.

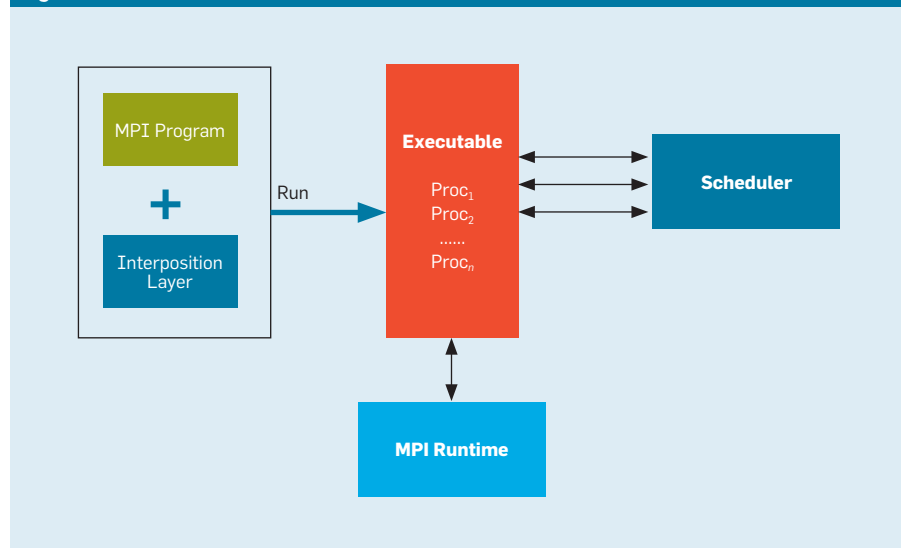
ISP's active testing approach (see Figure 3) means if P₂'s MPI `_Isend` can match P₁'s MPI `_Irecv`, the test encounters a bug. But can such a match occur? Yes, and here's how; first, let P₀ issue its non-blocking MPI `_Isend` call and P₁ its non-blocking MPI `_Irecv` call; then allow the execution to cross the MPI `_Barrier` calls; after that, P₂ can issue its MPI `_Isend`. The MPI runtime then faces a nondeterministic choice of matching either MPI `_Isend`. The system achieves this particular execution sequence only if the MPI `_Barrier` calls are allowed to match *before* the MPI `_Irecv` matches. Existing MPI testing tools cannot exert such fine control over MPI executions. By interposing a scheduler,

as outlined in Figure 2, ISP can reorder, at runtime, MPI calls issued by the program. In the example, ISP's scheduler intercepts all MPI calls coming to it in program order and dynamically reorders the calls going into the MPI runtime (ISP's scheduler sends `Barriers` first, an order allowed by the MPI semantics), at which point it discovers the nondeterminism.

When ISP determines two matches could occur, it re-executes (replays from the beginning) the program in Figure 3 twice, once with the `Isend` from P₀ matching the receive, the second `Isend` from P₂ matching it. To ensure these matches occur, ISP dynamically rewrites `Irecv(from:*)` into `Irecv(from:0)` and into `Irecv(from:2)` in these replays. If the algorithm does not do this but instead issues `Irecv(from:*)` into the MPI runtime, coverage of both process sends is no longer guaranteed. ISP discovers the maximal extent of nondeterminism through dynamic MPI call reordering and achieves scheduling control of relevant interleavings through dynamic API call rewriting. While pursuing relevant interleavings, ISP additionally detects three basic error conditions: deadlocks, resource leaks (such as MPI object leaks), and violations of C assertions in the code.

Developers should bear in mind that MPI programmers often use non-blocking MPI calls to enhance computation/communication overlap and nondeterministic MPI calls in master/worker patterns to detect which MPI

Figure 2. Overview of ISP.



process finishes first, so more work can be assigned to it. When these operations, together with “collective” operations (such as Barriers), are all employed in the same example, a developer can obtain situations like the one in Figure 3. The safety net provided by ISP and other such tools is therefore essential for efficiency-oriented MPI programming.

ISP guarantees MPI communication nondeterminism coverage under the given test harness and helps avoid exponential interleaving explosion primarily by avoiding redundantly examining equivalent behaviors (such as by not examining the $n!$ different orders in which an MPI barrier call might be invoked); testing tools typically fall victim to this explosion. ISP also includes execution-space sampling options.

ISP has examined many large MPI programs, including those making millions of MPI calls. Some of the authors have also built the Graphical Explorer of Message passing (GEM) tool,¹² which hosts the ISP verification engine. GEM is an official component of the Eclipse Parallel Tools Platform, or PTP,⁹ (PTP version 3.0 onward), making dynamic

formal verification of MPI available seamlessly within a popular integrated development environment. GEM also serves as a formal-methods supplement to a popular MPI textbook²⁶ by providing chapter examples as readily available MPI C projects.

Dynamic verification using DAMPI. A widely used complexity-reduction approach is to debug a given program after first suitably downscaling it. However, a practical difficulty in carrying out such debugging is that many programs are poorly parameterized. For them, if a problem parameter is reduced, it is often unclear whether another parameter should be reduced proportionally, logarithmically, or through some other relationship. A more serious difficulty is that some bugs are manifest only when a problem is run at scale. The algorithms employed by applications and/or the MPI library itself can change depending on problem scale. Also, resource bugs (such as buffer overflows) often show up only at scale.

While user-level dynamic verification supported by ISP resolves significant nondeterminism, testing at larger scales requires a decentralized

approach where supercomputing power aids verification, an idea the authors implemented in their tool framework DAMPI³⁴ (see Figure 4).

The key insight that allowed them to design the decentralized scheduling algorithm of DAMPI is that a non-deterministic operation, as in `MPI_Irecv(MPI_ANY_SOURCE)` and `MPI_Iprobe(MPI_ANY_SOURCE)`, represents a point on the timeline of the issuing process when the operation commits to a match decision. It is natural for an HPC programmer to view each such event as starting an epoch, an interval stretching from the current nondeterministic event up to (but not including) the next nondeterministic event. All deterministic receives can be assigned the same epoch as the one in which they occur. Even though the epoch is defined by a non-deterministic receive matching another process’s send, how can the tool determine all other sends that match it? The solution is to pick all the sends that are *not causally after* the nondeterministic receive (and subject to MPI’s “non-overtaking” rules). DAMPI determines these sends through an MPI-specific version of Lamport clocks,²⁰ striking a good compromise between scalability and omissions.

Experimental results show DAMPI effectively tests realistic problems running on more than 1,000 CPUs by exploiting the parallelism and memory capacity of clusters. It has examined all benchmarks from the Fortran NAS Parallel Benchmark suite,²⁴ with instrumentation overhead less than 10% compared to ordinary testing, but able to provide nondeterminism coverage not provided by ordinary testing.

Recent experiments by some of the authors found a surprising fact: None of the MPI programs in the NAS Parallel Benchmarks employing `MPI_Irecv(MPI_ANY_SOURCE)` calls actually exhibit nondeterminism under DAMPI. This means these benchmarks were “determinized,” perhaps through additional MPI call arguments and is further confirmation of the value of dynamic analysis in providing precise answers.

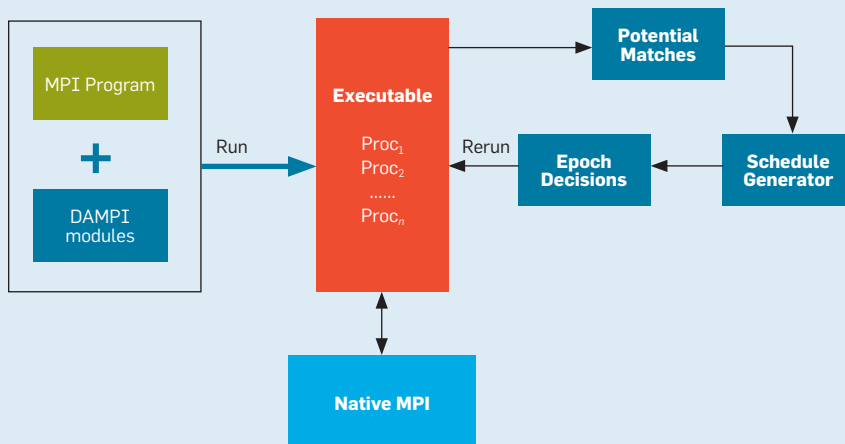
Full-Scale Debugging

The approach described here targets the large-scale systems that will

Figure 3. Bug manifests on some runtimes.

P ₀	P ₁	P ₂
<code>Isend(to : 1, 22);</code>	<code>Irecv(from : *, x)</code>	<code>Barrier;</code>
<code>Barrier;</code>	<code>Barrier;</code>	<code>Isend(to : 1, 33);</code>
	<code>if (x == 33) bug;</code>	

Figure 4. Distributed MPI analyzer.



emerge over the next few years; current estimates anticipate half a billion to four billion threads in exascale systems. With such concurrency, developers of verification tools must target debugging techniques able to handle these scales, as bugs are often not manifest until a program is run at its largest scale. Bugs often depend on input, which can differ significantly across full-scale runs. Furthermore, certain types of errors (such as integer overflows) often depend directly on the number of processors.

However, most debugging techniques do not translate well to full-scale runs. The traditional paradigm of stepping through code has significant performance limitations with large processor counts, as well as being impractical with thousands of processes or threads, let alone billions. Dynamic-verification techniques offer paradigmatic scaling but have even more performance limitations, particularly when the number of interleavings depends on process count.

Faced with scaling requirements, HPC developers require new techniques to limit the scope of their debugging efforts. Some of the authors developed mechanisms for identifying behavioral-equivalence classes based on the observation that when errors occur in large-scale programs, they do not exhibit thousands or millions of different behaviors. Rather, they exhibit a limited set of behaviors in which all processes follow the same erroneous path (a single common behavior) along which one or a few processes follow an erroneous path that can then lead to changes in the behavior of a few related processes (two or three behaviors). While the effect may trickle further out, developers rarely observe more than a half-dozen behaviors, regardless of the total number of processes in an MPI program.

Given the limited behaviors that are exhibited, developers can then focus on only debugging representative processes from each behavioral class, rather than all processes at once, thereby enabling the debugging of problems previously not debuggable.

The Stack Trace Analysis Tool (STAT)² achieves this debugging goal by attaching to all processes in a large-scale job and gathering stack traces

sampled over time in a low overhead and distributed manner. It then merges these stack traces to identify which processes are executing similar code. The tool considers a variety of equivalence relations; for example, for any $n \geq 1$, it considers two processes as equivalent if they agree on the first n function calls issued. Increasing n refines this equivalence relation, giving the developer control of the precision-accuracy trade-off.

The resulting tree readily identifies different execution behaviors. For example, Figure 5 shows the top levels of the tree obtained from a run of the Community Climate System Model (CCSM), an application that uses five separate modules to model land (CSM), ice, ocean (POP), and atmosphere (CAM) and couple the four models. In it, the developer can quickly identify that MPI processes 24–39 are executing the land model, 8–23 the ice model, 40–135 the ocean model, and 136–471 the atmospheric model, while 0–7 are executing the coupler. If a problem should be observed in one of them, the developer can then concentrate on this subset of tasks; in the case of a broader error, the developer can pick representatives from the five classes, thereby reducing the initial debugging problem to five processes. The STAT tool has been used to debug several codes with significantly shortened turnaround

time, including an Algebraic Multigrid (AMG) package, which is fundamental for many HPC application codes.

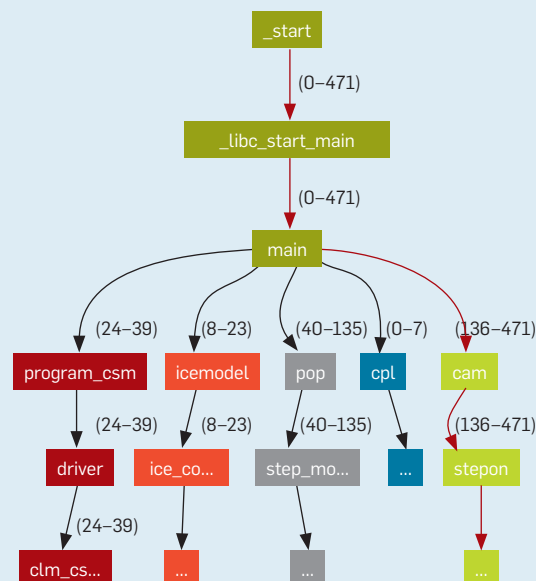
Tools like STAT also detect outliers that can directly point to erroneous behavior without further debugging; for example, the STAT tool was used on the CCSM code when it hung on more than 4,096 processes. The stack trace tree showed one task executing in an abnormally deep stack, and, on closer examination of the stack trace, not only that a mutex lock operation within the MPI implementation was called multiple times, creating the deadlock, but also exactly where in the code the respective erroneous mutex lock call occurred. This led to a quick fix of the MPI implementation.

The STAT developer group's efforts now include extensions that provide better identification of the behavior equivalence classes, as well as techniques to discern relationships among the classes.¹ Additional directions include using the classes to guide dynamic verification techniques.

Symbolic Analysis of MPI

The basic idea of symbolic execution is to execute the program using symbolic expressions in place of the usual (concrete) values held by program variables.¹⁸ The inputs and initial values of the program are symbolic constants $X_0; X_1, \dots$, so-called because they repre-

Figure 5. STAT process equivalence classes.



sent values that do not change during execution. Numerical operations are replaced by operations on symbolic expressions; for example, if program variables u and v hold values X_0 and X_1 , respectively, then $u+v$ will evaluate to the symbolic expression $X_0 + X_1$.

The situation is more complicated at a branch point. Suppose a branch

is governed by condition $u+v>0$. Since the values are symbolic, it is not necessarily possible to say whether the condition evaluates to *true* or *false*; both possibilities must be explored. Symbolic execution handles this problem by introducing a hidden Boolean-valued symbolic variable, the path condition pc , to record the choices

made at branch points. This variable is initialized to *true*. At a branch, a non-deterministic choice is made between the two branches, and pc is updated accordingly. To execute the branch on $u+v > 0$, pc would be assigned the symbolic value $pc \wedge u+v > 0$ if the true branch is selected; if this is the first branch encountered, pc will now hold the symbolic expression $X_0 + X_1 > 0$. If the false branch is chosen instead, pc will hold $X_0 + X_1 \leq 0$. Hence the path condition records the condition the inputs must satisfy for a particular path to be followed. Model-checking techniques can then be used to explore all nondeterministic choices and verify a property holds on all executions¹⁷ or generate a test set. An automated theorem prover (such as CVC3⁴) can be used to determine if pc becomes unsatisfiable, in which case the current path is infeasible and pruned from the search.

One advantage of symbolic techniques is they map naturally to message-passing-based parallel programs. The Verified Software Lab's Toolkit for Accurate Scientific Software (TASS),²⁷ based on CVC3, uses symbolic execution and state-exploration techniques to verify properties of such programs. The TASS verifier takes as input the MPI/C source program and a specified number of processes and instantiates a symbolic model of the program with that process count. TASS maintains a model of the state of the MPI implementation, including that of the message buffers. Like all other program variables, the buffered message data is represented as symbolic expressions. The TASS user may also specify bounds on input variables in order to make the model finite or sufficiently small. An MPI-specific partial-order-reduction scheme restricts the set of states explored while still guaranteeing that if a counterexample to one of the properties exists (within the specified bounds), a violation is reported. Examples are included in the TASS distribution, including where TASS reveals defects in the MPI code (such as a diffusion simulation code from the Functional Equivalence Verification Suite at <http://vsl.cis.udel.edu/fevs/>).

TASS can verify the standard safety properties, but its most important feature is the ability to verify that two programs are functionally equivalent;

Figure 6. Programs that read an array from a file, sum positive elements, and output the result.

```
for (i=0; i<n; i++) a[i] = read element i;
sum = 0.0;
for (i=0; i<n; i++)
    if (a[i]>0.0) sum += a[i];
output sum;

(a) adder_seq: sequential version

int first = n*rank/nprocs;
int count = n*(rank+1)/nprocs - first;
for (i=0; i<count; i++) a[i]=read element first+i;
sum = 0.0;
for (i=0; i<count; i++)
    if (a[i]>0.0) sum += a[i];
if (rank == 0) {
    for (j=1; j<nprocs; j++) {
        recv into buffer from rank j;
        sum += buffer;
    }
    output sum;
} else { send sum to rank 0; }
```

(b) adder_par: parallel version

Figure 7. Excerpts from MPICH2 broadcast code; the fault occurs when the highlighted expression is negative.

```
relative_rank = (rank >= root ?
    rank - root : rank - root + comm_size);
nbytes = type_size * count;
scatter_size =
    (nbytes + comm_size - 1)/comm_size;
mask = 0x1; i = 0;
while (mask < comm_size) {
    relative_dst = relative_rank ^ mask;
    dst_tree_root = relative_dst >> i;
    dst_tree_root <= i;
    recv_offset = dst_tree_root * scatter_size;
    if (relative_dst < comm_size)
    { ... MPIC_Sendrecv(...,
        nbytes-recv_offset, ...); ... }
    mask <= 1; i++;
}
```

(a) MPIR_Bcast_scatter_doubling_allgather

```
else { /* (nbytes >= MPIR_BCAST_SHORT_MSG)
    && (comm_size >= MPIR_BCAST_MIN_PROCS) */
    if ((nbytes < MPIR_BCAST_LONG_MSG) &&
        (MPIU_is_pof2(comm_size, NULL))) {
        MPIR_Bcast_scatter_doubling_allgather
```

(b) invocation context

that is, if given the same input, they always return the same output. This is especially useful in scientific computing where developers often begin with a simple sequential version of an algorithm, then gradually add optimizations and parallelism. The production code is typically much more complex but intended to be functionally equivalent of the original. The symbolic technique used to compare two programs for functional equivalence is known as “comparative symbolic execution.”²⁸

To illustrate the comparative symbolic technique, see Figure 6, where the sequential program reads n floating-point numbers from a file, sums the positive elements, and returns the result. A parallel version divides the file into approximately equal-size blocks. Each process reads one block into a local array and sums the positive elements in its block. On all processes other than process 0, this partial sum is sent to process 0, which receives the numbers, adds them to its partial sum, and outputs the final result.

Ignoring round-off error, the two programs are functionally equivalent; given the same file, they output the same result. To see how the comparative symbolic technique establishes equivalence, consider the case $n = \text{nprocs} = 2$ and call the elements of the file X_0 and X_1 . There are four paths through the sequential program, due to the two binary branches if $a[i] > 0.0$. One of these paths, arising when both elements are positive, yields the path condition $X_0 > 0 \wedge X_1 > 0$ and output $X_0 + X_1$. The comparative technique now explores all possible executions of `adder_par` in which the initial path condition is $X_0 > 0 \wedge X_1 > 0$; there are many such executions due to the various ways the statements from the two processes can be interleaved. In each, the output is $X_0 + X_1$. A similar fact can be established for the other three paths through the sequential program. Taken together, these facts imply the programs will produce the same result on any input (for $n = \text{nprocs} = 2$).

The ability to uncover defects at small scales is an important advantage of symbolic approaches. Isolating and repairing a defect that manifests only in tests with thousands of processes and huge inputs is difficult. Several research projects have focused on mak-



We propose a continuum of tools based on static analysis, dynamic analysis, symbolic analysis, and full-scale debugging, complemented by more traditional error-checking tools.



ing traditional debuggers scale to thousands of processes for just this reason. However, it would be more practical to force the same defect to manifest itself at smaller scales and then isolate the defect at those scales.

A real-life example illustrates this point: In 2008, a user reported a failure in the MPICH2 MPI implementation when calling the broadcast function `MPI_Bcast`, which used 256 processes and a message of just over `count = 3,200` integers. Investigation revealed the defect was in a function used to implement broadcasts in specific situations (see Figure 7a). For certain inputs, the “size” argument (`nbytes-recv_offset`) to an MPI point-to-point operation—an argument that should always be nonnegative—could in fact be negative. For 256 processes and integer data (`type_size = 4`), this fault occurs if and only if $3,201 \leq \text{count} \leq 3,251$.

The problematic function is guarded by the code in Figure 7b, referring to three compile-time constants—`MPIR_BCAST_SHORT_MSG`, `MPIR_BCAST_LONG_MSG`, and `MPIR_BCAST_MIN_PROCS`—defined elsewhere as 12,288, 524,288, and 8, respectively. Essentially, the function is called for “medium-size” messages only when the number of processes is a power of 2 and above a certain threshold. With these settings, the smallest configuration that would reveal the defect is 128 processes, with `count = 3,073`.

A symbolic execution technique that checks that the “size” arguments to MPI functions are always non-negative would readily detect the defect. If the tool also treats the three compile-time constants as symbolic constants, the defect can be manifest at the much smaller configuration of eight processes and `count = 1` (in which case `nbytes-recv_offset = -1`). Such an approach would likely have detected this defect earlier and with much less effort.

Arithmetic. In our analysis of the `adder` example, we interpreted the values manipulated by the program as the mathematical real numbers and the numerical operations as the (infinite precision) real operations. If instead these values are interpreted as (finite-precision) floating-point values and operations, the two programs

are not functionally equivalent, since floating-point addition is not associative.¹¹ Which is right? The answer is it depends on what the user is trying to verify. For functional equivalence, the specification and implementation are rarely expected to be “bit-level” equivalent (recall the adder example), so real equivalence is probably more useful for the task. TASS uses a number of techniques specialized for real arithmetic, as when all real-valued expressions are put into a canonical form that is the quotient of two polynomials to facilitate the matching of expressions. For other tasks (such as detecting the defect in Figure 7), bit-level reasoning is more appropriate. Klee⁸ is another symbolic execution tool for (sequential) C programs that uses bit-precise reasoning. There is no reason why these techniques could not be extended to parallel MPI-based programs.

Static Analysis of MPI

In the sequential arena, compiler techniques have been successful at analyzing programs and transforming them to improve performance. However, analyzing MPI applications is difficult for four main reasons: the number of MPI processes is both unknown at compile time and unbounded; since MPI processes are identified by numeric ranks, applications use complex arithmetic expressions to define the processes involved in communications; the meaning of ranks depends closely on the MPI communicators used by the MPI calls; and MPI provides several non-deterministic primitives (such as `MPI_ANY_SOURCE` and `MPI_Waitsome`). While some prior research (such as Strout et al.²⁹) explored analysis of MPI applications, none successfully addressed this challenge.

Some approaches treat MPI applications as sequential codes, making it possible to determine simple application behaviors (such as the relationship between writing to a buffer and sending the buffer). However, these approaches cannot represent or analyze the application’s communication topology. Other techniques require knowledge of the number of processes to be used at runtime, analyzing one copy of the application for each process. While this analysis can capture the application’s full parallel structure, it



The shortage of formal-methods researchers interested in HPC problems is perhaps the result of the severe historical disconnect between “traditional computer scientists” and HPC researchers.



is inflexible and non-scalable.

Over the past few years, we have developed a novel compiler-analysis framework that extends traditional dataflow analyses to MPI applications, extracting the application’s communication topology and matching the send and receive operations that may communicate at runtime.⁶ The framework requires no runtime bound on number of processes and is formulated as a dataflow analysis over the Cartesian product of control flow graphs (CFGs) from all processes we refer to as a parallel CFG, or pCFG. The analysis procedure symbolically represents the execution of multiple sets of processes, keeping track of any send and receive. Process sets are represented through abstractions (such as lower and upper bounds on process ranks) and predicates (such as “ranks divisible by 4”). Sends and receives are periodically matched to each other, establishing the application’s communication topology. Tool users can instantiate the analysis framework through a variety of “client analyses” that leverage the communication-structure information derived by the framework to propagate dataflow information, as they do with sequential applications. Analyses and transformations include optimizations, error detection and verification, and information-flow detection.

Finally, since topological information is key to a variety of compiler transformations and optimizations, our ongoing work focuses on source-code annotations that can be used to describe a given MPI application’s communication topology and other properties. The techniques will then exploit this information to implement novel scalable analyses and transformations to enable valuable optimizations in complex applications.

Conclusion

This article’s main objective is to highlight the fact that both formal and semi-formal methods are crucial for ensuring the reliability of message-passing programs across the vast scale of application sizes. Unfortunately, discussion of these techniques and approaches is rare in the literature. To address this lacuna, we presented the perspectives of academic researchers, as well as HPC researchers in

U.S. national laboratories, engaged in cutting-edge HPC deployment. We propose a continuum of tools based on static analysis, dynamic analysis, symbolic analysis, and full-scale debugging, complemented by more traditional error-checking tools.

Unfortunately, we only barely scratched the surface of a vast problem area. The shortage of formal-methods researchers interested in HPC problems is perhaps the result of the severe historical disconnect between “traditional computer scientists” and HPC researchers. This is especially unfortunate considering the disruptive technologies on the horizon, including many hybrid concurrency models to program many-core systems. There are also emerging message-passing-based standards for embedded multicores (such as MCAP²³), with designs and tool support that would benefit from lessons learned in the MPI arena.

We propose two approaches to accelerate use of formal methods in HPC: First and foremost, researchers in formal methods must develop verification techniques that are applicable to programs employing established APIs. This would help sway today’s HPC practitioners toward being true believers and eventually promoters of formal methods. Moreover, funding agencies must begin tempering the hoopla around performance goals (such as “ExaFLOPs in this decade”) by also setting formal correctness goals that lend essential credibility to the HPC applications on which science and engineering depend.

Acknowledgments

This work is supported in part by Microsoft, National Science Foundation grants CNS-0509379, CCF-0811429, CCF-0903408, CCF-0953210, and CCF-0733035, and Department of Energy grant ASCR DE-AC0206CH11357. Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. 

References

- Ahn, D.H., de Supinski, B.R., Laguna, I., Lee, G.L., Liblit, B., Miller, B.P., and Schulz, M. Scalable temporal order analysis for large-scale debugging. In *Proceedings of the ACM/IEEE Conference on Supercomputing* (Portland, OR, Nov. 14–20). ACM Press, New York, 2009.
- Arnold, D.C., Ahn, D.H., de Supinski, B.R., Lee, G.L., Miller, B.P., and Schulz, M. Stack trace analysis for large-scale debugging. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium* (Long Beach, CA, Mar. 26–30). IEEE Computer Society, 2007, 1–10.
- Balay, S., Gropp, W.D., McInnes, L.C., and Smith, B.F. Efficient management of parallelism in object-oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen, Eds. Birkhauser Press, 1997, 163–202.
- Barrett, C. and Tinelli, C. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification*, Vol. 4590 LNCS (Berlin, July 3–7). Springer, Berlin, 2007, 298–302.
- Blackford, L. *Scalpack User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.
- Bronevetsky, G. Communication-sensitive static dataflow for parallel message passing applications. In *Proceedings of the International Symposium on Code Generation and Optimization* (Seattle, Mar. 22–25, 2009), 1–12.
- Butenhof, D.R. *Programming with POSIX Threads*. Addison-Wesley, Boston, 2006.
- Cadar, C., Dunbar, D., and Engler, D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the Eighth USENIX Symposium on Operating Systems Design and Implementation* (San Diego, Dec. 7–10). USENIX Association, 2008, 209–224.
- Eclipse Foundation, Inc. Parallel Tools Platform. Ottawa, Ontario, Canada; <http://www.eclipse.org/ptp>
- Godefroid, P. Model checking for programming languages using Verisoft. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Paris, Jan. 15–17). ACM Press, New York, 1997, 174–186.
- Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys* 23, 1 (Mar. 1991), 5–48.
- Graphical Explorer of MPI Programs. ISP Eclipse plug-in; University of Utah, School of Computing; http://www.cs.utah.edu/formal_verification/GEM
- Gropp, W., Lusk, E., and Thakur, R. *Using MPI-2: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 1999.
- Hilbrich, T., Schulz, M., de Supinski, B., and Müller, M.S. MUST: A scalable approach to runtime error detection in MPI programs. In *Tools for High Performance Computing*, Springer, Berlin, 2009, 53–66.
- International Exascale Software Project; http://www.exascale.org/iesp/Main_Page
- Karypis Lab. ParMETIS: Parallel Graph Partitioning and Fill-Reducing Matrix Ordering. Minneapolis, MN; <http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview>
- Khurshid, S., Păsăreanu, C.S., and Visser, W. Generalized symbolic execution for model checking and testing. In *Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Vol. 2619 LNCS, H. Garavel and J. Hatcliff, Eds. (Warsaw, Apr. 7–11). Springer, 2003, 553–568.
- King, J.C. Symbolic execution and program testing. *Commun. ACM* 19, 7 (July 1976), 385–394.
- Krammer, B., Bidmon, K., Müller, M.S., and Resch, M.M. MARMOT: An MPI analysis and checking tool. In *Proceedings of the Parallel Computing Conference* (Dresden, Sept. 2–5, 2003), 493–500.
- Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (July 1978), 558–565.
- Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2*, Sept. 4, 2009; <http://www.mpi-forum.org/docs/>
- MPICH2: High performance and widely portable MPI; <http://www.mcs.anl.gov/mp/mpi/mvch>
- Multicore Association. Multicore Communications API, El Dorado Hills, CA; <http://www.multicore-association.org>
- NASA Advanced Supercomputing Division. *Parallel Benchmarks*; <http://www.nas.nasa.gov/Resources/Software/npb.html>
- Open MPI: Open Source High Performance MPI. Indiana University, Bloomington, IN; <http://www.open-mpi.org/>
- Pacheco, P. *Parallel Programming with MPI*. Morgan Kaufmann, San Francisco, 1996.
- Siegel, S.F. et al. *The Toolkit for Accurate Scientific Software*. Verified Software Laboratory, University of Delaware, 2010; <http://vsl.cis.udel.edu/tass>
- Siegel, S.F., Mironova, A., Avrunin, G.S., and Clarke, L.A. Combining symbolic execution with model checking to verify parallel numerical programs. *ACM Transactions on Software Engineering and Methodology* 17, 2 (Apr. 2008), 1–34.
- Strout, M.M., Kreaseck, B., and Hovland, P.D. Data-flow analysis for MPI programs. In *Proceedings of the 2006 International Conference on Parallel Processing* (Columbus, OH, Aug. 14–18). IEEE Computer Society, 2006, 175–184.
- Vakkalanka, S. *Efficient Dynamic Verification Algorithms for MPI Applications*. Ph.D. dissertation, University of Utah, 2010; <http://www.cs.utah.edu/fv>
- Vakkalanka, S., Vo, A., Gopalakrishnan, G., and Kirby, R. M. Reduced Execution Semantics of MPI: From Theory to Practice. In *Proceedings of Formal Methods, Second World Congress Lecture Notes in Computer Science 5850* (Eindhoven, The Netherlands, Nov. 2–6). Springer 2009, 724–740.
- Vetter, J.S. and de Supinski, B.R. Dynamic software testing of MPI applications with Umpire. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing* (Dallas, Nov. 4–10). IEEE Computer Society Press, 2000.
- Vo, A., Gopalakrishnan, G., Kirby, R.M., de Supinski, B.R., Schulz, M., and Bronevetsky, G. Large-scale verification of MPI programs using Lamport clocks with lazy updates. In *Proceedings of the 20th International Conference on Parallel Architectures and Compilation Techniques* (Galveston, TX, Oct. 10–14). IEEE Computer Society Press, 2011, 329–338.
- Vo, A., Aananthakrishnan, S., Gopalakrishnan, G., de Supinski, B.R., Schulz, M., and Bronevetsky, G. A scalable and distributed dynamic formal verifier for MPI programs. In *Proceedings of the ACM/IEEE Conference on Supercomputing* (New Orleans, Nov. 13–19). IEEE Computer Society Press, 2010.
- Vo, A., Vakkalanka, S., DeLisi, M., Gopalakrishnan, G., Kirby, R.M., and Thakur, R. Formal verification of practical MPI programs. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Raleigh, NC, Feb. 14–18). ACM Press, New York, 2009, 261–269.

Ganesh Gopalakrishnan (ganesh@cs.utah.edu) is a professor in the School of Computing at the University of Utah, Salt Lake City, UT, where he is director of the Center for Parallel Computing at Utah.

Robert M. (Mike) Kirby (kirby@cs.utah.edu) is an associate professor in the School of Computing and Scientific Computing and Imaging Institute at the University of Utah, Salt Lake City, UT.

Stephen F. Siegel (siegel@udel.edu) is an assistant professor in the departments of Computer & Information Sciences and Mathematical Sciences at the University of Delaware, Newark, DE.

Rajeev Thakur (thakur@mcs.anl.gov) is a computer scientist in the Mathematics and Computer Science Division of Argonne National Laboratory, Argonne, IL.

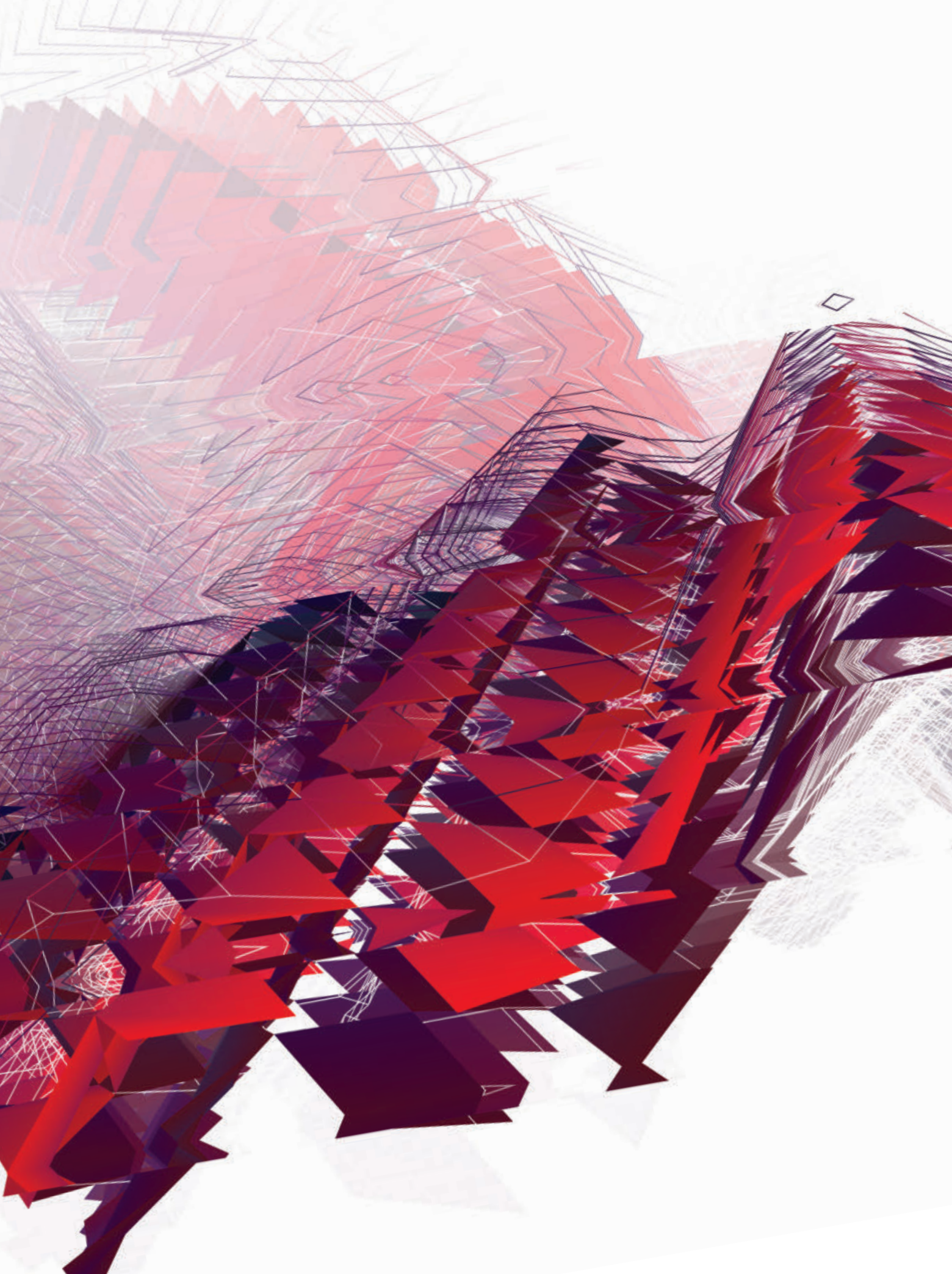
William Gropp (wgropp@illinois.edu) is the Paul and Cynthia Saylor Professor of Computer Science at the University of Illinois in Urbana-Champaign, and a fellow of the ACM, IEEE, and SIAM and a member of the National Academy of Engineering.

Ewing Lusk (lusk@mcs.anl.gov) is an Argonne Distinguished Fellow in the Mathematics and Computer Science Division at Argonne National Laboratory, Argonne, IL.

Bronis R. de Supinski (bronis@llnl.gov) is the principal investigator and leader of the Exascale Computing Technologies project and co-leader of the Advanced Simulation and Computing program's Application Development Environment and Performance Team at Lawrence Livermore National Laboratory, Livermore, CA.

Martin Schulz (schulzm@llnl.gov) is a computer scientist at Lawrence Livermore National Laboratory, Livermore, CA.

Greg Bronevetsky (bronevetsky@llnl.gov) is a computer scientist at the Lawrence Livermore National Laboratory, Livermore, CA, and a Department of Energy Early Career Investigator.



The motivation and key concepts behind answer set programming—a promising approach to declarative problem solving.

BY GERHARD BREWKA, THOMAS EITER,
AND MIROSLAW TRUSZCZYŃSKI

Answer Set Programming at a Glance

CAN SOLVING HARD computational problems be made easy? If we restrict the scope of the question to computational problems that can be stated in terms of constraints over *binary* domains, and if we understand “easy” as “using a simple and intuitive modeling language that comes with software for processing programs in the language,” then the answer is Yes! *Answer Set Programming* (ASP, for short) fits the bill.

While already well represented at research conferences and workshops, ASP has been around for barely more than a decade. Its origins, however, go back a long time; it is an outcome of years of research in knowledge representation, logic programming, and constraint satisfaction—areas that sought and studied declarative languages to model domain knowledge, as well as general-purpose computational tools for processing programs and theories that represent problem specifications in these languages. ASP borrows from each of these areas, all the time aiming

ILLUSTRATION BY GWEN VANHEE

to maintain a balance between expressivity, ease of use, and computational effectiveness. To give just a few examples, emerging applications in molecular biology, decision support systems for space shuttle controllers, and team building at Gioia Tauro Seaport (see sidebar here) bear witness to its potential.

Programs and Answer Sets

We start our ASP discussion with the propositional setting. The building blocks for programs are *atoms*, *literals*, and *rules*. *Atoms* are elementary propositions (factual statements) that may be true or false; *literals* are atoms a and their negations *not* a . *Rules* are expressions of the form

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n \quad (1)$$

where a and all b_i 's and c_j 's are atoms. Intuitively, a rule (1) is a justification to “establish” or “derive” that a (the so-called *head*) is true, if all literals to the right of $\leftarrow$ (the so-called *body*) are true in the following sense: a non-negated literal b_i is true if it has a derivation, a negated one, *not* c_j , is true if the atom c_j does not have one. For instance, the rule

$$\text{light_on} \leftarrow \text{power_on}, \text{not broken}$$

informally means we can assert that the light is on, if we established the power is on and there is no reason to think the lamp is broken. Rules may have no body. For instance, we may have a rule:

$$\text{power_on} \leftarrow .$$

Such rules are called *facts*, as the head is unconditionally true, and the arrow $\leftarrow$ is typically omitted.

Programs are finite collections of rules. They are thought of as “justifications” for sets of atoms that contain precisely those atoms that can be established. It is important to point out that *not* is not a standard negation operator. Rather, it is meant to stand for a modality “non-derivable.” Looking at the small program with the two rules mentioned here, *power_on* should be derived (as it is given as a fact), while intuitively *broken* should not (the program, which describes

ASP-based Team Building at Gioia Tauro Seaport

The seaport of Gioia Tauro, Reggio Calabria, Italy, is the largest transshipment terminal on the Mediterranean coast. A crucial management task for a port of this size is to build teams of employees to handle incoming ships. This is difficult and time consuming, as one must ensure that teams have appropriate skills, the workload is divided fairly, and legal workload regulations are met. Until recently this task was performed manually, which took several hours per day.

In cooperation with Exeura Srl, a University of Calabria (UNICAL) spin-off, and ICO BLG, an Italian logistics company, Nicola Leone's group at UNICAL has developed an ASP-based system for team building based on the DLV solver.³⁸ Rules describe the requirements that should be fulfilled regarding: necessary skills of team members; availability of employees; fairness of workload distribution; and distribution of “heavy” or “risky” tasks. Since in practice not all requirements can be satisfied, the system has an implicit conflict handling strategy that gives higher priority to more important criteria.

The system, which has been adopted by ICO BLG for work-force management, can generate shift plans for 130 employees within a few minutes. In addition, the plan quality turned out to be considerably better and overtime was decreased by 20%.

Key factors for the success of ASP in this application were its high expressiveness and the possibility to evolve an executable specification in close interaction with domain experts on site who, although not computer experts, could help getting it right in short time.

what we know, has no rule to derive *broken*). This in turn allows us to derive *light_on*.

Formalizing these intuitions posed a challenge to the knowledge representation and logic programming communities for years. Eventually, *answer sets* provided a solution that gained acceptance.

Answer sets. To trace the key points of answer sets, we consider two further examples. Let P_1 be the program consisting of the following rules:

$$\begin{aligned} \text{high_salary} &\leftarrow \text{employed}, \text{educated} \\ \text{educated} &\leftarrow \text{high_salary} \\ \text{employed} &\leftarrow \text{motivated} \\ \text{motivated} & \end{aligned}$$

We can regard *motivated* as established as it is the head of a rule that has no preconditions. Consequently, the third rule allows us to derive *employed*. Can we obtain anything else? To get *high_salary* we need to have established *educated* and, similarly, to get *educated* we need to have established *high_salary*. This “vicious cycle” of dependencies cannot be broken as there is no other rule with *high_salary* or *educated* in the head. Hence, neither *high_salary* nor *educated* can be derived given the information in the program. We conclude the set $\{\text{motivated}, \text{employed}\}$ is the only one the program “justifies.”

This bottom-up process can be extended to an arbitrary program without the *not* operator. In the general case, however, once negation is allowed the situation gets more complicated. For instance, let P_2 consist of two rules:

$$\begin{aligned} \text{open} &\leftarrow \text{not closed} \\ \text{closed} &\leftarrow \text{not open}. \end{aligned}$$

In the first example it was clear how to start and how to proceed. It is not so here. The reason is we do not know which atoms cannot be derived, therefore, we cannot verify the conditions for applying any of the rules.

A way out of the problem is to start by *assuming* which atoms will *not* be derived. For instance, let us assume that *closed* will not be derived. Then, the first rule can be used and we can establish *open*. Since *open* is established, the second rule cannot be used and *closed* indeed will not be established, *verifying* our assumption. Thus, the set $\{\text{open}\}$ is justified by the program in the following sense. Assuming that atoms not contained in the set cannot be derived, and using program rules (under our intuitive understanding of how they work), we can derive in the bottom-up fashion precisely those atoms that are in the set. Interestingly and importantly, $\{\text{open}\}$ is not the only set justified by the program P_2 . Another one is $\{\text{closed}\}$: if we assume that

open cannot be derived, we can use the second rule to derive *closed*. Having derived *closed*, we have that *open* cannot be derived, confirming the assumption we made.

Our examples suggest the case of programs that contain no rules with *not* in the body is easier. We do not need to make any assumptions about what cannot be derived, as no rule has negated atoms in its body. Instead, we proceed in an iterative fashion collecting atoms that can be established, in each step using atoms derived already to establish new ones. When no more atoms can be derived, the process terminates. The unique set of atoms derived in this way is justified by the program, and we call it the *answer set* of the program.

The concept of an answer set for negation-free programs (also called Horn programs) is a springboard to the general definition. The intuitions we discussed earlier in the context of the program P_2 are crucial. We start with a set M of atoms (in our example, with $\{open\}$) and make an *assumption* that no atom outside M can be derived. Given this assumption, rules that contain a negated atom *not a*, where a is in M , become unusable (as the non-derivability of a is not *assumed*; in our example, $closed \leftarrow not\ open$ is unusable). These rules are “blocked” by M and can be disregarded. Therefore, we remove them from the program. In every other rule, if an atom is negated, it must have been assumed non-derivable, otherwise, the rule would have been removed. According to our

The answer set semantics of programs is the foundation of ASP. But equally important is the understanding of how programs encode search problems and their instances.

reading of the rules the corresponding literal can be eliminated from the body without affecting the usability of the rule. Once this is done, we are left with a negation-free program, called the *reduct* of the program with respect to M . If the set of atoms we can derive from that program or, in other words, the answer set of that program, coincides with M , all non-derivability assumptions we made based on M are confirmed, and all atoms in M can be derived. Thus, M is justified by P . We call each such set M an *answer set* of P . The definitions of the reduct and an answer set are due to Gelfond and Lifschitz.²⁰ Originally, they used the term stable model and introduced the term answer set later for a generalization of the concept to a broader class of programs that feature *strong negation* and *disjunction*, which we will discuss. The new term eventually took over.

There is some similarity between rules and propositional logic implications. Indeed, the rule (1) looks like the implication

$$(b_1 \wedge \dots \wedge b_m \wedge \neg c_1 \wedge \dots \wedge \neg c_n) \rightarrow a \quad (2)$$

written in a “reversed” fashion. Each answer set of a program is a model of the program viewed as a set of implications (models are truth value assignments to atoms such that each implication evaluates to *true*). However, not all models are answer sets as not all models satisfy the *foundedness* requirement that atoms be derivable in the sense described here.

It should be noted that ASP has solid logic foundations, and is closely linked to nonmonotonic reasoning. In fact, programs under answer set semantics can be seen as a fragment of Reiter’s Default Logic and as theories in nonmonotonic modal logics, including Moore’s Autoepistemic Logic and nonmonotonic KD45.³¹ David Pearce showed that the answer set semantics can be elegantly captured by a nonmonotonic variant of the logic of here and there,³⁵ a logic located between intuitionistic and classical logic.

Close connection to nonmonotonic logics provides ASP with the power to model default negation and, more generally, to deal with incomplete information. We illustrate that by continuing our *light_on* example. The rule

» key insights

- Answer set programming is an emerging approach to modeling and solving search and optimization problems. It combines an expressive representation language, a model-based problem specification methodology, and efficient solving tools.
- The answer set programming language allows domain and problem-specific knowledge, including incomplete knowledge, defaults, and preferences, to be represented in an intuitive and natural way.
- Because of its strong declarative aspect, the language of answer set programming supports rapid prototyping and development of software for solving search and optimization problems, and facilitates modifications and refinements leading to better performance.

$broken \leftarrow lightning, not\ lightning_rod$

specifies that the lamp breaks when a lightning strikes, unless a lightning rod was installed. With this rule appended to the program here, we still derive $light_on$, as we cannot derive $broken$. However, things change if we further add the fact $lightning$. As $lightning_rod$ cannot be derived, we can establish $broken$, and so $light_on$ can no longer be derived. Thus, answer set programs behave *nonmonotonically*—conclusions may have to be retracted when more rules or facts are added to the theory. Further, if we add one more fact $lightning_rod$, the situation changes again; we can no longer derive $broken$, and thus $light_on$ will be derived. What this shows is that ASP provides convenient ways for handling exceptions and nested exceptions.

Shorthands and further connectives. A common and important type of rules has its head atom occur negated in the body:

$a \leftarrow B, not\ a.$

If such a rule, let us denote it by r , is added to a program P that has no occurrences of a , then r works as a *constraint*. Namely, a set M of atoms is an answer set of the program $P \cup \{r\}$ if and only if M is an answer set of the program P and does not satisfy (as in propositional logic) the conjunction of literals B . In other words, adding r to P simply eliminates those answer sets of P that satisfy B . As atom a is auxiliary and thus irrelevant (we do not allow it in P), a common way to write a constraint is as a “headless” rule

$\leftarrow B$

which conveys the intuition of a constraint: satisfying B results in a contradiction.

It is also quite common that programs contain pairs of rules

$a \leftarrow B, not\ \bar{a}$

$\bar{a} \leftarrow B, not\ a,$

where neither a nor $\bar{a}$ appear as the head of any other rule in the program, and B is a conjunction of literals. This happens, in particular, when the programmer wants to refer in the program

both to an atom a and to its (standard) negation. To represent the latter, the programmer introduces a new atom $\bar{a}$ and includes in the program the two rules here. Intuitively, the role of these rules is to select, in case B is satisfied, exactly one of a and $\bar{a}$; this is precisely what they do under the answer set semantics. Pairs of such rules are often written in a shorthand notation as a single *choice* rule

$\{a\} \leftarrow B.$

Strong negation, denoted with the standard negation symbol $\neg$, allows us to distinguish between having no justification for an atom a , expressed by *not* a , and having one for the negation of a , expressed by $\neg a$. In program rules, $\neg$ can only appear in front of atoms. Gelfond and Lifschitz showed that the definition of answer sets extends to programs of this form almost literally.²¹ Every program P with strong negation can be reduced to an ordinary program $\bar{P}$: we simply have to replace each literal $\neg a$ in P by a new atom $\bar{a}$. It can be shown that a consistent set of literals S is a (generalized) answer set of P if and only if the set $\bar{S}$ obtained from S by the same modification is an answer set of $\bar{P}$. Thus, strong negation is only a modeling convenience. However, it makes formulating defaults as in Reiter’s Default Logic easier. For example, a rule

$closed_{t+1} \leftarrow closed_t, not\ \neg\ closed_{t+1}$

might be interpreted as saying that by default, the valve remains closed at time $t+1$ if it was closed at time t (that

is, unless there are specific reasons for it not to be). Such default rules, which embody the law of inertia, allow for an elegant solution of the frame problem that arises when one reasons about actions and their effects, for instance when modeling and solving planning problems.¹

Modeling considerations also motivated allowing *disjunctions* in the heads of rules. Disjunctive rules

$a_1 \vee \dots \vee a_k \leftarrow b_1, \dots, b_m, not\ c_1, \dots, not\ c_n$

often make representations more intuitive, for example, in a rule like

$open \vee closed \leftarrow valve.$

To eliminate the possibility for a valve to be both, a form of minimality is needed. It is reflected in the answer sets of a disjunctive program.²¹ The definition uses the same process as before to “reduce” the program with respect to a candidate atom set M and yields the reduct that is free of (default) negation. However, the reduct may have disjunctions in the heads of its rules and thus, in general, there might be multiple minimal sets of atoms that satisfy all rules (and some are guaranteed to exist). The idea now is to check whether M is one of these minimal sets of the reduct. If this is the case, then M is an answer set. Importantly, unlike strong negation $\neg$, disjunction in the rule heads does increase the problem-solving capacity of programs, as witnessed by results on complexity and expressive power (see the accompanying sidebar “Complexity of ASP”).

Complexity of ASP

To decide whether a given program has some answer set is *NP-complete*,²⁹ thus as complex as the classical propositional satisfiability problem (SAT); in the presence of disjunctive rules, the problem is *NP^{NP}-complete*¹¹ (*NP^{NP}* are the problems decidable in *NP* with an oracle for *NP* problems); roughly speaking, this means *NP-completeness* even if calls to a subroutine for SAT are for free.^{9,22} Predicate programs have exponentially higher complexity (intuitively, this is because the reduction by grounding causes an exponential blow up in general). Regarding search problems, ASP can express all *NP*-search problems, that is, those solvable using a nondeterministic Turing machine in polynomial time, in such a way that the answer sets encode the solutions. In fact, each such problem (for example, finding some Hamiltonian cycle) is expressible by a fixed predicate program to which logical facts encoding a given problem instance (for example, a graph) are added. Again, additional constructs like disjunctive rules may increase the expressivity.

Table 1. ASP grounders.

LPARSE	www.tcs.hut.fi/Software/smodels/
DLV	www.dbai.tuwien.ac.at/proj/dlv/ or www.dlvsystem.com/
GRINGO	potassco.sourceforge.net/#gringo/

Table 2. Some ASP systems.

ASSAT	assat.cs.ust.hk/
CLASP 1	potassco.sourceforge.net/#clasp/
CMODELS	www.cs.utexas.edu/users/tag/cmodels/
DLV 2	www.dbai.tuwien.ac.at/proj/dlv/ or www.dlvsystem.com/
GNT	www.tcs.hut.fi/Software/gnt/
SMODELS	www.tcs.hut.fi/Software/smodels/
XASP	xsb.sourceforge.net/, distributed with XSB

¹ + CLASP, CLINGO, CLINGCON, among others; <http://potassco.sourceforge.net/>

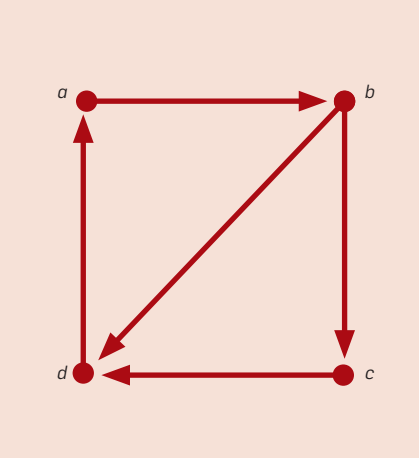
² + DLVHEX, DLVDB, DLT, DLV-COMPLEX, ONTO-DLV, and others.

Predicate programs. The propositional case is crucial for the definition of answer set semantics. But it is the predicate version of the formalism that facilitates modeling and makes ASP an effective problem-solving technique. The language has *relation* (or *predicate*) symbols, *constant* symbols and *variables*, as well as the logical connectives we discussed earlier, but no *function* symbols (we will discuss this restriction later). A rule is an expression of the form

$$A \leftarrow B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n \quad (3)$$

where A , B_i , and C_i are atomic formulas in the language. Rules are regarded as being implicitly universally quantified. The concepts of the head and body of the rule are defined as before and we interpret a rule (3) similarly as before, too. That is, we understand it as a device that, under some conditions, allows us to derive its head.

More formally, the semantics of a predicate program P is defined in terms of its ground version $grnd(P)$. The program $grnd(P)$ consists of all ground instantiations of rules in P with respect to constants that appear in P . In case P contains no constants (a situation that does not occur in practice), one is selected arbitrarily and used to produce $grnd(P)$. The program $grnd(P)$ can be regarded as a propositional one over all ground atoms in the language,

A graph for the Hamiltonian cycle problem.

and the answer sets of P are defined to be those of $grnd(P)$.

The ASP Paradigm

ASP is an approach to solving search problems. The answer set semantics of programs is the foundation of ASP. But equally important is the understanding of how programs encode search problems and their instances. Niemelä³² and Marek and Truszczyński³⁰ first formulated explicitly the basic principles of the ASP approach, Lifschitz²⁶ was the one to propose the term. In our discussion we rely on a rather intuitive understanding of a *search problem*. Namely, we assume that a search problem Π consists of a set of *instances*, D_Π , with each instance I assigned a finite set $S_\Pi(I)$ of solu-

tions. The set $S_\Pi(I)$ may be empty, that is, problem Π may have no solution for instance I .

To solve a search problem Π , a program P_Π is designed that captures the problem specifications so that when extended with facts $D(I)$, representing an instance I of the problem, the answer sets of $P_\Pi \cup D_\Pi(I)$ describe all solutions of problem Π for the instance I . The upshot of this design is that solving the problem is reduced in a uniform way (the program P_Π is fixed and only the data component changes) to the task of finding answer sets.

We now illustrate how ASP works by analyzing the problem of finding a Hamiltonian cycle in a directed graph. The choice is not arbitrary: this is an important combinatorial problem, arising in several practical situations (for example, as an essential component of the well-known Traveling Salesperson problem). While simple to state, it is still complex enough to allow us to emphasize all key aspects of ASP. In the problem, we are given a directed graph $G = (V, E)$, where V is the set of vertices and E the set of (directed) edges of G . The goal is to find a *Hamiltonian cycle* in G , that is, a set of edges that induce in G a directed cycle going through each vertex exactly once.

We will use two relation symbols to represent graphs: *vtx* and *edge*. Let us consider the graph G shown in the accompanying figure.

We represent the graph G as the set of ground atoms

$$D_{hc}(G) = \{vtx(a), vtx(b), vtx(c), vtx(d)\} \cup \{edge(a, b), edge(b, c), edge(c, d), edge(d, a), edge(b, d)\}.$$

Next, we need to capture the specification of the problem. A key part is the *definition* of a Hamiltonian cycle. According to our description, it must be a *subset* of the edges of the graph. To describe this subset formally, we use a relation symbol *in* and expressions $in(a, b)$ that informally read: the edge (a, b) is selected for a Hamiltonian cycle. To indicate that any edge (X, Y) can be “selected” to be in a Hamiltonian cycle, we use the choice rule:

$$(HC1) \{in(X, Y)\} \leftarrow edge(X, Y).$$

Next, we stipulate that no two selected

edges start or end in the same vertex. To this end, we use two constraint rules:

(HC2) $\leftarrow in(V2, V1), in(V3, V1), V2 \neq V3$
 (HC3) $\leftarrow in(V1, V2), in(V1, V3), V2 \neq V3$.

We stress the use of the relation symbol $\neq$ here. In the predicate version of ASP, we assume the set of constants includes integers, and the set of relation symbols includes symbols such as $=$, $\neq$, $\leq$, $<$, $\geq$, and $>$, as well as symbols for (bounded) arithmetic operations such as $+$. To be consistent with standard notation, we use the infix notation and write $X \leq Y$ instead of $\leq(X, Y)$. Similarly, we write $X + Y = Z$ instead of $+(X, Y, Z)$. All these symbols are always interpreted in the standard way.

To be a Hamiltonian cycle, the set of edges $in(x, y)$ must determine a single cycle. To enforce this condition, we need a concept of one vertex being reachable from another. To this end, we use an auxiliary relation symbol *rchble* and the following rules:

(HC4) *rchble*(V, V)
 (HC5) *rchble*($V1, V3$) $\leftarrow in(V1, V2),$
 rchble($V2, V3$).

The rules (HC4) and (HC5) define the *transitive closure* of the relation *in*^a; that is, all pairs of vertices (x, y) such that y can be reached from x by following zero or more edges that are “in.” Clearly, the selected edges form a Hamiltonian cycle if and only if every pair of vertices is in the transitive closure. This condition is captured by the following constraint rule:

(HC6) $\leftarrow vtx(V1), vtx(V2),$
 not rchble($V1, V2$).

Let P_{hc} be the program consisting of the rules (HC1) - (HC6). One can show that a set of edges H is a Hamiltonian cycle in a graph G if and only if $H = \{(x, y) \mid in(x, y) \in M\}$ for some answer set M of $P_{hc} \cup D_{hc}(G)$.

Finding a Hamiltonian cycle of an arbitrary input G is an NP-hard problem, and under a suitable notion complete for all NP-search problems. In fact, complexity theory (see the sidebar

a It is well-known that this is not expressible in first-order logic.

ASP for Repairing Large-Scale Biological Networks

New high-throughput methods have led to a dramatic increase of measurable data in modern molecular biology, and a number of corresponding knowledge repositories are available on the Web. However, both the data and the available biological networks are highly incomplete and error-prone, and inconsistencies are the rule rather than the exception.

In a joint project by Potsdam University, INRIA, and Institut Cochin, led by Torsten Schaub, an approach for repairing biological networks based on ASP has been developed.¹⁹ It builds on a range of available repair actions inspired by biological use cases. Examples are modifications of the role of a node in a biological influence graph (for example, from inhibitor to activator), additions of missing links between nodes, or modifications of experimental data in cases where it is plausible to assume errors in the measurements.

The program rules encode biological knowledge about the repair actions needed and possible in a particular situation. A possible repair is then achieved by minimizing, according to a variety of strategies, the set of applied repair actions. The system uses not more than 20 rules to encode five types of repair actions with different targets.

“Complexity of ASP”) tells us that each NP-search problem Π is expressible by a program P_{Π} as noted earlier.

Processing Answer Set Programs

Current tools for computing with answer set programs support several basic reasoning tasks, which include computing a single answer set (or determining that none exist), computing a given number of answer sets, and computing all of them. Most tools also support deciding whether an atom is true in every (resp. some) answer set, known as *cautious* (resp. *brave*) reasoning. These modalities are important for reasoning applications; for example, when we want to know whether a fact is true in every (resp. some) possible evolution of a system executing a sequence of actions of bounded length.

ASP processing typically works in two stages. First, the predicate program is replaced with an equivalent propositional program by so-called *variable replacement* or *grounding*. Second, that program is processed by a propositional ASP solver. Most implemented ASP processing systems make a clear distinction between the two stages and offer separate tools for each, others integrate them.

Grounding. The naive approach to grounding is to replace a program P with $grnd(P)$; but generally this is not

efficient. Consider the rule $p(X) \leftarrow p(X_1), \dots, p(X_n)$ and assume it needs to be grounded for two constants a and b . Then, the naive grounding will produce 2^{n+1} ground instances, as we can choose for X and each X_i either a or b . However, in this case, the full grounding amounts to just two propositional rules $p(a) \leftarrow p(b)$ and $p(b) \leftarrow p(a)$, as repeated literals in the bodies of rules and tautological rules, where the head atom occurs non-negated in the body, can be eliminated without affecting the answer sets. Intelligent grounding techniques incorporate such equivalences and many further optimizations. They aim to produce, given a predicate program P , a *possibly small* propositional program, not necessarily a subset of $grnd(P)$, that is equivalent to P , that is, has the same answer sets. Current grounders exploit techniques such as partial evaluation, rewriting, and a great deal of database technology to make grounding efficient. We refer to Table 1 for information on the three grounders most broadly used in ASP. Their input formats serve as de facto specifications of three most popular ASP dialects. They are quite close to each other. Nevertheless, the need for standardization is recognized by the ASP community. Extensions to the GRINGO grounder are an important step in this direction, making its input language much closer to that of

the DLV grounder.

Propositional solving. Table 2 provides pointers to several current ASP solvers. All of them more or less directly exploit methods developed in the field of satisfiability solving. Some ASP solver algorithms, often referred to as *native* (to ASP), follow the general backtracking search pattern of SAT solvers but append SAT-based propagation techniques with ones implied by an additional *foundedness* condition that models must satisfy to be answer sets.²⁵ It means every atom that is true in a model must be *derived* (in a certain precise sense) by a rule in the program. The search backtracks when either a contradiction is derived, or a complete and consistent assignment is found but some atoms that are true lack a derivation (are not founded). In each case, the need to backtrack indicates that some decisions made in the search earlier are incompatible with any answer set of the program and must be changed. This group of algorithms embodies a perspective on answer sets best captured by a catchphrase

(*propositional*) $ASP = SAT + \text{foundedness}$.

The answer set search outlined earlier can be improved by sophisticated search heuristics and techniques like backjumping and clause learning developed in the field of SAT solvers. The current ASP solvers take full advantage of these techniques. The native ASP solver CLASP, dressed as a SAT solver, won two tracks of the 2009 SAT solver competition.

Other successful ASP solver algorithms are based on reductions of answer set solving to satisfiability testing. They modify the formula corresponding to a program so that its models are exactly (or up to trivial projections) the answer sets of the program. One approach is to produce the so-called program *completion*. It reflects the idea that the program provides *all* conditions under which atoms are true; that is, it is a definition of the atoms in its rule heads. Accordingly, the completion is the formula containing for each atom a an equivalence saying that a holds if and only if the disjunction of the bodies of all rules with a in the head

holds. The completion captures some aspects of the foundedness condition, but not all. To capture it entirely, the completion must be extended by *loop formulas*, that exclude self-supporting derivations.²⁸ Loosely speaking, this approach could be cast as

$ASP = \text{completion} + \text{loop formulas}$.

Once the completion and loop formulas are built, an off-the-shelf SAT solver is used to find models of the resulting theory and so, answer sets of the original program. In the worst case, there can be exponentially many loop formulas, which complexity theory tells is somehow unavoidable. Therefore, some ASP solvers based on this idea, for example, ASSAT, add loop formulas incrementally and test whether models are already answer sets, while others, such as Cmodels2, similarly employ special techniques to select promising loop formulas to add and to “forget” them later.

Other reductions of ASP computation to SAT solving use auxiliary atoms for level rankings to represent founded derivation by keeping track of successive rule applications. Following this direction, translations of ASP to SAT modulo difference logic have been proposed that exploit fast solvers for theories in that formalism.³³

ASP Extensions

Motivated by the needs of applications, several extensions of the basic ASP paradigm have been proposed.

Constraints and aggregates. Constraints on sets of atoms are particu-

larly common. For instance, one often needs to say that exactly one out of a given set of atoms is true. In the well-known n -queens problem, we must place n queens on the $n \times n$ chessboard so that no two queens attack each other. Here one of the constraints is that exactly one queen is in each row. Even though this can be naturally encoded in the basic ASP language, the grounding will result in a large number of rules. ASP input languages thus provide constructs for constraints on sets of atoms that ASP solvers handle suitably. Basically, there are two approaches.

The first approach, which originated with LPARSE, uses the concept of a *cardinality atom*. In the propositional case, it has the form

$$l \{a_1, \dots, a_n\} k$$

and reads: at least l and at most k atoms in the set $\{a_1, \dots, a_n\}$ are true (if l or k are missing, it implies no restriction from the respective side). In the predicate language, one can be even more concise and write expressions such as

$$L \{a(X) : p(X, Y)\} K,$$

where L, K, X , and Y are variables. The expression captures a condition that given a value for Y , for at least L and at most K of the values of X such that $p(X, Y)$ holds, $a(X)$ is true. To ensure the grounding process is well defined, syntactic conditions on variables are used.

Let us denote by $q(X, Y)$ that some queen is in row X and column Y . We can state the uniqueness constraint on

ANTON—An ASP-based Music Composition System

ANTON,⁴ developed at University of Bath in cooperation with University of Glamorgan, is an automatic system for the composition of Renaissance-style music. It represents musical knowledge in the form of about 500 ASP rules. The rules describe the progression of a melody, both at the local level (the choice of the next note) and at the global level (the overall structure), the harmony that arises from the relationship between the melodic line and the supporting instruments, and also the rhythm, such as the intervals between notes, of a piece.

Given some initial information, for example, fixed notes or number of parts, the program generates answer sets representing musical pieces that satisfy the composition rules. With minor modifications, the system can also be used to detect violations of composition rules in given pieces of music.

queens in each row concisely by the following two constraint rules:

$$\leftarrow 2 \{q(X, Y) : col(Y)\}, row(X)$$

$$\leftarrow \{q(X, Y) : col(Y)\} 0, row(X).$$

The first rule states that for no row X there are distinct Y and Y' such that $q(X, Y)$ and $q(X, Y')$ are true (no row contains two or more queens). The second rule states that for no row X , it holds that all atoms $q(X, Y)$ are false (there is no row without queens). There is a more general version of cardinality constraints, *weight* constraints, where each atom is associated with a weight and the bounds constrain the sum of the weights of atoms that have some property.

The second approach to modeling constraints on sets of atoms follows the idea of aggregates familiar from SQL in databases.¹⁶ Those implemented in ASP languages include *count*, *sum*, *maximum*, and *minimum* and follow closely the database syntax. In the DLV input language, the unique-queen constraint is expressible by

$$\leftarrow 1 ! = \#count\{Y : q(X, Y)\}, row(X).$$

The input language of GRINGO also recognizes aggregates such as *count* and *sum* but specifies bounds as in cardinality constraints; this points to the need for standardization of ASP input languages.

Preferences. A basic assumption of the ASP paradigm is that problems are modeled in a way such that answer sets represent their solutions. However, it is impossible to further distinguish between better and poorer solutions. One way to address this problem is to introduce preferences. Simple forms of preferences can be expressed using *#minimize* and *#maximize* statements that are supported by several of the existing ASP solvers. They allow us to associate weights with specific literals. The generated answer sets then are those for which the sum of the weights of satisfied literals is minimal/maximal. The DLV system provides so-called “weak constraints,” which carry a weight of importance; they should be satisfied if possible, but their violation does not “kill” answer sets. The answer sets of a program P plus a set W of weak constraints are those answer

sets of P that minimize the sum of the weights of violated weak constraints. Other, non-numerical approaches use an external partial preference order on rules or special syntactic constructs in the rules; for example, Brewka et al.⁶ In each case the available preference information induces a corresponding ordering on answer sets, and the best ones are chosen.

Modularity and external data access. Modularity is an important notion in software development. In the context of ASP it is only beginning to receive the attention it deserves but already several key concepts and ideas have been developed.^{10,23} Modularization is a way to structure and ease the program development process. Modular ASP programs consist of modules that are combined through suitable interfaces. This way parts of a program can be developed and verified independently, and they can be more easily reused. A related issue is to integrate external sources into ASP programs. In a rule one would often like to access a database, an ontology or some other source of information. To serve this, HEXprograms¹³ provide a universal interface for arbitrary sources of external computation through the notion of external atom, which is akin to a remote procedure call but facilitates proper recursion.

Applications

The ASP paradigm is rather new but it has already led to many successful applications. We briefly discuss a few examples in different categories. Further examples can be found in the team-building sidebar noted earlier as well as the ones entitled “ASP for Repairing Large-Scale Biological Networks” and “ANTON—An ASP-based Music Composition System.”

Applications in science and humanities. An illustrative example is phylogenetic systematics—the study of evolutionary relations between species based on their shared traits.¹⁵ These relations can form a tree (called a “phylogeny”) where leaves represent the species, internal vertices their ancestors, and edges the genetic relationships between them. The computational task is to construct phylogenies, and researchers demonstrated the applicability and

effectiveness of ASP-based methods for these tasks by analysis of natural languages and parasite-host systems species of oak trees.

Industrial applications. An early, almost prototypical industrial application for ASP is product configuration.³⁹ The general idea is to have rules in a program that generate the space of all combinations of product components. Constraint rules then filter out configurations that are impossible, either due to some given, fixed restrictions on how components can be combined, or due to a violation of specific user requirements. Another early application is a decision support system for the space shuttle.³⁴ During normal shuttle operations, astronauts follow pre-scripted plans. However, in case of failure different courses of action are needed to ensure safety of the crew and completion of the mission. As exponentially many failures are possible, pre-planning for all exceptional circumstances is unfeasible, and decision support is needed. Based on failure information, the ASP system suggests a course of action.

Data management. INFOMIX^b is a project on advanced information integration. The main task is to provide a uniform interface to pre-existing data sources, where an information integration system frees the user from finding and accessing relevant data sources, and from cleaning and combining data in them. Here, in particular, proper handling of incomplete and inconsistent data is challenging. The INFOMIX prototype showed that ASP provides effective technology to deal with advanced information integration tasks. ASP also proved to be a valuable host for realizing query engines in the context of the Web. In fact, one of the first SPARQL reasoning engines for querying RDF data sources has been realized via an ASP encoding.³⁷

Artificial intelligence. Given the fact that ASP has roots in knowledge representation and nonmonotonic reasoning, its usage for problem solving in artificial intelligence (AI) has been investigated early on. Classic AI problems including planning, diagnosis, and agent decision making have been reduced to ASP, resulting

^b www.mat.unical.it/infomix/

in effective realizations (several are available, for example, as DLV frontends). As it turned out, thanks to its features—high expressiveness, non-determinism via multiple answer sets, and high declarativity—ASP is a valuable host language for domain-specific AI formalisms, allowing for quick experimental prototyping. A recent example of this is repair of Web-service workflows,¹⁸ where these features were fruitfully exploited.

Relation to Other Formalisms

ASP is just one of many ways to solve search problems by means of logic reasoning procedures. We briefly comment on three formalisms for declarative problem solving that are both related and relevant to ASP.

ASP and SAT solving. The key idea of ASP—to encode the solutions of a search problem in the models of a logical theory for declarative problem solving—had been exploited before. In a landmark paper, Kautz and Selman²⁴ showed that encoding a planning problem as a theory in propositional logic, with plans represented by models, and using SAT solvers to find models and so plans, could outperform specialized planners. Applications of similar nature led to a boom in SAT solver technology.

While both SAT solving and core ASP apply in principle to the same problems, there are differences. First, ASP supports variables that range over finite domains and enable uniform and compact representation of problems independently of data. In the Hamiltonian cycle example, we have a *single, fixed* program that works uniformly with *all* input graphs. ASP grounders produce instances for ASP solvers based on the program and the input graph. Having problem specifications separate from data facilitates debugging and testing, supports optimization and developing reusable problem modules, all topics currently under research. There is no such separation of problem specification and data in SAT, where the two are hard-wired into programs that generate satisfiability instances to be solved. This makes development of software engineering techniques for SAT difficult. Second, any problem that can be modeled in SAT can be modeled



Thanks to its features—high expressiveness, nondeterminism via multiple answer sets, and high declarativity—ASP is a valuable host language for domain-specific AI formalisms, allowing for quick experimental prototyping.



equally well in ASP. But there are problems—typically involving concepts defined inductively such as reachability in graphs—that are easy to cast in ASP, but representing them appropriately for SAT solving results in larger instances that slow down solving. In a similar vein, the language of ASP offers constructs such as “minimized” disjunction, aggregates and priorities that are useful in practical applications, are easy to use, and are supported by most current ASP solvers. These constructs require specialized ad hoc treatment when modeling for SAT solving. For some of them concise representations are not even possible.

ASP and Prolog. Prolog is the most widely known logic programming language. For some time, however, the interest in Prolog has been declining, in part because expectations of ambitious endeavors like the Fifth Generation Project could not be met. Is ASP, which is sometimes called *Answer Set Prolog*, a better Prolog? The two are similar in syntax and there are semantic connections, too. For a large class of programs, if Prolog returns “yes” (respectively “no”) to a ground query, then the query belongs (respectively, does not belong) to the unique answer set of the program. But in spite of these similarities, ASP and Prolog are actually quite different. Prolog was designed as a general purpose, Turing-complete programming language. It uses function symbols for nested terms to build potentially infinite data structures, and recursion for unbounded computation; solutions are computed by query answering, which amounts to *proof search*. In contrast, ASP was not conceived for such generality and works over a finite domain of “flat” data (though work on function symbols in ASP is under way); solutions are encoded in answer sets; that is, in *models*, and thus model-finding, not proof-finding, methods matter.

To be an effective Prolog programmer one needs to understand how to use terms as data structures, not quite intuitive, and not part of any standard CS curriculum, and to understand Prolog’s evaluation strategy, SLD resolution with unification, which is arguably quite difficult to master with no adequate logic background. To

complicate matters more, the order of rules in a Prolog program and of subgoals (literals) in rule bodies matters. Changing it may turn a working program useless. These features give a programmer control over the execution of search and make Prolog a programming language, a formalism in which one can implement algorithms. In this sense, Prolog misses *true declarativity*. ASP, on the other hand, offers ways to model specifications yet does not allow the programmer to control the search. Consequently, while less expressive, ASP is “more declarative:” it is intuitive, requires less background in logic, and its semantics is robust to changes in the order of literals in rules and rules in programs. Still, to solve practical application problems in ASP efficiently some experience is required. Typically, there are alternative ways to model a problem as an answer set program, and the resulting programs may perform quite differently. One of the more obvious and in the same time more important considerations for designing efficient answer set programs is that the size of the ground program be possibly small.

ASP and constraint programming.

Constraint programming is concerned with modeling and solving problems, where solutions are assignments of values from finite domains to *decision* variables. These assignments are subject to constraints given in the problem statement.

For instance, we can specify the n -queens problem as follows: assign to each of n decision variables $x_1, \dots, x_n$, a value from $1, \dots, n$ so that $x_i \neq x_j$, for $i \neq j$, and $|x_i - x_j| \neq |i - j|$. To solve a problem like this in constraint programming, one describes it in some high-level *modeling* language, such as ESSENCE or ZINC, and then maps the description into a set of constraints in some low-level format or, in other words, into a constraint satisfaction problem (CSP), which is then solved. The similarities with ASP—modeling in a high-level language and compiling to a low-level representation—are evident. But there are differences.

High-level languages including those mentioned here closely follow mathematical notation and, in particular, support using sets, relations,

functions and partitions as possible values of decision variables; modeling requires some mathematical sophistication. Mapping the high-level specification of a problem into constraints that will lend themselves well to processing also requires certain mathematical background, and expertise in constraint modeling and solving. On the other hand, the language of ASP and its extensions were developed with knowledge representation applications in mind and their constructs were designed to capture patterns of natural language statements, definitions, and default negation. The language is simple and intuitive to use. In addition, once a problem is modeled in ASP all subsequent steps are performed automatically. A grounder compiles a program into its propositional form and a solver computes solutions. There are also differences at the solving stage. For constraint programming this step consists of solving a CSP over an arbitrary but finite value domain. For ASP, all domains are binary (the variables are propositional atoms). This restriction opens a way to highly efficient implementations, as witnessed by the recent impressive advances in SAT solving technology.

Ongoing Developments

ASP processing tools are under continuous development and already achieved levels that make them effective in large-scale practical applications. Efforts to increase efficiency by new grounding technology and solving methods, but also non-ground evaluation are under way. To a large degree the advances are the result of a communitywide effort to build benchmarks, collect hard test problems and instances, and organize regular ASP system competitions.

However, the situation is quite different as concerns basic *software development support* in ASP. Although the first integrated development environment ASPIDE^c was recently announced, much remains to be done. One of the areas in need of progress is program debugging. Even if developing answer set programs benefits from the declarative nature of ASP,

discovering errors is difficult. There is some research in this direction already,^{5,7} but the ideas proposed need to be explored further. Methodologies for development and optimization are also important issues. Much progress was made in understanding the theory behind modularity of answer set programs. We discussed some of that research earlier. Here we mention research on *strong equivalence*²⁷ or, to put it informally, equivalence for replacement within larger systems, and further notions of equivalence.⁴⁰ Elegant technical results are now available, but their impact on practical developments remains open.

Function symbols often make modeling easier and the resulting encodings more readable and concise. Thus, not allowing them in ASP (except in built-ins for arithmetic) was perceived as a limitation. But allowing uninterpreted function symbols renders most of the ASP program processing techniques useless, as ground programs typically become infinite. A middle ground can be found, though. It requires imposing restrictions on how function symbols can occur in programs. Some globally constrain atom dependency in the grounded program,^{3,8} while others locally constrain the rule syntax.¹⁴ The LPARSE grounder was the first to offer (albeit limited) support of function symbols, while GRINGO and the DLV system (latest release) include some of the more recent advances. Recent research indicates that ASP can provide a full first-order language for non-monotonic reasoning, with the notion of an answer set extended to this setting.^{17,36} Computational support and further research will be required, however, to make this available for practical applications.

Integration of SAT solving with constraint solving techniques known as Satisfiability Modulo Theories has proved successful for SAT. The ASP community has recently taken up this idea, with CLINGCON (see Table 2) being a very promising system combining ASP with specialized constraint solvers.

Quantitative methods turned out to be extremely effective in knowledge representation applications in which uncertainty cannot be avoided. ASP as

^c www.mat.unical.it/~ricca/aspide/

it exists now is not designed for such applications. This is a drawback and so there are already research efforts to enhance ASP with means to combine probabilities and utilities with qualitative representations of uncertainty.² This research direction has not yet matured, though, and it is too early to say how successful such integration will turn out to be.

Conclusion

The aim of this article was to provide the reader with a basic understanding of the main motivation, the most important concepts, and the relevant techniques underlying ASP, a rather new yet highly promising declarative problem-solving paradigm.

We covered answer set semantics, both for propositional and predicate programs, discussed the ASP paradigm, and related it to some other problem-solving approaches. Moreover, we presented algorithms and solvers, several extensions of the basic approach, and some illustrative applications. This article should not be viewed as a complete overview of the field. It is meant as an appetizer. For a more complete picture we recommend Eiter et al.¹² or Baral.¹

Acknowledgments

The authors are grateful to the reviewers for comments that helped improve the presentation of the material. Brewka's work was supported by the DFG grant Br1817/3; Eiter's work was supported by the Austrian Science Fund (FWF) grants P20840 and P20841, Vienna Science and Technology Fund (WWTF) ICT08-020, and the European Commission grant ICT FP7 231875. Truszczyński's work was supported by NSF grant IIS-0913459. 

References

- Baral, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
- Baral, C., Gelfond, M. and Rushton, J.N. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming* 9, 1 (2009), 57–144.
- Baselice, S., Bonatti, P.A. and Crisculo, G. On finitely recursive programs. *Theory and Practice of Logic Programming* 9, 2 (2009), 213–238.
- Boenn, G., Brain, M., Vos, M.D. and Fitch, J. Automatic music composition using answer set programming. *Theory and Practice of Logic Programming* 11, 2-3 (2011), 397–427.
- Brain, M. and Vos, M.D. Debugging logic programs under the answer set semantics. In *Proc. 3rd International Workshop on Answer Set Programming, CEUR Workshop Proceedings* 142, 2005. M. De Vos and A. Provetti, Eds.
- Brewka, G., Niemelä, I. and Truszczyński, M. Answer set optimization. In *Proc. 18th International Joint Conference on Artificial Intelligence*. G. Gottlob and T. Walsh, Eds. Morgan Kaufmann, 2003, 867–872.
- Brummayer, R. and Järvisalo, M. Testing and debugging techniques for answer set solver development. *Theory and Practice of Logic Programming* 10, 4-6 (2010) 741–758.
- Calimeri, F., Cozza, S., Ianni, G. and Leone, N. Computable functions in ASP: Theory and implementation. In *Proc. 24th International Conference on Logic Programming*, LNCS 5366. M. Garcia de La Banda and E. Pontelli, Eds. Springer, 2008, 407–424.
- Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33, 3 (2001), 374–425.
- Dao-Tran, M., Eiter, T., Fink, M. and Krennwallner, T. Modular nonmonotonic logic programming revisited. In *Proc. 25th International Conference on Logic Programming*, LNCS 5649. P. M. Hill and D.S. Warren, Eds. Springer, 2009, 145–159.
- Eiter, T. and Gottlob, G. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15, 3/4 (1995), 289–323.
- Eiter, T., Ianni, G., and Krennwallner, T. Answer set programming: A primer. *Reasoning Web*, LNCS 5689. S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. A. Schmidt, Eds. Springer, 2009, 40–110.
- Eiter, T., Ianni, G., Schindlauer, R. and Tompits, H. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proc. 19th International Joint Conference on Artificial Intelligence*. L. P. Kaelbling and A. Saffioti, Eds. 2005, 90–96.
- Eiter, T. and Simkus, M. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Trans. Computational Logic* 11, 2 (2010).
- Erdem, E. Applications of answer set programming in phylogenetic systematics. *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, LNCS 6565. M. Balduccini and T. C. Son, Eds. Springer, 2011, 415–431.
- Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T. and Ielpa, G. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming* 8, 5-6 (2008), 545–580.
- Ferraris, P., Lee, J. and Lifschitz, V. Stable models and circumscription. *Artificial Intelligence* 175, 1 (2011), 236–263.
- Friedrich, G., Fugini, M., Mussi, E., Pernici, B. and Tagni, G. Exception handling for repair in service-based processes. *IEEE Trans. on Software Engineering* 36, 2 (2010) 198–215.
- Gebser, M., Guziolowski, C., Ivanchev, M., Schaub, T., Siegel, A., Thiele, S. and Veber, P. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *Proc. 12th International Conference on Principles of Knowledge Representation and Reasoning*. F. Lin, U. Sattler, and M. Truszczyński, Eds., 2010, 497–507.
- Gelfond, M. and Lifschitz, V. The stable model semantics for logic programming. *Logic Programming: The 5th International Conference and Symposium*. R.A. Kowalski and K. Bowen, Eds. MIT Press, Cambridge, MA, 1988, 1070–1080.
- Gelfond, M. and Lifschitz, V. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9 (1991), 365–385.
- Greco, S., Molinaro, C., Trubitsyna, I. and Zumpano, E. NP datalog: A logic language for expressing search and optimization problems. *Theory and Practice of Logic Programming* 10, 2 (2010), 125–166.
- Janhunen, T., Oikarinen, E., Tompits, H. and Woltran, S. Modularity aspects of disjunctive stable models. *Journal of Artificial Intelligence Research* 35 (2009), 813–857.
- Kautz, H.A. and Selman, B. Planning as satisfiability. In *Proc. 10th European Conference on Artificial Intelligence*. B. Neumann, Ed. 1992, 359–363.
- Leone, N., Rullo, P. and Scarcello, F. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation* 135, 2 (June 1997), 69–112.
- Lifschitz, V. Answer set programming and plan generation. *Artificial Intelligence* 138 (2002), 39–54.
- Lifschitz, V., Pearce, D. and Valverde, A. Strongly equivalent logic programs. *ACM Trans. Computational Logic* 2, 4 (2001), 526–541.
- Lin, F. and Zhao, Y. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, 2002, 112–117.
- Marek, V.W. and Truszczyński, M. Autoepistemic logic. *J. ACM* 38, 3 (1991) 588–619.
- Marek, V.W. and Truszczyński, M. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm—A 25-Year Perspective*. K. Apt, K.V. W. Marek, M.W. Truszczyński and D.S. Warren, Eds. Springer, 1999, 375–398.
- Marek, V.W. and Truszczyński, M. *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer, 1993.
- Niemelä, I. Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3-4 (1999), 241–273.
- Niemelä, I. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence* 53, 1 (2008), 313–329.
- Nogueira, M., Balduccini, M., Gelfond, M., Watson, R. and Barry, M. A Prolog decision support system for the space shuttle. In *Proc. 1st International Workshop on Answer Set Programming*. A. Provetti and T. C. Son, Eds. 2001.
- Pearce, D. Equilibrium logic. *Annals of Mathematics and Artificial Intelligence* 47, 1-2 (2006), 3–41.
- Pearce, D. and Valverde, A. Towards a first order equilibrium logic for nonmonotonic reasoning. In *Proc. 9th European Conference on Logics in Artificial Intelligence*, LNCS 3229. Springer, 2004, 147–160.
- Polleres, A. From SPARQL to rules (and back). In *Proc. 16th International Conference on World Wide Web*. C.L. Williamson, M.E. Zurko, P.F. Patel-Schneider, and P.J. Shenoy, Eds. ACM, 2007, 787–796.
- Ricca, F., Grasso, G., Alviano, M., Manna, M. Lio, V. Liritano, S. and Leone, N. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 2011; doi:10.1017/S147106841100007X.
- Soininen, T. and Niemelä, I. Developing a declarative rule language for applications in product configuration. In *Proc. 1st International Workshop on Practical Aspects of Declarative Languages*, LNCS 1551. G. Gupta, Ed. Springer, 1999, 305–319.
- Woltran, S. A common view on strong, uniform, and other notions of equivalence in answer-set programming. *Theory and Practice of Logic Programming* 8, 2 (2008), 217–234.

Gerhard Brewka (brewka@informatik.uni-leipzig.de) is a professor of computer science at University of Leipzig's Informatics Institute, Leipzig, Germany.

Thomas Eiter (eiter@kr.tuwien.ac.at) is a professor of computer science at Vienna Univ. of Technology's Institute of Information Systems, Vienna, Austria.

Miroslaw Truszczyński (mirek@cs.uky.edu) is a professor at University of Kentucky's Department of Computer Science, Lexington, KY.

CAREERS

American University

Assistant Professor in Computational Neuroscience

The College of Arts and Sciences at **American University** (Washington, DC) invites applications for a full-time, tenure-track, **Assistant Professor position, beginning in August 2012, in computational neuroscience** (broadly defined, including but not limited to neural networks, simulation, image processing, and bio-informatics). The appointee's tenure home and departmental affiliation will depend on his or her research background. Applicants must have a PhD in a relevant discipline. Teaching and post-doctoral experience are preferred. Responsibilities include: teaching and curriculum development; establishing an internationally recognized research program, preferably one that can involve undergraduate research participation; strengthening connections to neurosciences across campus; and service to the appointee's home department and the wider university.

American University has made other recent hires in neuroscience, and benefits from proximity to other scientific institutions in the Washington area. (For example, NIH is three metro stops from the AU campus.) The College of Arts and Sciences offers a variety of degrees at the undergraduate, masters, and doctoral levels. For more information about our programs, visit www.american.edu/cas/.

Applicants should submit a cover letter, curriculum vitae, teaching statement, and research statement, and applicants must arrange for three letters of recommendation to be sent directly to the search committee. Materials can be submitted online (highly preferred) at <http://academicjobsonline.org/ajob>, or via email to CompNeuroSearch@american.edu, or in hard copy to Computational Neuroscience Search Committee, Department of Mathematics and Statistics, American University, Washington, DC 20016-8050. Applications received by December 10, 2011 will receive full consideration. American University is an EEO/AA institution, committed to a diverse faculty, staff, and student body. Women and minority candidates are strongly encouraged to apply. American University offers employee benefits to same-sex domestic partners of employees and prohibits discrimination on the basis of sexual orientation/preference and gender identity/expression.

Baylor University

Assistant, Associate or Full Professor of Computer Science

The Department of Computer Science seeks a productive scholar and dedicated teacher for a tenure-track position beginning August, 2012. The ideal candidate will hold a terminal degree in Computer Science or closely related field, demonstrate scholarly capability and an established and active independent research agenda in one of sev-

eral core areas of interest, including, but not limited to, game design and development, software engineering, computational biology, machine learning or large-scale data mining. A successful candidate will also exhibit a passion for teaching and mentoring at the graduate and undergraduate level. For position details and application information please visit: <http://www.baylor.edu/hr/index.php?id=81302>

The Department: The Department offers a CS-AB-accredited B.S. in Computer Science degree, a B.A. degree with a major in Computer Science, a B.S. in Informatics with a major in Bioinformatics, and a M.S. degree in Computer Science. The Department has 13 full-time faculty members, over 250 undergraduate majors and approximately 30 master's students. We are currently seeking approval to offer a dual Ph.D. degree in cooperation with a well-established partner institution. Interested candidates may contact any faculty member to ask questions and/or visit the web site of the School of Engineering and Computer Science at <http://www.ecs.baylor.edu>.

The University: Chartered in 1845 by the Republic of Texas, Baylor University is the oldest university in Texas and the world's largest Baptist University. It is situated on a 500-acre campus next to the Brazos River and annually enrolls more than 14,000 students in over 150 baccalaureate and 80 graduate programs. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and strengthening its distinctive Christian mission as described in Baylor 2012 (www.baylor.edu/vision/).

Application Procedure: Applications, including detailed curriculum vitae, a statement demonstrating an active Christian faith, and contact information for three references should be sent to: Chair Search Committee, Department of Computer Science, Baylor University, One Bear Place #97356, Waco, TX 76798-7356.

Appointment Date: Fall 2012. For full consideration, applications should be received by January 1, 2012.

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an Affirmative Action/Equal Employment Opportunity employer, Baylor encourages minorities, women, veterans, and persons with disabilities to apply.

Baylor University

Assistant or Associate Professor of Computer Science

Chartered in 1845 by the Republic of Texas, Baylor University is the oldest university in Texas and the

world's largest Baptist University. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and strengthening its distinctive Christian mission as described in Baylor 2012 (www.baylor.edu/vision/). The combination of teaching, research and service has made Baylor one of the best universities for faculty, according to the Chronicle of Higher Education <http://chronicle.com/article/Great-Colleges-to-Work-For/128312/>.

The Department of Computer Science seeks a productive scholar and dedicated teacher for a tenure-track position beginning August, 2012. All specializations will be considered. Game/simulated environments, mobile computing, and graphics are of particular interest. The successful candidate will hold a terminal degree in Computer Science or a closely related field, demonstrate scholarly capability in his or her area of specialization, and exhibit a passion for teaching and mentoring at the graduate and undergraduate level. For position details and application information please visit: <http://www.ecs.baylor.edu>.

The Department: The Department offers a CS-AB-accredited B.S. in Computer Science degree, a B.A. degree with a major in Computer Science, a B.S. in Informatics with a major in Bioinformatics, and a M.S. degree in Computer Science. We are currently seeking approval to offer a dual Ph.D. degree in cooperation with a well-established European institution. The Department has 15 full-time faculty, over 370 undergraduate majors and 30 master's students. The Department's greatest strength is the faculty's dedication to the success of the students and each other. Interested candidates may contact any faculty member to ask questions and/or visit the web site of the School of Engineering and Computer Science at <http://www.ecs.baylor.edu>.

The University: Baylor University, situated on a 500-acre campus next to the Brazos River. It annually enrolls more than 14,000 students in over 150 baccalaureate and 80 graduate programs through: the College of Arts and Sciences; the Schools of Business, Education, Engineering and Computer Science, Music, Nursing, Law, Social Work, and Graduate Studies; plus Truett Seminary and the Honors College. For more information see <http://www.baylor.edu>.

Application Procedure: Please submit a letter of application, current curriculum vitae, and transcripts. Include names, addresses, and phone numbers of three individuals from whom you have requested letters of recommendation to: Jeff Donahoo, Ph.D., Search Committee Chair, Baylor University, One Bear Place #97356, Waco, Texas 76798-7356. Materials may be submitted to: Jeff_Donahoo@baylor.edu

Appointment Date: Fall 2012. For full consideration, applications should be received by January 1, 2012. However, applications will be accepted until the position is filled.

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an Affirmative Action/Equal Employment Opportunity employer, Baylor encourages minorities, women, veterans, and persons with disabilities to apply.

Bradley University
Assistant Professor
Department of Computer Science and
Information Systems

The CS&IS Department invites applications for a tenure track Assistant Professor position starting in August 2012. The position requires that a PhD in Computer Science or a closely related field is completed prior to the start date. We will consider strong candidates from all areas of computer science but applicants with expertise in mobile computing, architecture, security, and gaming are especially encouraged.

The successful candidate is expected to develop into an outstanding teacher at the undergraduate and graduate levels, establish and maintain an active and rigorous research program, actively seek funding in support of his/her research activities, and involve students in his/her research activities. The teaching load is three courses per semester. Some teaching experience is preferred.

The CS&IS Department at Bradley University currently consists of 9 full time tenure/tenure-track faculty professorial positions and one lecturer position. We currently offer bachelors and masters degrees in both Computer Science and Computer Information Systems.

To apply, send a letter of application, a curriculum vitae, transcripts, and at least three letters of reference to:

Chair Search Committee
Department of CS&IS
Bradley University
Peoria, IL 61625
e-mail: jawalter@bradley.edu

Initial screening will begin on February 1, 2012 and will continue until the position is filled.

Employment with Bradley University is contingent upon satisfactory completion of a criminal background check. Bradley University is an EEO/AA Employer.

Bradley University is an Equal Opportunity Employer. The administration, faculty and staff are committed to attracting qualified candidates from groups currently underrepresented on our campus.

Bucknell University
Assistant Professor of Computer Science

Applications are invited for a tenure track assistant professor position in computer science, preferably at the entry-level (four or fewer years of full-time teaching experience with a recent

Ph.D.), beginning mid-August 2012. Outstanding candidates in all areas will be considered. We are particularly interested in candidates whose research area is compatible with teaching a theory of computation course. Background in the area of programming languages would be a plus. Candidates are expected to have completed their Ph.D. requirements in computer science or a closely related field by August 15th, 2012. A strong commitment to excellence in teaching and scholarship is required. The successful candidate must be able to participate in the teaching of required core courses and be able to develop elective courses in the candidate's area of expertise. Bucknell is a highly selective private university emphasizing quality undergraduate education in engineering and in liberal arts and sciences. The B.S. programs in computer science are ABET accredited. The computing environment is Linux/Unix-based. More information about the department can be found at: <http://www.bucknell.edu/ComputerScience/> Applications will be considered as received and recruiting will continue until the position is filled. Candidates are asked to submit a cover letter, CV, a statement of teaching philosophy and research interests, and the contact information for three references. Please submit your application to <http://jobs.bucknell.edu/> by searching for the "Computer Science Faculty Position". Please direct any questions to Professor Stephen Guattery of the Computer Science Department at guattery@bucknell.edu. Bucknell University values a diverse college community and is committed to excellence through diversity in its faculty, staff

CALL FOR PhD STUDENTS

The Graduate School at IST Austria invites applicants from all countries to its PhD program. IST Austria is a new institution located on the outskirts of Vienna dedicated to cutting-edge basic research in the natural sciences and related disciplines. The language at the Institute and the Graduate School is English.

The PhD program combines advanced coursework and research, with a focus on Biology, Computer Science, Neuroscience, and interdisciplinary areas. IST Austria offers internationally competitive PhD salaries supporting 4-5 years of study. Applicants must hold either a BS or MS degree or equivalent.

The Institute offers PhD students positions with the following faculty:

- **Nick Barton** Evolutionary and Mathematical Biology
- **Jonathan P. Bollback** Evolutionary Biology
- **Sylvia Cremer** Evolutionary and Behavioral Biology
- **Caroline Uhler** Statistics
- **Tobias Bollenbach** Biophysics and Systems Biology
- **Călin C. Gue** Systems and Synthetic Biology
- **Carl-Philipp Heisenberg** Cell and Developmental Biology
- **Harald Janovjak** Molecular and Cellular Biophysics
- **Daria Siekhaus** Cell and Developmental Biology
- **Michael Sixt** Cell Biology and Immunology

- **Jozsef Csicsvari** Systems Neuroscience
- **Peter Jonas** Neuroscience
- **Gašper Tkačik** Theoretical Biophysics and Neuroscience
- **Krishnendu Chatterjee** Game Theory and Software Systems Theory
- **Herbert Edelsbrunner** Algorithms, Geometry, and Topology
- **Thomas A. Henzinger** Software Systems Theory
- **Vladimir Kolmogorov** Computer Vision and Graph Algorithms
- **Christoph Lampert** Computer Vision and Machine Learning
- **Krzysztof Pietrzak** Cryptography
- **Chris Wojtan** Computer Graphics

Additional faculty members will be announced on the IST website www.ist.ac.at.

For further information and access to the online application please consult www.ist.ac.at/gradschool. For inquiries, please contact gradschool@ist.ac.at. For students wishing to enter the program in the fall of 2012, the deadline for applications is **January 15, 2012**.

IST Austria values diversity and is committed to equality. Female students are encouraged to apply.



and students. An Equal Opportunity/Affirmative Action Employer, Bucknell University especially welcomes applications from women and minority candidates.

California State Polytechnic University, Pomona

Computer Science Department
<http://www.csupomona.edu/~cs/>

The Computer Science Department invites applications for a tenure-track position at the rank of Assistant Professor to begin Fall 2012. We are particularly interested in candidates with specialization in Secure Software Engineering, although candidates in all areas of Computer Science will be considered, and are encouraged to apply. Cal Poly Pomona is 30 miles east of L.A. and is one of 23 campuses in the California State University. The department offers an ABET-accredited B.S. program and an M.S. program.

Qualifications: Possess, or complete by September 2012, a Ph.D. in Computer Science or closely related area. Demonstrate strong English communication skills, a commitment to actively engage in the teaching, research, and curricular development activities of the department at both undergraduate and graduate levels, and ability to work with a diverse student body and multicultural constituencies. Ability to teach a broad range of courses, and to articulate complex subject matter to students at all educational levels. First consideration will be given to completed applications received no later than January 9, 2012.

Contact: Faculty Search Committee, Computer Science Department, Cal Poly Pomona, Pomona, CA 91768. Email: cs@csupomona.edu.

Cal Poly Pomona is an Equal Opportunity, Affirmative Action Employer.

Position announcement available at: <http://academic.csupomona.edu/faculty/positions.aspx>

Lawful authorization to work in US required for hiring.

California State University, Chico

Assistant Professor

California State University, Chico, Dept. of Computer Science has two full time tenure track Asst. Prof positions, starting 8/2012. EOE Employer. Please see the full Announcement at: <http://csci.ecst.csuchico.edu/jobs>

California State University, Fullerton

Assistant Professor

The Department of Computer Science invites applications for a tenure-track position at the Assistant Professor level starting Fall 2012. For a complete description of the department, the position, desired specialization and other qualifications, please visit <http://diversity.fullerton.edu/>.

Carleton College

Assistant Professor of Computer Science

Carleton College invites applications for a one-year position (potentially renewable for a second

year) in computer science, in any area of specialization, beginning September 1, 2012.

Carleton is a highly selective liberal arts college with outstanding, enthusiastic students. We seek an equally enthusiastic computer scientist committed to excellence in teaching, curriculum design, ongoing research, and undergraduate research advising. We are particularly interested in applicants who will strengthen the departmental commitment to students from underrepresented groups. To learn more about the position or to apply, visit jobs.carleton.edu. Applications completed by December 16, 2011 will receive full consideration.

Carleton College does not discriminate in providing employment. Please view the description for this position at jobs.carleton.edu for Carleton's full anti-discrimination statement.

Carnegie Mellon University, Silicon Valley (CMUSV)

Senior Faculty

We are seeking applicants with both industrial experience and traditional academic credentials to fill a senior position in our growing software engineering (SE) program. The faculty member will play key roles in expanding our software engineering research program, in teaching, in recruiting, and in overall campus leadership. Familiarity with software development practices used in Silicon Valley (SV) is a significant advantage. We place a strong emphasis on written and spoken communication.

Our MS program in SE currently enrolls about 80 students, both full-time and part-time, with many employed at top SV companies. The program is project-based, team-oriented, and follows a learning-by-doing approach, with small seminar-style classes. Faculty work directly as advisors to student teams on their deliverables, teaching knowledge and skills on a just-in-time basis. CMUSV is growing its research activities, emphasizing mobility, networking, and security. We are building up research in agile methods, cloud computing, and mobile and embedded system development. CMUSV also offers PhD degrees through CMU's ECE Dept.

The ideal candidate for this position will have sufficient experience to justify an appointment to a senior faculty position. We will give strong consideration to candidates who have spent most of their professional career in industry, and are now seeking an academic position. Please provide us with your curriculum vita with publication list, a statement about your practical experience, and letters from five references. Starting date is August, 2012, or sooner.

Apply for this position (#8652) at <http://sv.cmu.edu/se-positions>

More information on CMUSV may be found at <http://sv.cmu.edu>. Direct queries to SeniorS-Esearch@sv.cmu.edu

Carnegie Mellon University does not discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex, handicap or disability, age, sexual orientation, gender identity, religion, creed, ancestry, belief, veteran status, or genetic information.



INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

Ahmedabad University-AU

A State Private University, Gujarat, India
ict.ahduni.edu.in

AU is in the process of establishing a new Institute of ICT by July 2012. Institute of ICT, AU invites applications for faculty positions at the level of Director, Professors, and Associate/Assistant Professors. Academicians committed to teaching and research, excited by institution building are invited to participate in our vision to establish a leading new institute of ICT.

The institute aims to redefine ICT education -where high powered technological innovations will complement sustainable growth in various sectors such as healthcare, energy, finance. We are effectively looking at redefining how ICT education is handled in the country today and bolster our efforts by training the students not only in fundamentals of computing/engineering but also in handling multidisciplinary product development, team work and real-time problem solving. To build a high quality research driven academic program (B Tech, M Tech and PhD), the school will leverage its multi-disciplinary position as one of the four schools planned under the umbrella of Institute of Science and Technology, AU: Engineering, Life sciences, Physical sciences & ICT. AU is also engaged in developing a network cluster with leading institutes in a variety of disciplines.

Candidates, from any branch of ICT or related cross disciplinary fields such as computer science, electrical engineering, bio informatics, and maths, physics may apply. For all positions, a PhD in related field, significant demonstrated research record commensurate with the level of the position being applied for are required. Being a State University (privately funded) we offer more attractive remuneration package as compared to other institutions in the country. Faculty will be encouraged and supported to establish research labs and get involved in institution building, innovate, teach, consult and conduct collaborative research.

Applications should consist of a cover letter, CV, a research statement, names and contact information of at least 3 references, and URLs' /Pdf of at least 3-5 papers. Submit CV and queries to: ict@ahduni.edu.in

Carnegie Mellon University
Tepper School of Business
Tenure-track faculty opening in Information Systems, starting in September 2012

We invite applications from individuals who are interested in how information technology will transform businesses, markets, and economic processes. We have a particular interest in applicants who can access and make sense of the large volume of data that is available today and help design new systems and improve the design of existing systems. Applicant should send a current curriculum vita, evidence of research such as publications, working papers, or dissertation proposal to: isgroup@andrew.cmu.edu and three recommendation letters (via the Postal Service) to Mr. Phillip Conley, Information Systems Faculty Recruiting, Carnegie Mellon University, Tepper School of Business, Room 369 Posner Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890 (Phone: 412-268-6212). In order to ensure full consideration, completed applications must be received by December 31, 2011.

Applicants may hold a doctoral degree in any business discipline, Information Systems, Computer Science, Economics or Statistics. We are primarily seeking candidates at the Assistant Professor level. Applicants should have completed or be nearing completion of a Ph.D., and should demonstrate potential of excellence in research and teaching. Teaching assignments encompass BS, Masters, and Ph.D. programs.

Carnegie Mellon is an equal opportunity, affirma-

tive action employer with particular interest in identifying women and minority applicants for faculty positions.

Colgate University
Department of Computer Science

The Department of Computer Science invites applications for a tenure-stream position in computer science at the rank of Assistant Professor, beginning fall semester 2012. Completion of a Ph.D. in computer science is expected prior to or shortly after the date of hire. Preference will be given to applicants with expertise in either Data Mining or Graphics though applicants in other areas will also be considered. A letter of application, curriculum vitae and three letters of recommendation should be submitted through <https://academicjobsonline.org/ajo/jobs/1132> where you will also find a full ad description.

Review of applications will begin January 10, 2012, and continue until the position is filled. Colgate University is an EO/AA Employer; women and minorities are especially encouraged to apply.

Colorado School of Mines
Assistant Professor

Applications are invited for an anticipated tenure-track faculty position in data mining, data management, machine learning, or high performance computing.

The successful candidate will be expected to teach at both the undergraduate and graduate levels, direct graduate research, and develop a strong externally funded research program.

An earned Ph.D. in Computer Science or a closely related field by the time of appointment is required. Candidates must provide evidence of teaching competence in computer science and should demonstrate the potential to develop a self-sustaining research program.

For the complete job announcement and directions on how to apply, visit: <http://inside.mines.edu/HR-Academic-Faculty>

Colorado State University
Department of Computer Science
Assistant Professor

Colorado State University has an opening for one or more tenure-track assistant professor positions in Computer Science, beginning fall 2012. Areas of interest include computational biology, parallel computing, programming languages that focus on parallel programming models, mobile computing and HCI. For more information go to: <http://www.cs.colostate.edu>. Applications must be received by January 9, 2012 at <http://www.natsci.colostate.edu/employment/compsci/>. Complete applications of semi-finalists will be available to department faculty for review.

CSU is an EO/EA/AA employer. Colorado State University conducts background checks on all final candidates.



IST AUSTRIA IS LOOKING FOR Professors and Assistant Professors

IST Austria (Institute of Science and Technology Austria) invites applications for Professors and Assistant Professors in all fields of the natural and mathematical sciences and related disciplines. Outstanding scientists in software systems (operating, distributed, database systems) are especially encouraged to apply.

The Institute, which is situated on the outskirts of Vienna, was established by the Austrian government with a focus on basic research. The campus opened in 2009 and is expected to grow to 45 research groups and over 500 employees by 2016. IST Austria is entitled to award PhD degrees and includes an English-language graduate school. It aims to achieve an international mix of scientists and chooses them solely on the basis of their individual excellence and potential contribution to research.

The Institute recruits tenured and tenure-track leaders of independent research groups. The successful candidates will receive a substantial annual research budget but are expected to also apply for external research grants.

For further information and access to the online application material, please consult:

www.ist.ac.at/professor-applications

Deadline for receiving Assistant Professor applications: **January 15, 2012**

IST Austria values diversity and is committed to equality. Female researchers are encouraged to apply.





Lane Department of Computer Science & Electrical Engineering College of Engineering & Mineral Resources

Tenure-Track Position in Cyber Security and Information Assurance

The Lane Department of Computer Science and Electrical Engineering (LCSEE) invites applications for a tenure track faculty position in cyber security and information assurance. The position is open at the assistant or associate professor level. Areas of interest include, but are not limited to applications, systems, and network security; wireless and mobile security; digital forensics; secure programming; and software and information assurance. An earned Ph.D. in computer science or a closely related discipline is required. Successful candidates are expected to develop a vigorous extramurally funded research program, build effective collaborations, and demonstrate commitment to teaching excellence. The rank of initial appointment will be commensurate with experience and accomplishments.

West Virginia University (www.wvu.edu) is a comprehensive land grant research institution enrolling 30,000 students in 113 degrees programs, including engineering and health sciences. WVU has a Carnegie classification of Research – High. The Lane Department (www.csee.wvu.edu) has 35 tenure-track faculty members, 350 undergraduate students, and 250 graduate students. It offers BS degrees in Computer Science, Computer Engineering, Electrical Engineering, and Biometric Systems; MS degrees in Computer Science, Software Engineering, and Electrical Engineering; and Ph.D. degrees in Computer Science, Computer Engineering and Electrical Engineering. It also offers Graduate Certificates in Computer Forensics, Biometrics and Information Assurance, and Software Engineering. In addition, WVU is recognized by the NSA/DHS as a Center for Academic Excellence in Information Assurance Education, as well as a Center for Academic Excellence in Information Assurance Research. The Department conducts approximately \$6 million annually in externally sponsored research, with major research activities in the areas of biometric identification, nanotechnology, power systems, software and systems engineering, and wireless networks. Strong opportunities exist for building collaborative partnerships with nearby federal research facilities, including the Department of Defense, Department of Energy, FBI, and NASA.

To apply for this position, interested candidates should submit a letter of application, curriculum vitae, research statement, statement of teaching philosophy, and contact information for at least three technical references (as a single PDF document) to: **CEMR-LCSEE-Search@mail.wvu.edu**. Please include "Cyber security" as a subject line. Applications will be processed until the position is filled but those received by January 10, 2012 will receive full consideration. For further information, contact Dr. Katerina Goseva-Popstojanova, Search Chair, at katerina.goseva@mail.wvu.edu (queries only).

Tenure-Track Position in Biomedical Imaging

The Lane Department of Computer Science and Electrical Engineering (LCSEE) invites applications for a tenure track faculty position in the area of biomedical imaging. The position is open at the assistant or associate professor level. Areas of interest include, but are not limited to imaging with application to medical systems, visualization and graphics, medical informatics, data fusion, feature extraction, network biology, and biomedical applications of artificial intelligence. An earned Ph.D. in computer science or a closely related discipline is required. Successful candidates are expected to develop a vigorous extramurally funded research program, build effective collaborations, and demonstrate commitment to teaching excellence. The successful candidate should be qualified to teach core undergraduate courses in the field of Computer Science such as programming languages, operating systems, and computer organization, in addition to developing courses in their area of expertise. The rank of initial appointment will be commensurate with experience and accomplishments.

West Virginia University (www.wvu.edu) is a comprehensive land grant research institution enrolling 30,000 students in 113 degrees programs, including engineering and health sciences. WVU has a Carnegie classification of Research – High. The Lane Department (www.csee.wvu.edu) has 35 tenure-track faculty members, 350 undergraduate students, and 250 graduate students. It offers BS degrees in Computer Science, Computer Engineering, Electrical Engineering, and Biometric Systems; MS degrees in Computer Science, Software Engineering, and Electrical Engineering; and Ph.D. degrees in Computer Science, Computer Engineering and Electrical Engineering. The Robert C. Byrd Health Science Center at WVU has a comprehensive medical school and offers opportunities for collaboration with nationally recognized Centers in Neuroscience, Advanced Imaging, Cancer Research, Low Vision Research, and Rural Medicine. The Department conducts approximately \$6 million annually in externally sponsored research, with major research activities in the areas of biometric identification, nanotechnology, power systems, software and systems engineering, and wireless networks. Strong opportunities exist for building collaborative partnerships with nearby federal research facilities, including the Department of Energy, DOD, FBI, and NASA.

To apply for this position, interested candidates should submit a letter of application, curriculum vitae, research statement, statement of teaching philosophy, and contact information for at least three technical references (as a single PDF document) to **CEMR-LCSEE-Search@mail.wvu.edu**. Please include "Biomedical Imaging" as a subject line. Applications will be processed until the position is filled but those received by Jan. 10, 2012 will receive full consideration. For further information, contact Dr. Don Adjeroh, Search Chair, at Donald.adjeroh@mail.wvu.edu (queries only).

West Virginia University is an affirmative action, equal opportunity employer dedicated to building a culturally diverse and pluralistic faculty and staff committed to teaching and working in a multicultural environment. West Virginia University is the Recipient of an NSF ADVANCE Award for Gender Equity. Applications are strongly encouraged from women, minorities, individuals with disabilities and covered veterans. Dual career couples are also encouraged to apply.

Columbia University
Department of Computer Science
Open-rank Faculty Position
in Computer Science

Columbia Engineering's Department of Computer Science at Columbia University in New York City invites applications for tenured or tenure-track faculty positions. One or more appointments at the assistant professor, associate professor and full professor, will be considered.

Columbia Engineering's strategic theme areas are Health, Information and Sustainability, and the successful candidate should contribute to the advancement of the department in these areas by developing an externally funded research program, being a thought leader in the profession, contributing to the undergraduate and graduate educational mission of the Department and providing active service to professional societies. The successful candidate is expected to establish multidisciplinary research and educational collaborations with academic departments and units across Columbia University. The Department is especially interested in qualified candidates who can contribute, through their research, teaching, and/or service, to the diversity and excellence of the academic community.

Applications are specifically sought in any of the areas that fall under the umbrella of Computer Systems, Software, Artificial Intelligence, Theory, and Computational Biology, with particular emphasis on, but not limited to: Computer Graphics, Human-Computer Interaction, Simula-

tion, and Animation. Applicants whose work contributes to CyberBioPhysical Systems, where the biological, physical, and digital worlds fuse, are also encouraged to apply.

Candidates must have a Ph.D. or its professional equivalent by the starting date of the appointment. Applicants for this position at the Assistant Professor and Associate Professors without tenure must have the potential to do pioneering research and to teach effectively. Applicants for this position at the tenured level (Associate or Full Professor) must have a demonstrated record of outstanding research accomplishments, excellent teaching credentials and established leadership in the field.

Candidates should apply online at:
academicjobs.columbia.edu/applicants/Central?quickFind=55402

and should submit electronically the following: curriculum-vitae including a publication list, a description of research accomplishments, a statement of research/teaching interests and plans, contact information for three people who can provide letters of recommendation, and up to three pre/reprints of scholarly work. The position will close no sooner than 12/31/2011, and will remain open until filled.

Applicants can consult www.cs.columbia.edu for more information about the department.

Columbia is an affirmative action/
equal opportunity employer with a strong
commitment to the quality of faculty life.

Cornell University
Assistant Professor

The SIBLEY SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING, CORNELL UNIVERSITY, Ithaca, NY, invites applications for a faculty position in DIGITAL MANUFACTURING and DIGITAL DESIGN. Areas of particular interest include additive manufacturing processes, models, and materials; on-demand, autonomous, flexible manufacturing; programmable/smart/electronic materials; energy-efficient manufacturing; bio-printing, micro-manufacturing; computational design automation and optimization. Details at, <http://www.mae.cornell.edu/>.

Candidates must hold a doctorate in an appropriate field and are expected to establish an outstanding, funded research program as well as contribute fully to both undergraduate and graduate instruction.

Cornell University is an equal opportunity, affirmative action educator and employer.

Dartmouth College
Department of Computer Science
Roth Family Distinguished Professorship

The Department of Computer Science at Dartmouth College invites applications for the inaugural Roth Family Distinguished Professorship. We seek candidates with a strong academic or industry track record in the general area of Digital Arts (including, but not limited to, Computer Graphics, Computer Vision, Visualization, Human-Computer Interaction, Design & Media Arts). Candidates at the level of full professor or senior-level associate professor will be considered.

Dartmouth is home to a growing program in the Digital Arts with affiliated faculty and students in Computer Science, English, Film & Media Studies, Mathematics, Music, Psychology, Studio Art, and Theater. In the coming years we expect continued investment in the Digital Arts. The Roth Family Distinguished Professor will help shape what is expected to be a leading undergraduate and graduate program in Digital Arts.

The Computer Science department (www.cs.dartmouth.edu) is home to 17 tenured and tenure-track faculty members whose research encompasses the areas of digital arts, graphics, vision, algorithms, theory, systems, security, robotics, and computational biology. The Computer Science department is in the School of Arts and Sciences, and it has strong Ph.D. and M.S. programs and outstanding undergraduate majors. The department is affiliated with Dartmouth's M.D.-Ph.D. program and has strong collaborations with the Tuck School of Business and Dartmouth Medical School.

Dartmouth College is located in Hanover, New Hampshire. Dartmouth has a beautiful, historic campus, located in a scenic area on the Connecticut River. Recreational opportunities abound in all four seasons. Dartmouth hosts an annual film festival, as well as renowned musical and theatrical performers. Convenient public transportation to Boston and New York is available. Airports with commercial service are located 15 minutes away (Lebanon, New Hampshire) and 75 minutes away (Manchester-Boston Regional Airport).

Applicants are invited to send their CV, re-



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact:

acmm mediasales@acm.org

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at:

<http://jobs.acm.org>

Ads are listed for a period of 30 days.

For More Information Contact:

**ACM Media Sales
at 212-626-0686 or
acmm mediasales@acm.org**



Leslie Pack Kaelbling

MIT's Professor of Computer Science and Engineering Research Director of the Computer Science and Artificial Intelligence Laboratory (CSAIL) Coordinator of SUTD's curriculum development for Information Systems Technology & Design Pillar

INSPIRE THE NEXT GENERATION TO BUILD A BETTER WORLD.

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 multi-disciplinary curriculum, has a mission to advance knowledge and nurture technically grounded leaders and innovators to serve societal needs. SUTD is characterized 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. SUTD invites applicants for tenure-track or tenured appointments in all areas of computer science and information technology, with particular interest in candidates with expertise in operating systems, databases, networking, security, cryptography, and applied algorithms. 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

A BETTER WORLD BY DESIGN.



SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Established in collaboration with MIT

search statement, teaching statement, and names of at least four references. All material and inquiries should be sent to roth2012@cs.dartmouth.edu. Application review will start on January 15, 2011.

Applicants should arrange to have letters of recommendation sent directly by the recommender, either by email or physical mail, to:

Roth Family Professorship Search
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755
roth2012@cs.dartmouth.edu

Dartmouth College is an equal opportunity/affirmative action employer and encourages applications from women and members of minority groups.

Dartmouth College **Assistant Professor**

We invite applications for tenure-track faculty assistant professor of Computer Science, in the areas of systems, computer architecture (especially with applications to computer graphics), networking, or human-computer interaction. For detailed information, see www.cs.dartmouth.edu.

Florida International University (FIU) **School of Computing and Information Sciences** **Multiple tenure-track & tenured faculty** **Positions at all levels**

FIU is a multi-campus public research university located in Miami, a vibrant, international city. FIU offers more than 180 baccalaureate, masters, professional and doctoral degree programs to over 42,000 students. As one of South Florida's anchor institutions, FIU is worlds ahead in its local and global engagement and is committed to finding solutions to the most challenging problems of our times.

The School of Computing and Information Sciences seeks exceptionally qualified candidates for multiple tenure-track and tenured faculty positions at all levels. Outstanding candidates are sought in areas of bio/medical/health informatics, computer architecture, computer graphics, large-scale data management, search, and visualization, human-computer interaction (HCI), networking, programming languages, robotics and game theory, and telecommunication. Exceptional candidates in other areas will be considered as well. Preference will be given to candidates who will enhance or complement our existing research.

Ideal candidates for junior positions should have a record of exceptional research in their early careers. Candidates for senior positions must have an active and proven record of excellence in funded research, publications, and professional service, as well as a demonstrated ability to develop and lead collaborative research projects. In addition to developing or expanding a high-quality research program, all successful applicants must be committed to excellence in teaching at both graduate and undergraduate levels. An earned Ph.D. in Computer Science or related disciplines is required.

Florida International University (FIU), the state university of Florida in Miami, is ranked by the Carnegie Foundation as a comprehensive doctoral research university with high research activity. The School of Computing and Information Sciences (SCIS) is a rapidly growing program of excellence at the University, with 31 faculty members and 1,400 students, including 65 Ph.D. students. SCIS offers B.S., M.S., and Ph.D. degrees in Computer Science, an M.S. degree in Telecommunications and Networking, and B.S., B.A., and M.S. degrees in Information Technology. SCIS has received approximately \$12.6M in the last three years in external research funding, has six research centers/clusters with first-class computing infrastructure and support, and enjoys broad and dynamic industry and international partnerships.

HOW TO APPLY:

Applications, including a letter of interest, contact information, curriculum vitae, and the names of at least three references, should be submitted directly to the FIU J.O.B.S Link **website at <https://www.fiujobs.org>; refer to Position # 33334**. The application review process will begin on January 16, 2012, and will continue until the position is filled. Further information can be obtained from the School website <http://www.cis.fiu.edu>, or by e-mail to recruit@cis.fiu.edu.

FIU is a member of the State University System of Florida and is an Equal Opportunity, Equal Access Affirmative Action Employer.

Florida State University **Assistant Professor**

Tenure-Track Assistant Professor Positions

The Department of Computer Science at the Florida State University invites applications for multiple tenure-track Assistant Professor positions to begin August 15, 2012. Positions are 9-mo, full-time, tenure-track, and benefits eligible. We encourage strong applicants in all areas of Computer Science to apply. Preference may be given to applicants with research experience in the areas of Databases and Security. Applicants should hold a PhD in Computer Science or closely related field, and have excellent research and teaching accomplishments/potential. The department offers degrees at the BS, MS, and PhD levels. The department is an NSA Center of Academic Excellence in Information Assurance Education (CAE/IAE) and Research (CAE-R).

FSU is classified as a Carnegie Research I university. Its primary role is to serve as a center for advanced graduate and professional studies while emphasizing research and providing excellence in undergraduate education. The department has experienced rapid growth in the major and new degree programs. Further information can be found at

<http://www.cs.fsu.edu>

Screening will begin January 1, 2012 and will continue until the positions are filled. Please apply online with curriculum vitae, statements of teaching and research philosophy, and the names of five references, at <http://www.cs.fsu.edu/positions/apply.html>

Questions can be e-mailed to Prof. Mike Bur-

mester, Chair Search Committee, recruitment@cs.fsu.edu or to Prof. Robert van Engelen, Department Chair, chair@cs.fsu.edu.

The Florida State University is a Public Records Agency and an Equal Opportunity/Access/Affirmative Action employer, committed to diversity in hiring.

George Mason University **Department of Computer Science** **Assistant Professor**

The Department of Computer Science at George Mason University invites applications for a tenure-track faculty position at the rank of Assistant Professor beginning Fall 2012.

We are seeking a faculty member who can establish strong research and teaching programs in the area of computer game design. Applicants must have a research focus in an area of computer games technology - for example, in artificial intelligence, multi-agent systems, computer graphics, real-time animation, simulation and modeling, networked and distributed systems, or software engineering, as applied to computer games. Minimum qualifications include a Ph.D. in Computer Science or a related field, demonstrated potential for excellence and productivity in research, and a commitment to high quality teaching.

The department currently offers a graduate certificate in Computer Games Technology and a concentration in Computer Game Design at the undergraduate level. The Computer Game Design concentration is offered in collaboration with faculty in the College of Visual and Performing Arts at Mason. For more information on these and other programs offered by the department, visit our Web site: <http://cs.gmu.edu/>

The department has over 40 faculty members with wide-ranging research interests including artificial intelligence, algorithms, computer graphics, computer vision, databases, data mining, distributed virtual environments, expert systems, human computer interaction, parallel and distributed systems, real-time systems, robotics, security, software engineering, and wireless and mobile computing.

George Mason University is located in Fairfax, Virginia, a suburb of Washington, DC, and home to one of the highest concentrations of high-tech firms in the nation. There are excellent opportunities for interaction with government agencies and industry, including many game and serious game development companies. In particular, the Washington DC region is fast becoming a hub for the serious games industry. Fairfax is consistently rated as being among the best places to live in the country, and has an outstanding local public school system.

For full consideration please submit application and application materials on-line at <http://jobs.gmu.edu> (position number F9542Z). To apply, you will need a statement of professional goals including your perspective on teaching and research, a complete C.V. with publications, and the names of four references. The review of applications will begin immediately and will continue until the position is filled. GMU is an equal opportunity/affirmative action employer. Women and minorities are strongly encouraged to apply.

The Institute for Interdisciplinary Information Sciences (IIIS) **Tenure-track Assistant/Associate/Full Professor**

IIIS invites applications from highly-qualified candidates in areas including (but not limited to) Computer Systems, Algorithms and Complexity, Machine Learning, Multimedia, Databases, Computer Networks, Wireless Sensor Networks, Information Security, Web Technologies, Energy-Efficient Computing, Computational Finance, Quantum Information, Computational Biology. Positions at Assistant/Associate/Full Professor levels are available.

Apply for this job:

Email: iiisdean@mail.tsinghua.edu.cn

Tel: +86-01-62789157

Iowa State University **Software Engineering Program** **Tenure-track or tenured faculty position**

The **Software Engineering Program at Iowa State University**, Ames, IA, has an immediate opening for a **tenure-track or tenured faculty position** that will commence in August 2012. Appointments will be considered at all experience levels.

Duties for the position will include undergraduate and graduate education; mentoring and engaging undergraduate as well as prospective students; developing and sustaining externally-funded research; graduate student supervision and mentoring; and professional and institutional service.

An earned Ph.D. or equivalent in software engineering, computer science, computer engineering or a closely related field is required. For appointment at the level of assistant professor, the successful candidate must have demonstrated potential to establish and maintain a productive externally funded research program and potential to excel in the classroom. Commensurate experience and a proven track record will be expected for appointment at a more senior level. The tenure home in either the Department of Computer Science or the Department of Electrical and Computer Engineering will be decided in consultation with the successful candidate, with joint appointment in both departments.

Apply for this job:

Contact Person: Sara K. Harris

Email Address: skharris@iastate.edu

Phone: 515-294-1097

Fax: 515-294-3637

Apply URL: <http://www.se.iastate.edu/careers/faculty-staff-openings/faculty-position>

Candidates are subject to a background check. ISU is an EO/AA employer.

King Abdullah University of Science and Technology **Postdoc, Computer Science**

The InfoCloud group at KAUST invites applications for PostDoc positions in all areas of Databases, Data Mining, Cloud Computing, Parallel/Distributed Systems and High-performance Computing. The positions are for 1 to 3 years. An ideal candidate should have (or be expecting soon) a PhD in Computer Science or a related

field, with strong publication record.

KAUST offers: Very attractive salary and benefits; generous research funding; state-of-the-art research facilities, including one of the fastest supercomputers in the world; collaboration with top institutions such as Stanford, Texas A&M, IBM Watson, etc.

KAUST is an international graduate-only research university located on the coast of the Red Sea, near Jeddah, Saudi Arabia. All activities of the University are conducted on the basis of equality, without regard to race, color, religion or gender.

Further information can be found at: <http://cloud.kaust.edu.sa>

For enquiries, please contact Dr. Panos Kalnis: panos.kalnis@kaust.edu.sa

Lawrence Technological University **Assistant Professor of Computer Science**

For appointment in August 2012. The ideal candidate will have a Ph.D. degree in computer science, will have experience with intelligent robotic systems, be primarily committed to the development of undergraduate and professional graduate computer science students through teaching, applied projects and scholarship, be able to work effectively in interdisciplinary teams, and believe strongly in the value of both theory and application. Applicants should email a cover letter, curriculum vitae, statement of teaching philosophy and research interests, and three letters of recommendation. Computer Science Search Committee; cssearch@ltu.edu

Marist College **Lecturer, Assistant or Associate Professor of Computing Technology**

Marist College's School of Computer Science and Mathematics invites applications for two faculty positions. Marist College is a highly selective, independent, liberal arts institution located in the historic Hudson River Valley, 60 miles north of New York City. Marist currently enrolls 4,200 traditional undergraduate, 950 graduate and 530 continuing education students. The College has been recognized for excellence by U.S. News & World Report, The Princeton Review, Entrepreneur Magazine, and is noted for its leadership in the use of technology to enhance the teaching and learning process.

PhD in CS, IT, IS or closely related field preferred; Master's degrees with significant industry experience will be considered. Candidates with expertise in software development, security, applied networking, business analytics, and management information systems is highly desirable.

To learn more or to apply, please visit <http://jobs.marist.edu>. Only online applications are accepted.

AN EQUAL OPPORTUNITY/AFFIRMATIVE ACTION EMPLOYER

Massachusetts Institute of Technology **Faculty Positions**

The Department of Electrical Engineering and Computer Science (EECS) seeks candidates for faculty positions starting in September 2012. Ap-

pointment will be at the assistant or untenured associate professor level. In special cases, a senior faculty appointment may be possible. Faculty duties include teaching at the graduate and undergraduate levels, research, and supervision of student research. We will consider candidates with backgrounds and interests in any area of electrical engineering and computer science. Faculty appointments will commence after completion of a doctoral degree.

Candidates must register with the EECS search website at <https://eeecs-search.eecs.mit.edu>, and must submit application materials electronically to this website. Candidate applications should include a description of professional interests and goals in both teaching and research. Each application should include a curriculum vita and the names and addresses of three or more individuals who will provide letters of recommendation. Letter writers should submit their letters directly to MIT, preferably on the website or by mailing to the address below. Please submit a complete application by December 15, 2011.

Send all materials not submitted on the website to:

Professor Anantha Chandrakasan
Department Head, Electrical Engineering
and Computer Science
Massachusetts Institute of Technology
Room 38-401
77 Massachusetts Avenue
Cambridge, MA 02139

M.I.T. is an equal opportunity/affirmative action employer.

Max Planck Institute for Software Systems (MPI-SWS) **Tenure-track openings**

Applications are invited for tenure-track and tenured faculty positions in all areas related to the study, design, and engineering of software systems. These areas include, but are not limited to, data and information management, programming systems, software verification, parallel, distributed and networked systems, and embedded systems, as well as cross-cutting areas like security, machine learning, usability, and social aspects of software systems. A doctoral degree in computer science or related areas and an outstanding research record are required. Successful candidates are expected to build a team and pursue a highly visible research agenda, both independently and in collaboration with other groups. Senior candidates must have demonstrated leadership abilities and recognized international stature.

MPI-SWS, founded in 2005, is part of a network of eighty Max Planck Institutes, Germany's premier basic research facilities. MPIs have an established record of world class, foundational research in the fields of medicine, biology, chemistry, physics, technology and humanities. Since 1948, MPI researchers have won 17 Nobel prizes. MPI-SWS aspires to meet the highest standards of excellence and international recognition with its research in software systems.

To this end, the institute offers a unique environment that combines the best aspects of a university department and a research laboratory:

a) Faculty receive generous base funding to build and lead a team of graduate students and

post-docs. They have full academic freedom and publish their research results freely.

b) Faculty supervise doctoral theses, and have the opportunity to teach graduate and undergraduate courses.

c) Faculty are provided with outstanding technical and administrative support facilities as well as internationally competitive compensation packages.

MPI-SWS currently has 11 tenured and tenure-track faculty, and is funded to support 17 faculty and about 100 doctoral and post-doctoral positions. Additional growth through outside funding is possible. We maintain an open, international and diverse work environment and seek applications from outstanding researchers regardless of national origin or citizenship. The working language is English; knowledge of the German language is not required for a successful career at the institute.

The institute is located in Kaiserslautern and Saarbrücken, in the tri-border area of Germany, France and Luxembourg. The area offers a high standard of living, beautiful surroundings and easy access to major metropolitan areas in the center of Europe, as well as a stimulating, competitive and collaborative work environment. In immediate proximity are the MPI for Informatics, Saarland University, the Technical University of Kaiserslautern, the German Center for Artificial Intelligence (DFKI), and the Fraunhofer Institutes for Experimental Software Engineering and for Industrial Mathematics.

Qualified candidates should apply online at <http://www.mpi-sws.org/application>. The review of applications will begin on January 3, 2012, and applicants are strongly encouraged to apply by that date; however, applications will continue to be accepted through January 2012.

The institute is committed to increasing the representation of minorities, women and individuals with physical disabilities in Computer Science. We particularly encourage such individuals to apply.

North Dakota State University **Assistant/Associate Professor**

Department of Computer Science at North Dakota State University seeks to fill a tenure-track Assistant Professor position starting Fall of 2012. PhD required. NDSU offers degrees in Computer Science and Software Engineering. Research and teaching excellence is expected. Normal teaching loads are 3 courses per year. Salary is competitive.

The Department has 18 Faculty in diverse areas, approximately 180 graduate (PhD and MS) students and 240 BS/BA students. NDSU is a Carnegie research extensive class institution.

Fargo is a clean, growing metropolitan area of 250,000 that consistently ranks very high in national quality-of-life surveys. We have low levels of crime and pollution, excellent schools, short commutes and proximity to the Minnesota lakes country. The community has a symphony, an opera, a domed stadium, a community theater, three universities and many other opportunities for advancement, recreation and entertainment.

Minimum Qualifications:

Ph.D in Computer Science or a closely related area. Experience or other evidence of potential

for excellence in research, undergraduate and graduate teaching and service. Effective oral and written communication skills.

Qualifications preferred but not required:

Experience or other evidence of potential for excellence in research in high performance computing, mobile and distributed computing. Security, networks or a related area. The capability for teaching a variety of courses in computer science.

To Apply: see <https://jobs.ndsu.edu/>

For additional information see <http://cs.ndsu.edu/positions.htm>

Review of applications begins January 15, 2012. Applications will be accepted until position is filled.

North Dakota State University is an Equal Opportunity/AA employer.

This position is exempt from North Dakota Veterans' Preference requirements.

Northeastern Illinois University **Computer Science Department** **Assistant Professor**

The Computer Science Department of Northeastern Illinois University in Chicago invites interested individuals to apply for a tenure-track assistant professor position starting Fall 2012. PhD in Computer Science or closely related field required. Preference will be given to candidates with interests in bioinformatics, software engineering, computer networks and security, and emerging technologies. Strong candidates in other areas will be considered. AA/EOE View complete job posting at: http://www.neiu.edu/~compsec/cs_faculty_search.html

Peking University **School of EECS** **Tenure-track Faculty Positions - Center for Energy-Efficient Computing and Applications**

The School of EECS at Peking University invites applications for tenure-track positions in the areas of energy efficient computing (including but not limited to energy-efficient architectures, compilation, communication, and system software) and applications (such as mobile computing, wireless health, and cloud computing). These positions are associated with the Center for Energy-Efficient Computing and Applications (<http://ceca.pku.edu.cn>), which offers a new level of startup and compensation packages. Applications from distinguished candidates at senior levels are also encouraged. To apply, please email the resume, statements of research and teaching, and at least three names for references to Dr. Tao Wang (ceca_recruiting@pku.edu.cn). Applications received by January 20, 2012 will be given full consideration.

The Pennsylvania State University **Faculty Position**

Applications are invited for several tenure-track faculty positions at all ranks. Outstanding candidates in all areas of Computer Science and Engineering will be considered. Areas of particular

interest are: software systems with an emphasis on operating systems; machine learning with an emphasis on cross-disciplinary research and applications; and programming languages with an emphasis on parallel languages and parallel processing. For all of these fields there is special interest in interdisciplinary work such as cyber-physical systems and predictive computational modeling and simulation.

The department has 32 tenure-track faculty representing major areas of computer science and engineering. Fifteen members of our faculty are recipients of the NSF Career Award. Two faculty members have received the prestigious NSF PECASE Award. In recent years, our faculty received seven NSF ITR Grants, a \$35M Network Science Center Award, over \$4.5M in computing and research infrastructure and instrumentation grants from NSF, eleven NSF Cyber Trust and Networking awards, and several awards from DARPA, DOE, DTRA and DoD. There are state-of-the-art research labs for computer systems, computer vision and robotics, Microsystems design and VLSI, networking and security, high performance computing, bioinformatics and virtual environments. The department offers a graduate program with over 40 Masters students and 167 Ph.D. students, and undergraduate programs in computer science and computer engineering. The university is committed to growing the faculty ranks over the next several years and promoting interdisciplinary research toward cyber-enabled discovery and design.

Penn State is a major research university and is ranked 3rd in the nation in industry-sponsored research. Computer science is ranked 6th in the nation in research expenditures. *U.S. News and World Report* consistently ranks Penn State's College of Engineering undergraduate and graduate programs in the top 15 of the nation.

The university is located in the beautiful college town of State College in the center of Pennsylvania. State College has 40,000 inhabitants and offers a variety of cultural and outdoor recreational activities nearby. The university offers outstanding events from collegiate sporting events to fine arts productions. Many major population centers on the east coast (New York, Philadelphia, Pittsburgh, Washington D.C., Baltimore) are only a few hours drive away and convenient air services to several major hubs are operated by four major airlines out of State College.

Applicants should hold a Ph.D. in Computer Science, Computer Engineering, or a closely related field and should be committed to excellence in both research and teaching. Support will be provided to the successful applicants for establishing their research programs. We encourage dual career couples to apply. Applications should be received by January 31, 2012 to receive full consideration. To apply by electronic mail, send your resume (including curriculum vitae and the names and addresses of at least three references) as a pdf file to recruiting@cse.psu.edu.

Chair, Faculty Search Committee
The Pennsylvania State University
Department of Computer Science
and Engineering
Box A
University Park, PA 16802-6106 USA

For more information about the Department of CSE at PSU, see <http://www.cse.psu.edu>. Click here to fill out an Affirmative Action Applicant

Data Card. Our search number is **015-92**. You **MUST** include this search number in order to submit this form.

Penn State is committed to affirmative action, equal opportunity and the diversity of its workforce.

Princeton University
Computer Science Department
Tenure-Track Positions, Assistant Professor

The Department of Computer Science at Princeton University invites applications for faculty positions at the Assistant Professor level. We are accepting applications in all areas of Computer Science.

Applicants must demonstrate superior research and scholarship potential as well as teaching ability. A PhD in Computer Science or a related area is required. Successful candidates are expected to pursue an active research program and to contribute significantly to the teaching programs of the department. Applicants should include a CV and contact information for at least three people who can comment on the applicant's professional qualifications.

There is no deadline, but review of applications will start in December 2011. Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. You may apply online at: <http://jobs.cs.princeton.edu/>. Requisition Number: 0110422

Princeton University
Computer Science Department
PostDoctoral Research Associate

The Department of Computer Science at Princeton University is seeking applications for postdoctoral or more senior research positions in theoretical computer science. Candidates will be affiliated with the Center for Computational Intractability (CCI) or the Princeton Center for Theoretical Computer Science. Candidates should have a PhD in Computer Science, a related field, or on track to finish by August 2012.

Candidates affiliated with the CCI will have visiting privileges at partner institutions NYU, Rutgers University, and The Institute for Advanced Study. Review of candidates will begin on Jan 1, 2012, and will continue until positions are filled. Applicants should submit a CV and research statement, and contact information for three references. Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. Apply to: <http://jobs.princeton.edu/> Requisition# 0110698

Purdue University
Computer Science Department
Faculty Position

The Department of Computer Science at Purdue University invites applications for tenure-track positions at the assistant professor level beginning August 2012. Outstanding candidates in all areas of Computer Science and with a multi-disciplinary focus are encouraged to apply. Specific needs that have been identified include theory and software engineering.

The Department of Computer Science offers a stimulating and nurturing academic environment.

Forty-eight faculty members direct research programs in analysis of algorithms, bioinformatics, databases, distributed and parallel computing, graphics and visualization, information security, machine learning, networking, programming languages and compilers, scientific computing, and software engineering. Information about the department and a detailed description of the open position are available at <http://www.cs.purdue.edu>.

All applicants should hold a PhD in Computer Science, or a closely related discipline, be committed to excellence in teaching, and have demonstrated potential for excellence in research. The successful candidate will be expected to teach courses in computer science, conduct research in field of expertise and participate in other department and university activities. Salary and benefits are highly competitive. Applicants are strongly encouraged to apply online at <https://hiring.science.purdue.edu>. Hard copy applications can be sent to: Faculty Search Chair, Department of Computer Science, 305 N. University Street, Purdue University, West Lafayette, IN 47907. Review of applications will begin on November 10, 2011, and will continue until the position is filled. A background check will be required for employment in this position. Purdue University is an Equal Opportunity/Equal Access/Affirmative Action employer fully committed to achieving a diverse workforce.

Rutgers, The State University of New Jersey
Assistant Professor

The Department of Management Science and Information Systems of Rutgers Business School-Newark and New Brunswick invites applications for a tenure-track position at the Assistant Professor rank to start in September 2012.

This position is focused in the area of information systems and the candidate must be an active researcher and have a strong record of scholarly excellence. Special consideration will be given to candidates with knowledge in any of the areas: data mining, machine learning, security, data management and analytical methods related to business operations.

A letter of application articulating the candidate's fit (in terms of research and teaching) with the position description, a curriculum vitae, and the names and contact information of three persons that can provide references should be sent electronically to Luz Kosar at: kosar@business.rutgers.edu.

Luz Kosar,
 MSIS
 Rutgers Business School -
 Newark and New Brunswick
 1 Washington Park # 1068
 Newark, New Jersey 07102-1895

Rutgers University
Department of Computer Science and the Center for Computational Biomedicine, Imaging and Modeling (CBIM)
Tenure Track Faculty Position in Robotics

The Rutgers University Department of Computer Science and the Center for Computational Biomedicine, Imaging and Modeling (CBIM) seeks applicants at in robotics, for a tenure-track faculty position starting September 2012. We're

particularly interested in synergy with CBIM and thus we're excited about receiving applications primarily in areas related to multimodal sensing, decision making under uncertainty, planning, learning and novel designs for collaborative robots, co-robots, social robots, network-based robotics, underwater autonomous robots and heterogeneous swarms of robots. Rutgers University offers an exciting and multidisciplinary research environment and encourages collaborations between Computer Science and other disciplines.

Applicants for this research/teaching position must, at minimum, be in the process of completing a dissertation in Computer Science or a closely related field, and should show evidence of exceptional research promise, potential for developing an externally funded research program, and commitment to quality advising and teaching at the graduate and undergraduate levels. Hired candidates who have not defended their Ph.D. by September 2012 will be hired at the rank of Instructor, and must complete the Ph.D. by December 31, 2012 to be eligible for tenure-track title retroactive to start date. Senior applicants at the Associate or Full Professor level will need to have demonstrated significant funding, scholarship, collaborative, and leadership abilities.

Applicants should go to <http://www.cs.rutgers.edu/employment/> and submit their curriculum vitae, a research statement addressing both past work and future plans and a teaching statement along with three letters of recommendation. If electronic submission is not possible, hard copies of the application materials may be sent to:

Professor Dimitris Metaxas, Hiring Chair
 Computer Science Department
 Rutgers University
 110 Frelinghuysen Road
 Piscataway, NJ 08854

Applications should be received by January 31st, 2012 for full consideration.

Rutgers subscribes to the value of academic diversity and encourages applications from individuals with varied experiences, perspectives, and backgrounds. Females, minorities, dual-career couples, and persons with disabilities are encouraged to apply.

Rutgers is an affirmative action/equal opportunity employer.

State University of New York at Binghamton
Department of Computer Science

Applications are invited for a tenure-track Assistant Professor starting Fall 2012. Our preferred specializations are embedded systems, energy-aware computing and systems development. We have well-established BS (accredited), MS and PhD programs, with over 60 full-time PhD students. We offer a significantly reduced teaching load for junior faculty for at least the first three years. A new NSF supported industry-university collaborative research center on energy-efficient electronic systems offers an added venue for research and funding. Please submit a resume and the names of three references at: <http://binghamton.interviewexchange.com>

First consideration will be given to applications received by January 10, 2012.

We are an EE/AA employer.

Swarthmore College
Computer Science Department
Tenure Track and Two Year Leave Position

Applications are invited for a tenure track assistant professor position and for a two-year leave replacement position at the assistant professor level. Both positions begin Fall semester 2012. Swarthmore College is a small, selective liberal arts college located in a suburb of Philadelphia. The Computer Science Department offers majors and minors in computer science at the undergraduate level. The teaching load is 3 courses and 3 labs per year. Applicants must have teaching experience and should be comfortable teaching a wide range of courses at the introductory and intermediate level.

For the tenure track position, all areas of CS will be considered, but we are particularly interested in areas that complement our current offerings, including databases, networking, and other systems and algorithms areas. Candidates should additionally have a strong commitment to involving undergraduates in their research. A Ph.D. in CS by or near the time of appointment is required.

For the leave replacement position, all areas of CS will be considered. A Ph.D. in CS by or near the time of appointment is preferred (ABD required).

See <http://cs.swarthmore.edu/jobs> for application submission information and more details about both positions. We expect to begin interviewing by mid January 2012. Applications will be accepted until the positions are filled.

Swarthmore College has a strong institutional commitment to excellence through diversity in its educational program and employment practices and actively seeks and welcomes applications from candidates with exceptional qualifications, particularly those with demonstrable commitments to a more inclusive society and world.

Toyota Technological Institute Chicago
Computer Science at TTI Chicago
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. Applications are being accepted in all areas, but we are particularly interested in machine learning, speech processing, computational linguistics, Computer vision, computational biology and optimization. Positions are available at all ranks, and we have a large number of three year 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*

Trinity University
Assistant Professor

Trinity University seeks applications for a tenure-track assistant professor position in the Department of Computer Science, to commence August 2012.

Candidates must have a Ph.D. in Computer Science or a related area. Our new colleague will teach introductory and advanced courses in CS, advise first-year students and majors, and be willing to include undergraduates in research activities. He/she will participate in ongoing development of the CS curriculum and will be active in research and professional activities. The teaching load is 9 student contact hours (3 courses) per semester.

Trinity is an independent, coeducational, primarily undergraduate and residential university founded in 1869. Enrollment is approximately 2400 from all areas of the US and many foreign countries. Trinity has highly selective admission standards and occupies an attractive campus overlooking downtown San Antonio, a city rich in heritage and ethnic diversity. Trinity's new Entrepreneurship program is housed in our Department.

Perhaps unexpectedly, San Antonio has a particularly lively computing community! Indeed, San Antonio is second in the nation (only to the Washington, D.C. area) in cyber-security. While the position is open to all areas, a new colleague would find San Antonio to be a particularly good fit for information assurance & security, bioinformatics, or cloud computing.

The Department, 42 years old, is well integrated into campus life. Our students are among the best in the University and secure excellent offers for jobs and graduate schools. Late in 2013 the Department is scheduled to move into a new Center for Science and Innovation, housing all the STEM departments. Additional information may be found at

<http://web.trinity.edu/x605.xml>
<http://web.trinity.edu/x13585.xml>
<http://web.trinity.edu/>

Please send a letter of application; CV; teaching and research interests; and names and contact information of at least three references to Paul Myers, Chair
Department of Computer Science
Trinity University
1 Trinity Place
San Antonio, Texas 78212-7200
(210) 999-7398
pmyers@trinity.edu

Early review of applications will begin December 12, 2011.

Trinity University is an
Equal Opportunity Employer.

UCL
Université Catholique de Louvain, Belgium
Professor - Programming Systems

UCL invites applications for a full-time faculty position in Programming Systems. The successful candidate will carry out research in the field of programming systems, including but not limited to programming languages, adaptive software, artificial intelligence, algorithmics, software analysis and synthesis. Still, other areas of competence will also be considered, since qualifications take precedence over specialization.

Responsibilities include research, supervision of undergraduate and graduate students, as well as PhD theses, submission and management of research grants, and undergraduate/graduate teaching within the curricula in Computer Science

(<http://www.uclouvain.be/en-info.html>). Courses are taught in French and English.

The University is located in the new city of Louvain-la-Neuve, 25 kms southeast of Brussels, the capital of Belgium, in the heart of Europe.

Applications must be posted before December 15th, 2011 on <http://www.uclouvain.be/en-287770.html>

University of Arkansas
Assistant Professor

The Computer Science and Computer Engineering (CSCE) Department at the University of Arkansas seeks outstanding individuals to fill two tenure-track, assistant professor positions. Successful candidates will have an earned doctorate in computer science, computer engineering, or a related field. We are seeking candidates with expertise in cyber security, data mining or machine learning, human-computer interaction, or pervasive/ubiquitous computing. Outstanding candidates in other related areas are also encouraged to apply.

The CSCE department offers BS, MS, and PhD degrees in Computer Science and Computer Engineering to approximately 250 undergraduate and 60 graduate students. The department has fourteen faculty with an annual external research funding of \$1.8 million and is located in a new 90,000 sq. ft. building. For more information concerning the department see <http://www.csce.uark.edu>

Application materials (cover letter, teaching and research statement, curriculum vitae, and the names and addresses of three references) should be submitted to the CSCE Search Committee electronically at <http://www.csce.uark.edu/search> or mailed to:

Search Committee
CSCE Department, 504 JBHT
Fayetteville, AR 72701
search@csce.uark.edu

Completed applications received by January 15, 2012 will be assured full consideration. Late applications will be reviewed as necessary to fill the position.

The University of Arkansas is ranked as a Carnegie Foundation RU/VH (research university, very high research activity) university and is located in Fayetteville, ranked as the 4th "Best Metro" in Forbes' 2009 list of "Best Places for Business and Careers," and recognized as one of the "Top 10 Best Cities to Live, Work and Play" by Kiplinger in 2008.

The University of Arkansas is an equal opportunity, affirmative action employer. Qualified underrepresented minority and women candidates are especially invited to apply. All applicants are subject to public disclosure under the Arkansas Freedom of Information Act and persons hired must have proof of legal authority to work in the United States.

University of Calgary
Department of Computer Science
Assistant Professor Positions

The Department of Computer Science at the University of Calgary seeks outstanding candidates for two tenure-track positions at the Assis-

tant Professor level. Applicants from the areas of **Database Management and Scientific Visualization** are of particular interest. Details for each position appear at: <http://www.cpsc.ucalgary.ca/>. Applicants must possess a doctorate in Computer Science 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 large undergraduate and graduate programs 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**, Head, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, T2N 1N4 or via email to: search@cpssc.ucalgary.ca. Completed applications received by December 15, 2011 will receive full consideration, though the review process will continue until the positions are filled. Hiring decisions will be finalized in Spring 2012, with the successful candidates joining the U of C on July 1, 2012.

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 California, Los Angeles

Computer Science Department

Tenure Track Positions, All Areas of Computer Science & Computer Engineering Tracking #0145-1112-01

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. Salary is commensurate with education and experience.

UCLA 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 candidates are urged to ensure that their applications and letters of reference are received by January 1, 2012.

University of Colorado Boulder

Tenure-track positions in Computer Systems

University of Colorado Boulder: Department of Electrical, Computer, and Energy Engineering (ECEE) seeks outstanding candidates for **two tenure-track positions in computer systems**. The openings are targeted at the level of **Assistant**

Professor, but experienced candidates with outstanding credentials may be considered for **Associate or Full Professor**.

Candidates interested in rigorous and innovative approaches to the design and analysis of complex computing systems (from embedded and cyberphysical to large-scale distributed systems) should apply. We seek candidates with background in programming languages, concurrency, security, formal methods, verification, or system engineering. Preference will go to researchers whose work spans multiple areas.

The positions will help shape the cooperation with the Department of Computer Science on computing systems.

Candidates must have a Ph.D. in electrical engineering, computer engineering, computer science, or related discipline; they must have the ability to develop an independent research program, and enthusiasm for working with undergraduate and graduate students.

The University of Colorado Boulder is committed to diversity and equality in education and employment. We encourage applications from women, minority candidates, people with disabilities, and veterans.

Applications will be evaluated starting December 6, 2011 and until the positions are filled.

Applications must include a letter of application specifying the desired position and area of specialization, complete curriculum vitae, statements of research and teaching interests, and names and contact information of three references. Applications must be submitted on-line at <http://www.jobsatcu.com/> using posting number #815103 (computer systems). Additional information is available at that site.

University of Colorado, Colorado Springs

Assistant Professor

The University of Colorado, Colorado Springs invites applications for up to three tenure-track Assistant Professor positions in all areas of Computer Science and Software Engineering. The CS Dept offers Bachelor, Master and PhD degrees. See full ad and apply electronically at <http://www.JobsatCU.com>, refer to posting #815131. Review of applications will begin on January 15, 2012 and continue until the positions are filled.

University of Houston-Clear Lake

Assistant Professor of Computer Science

The University of Houston-Clear Lake Computer Science program invites applications for a tenure-track Assistant Professor of CS to begin August 2012. A Ph.D. in CS, or closely related field, is required. Applications accepted online only at <https://jobs.uhcl.edu>. See <http://sce.uhcl.edu/cs>. AA/EOE.

University of Illinois at Chicago

Faculty Position

Department of Computer Science, UIC

The Computer Science Department at the University of Illinois at Chicago invites applications for tenure-track positions at the rank of Assistant Pro-

fessor (exceptional candidates at other ranks may also be considered). Candidates in the following areas are especially encouraged to apply: Computer Security, Software Engineering, Machine Learning and Computer Systems (Mobile/Ubiquitous computing and other experimental subareas).

The University of Illinois at Chicago (UIC) ranks among the nation's top 50 universities in federal research funding. It is the largest research university in the Chicago area, and is one of the most diverse universities in the country. The Computer Science department has 27 tenure-track faculty representing major areas of computer science, and offers BS, MS and PhD degrees. Two of our faculty members are ACM Fellows and eight members are recipients of NSF CAREER awards. Our annual research funding has averaged \$6.5M over the last five years and includes major funding from NSF, DARPA, DoD and NASA, including two NSF IGERT awards, nine Trustworthy Computing awards and several other research and instrumentation grants; awards from state agencies such as the Illinois Department of Transportation, and from companies such as Google, Yahoo! and Motorola. Our department is home to many pioneering and discipline-defining efforts in the areas of virtual reality (CAVE), software engineering (Petri Nets, Model Checking), Data Management and Mining, and Computational Transportation. We have growing research programs in areas such as computational biology, learning technologies, mobile and distributed systems, and security and privacy. At UIC, there are plenty of opportunities for interdisciplinary work—UIC houses the largest medical school in the country, and our faculty are engaged with several cross-departmental collaborations with faculty from health sciences, social sciences and humanities, urban planning and the business school.

Chicago is the third most populous city in the USA. Located by the shore of Lake Michigan, the city offers an outstanding array of cultural and culinary experiences. As the birthplace of the modern skyscraper, Chicago boasts one of the world's tallest and densest skylines, combined with an extensive system of parks and public transit. Its primary airport is the second busiest in the world, with frequent non-stop flights to virtually anywhere. Yet the cost of living, whether in an 85th floor condominium downtown or on a tree-lined street in one of the nation's finest school districts, is surprisingly low.

Applications must be submitted at <https://jobs.uic.edu/>. Please include a resume, teaching and research statements, and names and addresses of at least three references in the online application. Applicants needing additional information may contact the Faculty Search Chair at search@cs.uic.edu.

Application processing will commence on Nov 15th. We will continue to accept and process applications after that date until all the positions are filled. The University of Illinois at Chicago is an Affirmative Action/Equal Opportunity Employer.

University of Massachusetts Amherst

Faculty Positions in Computer Science

We invite applications for tenure-track faculty positions in computer science with a preference for applicants with expertise in software engineering, programming languages, security, graphics,

robotics, and computer vision. Applicants must have completed (or be completing) a Ph.D. in Computer Science, or a related area, and should show evidence of exceptional research promise. Three positions are available at either the Assistant Professor or Associate Professor level.

Our department is highly supportive of junior faculty, providing both formal and informal mentoring. Many of our faculty are involved in interdisciplinary research, working closely with other departments including statistics/mathematics, electrical and industrial engineering, biology, physics, linguistics, and nursing, as well as new "green" initiatives. Amherst, a historic New England town, is the center of a vibrant and culturally rich area that includes four other colleges. For more information about the department, visit <http://www.cs.umass.edu/>.

To apply, please send a cover letter referencing search R41582 with your vita, a research statement, a teaching statement, and at least three letters of recommendation. Electronic submission of application materials in pdf format is preferred. Send to facrec@cs.umass.edu. Alternatively, paper copies of application materials may be sent to: Search R41582 c/o Chair of Faculty Recruiting, Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

We will begin to review applications on November 15, 2011 and will continue until the positions are filled. Salary will be commensurate with education and experience. Inquiries and requests for more information can be sent to: facrec@cs.umass.edu.

The University of Massachusetts is an Affirmative Action/Equal Opportunity employer. Women and members of minority groups are encouraged to apply.

University of Minnesota – Twin Cities **Department of Computer Science and Engineering** **Faculty Position**

The Department of Computer Science and Engineering at the University of Minnesota-Twin Cities invites applications from candidates in all areas of Computer Science for two faculty positions. We strongly encourage applications from women and members of minority groups. Candidates should have a PhD in Computer Science or a closely related discipline. The position is open until filled, but for full consideration apply at www.cs.umn.edu/employment/faculty by December 15, 2011. The University of Minnesota is an equal opportunity employer and educator.

The University of Mississippi **Department of Computer & Information Science** **Assistant Professor Positions**

The Department of Computer and Information Science at the University of Mississippi invites applications for two tenure-track Assistant Professor positions.

An applicant must hold a PhD or equivalent in computer science or a closely related field by August 15, 2012. An applicant must have the ability to teach both graduate and undergraduate students, conduct research in major areas of

computer and information science, and supervise MS and PhD students. An applicant must provide evidence of research potential, effective communication skills, and a broad background in computing.

The Department has an ABET/CAC-accredited undergraduate program and MS and PhD programs. See the website <http://www.cs.olemiss.edu> for more information about the Department.

The University is located in Oxford, one of America's top-ranked college towns. Oxford has a wonderful small-town atmosphere and excellent schools.

Individuals may apply online at <http://jobs.olemiss.edu>. The application requires a cover letter, curriculum vitae, research and teaching statements, and contact information for four references. Review of applications will begin immediately and continue until the position is filled or an adequate applicant pool is reached.

The University of Mississippi is an EEO/AA/Title VI/Title IX/Section 504/ADA/ADEA Employer.

University of North Carolina at Charlotte **Chairperson** **Department of Software and Information Systems**

The SIS Department invites applications for the position of Department Chair. Candidates for the position must have a Ph.D. in Computer Science, Information Technology, or a closely related field, a record of research and publication commensurate with that of a Full Professor, evidence of a commitment to excellence in teaching, and strong administrative skills. SIS has a fast-growing enrollment and expects to grow significantly in faculty and research funding over the next 5 years. The chair is expected to bring dynamic leadership in research, curricular, and faculty development, and to build consensus within the department for strategic initiatives such as large, multidisciplinary research efforts. The university is one of the most rapidly growing in the country, and the goal within the College of Computing and Informatics is that the college and the SIS Department will lead this growth both in enrollment and quality of programs. Applications must be made electronically at <https://jobs.uncc.edu> (position1770) and must include a CV, references, and a statement on teaching, management, leadership, and research goals. Informal inquiries can be made to the Search Committee Chair, William Ribarsky, ribarsky@uncc.edu. Review of applications will begin as they are received and continue until the position is filled. All inquiries and applications will be treated as confidential. The University of North Carolina at Charlotte is an EOE/AA employer and an NSF ADVANCE Institution. For additional information, please visit our website at <http://sis.uncc.edu>.

The Department of Software and Information Systems has 15 faculty members, a large Ph.D. program, and over 350 students. The department offers a B.A. degree in Software and Information Systems, an M.S. degree in Information Technology, and a Ph.D. degree in Computing and Information Systems. SIS also plays a key role in the interdisciplinary Professional Science Master's in Health Informatics. The department has focus areas in Security, Design, and Analytics, with research that includes Complex Adaptive Sys-

tems, Information Security and Assurance, Data Privacy, Formal Analytics, Modeling and Simulation and Human Computer Interaction. The Security and Privacy program is recognized by the National Security Agency as a National Center of Academic Excellence in Information Assurance Education and Research. SIS is part of the College of Computing and Informatics along with the Departments of Computer Science and Bioinformatics and Genomics. CCI is the university leader in external funding per faculty member and has the largest Ph.D. program at UNC Charlotte.

The University of North Texas **Department of Computer Science and Engineering** **Assistant/Associate/Full Professors**

The Department of Computer Science and Engineering at the University of North Texas (UNT) is seeking candidates for multiple tenure-track/tenured faculty positions at the Assistant, Associate or Full Professor level beginning August 15, 2012. The department plans to build on its existing strengths in 3 areas: **Computer Systems**, including operating systems, runtime systems for cloud and high performance or mobile and handheld devices, software engineering of net-centric, real-time and embedded systems, and energy efficient and low power circuits and systems; **Intelligent Systems**, including data mining, machine learning, information retrieval, scientific visualization, human-computer interaction, and computational life sciences; and **Security**, including information assurance, network security and intrusion detection, and secure software systems and vulnerability analysis. Candidates should have demonstrated the potential to excel in research in one or more of these areas and in teaching. A Ph.D. in Computer Science, Computer Engineering or closely related field is required at the time of appointment. At the Assistant Professor level, the applicant's record must include high quality publications. At the Associate Professor level, the applicant must have at least 5 years of experience beyond an earned doctoral degree with a significant record of publications and extramural funding. A Full Professor would be expected to be a leader in his/her field with a record of building and maintaining a large-scale research program of international renown.

The Computer Science and Engineering department is home to 730 Bachelors students, 136 Masters students and 77 Ph.D. students. Additional information about the department is available at the department's website at: www.cse.unt.edu.

Application Procedure:

All applicants must apply online to: <https://facultyjobs.unt.edu>. Submit nominations and questions regarding the position to Dr. Philip Sweany (sweany@cse.unt.edu).

Application Deadline:

The committee will begin its review of applications on December 1, 2011 and continue to accept and review applications until the positions are closed.

The University:

With about 36,000 students, UNT is the nation's 33rd largest university. As the largest, most comprehensive university in Dallas-Fort Worth, UNT

drives the North Texas region. UNT offers 97 bachelor's, 88 master's and 40 doctoral degree programs, many nationally and internationally recognized. A student-focused public research university, UNT is the flagship of the UNT System.

The University of North Texas is an AA/ADA/EOE committed to diversity in its educational programs.

University of Notre Dame
Department of Computer Science and Engineering
Assistant or Associate Professor

The Department of Computer Science and Engineering at the University of Notre Dame invites applications for Assistant or Associate Professor. Excellent candidates in all areas will be considered.

The Department offers the PhD degree and accredited undergraduate Computer Science and Computer Engineering degrees, with currently over 80 PhD students and over 150 undergraduate majors. Faculty are expected to excel in classroom teaching and to build and lead cutting-edge and highly visible research projects that attract substantial external funding.

The University of Notre Dame is a private, Catholic university with a doctoral research extensive Carnegie classification, and consistently ranks in U. S. News & World Report as a top-twenty national university. The South Bend area has a vibrant and diverse economy with affordable housing and excellent school systems, and is within easy driving distance of Chicago and Lake Michigan.

Applicants should send (pdf format preferred) a CV, statement of teaching and research interests, and contact information for three professional references to: **facultysearch AT cse.nd.edu**

The University of Notre Dame is an Equal Opportunity, Affirmative Action Employer.

University of Oregon
Department of Computer and Information Science
Faculty Position
Assistant Professor

The Department of Computer and Information Science (CIS) seeks applications for a tenure track faculty position at the rank of Assistant Professor, beginning Fall 2012. The University of Oregon is an AAU research university located in Eugene, two hours south of Portland, and within one hour's drive of both the Pacific Ocean and the snow-capped Cascade Mountains.

The CIS Department is part of the College of Arts and Sciences and is housed within the Lorry Lokey Science Complex. The department offers B.S., M.S. and Ph.D. degrees. More information about the department, its programs and faculty can be found at <http://www.cs.uoregon.edu>.

We offer a stimulating, friendly environment for collaborative research both within the department and with other departments on campus. Faculty in the department are affiliated with the Cognitive and Decision Sciences Institute, the Computational Science Institute, and the Neuro-Informatics Center.

The department seeks to hire faculty in the general area of systems, with specific specialization that complements existing faculty strengths

in systems-related fields. The successful candidate will have strong record of accomplishments that demonstrate a highly creative approach to systems research, knowledge of state-of-the-art techniques and technology, and significant collaborative project work. In addition, the candidate should have experience creating, evaluating, and applying experimental systems artifacts. Opportunities for interactions with systems-oriented faculty include: parallel and distributed systems, networking, databases, intelligent systems, informatics, and computational science. Applicants must have a Ph.D. in computer science or closely related field, a demonstrated record of excellence in research, and a strong commitment to teaching. A successful candidate will be expected to conduct a vigorous research program and to teach at both the undergraduate and graduate levels.

Applications will be accepted electronically through the department's web site (only). Application information can be found at <http://www.cs.uoregon.edu/Employment/>. Review of applications will begin March 01, 2012 and continue until the position is filled. Please address any questions to faculty.search@cs.uoregon.edu.

The University of Oregon is an equal opportunity/affirmative action institution committed to cultural diversity and is compliant with the Americans with Disabilities Act. We are committed to creating a more inclusive and diverse institution and seek candidates with demonstrated potential to contribute positively to its diverse community.

University of Rochester
Tenure Track Position in Computer Science

The University of Rochester Department of Computer Science seeks applicants for a tenure-track faculty position. We are particularly interested in researchers in human-computer interaction and machine learning, but will consider all outstanding candidates. See <http://www.cs.rochester.edu/recruit> for details. UR is an Equal Opportunity Employer.

University of Science and Technology of China
School of Computer Science and Technology
Faculty Positions

The School of Computer Science and Technology at University of Science and Technology of China (USTC) invites applications for tenure-track or tenured positions at all levels. Research areas of particular interest include programming languages and compilers, formal verification, robotics, computational intelligence, machine learning, data mining, computer architectures, parallel and high-performance computing, network and distributed systems. For more information about the positions, please see http://en.cs.ustc.edu.cn/join_us.

University of South Florida
Assistant Professor Positions
Computer Science and Engineering

Applications are invited for two **tenure-track Assistant Professor positions** in the Department of Computer Science and Engineering. The Department is hiring in all areas of computer engineer-

ing and computer security (cybersecurity). In both areas we seek candidates with a record of outstanding-quality research publications and potential for excellence in teaching.

The Department of Computer Science and Engineering (<http://www.cse.usf.edu>) has 23 faculty members and offers B.S., M.S., and Ph.D. degrees. The research program is well supported by federal and state agencies and industry. The University of South Florida serves over 47,000 students and is one of the nation's top public research universities.

For further information and for application instructions, please see our faculty search website: <http://www.cse.usf.edu/faculty-search/>. For questions please send email to faculty-search@cse.usf.edu. Applications will be considered starting immediately until the positions are filled.

According to Florida law, applications and meetings regarding them are open to the public. The University of South Florida is an Equal Opportunity/Equal Access/Affirmative Action Institution. Women and minorities are strongly encouraged to apply.

University of South Florida
Instructor Position
Computer Science and Engineering

Applications are invited for one Instructor position in the Department of Computer Science and Engineering. We are seeking an instructor who can teach a broad range of core computer science and computer engineering courses – both software and hardware – at the undergraduate level, as well as advise students. Candidates must have completed, or be near completion, of a Ph.D. degree in computer science or a related area. For exceptionally qualified candidates an M.S. degree may be considered.

The Department of Computer Science and Engineering (<http://www.cse.usf.edu>) has 23 faculty members and offers B.S., M.S., and Ph.D. degrees. The undergraduate program graduates approximately 80 students per year. The University of South Florida is one of the nation's top public research universities.

For further information and for application instructions, please see our faculty search website: <http://www.cse.usf.edu/faculty-search/>. For questions please send email to faculty-search@cse.usf.edu. Applications will be considered starting immediately until the position is filled.

According to Florida law, applications and meetings regarding them are open to the public. The University of South Florida is an Equal Opportunity/Equal Access/Affirmative Action Institution. Women and minorities are strongly encouraged to apply.

University of Texas at Austin
Computer Science Department
Tenure Track/Tenured Faculty Positions

The Department of Computer Science of the University of Texas at Austin invites applications for tenure-track positions at all levels. Excellent candidates in all areas will be seriously considered, especially in Computer Architecture and other areas of computer systems research. All tenured and tenure-track positions require a Ph.D. or

equivalent degree in computer science or a related area at the time of employment.

Successful candidates are expected to pursue an active research program, to teach both graduate and undergraduate courses, and to supervise graduate students. The department is ranked among the top ten computer science departments in the country. It has 43 tenured and tenure-track faculty members across all areas of computer science. Many of these faculty participate in interdisciplinary programs and centers in the University, including those in Computational and Applied Mathematics, Computational Biology, and Neuroscience.

Austin, the capital of Texas, is located on the Colorado River, at the edge of the Texas Hill Country, and is famous for its live music and outdoor recreation. Austin is also a center for high-technology industry, including companies such as IBM, Dell, Freescale Semiconductor, Advanced Micro Devices, National Instruments, AT&T, Intel and Samsung. For more information please see the department web page: <http://www.cs.utexas.edu/>. The department prefers to receive applications online, beginning October 17, 2011.

To submit yours, please visit <http://services.cs.utexas.edu/recruit/faculty>

If you do not have internet access, please send a curriculum vita, a homepage URL, a description of research interests, and a list of selected publications to the address below. Applicants for an assistant professor position must have at least three (3) referees send letters of reference directly to the address provided. Applicants for a tenured position (associate or full professor) must have at least six (6) referees send letters of reference directly. The address for all hard copy material is: Faculty Search Committee, Department of Computer Science, The University of Texas at Austin, 1616 Guadalupe Street, Suite 2.408, Austin, Texas 78701, USA.

Inquiries about your application may be directed to facultysearch@cs.utexas.edu. For full consideration of your application, please apply by January 31, 2012. Women and minority candidates are especially encouraged to apply. The University of Texas is an Equal Opportunity Employer.

University of Texas at San Antonio Faculty Positions in Computer Science

The Department of Computer Science at The University of Texas at San Antonio (UTSA) invites applications for tenure/tenure-track positions at the Assistant, Associate or Professor level, starting Fall 2012. All areas of computer science will be considered. We are particularly interested in candidates in theory and algorithms or computer architecture, with a focus on computer and information security as a cross-cutting concern.

The Department of Computer Science currently has 23 faculty members and offers B.S., M.S., and Ph.D. degrees supporting a dynamic and growing program with 560 undergraduates and more than 160 graduate students, including 81 Ph.D. students. See <http://www.cs.utsa.edu> for application instructions and additional information on the Department of Computer Science. Applications must be submitted by email in PDF format to fsearch@cs.utsa.edu. UTSA is an EO/AA Employer.

**Chair of Faculty Search Committee
Department of Computer Science
The University of Texas at San Antonio
One UTSA Circle
San Antonio, TX 78249-0667
Phone: 210-458-4436 Fax: 210-458-4437
fsearch@cs.utsa.edu**

University of Toronto Scarborough Lecturer Computer Science

The Department of Computer & Mathematical Sciences at the University of Toronto Scarborough invites applications for a Lecturer (teaching-stream) in Computer Science. For more information see <https://www.mathjobs.org/jobs/jobs/3173>

University of Virginia Senior Director for Enterprise Infrastructure

We invite applications for our Sr. Director position. Responsible for leading all operational aspects of the University's IT organization. Significant relevant technical expertise, coupled with the ability to lead a large-scale enterprise, and foster a collaborative, high performance environment is required.

Apply for this Job:

Contact Person: Catherine Brand

Email Address: cab3ae@virginia.edu

Website: <https://jobs.virginia.edu>

Yuba Community College District Information Systems Financial Aid Liaison

**CATEGORICALLY FUNDED – POSITION
CONTINGENT UPON CONTINUED FUNDING
AND BOARD APPROVAL
COMMENCING: As Soon As Possible
INFORMATION SYSTEMS FINANCIAL AID
LIAISON
DEPARTMENT: FINANCIAL AID**

FINAL FILING DATE: FRIDAY, SEPTEMBER 30, 2011 BY 12:00 NOON. (POSTMARKS ARE NOT ACCEPTED)

LOCATION: *SCHEDULED TO BE ASSIGNED TO:
YUBA COLLEGE

SALARY: \$3675.83 - \$4039.28/MO. (CSEA Salary Schedule, initial placement will not be higher than the above listed salary; the top step for this position is \$5122.60/MO)

BASIC FUNCTION: Under the direction of the Dean of Financial Aid, the Information Systems Financial Aid Liaison will act as liaison between Financial Aid and Information Systems providing: assessment, implementation and testing of updates, new features and new state or federal regulations using the District's administrative software, Colleague. This position will be physically located in the Financial Aid Office.

DISTINGUISHING CHARACTERISTICS: The Student Services Liaison will assume responsibility for overall report development, testing and implementation of new functionality, and content development that will interface with third party applications.

REPRESENTATIVE DUTIES:

Design, develop and plan efficient means of im-

plementing new or revised system components to improve Colleague use and reliability. (E)

Help develop plans for upgrading of software application systems, lead department teams, and use effective project management tools (E)

Test and analyze the impact of system upgrades/patches on custom programming. (E)

Conduct and evaluate department needs and provide analysis of computer systems and recommend solutions. (E) Design & provide specifications for subroutines and extensions to customize the integrated system components. (E) Provide documentation, train staff, and end users in system features and usage. (E)

Evaluate and implement state and federal reporting requirements for information collection and processing. (E) Provide analytical and report writing services. (E)

Determine appropriate methods for gathering, presenting and reporting data. (E)

Develop report formats and specifications. (E)

Conduct analysis of large data files; ability to make assessments & recommendations on reporting assessments. (E) Participate in short- and long-range planning. (E)

Keep current about new developments and technology regarding Colleague information analysis, planning, and reporting systems. (E)

Explore innovations and trends in technology with and for institutional applicability. (E)

Meet with staff in other cross-functional systems to identify and resolve problems. (E)

Participate in internal and external user groups; serve on committees as assigned. (E)

Other related duties as assigned.

KNOWLEDGE OF:

Uses, capabilities characteristics and limitations of computer systems and related equipment Principles and techniques of programming, data processing and programming documentation

ABILITY TO:

Analyze, design, write, test and resolve problems with complex and technical computer systems, programs and subsystems

Read, interpret, and apply technical publications, manuals, and other documents

Independently diagnose problems, develop solutions and communicate effectively with users

Interpret and implement requirements based on laws and regulations Prepare clear and concise reports

Work independently or as a member of a team

Communicate effectively both orally and in writing

Maintain accurate records

Establish and maintain effective working relationships with staff, faculty and vendors

EDUCATION AND EXPERIENCE:

Any combination equivalent to a BA/BS degree and two (2) years of professional experience providing end user support in an integrated database environment; experience programming and developing reports, working knowledge of database interfaces and reporting tools; including community college experience.

ENVIRONMENT:

Office environment; subject to constant interruptions

PHYSICAL ABILITIES:

Sitting and operating a keyboard to enter data into a computer terminal for extended periods of time

Hearing and speaking to exchange information
Moderate lifting up to 25 pounds

REQUIRED DUTIES:

Demonstrate sensitivity to and understanding of the diverse academic, socioeconomic, cultural, disability and ethnic backgrounds of community college students.

* This position is anticipated to be assigned to the Yuba College but may be assigned temporarily or permanently within the District.

WORKING CONDITIONS:

Categorically funded positions are contingent upon funding. Smoking is restricted in many areas of the Yuba Community College District. Woodland Community College is a tobacco free campus.

INTERVIEW:

A candidate selected for interview will be required to visit the Yuba College at his/her own expense upon a date selected by the District. Meeting the minimum qualifications does not guarantee an interview.

FOREIGN TRANSCRIPTS:

Must include a U.S. evaluation and translation. Please contact the Office of Human Resources for a list of agencies providing this service.

PRE-EMPLOYMENT REQUIREMENTS:

All Academic, Classified and Management employees shall be required to provide fingerprints to the District for the purpose of obtaining a criminal history as authorized by the California Education Code and all fees are the responsibility of the selected candidates. All prospective Administrative and Classified employees shall be

required to provide verification of freedom from tuberculosis.

EQUAL EMPLOYMENT:

Yuba Community College District is an Equal Employment Opportunity Employer and guarantees equal opportunity regardless of race, color, creed, national origin, ancestry, gender, marital status, disability, religious or political affiliation, age or sexual orientation and does not discriminate in its educational programs, in employment nor in any other of its activities.

PART-TIME (less than .60 FTE): Part-time positions less than .60 FTE are not entitled to any District paid fringe benefits. The District does however; provide the employee prorated leaves including vacation, sick leave and paid holidays. Employees less than .50 FTE contribute to an Alternative Retirement System (Apple). Employees whose FTE is between .50 and .60 contribute to the California Public Employees Retirement System (CalPERS).

BENEFITS/SALARY:

The District offers a comprehensive benefit package for employees and dependents for positions whose FTE is .60 or higher, valued at over \$16,000 annually with a \$14.50 monthly out of pocket expense to employees or dependents for monthly premiums. The package includes health, dental, vision, one (1) life insurance policy and an Employee Assistance program. Additional benefits include contributions to the Public Employee's Retirement System (PERS) which is integrated with Social Security, 457/403b options, Vacation days - 7.33 hrs per month for the first year, 96 hrs per year, 1-5, 12 sick days and 20 holidays. INI-

TIAL PLACEMENT WILL NOT BE HIGHER THAN THE LISTED SALARY, STEP 3 OF THE CSEA SALARY SCHEDULE, ACCORDING TO THE CLASSIFIED POLICY AND PROCEDURES HANDBOOK. THIS DOES NOT APPLY TO INTERNAL CANDIDATE'S; PLEASE REVIEW THE TOP OF THE FLYER (SALARY).

APPLICATION PROCEDURE & DEADLINE:

A District Classified application and the Diversity Statement are required. The application is available at the Human Resources Office, 2088 North Beale Road, Building 100A, Room 21, Marysville, CA 95901. Or you may call our TTY line at (530) 634-7760 OR visit our Web Site at <http://www.yccd.edu/> It is the sole responsibility of the applicant to ensure that all application materials are received by the final filing date in the Human Resources Office by FRIDAY, SEPTEMBER 30, 2011 BY 12:00 NOON.

All submitted materials become District property, will not be returned, will not be copied and will be considered for this opening only. Faxed, emailed, incomplete and/or late applications will not be forwarded for further consideration.

Central Washington University Assistant/Associate Professor

Central Washington University, Computer Science Dept - accepting applications for Ass't/Assoc Prof. Applicants with research potential in computational science areas are encouraged to apply. To apply online, visit: <https://jobs.cwu.edu>. AA/EEO/Title IX Institution.



Association for
Computing Machinery

Advancing Computing as a Science & Profession



MentorNet

You've come a long way.
Share what you've learned.



ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.



Make a difference to a student in your field.

Sign up today at: www.mentornet.net

Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.

research highlights

P. 122

Technical Perspective Safety First!

By Xavier Leroy

P. 123

Safe to the Last Instruction: Automated Verification of a Type-Safe Operating System

By Jean Yang and Chris Hawblitzel

P. 132

Technical Perspective Anonymity Is Not Privacy

By Vitaly Shmatikov

P. 133

Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Steganography

By Lars Backstrom, Cynthia Dwork, and Jon Kleinberg

Technical Perspective

Safety First!

By Xavier Leroy

SOFTWARE MISBEHAVES ALL too often. This is a truism, but also the driving force behind many computing techniques intended to increase software reliability, safety, and security, ranging from basic testing to full formal verification.

In this wide spectrum of approaches, a sweet spot is type and memory safety. Rather than attempting to rule out all bugs, type and memory safety focuses on strict enforcement of a few basic safety properties: a character string is not a code pointer; arrays are always accessed within bounds; memory blocks are not accessed after deallocation; pointers or object references cannot be forged from integers; and so on. Such properties are enforced through a combination of static (compile-time) type-checking, dynamic (runtime) checks such as array bound checks, and automatic memory management. These humble safety properties not only catch a number of common programming errors, but are also surprisingly effective at thwarting many security attacks such as buffer overrun attacks. Moreover, they can be leveraged to build software-enforced access control and isolation architectures such as the Java and .NET security managers; for if object references can be forged from integers, any software-only security infrastructure can be circumvented.

In the mid-1990s came the realization that type and memory safety is not just for high-level programming languages. Java and its bytecode verifier popularized the idea that the bytecode of a virtual machine can be made type-safe through a combination of load-time type-checking (bytecode verification) and runtime checks in the virtual machine. Going one step further “down,” Morrisett, Walker, Crary and Glew introduced *Typed Assembly Language* (TAL), which guarantees type and memory safety directly at

the level of assembly language for the ubiquitous x86 processor architecture. There are several benefits to enforcing type and memory safety at the level of bytecode or assembly language: the compilers no longer need to be trusted to preserve safety and therefore are no longer part of the trusted computing base. Moreover, type-safe interoperability between different source languages can be guaranteed.

The following work by Yang and Hawblitzel is a major milestone in an ambitious research project: that of guaranteeing end-to-end type and memory safety for a complete software stack. Leveraging the Bartok .NET-to-typed-x86 compiler and the corresponding TAL checker, it is possible to automatically obtain safety guarantees for most of the software stack written in C#—not just application code, but also large chunks of systems code such as network protocols. In particular, the paper shows that the major part of a safe, preemptive scheduler for multitasking can be developed this way, which may come as a surprise to many readers.

However, not all parts of an operating system and runtime system can be

The following work is a major milestone in an ambitious research project: that of guaranteeing end-to-end type and memory safety for a complete software stack.

shown memory-safe using only TAL. A major offender is the memory manager (allocator, garbage collector, among others), which has to treat memory in an essentially untyped way. Similar issues occur in the lowest layers of operating systems (context switching, interrupt handling, among others). The standard approach at this point is to leave these components in the trusted computing base and validate them only by testing. Instead, Yang and Hawblitzel succeeded in formally verifying these components—which they call the “Nucleus” of their Verve operating system—against mathematical specifications (pre- and post-conditions), using the Boogie deductive program verifier.

The minimalistic design of the Nucleus is elegant, and the interplay between its specifications and the generic safety guarantees of the TAL code is subtle. Perhaps the most impressive aspect of this work, however, is the remarkable economy of means by which it achieves end-to-end type and memory safety. The high degree of automation offered by the Boogie verifier and Z3 automatic theorem prover does wonders here, resulting in an overall verification effort that is remarkably low by today’s standards.

The formal verification of high-assurance software is making great progress lately. Yang and Hawblitzel’s work, along with other recent breakthroughs in software verification such as the seL4 verified microkernel of Klein et al. (see *Communications*, June 2010, p. 107), were unthinkable 10 years ago. Little by little, one point at a time, these results sketch a promised land where, with mathematical certainty, software does behave properly after all. 

Xavier Leroy (xavier.leroy@inria.fr) is a senior research scientist at INRIA Paris-Rocquencourt, France.

© 2011 ACM 0001-0782/11/12 \$10.00

Safe to the Last Instruction:

Automated Verification of a Type-Safe Operating System

By Jean Yang and Chris Hawblitzel

Abstract

Typed assembly language (TAL) and Hoare logic can be used to verify the absence of many kinds of errors in low-level code. We use TAL and Hoare logic to achieve highly automated, static verification of the safety of a new operating system called Verve. We have developed techniques and tools to mechanically verify the safety of every assembly-language instruction in the operating system, runtime system, drivers, and applications (in fact, every part of the system software except the boot loader). Verve consists of a “Nucleus” that provides primitive access to hardware and memory, a kernel that builds services on top of the Nucleus, and applications that run on top of the kernel. The Nucleus, written in verified assembly language, implements allocation, garbage collection, multiple stacks, interrupt handling, and device access. The kernel, written in C# and compiled to TAL, builds higher-level services, such as preemptive threads, on top of the Nucleus. A TAL checker verifies the safety of the kernel and applications. A Hoare-style verifier with an automated theorem prover verifies both the safety and correctness of the Nucleus. Verve is, to the best of our knowledge, the first operating system mechanically verified to guarantee both type and memory safety. More generally, Verve’s approach demonstrates a practical way to mix high-level typed code with low-level untyped code in a verifiably safe manner.

1. INTRODUCTION

High-level computer applications build on services provided by lower-level software layers, such as operating systems and language runtime systems. These lower-level software layers should be reliable and secure. Without reliability, users endure frustration and potential data loss when the system software crashes. Without security, users are vulnerable to attacks from the network, which often exploit low-level bugs such as buffer overflows to take over a user’s computer. Unfortunately, today’s low-level software still suffers from a steady stream of bugs, often leaving computers vulnerable to attack until the bugs are patched.

Many projects have proposed using safe languages to increase the reliability and security of low-level systems. Safe languages ensure type safety and memory safety: accesses to data are guaranteed to be well-typed and guaranteed not to overflow memory boundaries or dereference dangling pointers. This safety rules out many common bugs, such as buffer overflow vulnerabilities. Unfortunately, it is difficult to express a complete computer system entirely in a safe language, because safe languages deliberately omit unsafe, low-level features, such as explicit memory deallocation. To perform low-level tasks like memory management, a safe language usually relies on a runtime system written in

an unsafe language (e.g., C), and any bugs in this runtime system can undermine the safety of the entire language. For example, such bugs have left popular Web browsers, including Mozilla and Internet Explorer, open to attack.¹⁰

This paper presents Verve, an operating system and runtime system that we have verified to ensure type and memory safety. Verve has a simple mantra: every assembly-language instruction in the software stack must be mechanically verified for safety. This includes every instruction of every piece of software except the boot loader: applications, device drivers, thread scheduler, interrupt handler, allocator, garbage collector, etc. Because of this, Verve does not have to trust a high-level language compiler to enforce safety, nor does it have to rely on unverified library code.

The goal of formally verifying low-level OS and runtime system code is not new. Nevertheless, very little mechanically verified low-level OS and runtime system code exists, and that code still requires man-years of effort to verify.^{5, 8} This paper argues that recent programming language and theorem-proving technologies reduce this effort substantially, making it practical to verify strong properties throughout a complex system. The key idea is to split a traditional OS kernel into two layers: a critical low-level “Nucleus,” which exports essential runtime abstractions of the underlying hardware and memory, and a higher-level kernel, which provides more fully fledged services. Because of these two distinct layers, we can leverage two distinct automated technologies to verify Verve: TAL (typed assembly language¹¹) and automated theorem provers. Specifically, we verify the Nucleus using automated theorem proving (based on Hoare Logic) and we ensure the safety of the kernel using TAL (generated from C#).

A complete Verve system consists of a Nucleus, a kernel, and one or more applications. We wrote the kernel and applications in safe C#, which is automatically compiled to TAL. An existing TAL checker³ verifies this TAL code (again, automatically). We wrote the Nucleus directly in assembly language, hand-annotating it with assertions (preconditions, postconditions, and loop invariants). An existing Hoare-style program verifier called Boogie¹ verifies the assembly language against a specification of safety and correctness. This ensures the safety and correctness of the Nucleus’s implementation, including safe interaction with the TAL code and safe interaction with hardware (including memory, interrupts, timer, keyboard, and screen). Boogie relies on Z3,⁴ an automated theorem prover, to

The original version of this paper was published in *Programming Language Design and Implementation (PLDI)*, 2010, ACM.

check that the assertions are satisfied. Writing the assertions requires human effort, but once they are written, we can use Boogie and Z3 to verify them completely automatically. As a result, the Verve Nucleus requires only 2–3 lines of proof annotation per executable statement. Although it is difficult to compare annotation burdens across systems that use different proof environments and programming languages, similar projects based on interactive theorem provers^{5,8} have required more than 10 lines of proof annotation per line of code.

Verve boots and runs on real, off-the-shelf x86 hardware, and provides efficient support for realistic language features, including classes, virtual methods, arrays, and preemptive threads. Nevertheless, the current Verve system is still small compared to commodity operating systems and has many limitations. It lacks support for many C# features: exception handling, for example, is implemented by killing a thread entirely, rather than with try/catch. It lacks dynamic loading of code. It runs on only a single processor. Although it protects applications from each other using type safety, it lacks a more comprehensive isolation mechanism between applications such as Java Isolates or C# AppDomains. The verification does not guarantee termination. Finally, Verve uses verified garbage collectors⁷ that are stop-the-world rather than incremental or real time, and Verve keeps interrupts disabled throughout the collection. Except for multiprocessor support, none of the limitations in Verve's present implementation are fundamental.

We expect that with more time, the high degree of automation in Verve's verification will allow Verve to scale to a more realistic feature set, such as a large library of safe code and a verified incremental garbage collector. Indeed, we have already ported about 35,000 lines of safe C# code to run on top of Verve, including standard C# libraries, device drivers, and implementations of several Internet protocols.

In this paper, we describe our verification tools (Section 2), the interface that our Nucleus exports to the rest of the kernel (Section 3), the verification of the Nucleus (Section 4), the kernel (Section 5), the time it takes to verify Verve (Section 6), and related work in systems verification (Section 7).

2. TOOLS FOR BUILDING A SAFE OS

Two verification technologies, TAL and automated theorem proving, drive Verve's design. TAL is assembly language annotated with type information at the level of the assembly-language registers. We can use a simple type checker to verify that each assembly-language instruction in a TAL program respects the types of the instruction's operands. For instance, the checker would reject the use of an integer as a memory address.

While TAL reasons about types, theorem provers reason about logical formulas, attempting to prove formulas valid or invalid. *Automated* theorem provers run with little or no human assistance, in contrast to interactive theorem provers, which can prove a wider variety of formulas but often require considerable human assistance. Modern automated theorem provers can reason about various theories, such as integer arithmetic, bitwise arithmetic, and

arrays. Based on this reasoning, automated theorem provers can soundly prove deep properties about computer programs.

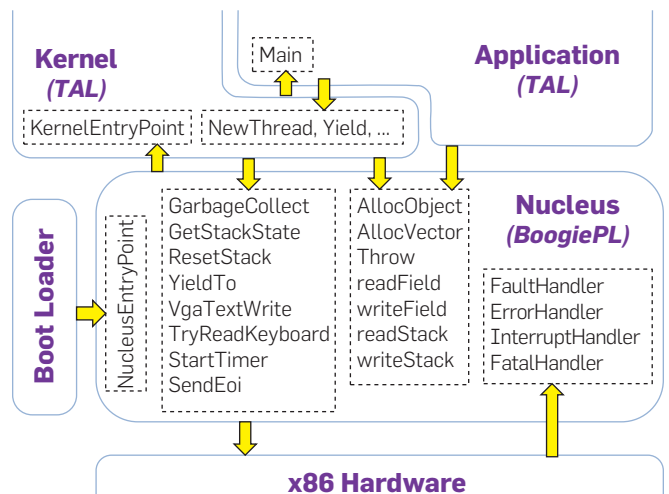
TAL and automated theorem proving are complementary technologies. On one hand, TAL is relatively easy to generate: because of the similarity between TAL types and high-level language types, a compiler can automatically turn high-level language code into TAL code, relying on the type annotations already present in the high-level language code. This enables TAL to scale easily to large amounts of code. Verve uses the Bartok compiler,³ which automatically generates TAL code from type-safe C# code.

On the other hand, we can use automated theorem provers to verify deeper logical properties about the code than a typical TAL type system can express, using a methodology discussed by Turing¹² and described by Floyd and Hoare in the 1960s, now commonly known as “Hoare logic.” In this methodology, a programmer annotates various points in the program, such as procedure entry points and loop entry points, with annotations describing the state of the machine. Such annotations are similar to type annotations, but specify properties of variables in much greater detail than usually found in type annotations. For example, a type annotation might merely say that registers `eax` and `ebx` both have type `int`, while a Hoare-style annotation might specify a precise formula about the values in `eax` and `ebx`, such as “`eax >= 10 && eax + ebx < 20`.” Because of this level of detail, writing these annotations requires substantial programmer effort.

To exploit the tradeoff between TAL and automated theorem proving, we decided to split the Verve operating system code into two parts, shown in Figure 1: a Nucleus, verified with Hoare logic and automated theorem proving, and a kernel, verified with TAL. The relative difficulty of Hoare logic motivated the balance between the two parts: only the functionality that we could not use TAL to verify as safe went into the Nucleus; all other code went into the kernel.

The Nucleus's source code is not expressed in TAL, but

Figure 1. Verve structure, showing all 20 functions exported by the Nucleus.



rather in Boogie’s programming language, called BoogiePL (or just Boogie), so that the Boogie verifier can check it. Since the Nucleus code consists of assembly-language instructions, these assembly-language instructions must appear in a form that the Boogie verifier can understand. As described in detail below, we decided to encode assembly-language instructions as BoogiePL statements, so that the Boogie verifier can check that each instruction’s requirements are satisfied. After Boogie verification, a separate tool called “BoogieAsm,” developed for an earlier project,⁷ extracts standard assembly-language instructions from the BoogiePL code. A standard assembler then turns these instructions into an object file.

A bug in the Boogie checker or TAL checker could allow unsafe code to pass into the Verve system. Therefore, we currently must trust that these checkers are correct; they are part of Verve’s “trusted computing base.” Figure 2 shows the various components of Verve’s trusted computing base. The only trusted components are the tools used to verify, assemble, link, and boot the verified Nucleus and kernel. Note that although we rely on various compilers (the Bartok compiler generates TAL code, while a separate compiler called “Beat” generates much of the BoogiePL code), none of our compilers are part of our trusted computing base: we do not need to trust the compilers to ensure the correctness of the Nucleus and safety of the Verve system as a whole.

The trusted computing base includes the specification of correctness for the Nucleus’s BoogiePL code. This includes specifications of the behavior of functions exported by the Nucleus, shown in Figure 1. (For example, the specification of “YieldTo” ensures that the Nucleus sets the stack pointer to the top of the correct stack during a yield.) It also includes specifications for assembly-language instructions and for interaction with hardware devices and memory; we took some of these specifications from existing work,⁷ and wrote some of them from scratch. All Boogie specifications

are written as logical formulas in the BoogiePL language.

By expressing and checking properties at a low level (assembly language), we can ensure nontrivial properties with high confidence. The bulk of this paper focuses on these properties, with an emphasis on the specification and verification of the Nucleus’s correctness properties. The next two sections discuss the Nucleus’s design and verification.

3. THE NUCLEUS INTERFACE

The core of our verification is the Nucleus, which provides a verified interface to the low-level functionality of the operating system. We verify the Nucleus using Hoare logic in Boogie, based on a trusted specification for x86 assembly-language instructions. In Verve, all access to low-level functionality must occur through the Nucleus—the kernel’s TAL code and application’s TAL code can access low-level functionality only indirectly, through the Nucleus. For example, TAL code cannot directly access devices. Furthermore, even though TAL code can directly read and write words of memory, it can read and write only words designated as safe-for-TAL by the Nucleus’s garbage collector.

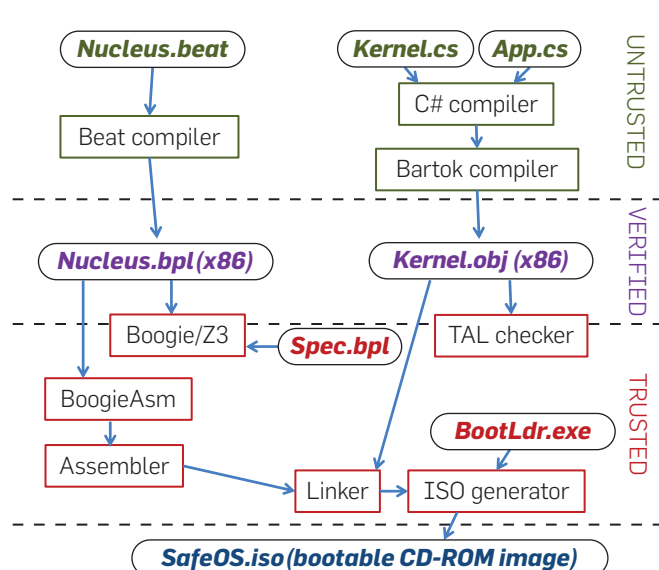
The Nucleus consists of a minimal set of functions necessary to support the TAL code that runs above it. We wanted a minimal set because even with an automated theorem prover, Hoare-style verification is still hard work; less code in the Nucleus means less code to verify. At the same time, the set has to guarantee safety in the presence of arbitrary TAL code; it can assume that the TAL code is well typed, but can make no further assumptions about the behavior of the TAL code. For example, when an interrupt occurs, the Nucleus tries to transfer control to a designated TAL interrupt handler. The Nucleus cannot assume that this handler is in the correct state to handle the interrupt, and must therefore check the handler’s state at runtime.

One design decision greatly simplified the Nucleus: following a design used in recent microkernels,^{6,8} no Nucleus function ever blocks. In other words, every Nucleus function performs a finite (and usually small) amount of work and then returns. The Nucleus may, however, return to a different thread than the thread that invoked the function. This allows the kernel built on top of the Nucleus to implement blocking thread operations, such as waiting on a semaphore.

Another design decision simplified reasoning about the Nucleus: following the approach taken by the recent verified L4 microkernel, seL4,⁸ Verve keeps interrupts disabled throughout the execution of any single Nucleus function. (On the other hand, interrupts may be enabled during the TAL kernel’s execution, with no loss of safety.) Since Nucleus functions do not block, Verve still guarantees that eventually, interrupts will always be re-enabled, and usually will be re-enabled very quickly. However, Verve’s current implementation sacrifices real-time interrupt handling because of one particularly long function: “GarbageCollect,” which performs an entire stop-the-world garbage collection. In the future, we hope to improve real-time behavior by using a verified incremental collector.¹⁰

Such design decisions led us to a small Nucleus API

Figure 2. Building the Verve system: trusted, untrusted components.



consisting of just 20 functions, all shown in Figure 1. These 20 functions, implemented with a total of about 1500 x86 instructions, include memory management (AllocObject, AllocVector, GarbageCollect, readField, writeField, readStack, and writeStack), stack management (GetStackState, ResetStack, YieldTo, Throw), device access (VgaTextWrite, TryReadKeyboard, StartTimer, SendEoi), fault handling (FaultHandler, ErrorHandler, FatalHandler, and InterruptHandler), and startup (NucleusEntryPoint). Most of the functions are intended for use by only the kernel, not by applications. However, applications may call AllocObject, AllocVector, and Throw directly.

4. THE VERIFIED NUCLEUS

To verify that the Nucleus behaves correctly, we have to specify what correct behavior is. Formally, this specification consists of preconditions and postconditions for each of the 20 functions exported by the Nucleus (Figure 1). The preconditions reflect the guarantees made by other components of the system when calling the Nucleus. For example, the precondition to NucleusEntryPoint describes the state of memory when the Nucleus begins execution; the (trusted) boot loader is responsible for establishing this precondition. The preconditions for functions exported to the kernel and applications describe the state of registers and the current stack when making a call to the Nucleus; the (trusted) TAL checker is responsible for guaranteeing that these preconditions hold when the (untrusted) kernel and applications transfer control to the Nucleus. Nucleus postconditions describe changes to the Nucleus state and reflect memory and guarantees the Nucleus makes to the rest of the kernel. Because preconditions are relatively weak for certain functions, the Nucleus must occasionally perform runtime checks to validate the values passed from the kernel and applications.

The Nucleus specification describes what the Nucleus must do, but does not specify exactly how the Nucleus must be implemented. For example, the Verve specification of garbage collection does not specify which algorithm the garbage collector should implement. Instead, following the approach of McCreight et al.,¹⁰ the specification just says that the garbage collector must ensure that the stack frames and heap objects contain the correct data, with no dangling pointers. We have built a verified mark-sweep and a verified copying collector for Verve, both obeying this same specification.

The Nucleus interacts with five components: memory, hardware devices, the boot loader, interrupt handling, and TAL code (kernel and application code). Memory and hardware devices export functionality to the Nucleus, such as the ability to read memory locations and write to hardware devices. The verification process ensures that the Nucleus satisfies the preconditions to each operation on memory and hardware. In turn, the Nucleus exports functionality to the boot loader (the Nucleus entry point), the interrupt handling (the Nucleus's interrupt handlers), and the TAL code (AllocObject, YieldTo, etc.).

4.1. Specification logistics

We express Verve's specification as first-order logical formulas in BoogiePL.¹ These formulas follow C/Java/C# syntax and consist of:

- Arithmetic operators: $+$, $-$, $*$, $>$, $==$, $!=$, $\dots$
- Boolean operators: $!$, $\&\&$, $||$, $\Rightarrow$, $\dots$
- Variables: `foo`, `Bar`, `old(foo)`, $\dots$
- Boolean constants: `true`, `false`
- Integer constants: `5`, $\dots$
- Bit vector constants: `5bv16`, `5bv32`, $\dots$
- Function application: `Factorial(5)`, `Max(3,7)`, `IsOdd(9)`, $\dots$
- Array indexing: `foo[3]`, $\dots$
- Array update: `foo[3] := Bar`, $\dots$
- Quantified formulas: $(\forall i:\text{int}::\text{foo}[i] == \text{Factorial}(i))$, $\dots$

BoogiePL bit vectors correspond to integers in C/Java/C#, which are limited to numbers that fit inside a fixed number of bits. BoogiePL integers, on the other hand, are unbounded mathematical integers. BoogiePL arrays are unbounded mathematical maps from some type (usually integers) to some other type. Unlike arrays in C/Java/C#, BoogiePL arrays are immutable values (there are no references to arrays and arrays are not updated in place). An array update expression `a[x := y]` creates a new array, which is equal to the old array `a` at all locations except `x`, where it contains a new value, `y`. For example, $(a[x := y])[x] == y$ and $(a[x := y])[x + 1] == a[x + 1]$.

BoogiePL procedures have preconditions and postconditions, written as BoogiePL logical formulas:

```
var a:int, b:int;
procedure P(x:int, y:int)
  requires a < b && x < y;
  modifies a, b;
  ensures a < b && a == x + old(a);
{
  a := a + x;
  b := b + y;
}
```

In this example, the procedure `P` can be called only in a state where global variable `a` is less than global variable `b`, and the parameter `x` is less than the parameter `y`. Upon exit, the procedure's postconditions ensure that `a` is still less than `b`, and that `a` is equal to `x` plus the old version of `a` (before `P` executed). Note that the procedure must explicitly reveal all the global variables that it modifies ("`modifies a, b;`" in this example), so that callers to the procedure will be aware of the modifications.

The Boogie tool relies on the Z3⁴ automated theorem prover. Z3 automatically checks logical formulas involving linear integer arithmetic (addition, subtraction, comparison), arrays, bit vectors, and functions. Z3 checks integer formulas faster than it checks bit-vector formulas, so we

chose integers over bit vectors where possible. Z3 also checks quantified formulas (formulas with `forall` and `exists`); however, Z3 relies on programmer-supplied hints in the form of *triggers* to help with quantifiers, since checking quantified formulas with arithmetic and arrays is undecidable in general.

For each function exported to the boot loader, interrupt handling, and TAL code, the Nucleus implements a BoogiePL procedure whose specification is given in terms of “requires,” “ensures,” and “modifies” clauses. The Nucleus implements these procedures in terms of more primitive hardware procedures: each BoogiePL procedure exported from the memory and hardware devices to the Nucleus corresponds to exactly one assembly-language instruction, such as an instruction to read a single memory location or write to a single hardware register. The rest of this section presents specifications for various BoogiePL procedures in Verve, and describes how the Nucleus implementation is verified against these specifications.

4.2. Memory

Verve’s initial memory layout is set by the boot loader. Verve uses an off-the-shelf boot loader, which sets up an initial virtual-memory address space (i.e., it sets up a page table), loads the executable image into memory, and jumps to the executable’s entry point, passing detailed information about the memory layout to the entry point. The boot-loader-supplied address space simply maps virtual memory directly to physical memory, except for a small range of low addresses that are left unmapped (to catch null pointer dereferences). A traditional operating system would create new virtual memory address spaces to protect applications from each other. Because Verve guarantees type safety, however, it can rely on type safety for protection and keep the initial boot-loader-supplied address space.

Verve’s mapped address space consists of three parts. First is the memory occupied by the executable image, including code, static fields, method tables, and memory layout information for the garbage collector. Verve may read this memory, but may write to only the static fields, not the code, method tables, or layout information. Second, the Verve specification reserves the memory just above the executable image for the interrupt table. Verve may write to the table, but it can write only values that obey the specification for interrupt handlers. Third, the remaining memory above the interrupt handler is general-purpose memory, free for arbitrary use. The specification describes the state of general-purpose memory using a global variable `Mem`, which is an array that maps integer byte addresses to integer values. For any 4-byte-aligned address i in general-purpose memory, `Mem[i]` contains the 32-bit memory contents stored at address i , represented as an integer in the range $0 \dots 2^{32} - 1$.

Each part of memory exports its own access functions to the Nucleus. The general memory exports two operations to the Nucleus, `Load` and `Store`:

```
procedure Load(ptr:int) returns (val:int);
  requires memAddr(ptr);
  requires Aligned(ptr);
  modifies Eip;
  ensures word(val);
  ensures val == Mem[ptr];

procedure Store(ptr:int, val:int);
  requires memAddr(ptr);
  requires Aligned(ptr);
  requires word(val);
  modifies Eip, Mem;
  ensures Mem == old(Mem)[ptr := val];
```

Each of these two operations requires a 4-byte-aligned pointer (“`Aligned(...)`”) to memory inside the general-purpose memory region (“`memAddr(...)`”). The loaded or stored value must be in the range $0 \dots 2^{32} - 1$ (“`word(...)`”). Any `Store` operation updates the contents of `Mem`, so that subsequent `Load` operations are guaranteed to see the updated value. Loads and stores have an additional side effect, noted in the modifies clause: they modify the current instruction pointer (program counter), “`Eip`.”

4.3. Hardware devices

Verve supports four basic hardware devices: a programmable interrupt controller (PIC), a programmable interval timer (PIT), a VGA text screen, and a keyboard. Verve specifies the interaction with this hardware using unbounded streams of events. The Nucleus delivers events to the PIC, PIT, and screen, and it receives events from the keyboard. For the screen, the events are commands to draw a character at a particular position on the screen. For the keyboard, events are keystrokes received from the keyboard. For the PIC and PIT, particular sequences of events initialize interrupt handling and start timers.

We present the keyboard specification as an example. Verve represents the stream of events from the keyboard as an immutable array `KbdEvents` mapping event sequence numbers (represented as integers, starting from 0) to events (also represented as integers). As the Nucleus queries the keyboard, it discovers more and more events from the stream. Two indices into the array, `KbdAvailable` and `KbdDone`, indicate the state of the Nucleus’s interaction with the keyboard. Events $0 \dots \text{KbdDone} - 1$ have already been read by the Nucleus, while events $\text{KbdDone} \dots \text{KbdAvailable} - 1$ are available to read but have not yet been read.

Two operations, `KbdStatusIn8` and `KbdDataIn8`, query the keyboard. Each of these procedures represents a single x86 assembly-language 8-bit I/O instruction, and BoogieAsm translates each call to these procedures into a single x86 “in” instruction. By invoking `KbdStatusIn8`, the Nucleus discovers the current state of `KbdAvailable` and `KbdDone`. If this operation places a 0 in the `eax` register’s lowest bit, then no events are available; if the operation places a 1 in `eax`’s lowest bit, then at least one event is available. If the Nucleus can prove that at least one event

is available (`KbdAvailable > KbdDone`), it may call `KbdDataIn8` to receive the first available event.

```
var KbdEvents:[int]int;
var KbdAvailable:int, KbdDone:int;
procedure KbdStatusIn8();
  modifies Eip, eax, KbdAvailable;
  ensures and(eax,1)==0 ==> KbdAvailable==KbdDone;
  ensures and(eax,1)!=0 ==> KbdAvailable> KbdDone;
procedure KbdDataIn8();
  requires KbdAvailable > KbdDone;
  modifies Eip, eax, KbdDone;
  ensures KbdDone == old(KbdDone) + 1;
  ensures and(eax,255) == KbdEvents[old(KbdDone)];
```

Given primitive x86 operations like Load, Store, `KbdStatusIn8`, and `KbdDataIn8`, we can implement and verify the procedures that make up the Nucleus. We illustrate this process with a small, but complete, example—the verified source code implementing `TryReadKeyboard` from Figure 1, along with a portion of its specification:

```
procedure TryReadKeyboard();
...
  ensures KbdAvailable==old(KbdDone) ==> eax==256;
  ensures KbdAvailable> old(KbdDone) ==>
    eax==KbdEvents[old(KbdDone)];

implementation TryReadKeyboard() {
  call KeyboardStatusIn8();
  call eax := And(eax, 1);
  call Go(); if (eax != 0) {goto skip;}
  call eax:=Mov(256);
  call Ret(old(RET)); return;
skip:
  call KeyboardDataIn8();
  call eax := And(eax, 255);
  call Ret(old(RET)); return;
}
```

`TryReadKeyboard`'s specification requires that the Nucleus return a keystroke (in the range 0–255) if one is available, and otherwise return the value 256. The implementation of the `TryReadKeyboard` function calls `KeyboardStatusIn8`, performs a bitwise AND operation to discover the status, and branches based on the status. To verify that the implementation meets the specification, we simply run the Boogie tool on `TryReadKeyboard`'s implementation. Boogie queries the Z3 theorem prover to check that the procedure satisfies its postconditions, and that all calls inside the procedure satisfy the necessary preconditions. Given the BoogiePL source code, this process is entirely automatic, requiring no scripts or human interactive assistance to guide the theorem prover.

BoogieAsm checks that each statement in the verified BoogiePL code corresponds to a simple, predetermined sequence of 0, 1, or 2 assembly-language instructions (e.g., “call `eax := And(eax, 1)`” corresponds to “and `eax, 1`”), and then transforms the BoogiePL code into valid assembly code:

```
_?TryReadKeyboard proc
  in al, 064h
  and eax, 1
  cmp eax, 0
  jne TryReadKeyboard$skip
  mov eax, 256
  ret
TryReadKeyboard$skip:
  in al, 060h
  and eax, 255
  ret
```

Note that some variables in the BoogiePL code, like `KbdEvents`, `KbdAvailable`, and `KbdDone`, are “specification variables” that exist only during verification, and do not exist in the generated assembly code.

4.4. Stacks and threads

Verve supports preemptive multithreading, which requires periodic switching between stacks. This stack switching requires the Nucleus specification to be explicit about the flow of control between the kernel and the Nucleus. Most Nucleus procedures have a simple control flow—the Nucleus procedure performs some work, and then uses the x86 `ret` instruction to return directly to the caller that called the Nucleus procedure (see `TryReadKeyboard` in the previous section, e.g.). However, a few Nucleus procedures return in more complicated ways. Interrupt handlers, for example, use a special x86 “return-from-interrupt” instruction, rather than the standard x86 `ret` instruction. More interestingly, several procedures, including `YieldTo` and interrupt handlers, may return to a caller in a different stack.

Verve uses a specification variable “RET” to specify how each procedure must return. RET equals one of two values: `ReturnToAddr(i)`, which specifies that the procedure must perform a normal return (the x86 `ret` instruction) to return address `i`, or `ReturnToInterrupted(i, cs, eflags)`, which specifies that the procedure must perform an interrupt return (the x86 `iretd` instruction) to return address `i`, restoring code segment `cs` and status flags `eflags`. Most Nucleus procedures are required to return to the return address pushed on the stack, pointed to by the stack pointer `esp`:

```
procedure TryReadKeyboard(...);
  requires RET == ReturnToAddr(Mem[esp]);
  ...
```

The Nucleus entry point, on the other hand, does not return to its caller (the boot loader), but instead returns to the TAL code located at address `KernelEntryPoint`, so that the kernel begins execution after the Nucleus has finished initialization:

```
procedure NucleusEntryPoint (...);
  requires RET == ReturnToAddr(KernelEntryPoint);
  ...
```

The kernel scheduler calls the `YieldTo` procedure to switch stacks. For example, the kernel's timer interrupt handler calls `YieldTo` to preempt one thread and switch to another thread. The exact behavior of `YieldTo` depends on the state of the target stack that is being yielded to. Each stack may be in one of four states at any time: empty, interrupted, yielded, or running. When switching to an empty stack (a stack with no stack frames), `YieldTo` is required to switch the stack pointer to the empty stack and then return to `KernelEntryPoint`, which starts a new thread running in the empty stack. When switching to a stack that had been interrupted earlier, `YieldTo` is required to use return-from-interrupt to return control to the interrupted instruction running in the interrupted stack. To switch to a stack that had earlier voluntarily yielded, `YieldTo` uses an ordinary return instruction. Finally, switching to the currently running thread has no effect; in this case, `YieldTo` simply returns:

```
procedure YieldTo (...);
  requires
    (StackState[s] == StackRunning && s == S
     && RET == ReturnToAddr(Mem[esp]))
  || (StackState[s] == StackYielded(...)
     && RET == ReturnToAddr(...))
  || (StackState[s] == StackInterrupted(...)
     && RET == ReturnToInterrupted(...))
  || (StackState[s] == StackEmpty
     && RET == ReturnToAddr(KernelEntryPoint)
     && ...);
  ...
```

In the specification above, stacks are numbered starting from 0, where `S` contains the current running stack and `s` contains the target stack to which `YieldTo` is switching. The array `StackState` maps stacks to their states (empty, interrupted, yielded, or running). The Nucleus's interrupt handler "InterruptHandler" has a specification similar to `YieldTo`'s specification, except that its target stack is always stack number 0, which runs the kernel's TAL code for scheduling and interrupt processing.

5. KERNEL

The Verve Nucleus implements low-level primitives to manage memory, switch stacks, and access hardware, but by itself the Nucleus does not provide the high-level

abstractions that applications expect from an operating system. Therefore Verve also includes a simple kernel on top of the Nucleus, written in C# and compiled to TAL. This kernel includes standard libraries for thread synchronization and scheduling. It also schedules garbage collections, bringing each thread to a safe point before invoking the Nucleus garbage collector.

The key novelty of Verve's kernel is that it is written entirely in safe code, relying on the Nucleus to perform operations that require unsafe code in traditional operating systems, such as memory deallocation and context switching. In general, the kernel decides policies, such as when a particular thread should run or when a garbage collection should take place, while the Nucleus implements mechanisms like context switching and garbage collection. An incorrect policy decision might cause incorrect behavior; for example, an incorrect scheduler implementation might fail to schedule a thread. However, an incorrect kernel implementation cannot violate the system's type safety.

As an example of a kernel policy, the current kernel implements round-robin preemptive threading, allowing threads to block on semaphores. The kernel manages the threads by keeping each thread in a queue, where one queue holds threads ready to run, one queue holds threads awaiting garbage collection, each semaphore contains a queue of blocked threads, and so on. Each queue is implemented as a linked list using standard C# objects, but queues could be implemented in other ways (e.g., using a tree or array to implement priority queues) without requiring changes to the Nucleus. Furthermore, other policies, such as priority-based scheduling or deadline-based scheduling, could be implemented with no changes to the Nucleus.

6. MEASUREMENTS

This section summarizes Verve's performance, quantifies the size of the Nucleus implementation (including annotations), and describes the time taken to verify Verve, for mechanical verification and in terms of person-months.

First, we wrote two simple micro-benchmarks to exercise the Nucleus's stack management. We wrote the benchmarks in C#, compiled them to TAL, verified the TAL code, and linked them with the kernel and Nucleus. We then ran them on a 1.8GHz AMD Athlon 64 3000+ with 1GB RAM, using the processor's cycle counters to measure time and averaging over multiple iterations, after warming the caches. We measured for two configurations—Verve built with a copying collector and Verve built with a mark-sweep (MS) collector:

	Copying (cycles)	MS (cycles)
2*YieldTo	98	98
2*Wait + 2*Signal	216	216

The `YieldTo` benchmark shows that the Verve Nucleus requires 98 cycles to switch from one stack to another

and back (49 cycles per invocation of YieldTo). The kernel builds thread scheduling and semaphores on top of the raw Nucleus YieldTo operation. Using semaphore wait and signal operations, it takes 216 cycles to switch from one thread to another and back (108 cycles per thread switch). The wait/signal performance is comparable to the round-trip IPC performance of fast microkernels such as L4 (242 cycles on a 166MHz Pentium⁹) and seL4 (448 cycles on an ARM processor⁸), although in fairness, IPC involves an address space switch as well as a thread switch.

We next present the size of various parts of the Nucleus specification and implementation. All measurements are lines of BoogiePL code, after removing blank lines and comment-only lines. The following table shows the size of various portions of the trusted specification:

Basic definitions	61
Memory and stacks	116
Interrupts and devices	111
x86 instructions	126
GC tables and layouts	317
Nucleus GC, allocation functions	239
Nucleus other functions	215
Total BoogiePL lines	1185

Overall, 1185 lines of BoogiePL is fairly large, but most of this is devoted to definitions about the hardware platform and memory layout. The GC table and layout information, originally defined by the Bartok compiler, occupies a substantial fraction of the specification. The specifications for all the functions exported by the Nucleus total $239 + 215 = 454$ lines.

We measured the size of the Nucleus implementation for two configurations of Verve, one with the copying collector and one with the mark-sweep collector (note that the trusted specifications are the same for both collectors); 1610 lines of BoogiePL are shared between the two configurations:

	Copying	MS
Shared BoogiePL lines	1610	1610
Private BoogiePL lines	2699	3243
Total BoogiePL lines	4309	4854
Specification BoogiePL lines	1185	1185
Total BoogiePL lines w/spec	5494	6039
x86 instructions	1377	1489
BoogiePL/x86 ratio	3.1	3.3
BoogiePL+spec/x86 ratio	4.0	4.1

In total, each configuration contains about 4500 lines of BoogiePL. From these, BoogieAsm extracts about 1400 x86 instructions. This corresponds roughly to a 3-to-1 ratio (or 4-to-1 ratio, if the specification is included) of BoogiePL to x86 instructions (or, roughly, 2-to-1 or 3-to-1 ratio of nonexecutable annotation to executable code). This is about an order of magnitude fewer lines of annotation and script than related projects.^{5, 8} The choice of using Boogie/Z3 and first-order logic, rather than a

less-automated interactive proof system designed for non-first-order logics, is the key reason for this relatively small amount of annotation: since Verve's annotations are written in BoogiePL's first-order logic, the Z3 first-order logic theorem prover is able to automatically prove properties that require thousands of lines of manual scripts in interactive proof systems.

It takes 272s for the Boogie/Z3 tools to verify all the Nucleus components, including both the mark-sweep and copying collectors, on a 2.4GHz Intel Core2 with 4GB of memory. The vast majority of this time is spent on verifying the collectors; only 33s were required to verify the system's other components.

This small verification time gave us the freedom to experiment with different designs. For example, mid-way through the project, we switched from an implementation based on blocking Nucleus calls to an implementation based on non-blocking Nucleus calls. We were able to make such changes in days rather than months, because we could make minor changes to large, Nucleus-wide invariants and then run the automated theorem prover to quickly re-verify the entire Nucleus. In the end, the Verve design, implementation, and verification described in this paper took just 9 person-months, spread between two people.

7. RELATED WORK

The Verve project follows in a long line of operating system and runtime system verification efforts. More than 20 years ago, the Boyer-Moore mechanical theorem prover was used to verify a small operating system (Kit) and a small high-level language implementation,² although the Kit OS was too limited to run on commodity hardware and to support standard programming languages.

More recently, the seL4 project verified all of the C code for an entire microkernel.⁸ The seL4 microkernel contains 8700 lines of C code, substantially larger than earlier verified operating systems like Kit. This allows seL4 to implement realistic primitives for page table management, multithreading, capabilities, and message passing, so that it can securely run realistic user-mode applications, written in standard languages like C, on real hardware. The features supported by seL4 are comparable, though not identical, to those supported by Verve: seL4 pages are analogous to Verve objects, seL4 capabilities are analogous to Verve object references, seL4 messages are analogous to Verve method invocations, and seL4 threads are similar to Verve threads. The verified seL4 microkernel is substantially larger than the verified Verve Nucleus (8700 lines of C vs. 1400 x86 instructions). On the other hand, the verification effort required by seL4 was larger than the effort required by Verve: they report 20 person-years of research devoted to developing their proofs, including 11 person-years specifically for the seL4 code base. The proof required 200,000 lines of Isabelle scripts—a 20-to-1 script-to-code ratio. We hope that while seL4 demonstrates that realistic microkernels are within the reach of interactive theorem proving, Verve demonstrates that automated theorem proving can provide a less time-consuming alternative to interactive theorem proving for realistic systems software verification.

The FLINT project sets an ambitious goal to build foundational certified machine code, where certification produces a proof about an executable program, and a very small proof checker can be used to verify the proof.⁵ Such a system would have a much smaller trusted computing base than Verve. So far, such a foundational approach has been labor-intensive, limiting the scope of the current implementation, which currently does not support standard programming languages and does not have more than a few hundred lines of application code. We hope that future advances in theorem proving technology will combine foundational proofs with more automated verification.

8. CONCLUSION AND FUTURE WORK

Using a combination of TAL and automated theorem proving, we have completely verified the safety of Verve at the assembly-language level, and completely verified the correctness (excepting termination) of Verve's Nucleus at the assembly-language level. So what happens when we boot and run Verve? Since it is verified, did it run perfectly (or at least safely?) every time we ran it? Almost—the good news is that as far as we know, every execution of Verve has run in accordance with its specification. The bad news is that the specification itself may contain bugs, or at least may contain unwarranted assumptions about the world that Verve interacts with. In fact, we did encounter two violations of type safety, due to violations of the assumptions made by the specification. First, Verve initially included an off-the-shelf, unverified debugger stub, written in C++. The presence of the C++ code undermined Verve's assumptions about memory, so we eventually decided to banish the debugger stub from Verve. Second, a linking issue caused Bartok's relocation information for GC tables to get dropped, resulting in incorrect return addresses in the GC tables at run time. After these two issues were resolved, Verve ran without incident.

Compared to the traditional operating system development, where vast numbers of bugs are discovered only at run time, the number of bugs encountered at run time in Verve has been extremely small. Nevertheless, the fact that any bugs existed is a good motivation for trying to reduce the size and complexity of the specification, for reducing the number of trusted components (such as the debugger stub and the linker), and for more systematically testing the specification.^{7, 8} It also motivates the idea of verifying the verification tools themselves: verifying the TAL checker (which currently contains over 12,000 lines of trusted code³) against a small specification would particularly help to reduce Verve's trusted computing base.

Based on the core Verve features described in this paper, we have implemented or ported approximately 35,000 lines of safe C# code to run on top of the Verve kernel and Nucleus. This C# code, which Bartok compiles to verifiable TAL code, includes various System libraries from .NET and support for networking protocols like ARP, IP, and UDP. To enable running networking protocols safely on real networking hardware, we recently extended Verve's

Nucleus to support a large class of PCI devices, using an IO memory management unit (IO/MMU) to protect Verve's memory from errant devices. These extensions added less than 400 assembly-language instructions to the Nucleus, and required less than 1000 lines of BoogiePL code and just a couple man-months to implement. While many challenges remain for future work (such as verified incremental garbage collection¹⁰ and support for multicore machines), our experience so far gives us confidence that Verve's approach will scale to larger systems and more features without requiring excessive human effort and without sacrificing type safety.

Acknowledgments

We would like to thank Jeremy Condit, Galen Hunt, Ed Nightingale, Don Porter, Shaz Qadeer, Rustan Leino, Juan Chen, Gregory Malecha, and David Tarditi for their suggestions and assistance. C

References

1. Barnett, M., Chang, B.-Y.E., DeLine, R., Jacobs, B., Leino, K.R.M. Boogie: A modular reusable verifier for object-oriented programs. In *Formal Methods for Components and Objects (FMCO)* (Amsterdam, the Netherlands, 2006), volume 4111.
2. Bevier, W.R., Hunt Jr., W.A., Moore, J.S., Young, W.D. An approach to systems verification. *J. Autom. Reason.* 5, 4 (1989), 411–428.
3. Chen, J., Hawblitzel, C., Perry, F., Emmi, M., Condit, J., Coetzee, D., Pratikakis, P. Type-preserving compilation for large-scale optimizing object-oriented compilers. *SIGPLAN Not.* 43, 6 (2008), 183–192.
4. de Moura, L.M., Bjørner, N. Z3: An efficient SMT solver. In *TACAS* (2008), 337–340.
5. Feng, X., Shao, Z., Dong, Y., Guo, Y. Certifying low-level programs with hardware interrupts and preemptive threads. In *PLDI* (2008), 170–182.
6. Ford, B., Hibler, M., Lepreau, J., McGrath, R., Tullmann, P. Interface and execution models in the Fluke kernel. In *OSDI* (1999), 101–115.
7. Hawblitzel, C., Petrank, E. Automated verification of practical garbage collectors. In *POPL* (2009), 441–453.
8. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe et al. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)* (Big Sky, MT, Oct. 2009), ACM, 207–220.
9. Liedtke, J., Elphinstone, K., Schönberg, S., Härtig, H., Heiser, G., Islam, N., Jaeger, T. Achieved IPC performance (still the foundation for extensibility). In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)* (Cape Cod, MA, May 5–6, 1997).
10. McCreight, A., Shao, Z., Lin, C., Li, L. A general framework for certifying garbage collectors and their mutators. In *PLDI* (2007), 468–479.
11. Morrisett, G., Walker, D., Crary, K., Glew, N. From System F to typed assembly language. In *POPL '98: 25th ACM Symposium on Principles of Programming Languages* (Jan. 1998), 85–97.
12. Turing, A. Checking a large routine. In *The Early British Computer Conferences*, MIT Press, Cambridge, MA, 1989, 70–72.

Jan Yang (jeanyang@csail.mit.edu), Massachusetts Institute of Technology, Cambridge, MA.

Chris Hawblitzel (chris.hawblitzel@microsoft.com), Microsoft Research, Redmond, WA.

Technical Perspective

Anonymity Is Not Privacy

By Vitaly Shmatikov

WE LIVE IN an era of data abundance. Every aspect of our online and offline behavior—every click on a Web site, every relationship in online social networks, every bit of information we disclose about ourselves—is captured and analyzed by multiple entities, ranging from Internet service providers to advertising agencies to credit bureaus. With this dramatic increase in data collection, the companies holding our data face the responsibility for protecting our privacy, especially as they sell and exchange information about us.

Existing privacy protection technologies are overwhelmingly based on anonymization. They remove a few data attributes (such as names) that could be used to identify individuals and consider the resulting anonymized datasets safe from privacy violations. This approach is pervasive in academic literature, as well as industry practices. Whether it's the chief privacy officer of a major online social network testifying to a U.S. Senate committee that there is a critical distinction between the use of information in "personally identifiable form" and the use, sharing, and dissemination of information in "non-personally identifiable form," or a popular Web site informing its customers that it shares non-personally identifiable information about them with hundreds of advertisers, the safe-keepers of our data act as if anonymity were equivalent to privacy. But is it, really?

To build meaningful protections for sensitive individual data, we must ask the right questions. What does it mean to compromise privacy? How can a potential adversary access and/or influence the data, both before and after anonymization? What are the adversary's capabilities, and what information might she employ to reverse anonymity? Unfortunately, many existing privacy technologies suffer from a certain poverty of imagination. For example, they assume the only way to reidentify anonymized re-

The following paper is an object lesson in how to do data privacy research.

cords is to link them with an external dataset by matching common demographic attributes. As a consequence, anonymization is easily broken by creative adversaries who use a different attack model.

The following paper by Lars Backstrom, Cynthia Dwork, and Jon Kleinberg is a landmark in privacy research because it asks all of the above questions and gives unexpected answers. The authors demonstrate fragility of data anonymization, invent several new techniques for reidentifying anonymized nodes in social networks, and radically change our understanding of what constitutes personally identifiable information.

Their first contribution is to investigate the meaning of anonymity in graph-structured data, which are very different from relational datasets traditionally considered in privacy research. They focus on online social networks, but their results apply broadly to telephone call graphs, survey data, and, in general, almost any dataset containing information about relationships between people.

Their second contribution is the insight that the basic topological structure of the social graph can act as an identifier. They show that patterns of social links—whether arising naturally or artificially introduced into the social network by the adversary—tend to be unique and efficiently recognizable even in a completely ano-

nymized graph, yet without knowing the pattern, it is difficult to determine whether the graph contains such a structure. This idea is very powerful because graph structure is not a "personally identifiable" attribute by any meaning of the term. Nevertheless, Backstrom et al. show how it can be used to reidentify (sub)graphs that have been anonymized according to the best legal standards and satisfy the strongest anonymity properties.

Their third contribution is a new class of attacks on anonymity; in particular, an active attack in which the adversary deliberately introduces random links into the social network so that the resulting subgraph can be recognized even after all information about identities has been erased from the network. Existing privacy technologies fail to account for the possibility that the adversary may influence the data prior to anonymization and thus do not provide a defense against this threat.

The era of data abundance is bringing new kinds of sensitive data about individuals, new understanding of privacy risks, new attacks, and new defenses. This work provides us with valuable insights in all of these areas. By showing that the basic structure of our social relationships can be as identifying as a name, they debunk the naive belief that simple removal of identifiers renders the data non-personally identifiable. The authors carry out a rigorous theoretical analysis of anonymity in social networks (including interesting connections to graph theory) and accompany it by the empirical evaluation of their reidentification techniques on a large, real-world social network of the LiveJournal blogging service. Their paper is an object lesson in how to do data privacy research. It should be required reading for anyone interested in this area. 

Vitaly Shmatikov (shmat@cs.utexas.edu) is an associate professor at the University of Texas at Austin.

© 2011 ACM 0001-0782/11/12 \$10.00

Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Steganography

By Lars Backstrom, Cynthia Dwork, and Jon Kleinberg

Abstract

In a social network, nodes correspond to people or other social entities, and edges correspond to social links between them. In an effort to preserve privacy, the practice of anonymization replaces names with meaningless unique identifiers. We describe a family of attacks such that even from a single anonymized copy of a social network, it is possible for an adversary to learn whether edges exist or not between specific targeted pairs of nodes.

1. INTRODUCTION

1.1. Anonymized social networks

Digital traces of human social interactions can now be found in a wide variety of online settings, and this has made them rich sources of data for large-scale studies of social networks. While a number of these online data sources are based on publicly crawlable blogging and social networking sites, where users have explicitly chosen to publish their links to others, many of the most promising opportunities for the study of social networks are emerging from data on domains where users have strong expectations of privacy—these include email, phone, and messaging networks, as well as the link structure of closed (i.e., “members-only”) online communities. As a useful working example, consider a “communication graph,” in which nodes are email addresses, and there is a directed edge (u, v) if u has sent at least a certain number of email messages or instant messages to v , or if v is included in u ’s address book. Here, we will be considering the “purest” form of social network data, in which there are simply nodes corresponding to individuals and edges indicating social interaction, without any further annotation such as timestamps or textual data.

In designing studies of such systems, one needs to set up the data to protect the privacy of individual users while preserving the global network properties. This is typically done through anonymization, a simple procedure in which each individual’s “name”—for example, email address, phone number, or actual name—is replaced by a random user ID, but the connections between the (now anonymized) people—encoding who spoke together on the phone, who corresponded with whom, or who instant-messaged whom—are revealed. The motivation behind anonymizing

is roughly as follows: while the social network labeled with actual names is sensitive and cannot be released, there may be considerable value in allowing researchers to study its structure. For such studies, researchers are not specifically interested in “who” corresponds to each node, but in the properties of the graph, such as its connectivity, node-to-node distances, frequencies of small subgraphs, or the extent to which it can be clustered. Anonymization is thus intended to exactly preserve the pure unannotated structure of the graph while suppressing the “who” information.

Can this work? The hope is that being handed an anonymized picture of a social network—just a graph with a random identifier attached to each node—is roughly akin to being given the complete social network of Mars, with the true Martian names attached to the nodes. Intuitively, the names are meaningless to earth-dwellers: we do not “know” the Martians, and it is completely irrelevant to us whether a given node in the graph is labeled “Groark” or “Zoark.” The difficulty with this metaphor, of course, is that anonymous social network data almost never exists in the absence of outside context, and an adversary can potentially combine this knowledge with the observed structure to begin compromising privacy, de-anonymizing nodes, and even learning the edge relations between explicitly named (de-anonymized) individuals in the system. Moreover, such an adversary may in fact be a user (or set of users) of the system that is being anonymized.

For distinguishing among ways in which an adversary might take advantage of context, it is useful to consider an analogy to the distinction between passive attacks and active attacks in cryptanalysis—that is, between attacks in which an adversary simply observes data as it is presented, and those in which the adversary actively tries to affect the data to make it easier to decipher. In the case of anonymized social networks, passive attacks are carried out by individuals who try to learn the identities of nodes only after the anonymized network has been released.

A previous version of this paper was published in the *Proceedings of the 16th International Conference on World Wide Web: WWW2007* (Banff, Alberta, Canada, May 8–12, 2007).

In contrast, an adversary in an active attack tries to compromise privacy by strategically creating new user accounts and links before the anonymized network is released so that these new nodes and edges will then be present in the anonymized network.

1.2. The present work: Attacks on anonymized social networks

In this paper, we present both active and passive attacks on anonymized social networks, showing that both types of attacks can be used to reveal the true identities of targeted users, even from just a single anonymized copy of the network, and with a surprisingly small investment of effort by the attacker.

We describe active attacks in which an adversary chooses an arbitrary set of users whose privacy it wishes to violate, creates a small number of new user accounts with edges to these targeted users, and creates a pattern of links among the new accounts with the goal of making it stand out in the anonymized graph structure. The adversary then efficiently finds these new accounts together with the targeted users in the anonymized network that is released. At a theoretical level, the creation of $O(\sqrt{\log n})$ nodes by the attacker in an n -node network can begin compromising the privacy of arbitrary targeted nodes, with high probability for any network; in experiments, we find that on a 4.4-million-node social network, the creation of 7 nodes by an attacker (with degrees comparable to those of typical nodes in the network) can compromise the privacy of roughly 2400 edge relations on average. Moreover, experimental evidence suggests that it may be very difficult to determine whether a social network has been compromised by such an active attack.

We also consider passive attacks, in which users of the system do not create any new nodes or edges—they simply try to find themselves in the released network and from this to discover the existence of edges among users to whom they are linked. In the same 4.4-million-node social network dataset, we find that for the vast majority of users, it is possible for them to exchange structural information with a small coalition of their friends and subsequently uniquely identify the subgraph on this coalition in the ambient network. Using this, the coalition can then compromise the privacy of edges among pairs of neighboring nodes.

There are some obvious trade-offs between the active and passive attacks. The active attacks have more potent effects, in that they are guaranteed to work with high probability in any network (they do not force users to rely on the chance that they can uniquely find themselves after the network is released), and the attacker can choose any users it wants to target. On the other hand, while the passive attack can only compromise the privacy of users linked to the attacker, it has the striking feature that this attacker can simply be a user of the system who indulges his or her curiosity; there is no observable “wrongdoing” to be detected. Moreover, since we find in practice that the passive attack will succeed for the majority of the population, it says in effect that most people in a large social network have laid the groundwork for a privacy-breaching attack simply through their everyday actions, without even realizing it.

These trade-offs naturally suggest the design of hybrid “semi-passive” attacks, in which a user of the system creates no new accounts but simply creates a few additional out-links to targeted users before the anonymized network is released. As we show later, this can lead to privacy breaches on a scale approaching that of the active attack, without requiring the creation of new nodes.

In the next section, we provide some background and context for our work in terms of the broader area of data privacy. We then present our two main classes of active attacks on anonymized social networks; we refer to them as *walk-based attacks* and *cut-based attacks*, with the names reflecting the underlying techniques being used. We then describe the use of passive attacks and conclude with a general discussion.

2. RELATED WORK

This work fits within a growing literature that has considered ways in which private online data can be divulged against users’ wishes, via carefully devised privacy-breaching attacks. Such attacks have been based on a variety of features in the data; for example, the queries entered by users into search engines can be used to uniquely identify them,¹⁷ and the writing styles of users in online discussion can likewise be used to find the same person writing under different pseudonyms.²² Temporal data can also be an effective feature in privacy-breaching attacks: since it is unlikely for two users to perform a nontrivial set of actions at almost exactly the same sets of times, the sequence of times at which a user performs these actions becomes a type of identifying signature.²⁰

We note that in our case, both the passive and active attackers do not have access to highly resolved data like timestamps or other textual or numerical attributes; they can only use the binary information about who links to whom, without other node attributes, and this makes their task more challenging. Indeed, the secret subgraph H constructed as part of our attacks can be thought of as a kind of *structural steganography*, hiding secret messages for later recovery using just the social structure of G .

In this way, our approach can be seen as a step toward understanding how fundamental techniques of data privacy (see, e.g., Dwork⁹ and the references therein) can inform how we think about the protection of even the most skeletal social network data. We discuss this further in the final section.

In the time since the conference proceedings version of our work appeared, there has been continued research exploring mechanisms by which private data can be revealed online. Concurrent with our work, Hay et al.¹⁵ considered a set of methods for identifying nodes in anonymized social networks by looking at successively larger neighborhoods of a node. More recently, Narayanan and Shmatikov²¹ have shown how access to multiple networks containing overlapping sets of people can enable approaches to de-anonymization based on approximately aligning the portions of the networks that overlap.

In a related but different direction, several lines of recent work have shown how the principle of *homophily*—that neighbors in social networks have similar characteristics—can be used to discover private information: even if a user

keeps profile information private, it can often be estimated from attributes in public profiles maintained by his or her friends.^{16, 19, 25} Finally, recent work shows how social links themselves can be discovered from information about the times and places at which individuals perform activities; using data from a photo-sharing site, Crandall et al.⁶ showed that if two users uploaded photos from approximately the same location at approximately the same time on multiple occasions, there was a sharply increased probability that they were linked on the site's social network.

3. THE WALK-BASED ATTACK

To set the stage for our first active attack, we begin with some definitions and notation. We assume the social network is an n -node graph $G = (V, E)$, representing interactions in an online system. Nodes correspond to user accounts and an edge (u, v) indicates that u has communicated with v (again, consider the example of an email or instant messaging network). The attacks become easier to carry out if the released graph data is directed; for most of the paper, we will therefore consider the harder case of undirected graphs, in which we assume that the curator of the data—the agent that releases the anonymized network—eliminates the directions on the edges.

3.1. Description of the attack

Let us consider the problem from the perspective of the attacker. For ease of presentation, we begin with a slightly simplified version of the attack and then show how to extend it to the attack we really use. Recall that as an attacker, our basic approach is to create a set of new user accounts with links among them that will “stand out” when the anonymized graph is released. Thus, we first choose a set of $k = \Theta(\log n)$ named users, $W = \{w_1, \dots, w_k\}$, that we wish to target in the network—we want to learn all the pairs (w_i, w_j) for which there are edges in G . We create a set of k new user accounts, $X = \{x_1, \dots, x_k\}$, which will appear as nodes in the system. We include each undirected edge (x_i, x_j) independently with probability $1/2$. This produces a random graph H on X .

We also create an edge (x_i, w_i) for each i . (In terms of the underlying social network, this involves having x_i send w_i a message, or include w_i in an address book, or some other activity depending on the nature of the network.) For describing the basic version of the attack, we also assume that, because the account x_i corresponds to a fake identity, it will not receive messages from any node in $G - H$ other than potentially w_i , and thus will have no link to any other node in $G - H$. We will see later that the attack can be made to work even when this latter assumption does not hold.

When the anonymized graph G is released, we need to find our copy of H , and to correctly label its nodes as $x_1, \dots, x_k$. Having found these nodes, we then find w_i as the unique node in $G - H$ that is linked to x_i . We thus identify the full labeled set W in G , and we can simply read off the edges between its elements by consulting G .

It is worth noting that this type of attack only involves the use of completely innocuous operations in the context of the system being compromised—the creation of new

accounts and the creation of links to existing accounts. In this sense, while the active attacker's aims are nefarious (and, in almost any imaginable scenario, prohibited either by research ethics guidelines or the terms of service of the system, or both), none of the individual steps from which the attack is constructed could be viewed at a syntactic level as “breaking into” parts of the system where it is not allowed.

A number of technical ingredients are needed in order to make this attack work based on whether certain subgraphs have the same structure as each other and whether they have any internal symmetries. To express such questions, we use the following terminology. For a set of nodes S , we let $G[S]$ denote the subgraph of G induced by the nodes in S . An *isomorphism* between two sets of nodes S and S' in G is a one-to-one correspondence $f: S \rightarrow S'$ that maps edges to edges and non-edges to non-edges: (u, v) is an edge of $G[S]$ if and only if $(f(u), f(v))$ is an edge of $G[S']$. In this case, $G[S]$ and $G[S']$ are *isomorphic*—they are the same graph up to relabeling. An *automorphism* is an isomorphism from a set S to itself—a relabeling of the nodes $f: S \rightarrow S$ that preserves graph's structure. An automorphism f is *nontrivial* if it is not the identity function.

Thus, the construction of H succeeds if

- (i) There is no $S \neq X$ such that $G[S]$ and $G[X] = H$ are isomorphic.
- (ii) The subgraph H can be efficiently found, given G .
- (iii) The subgraph H has no nontrivial automorphisms

If (i) holds, then any copy of H we find in G must in fact be the one we constructed; if (ii) holds, then we can in fact find the copy of H quickly; and if (iii) holds, then once we find H , we can correctly label its nodes as $x_1, \dots, x_k$, and hence find $w_1, \dots, w_k$.

The full construction is almost as described above, with the following three additions. First, the size of the targeted set W can be larger than k . The idea is that rather than connecting each w_i with just a single x_i , we can connect it to a subset $N_i \subseteq X$, as long as w_i is the only node in $G - H$ that is attached to precisely the nodes in N_i —this way w_i will still be uniquely identifiable once H is found. Second, we will explicitly randomize the number of links from each x_i to $G - H$, to help in finding H . And third, to recover H , it is helpful to be able to traverse its nodes in order $x_1, x_2, \dots, x_k$. Thus, we deterministically include all edges of the form (x_i, x_{i+1}) and randomly construct all other edges.

The Construction of H . With this informal discussion in mind, we now give the full specification of the attack.

- (1) We choose $k = (2 + \delta) \log_2 n$, for a small constant $\delta > 0$, to be the size of X . We choose two constants $d_0 \leq d_1 = O(\log n)$, and for each $i = 1, 2, \dots, k$, we choose an *external degree* $\Delta_i \in [d_0, d_1]$ specifying the number of edges x_i will have to nodes in $G - H$. Each Δ_i can be chosen arbitrarily, but in our experiments with the algorithm, it works well simply to choose each Δ_i independently and uniformly at random from the interval $[d_0, d_1]$.
- (2) Let $W = \{w_1, w_2, \dots, w_b\}$ be the users we wish to target,

for a value $b = O(\log^2 n)$. We also choose a small integer constant c ($c = 3$ will suffice in what follows). For each targeted node w_j , we choose a set $N_j \subseteq \{x_1, \dots, x_k\}$ such that all N_j are distinct, each N_j has size at most c , and each x_i appears in at most Δ_i of the sets N_j . (This gives the true constraint on how large $b = O(\log^2 n)$ can be.) We construct links to w_j from each $x_i \in N_j$.

- (3) Before generating the random internal edges of H , we add arbitrary further edges from H to $G - H$ so that each node x_i has exactly Δ_i edges to $G - H$. We construct these edges subject only to the following condition: for each $j = 1, 2, \dots, b$, there should be no node in $G - H$ other than w_j that is connected to precisely the nodes in N_j .
- (4) Finally, we generate the edges inside H . We include each edge (x_i, x_{i+1}) , for $i = 1, \dots, k-1$, and we include each other edge (x_i, x_j) independently with probability $1/2$. Let Δ'_i be the degree of x_i in the full graph G (this is Δ_i plus its number of edges to other nodes in X).

This concludes the construction. As a first fact, we note that standard results in random graph theory (see, e.g., Bollobás⁵) imply that with high probability, the graph H has no nontrivial automorphisms. We will assume henceforth that this event occurs, that is, that H has no nontrivial automorphisms.

We also note that the attack will work even if multiple copies of the construction are carried out simultaneously. That is, we can choose different sets of nodes to attack, $W_1, W_2, \dots, W_\ell$, each of size $\Theta(\log n)$; for each W_i , we add a distinct set of new nodes X_i to the graph G , building a graph H_i on each X_i with the different random constructions performed independently.

Efficiently Recovering H Given G . When the graph G is released, we want to identify H : that is, we want to find the subset of nodes of G that correspond to the set of nodes $x_1, x_2, \dots, x_k$ of H . Since we have constructed H to contain a path through the nodes $x_1, x_2, \dots, x_k$, we will search along k -node paths in G , looking for a k -node path P for which the edges induced among the nodes of P have precisely the structure of H .

At a high level (ignoring issues of efficiency, which we discuss next), our algorithm works simply as follows. For every k -node path $P = \{y_1, y_2, \dots, y_k\}$ in G , we visit the nodes of P in order, declaring P to have *failed* in the comparison to H as soon as we reach a node y_i that fails one of the following two tests.

- (i) **A degree test:** The degree of node y_i should be equal to the value Δ'_i , which we know to be the degree of node x_i in G .
- (ii) **An internal structure test:** For each $j < i$, there should be an edge (y_j, y_i) in G if and only if (x_j, x_i) is an edge of H .

Finally, if we reach the end of the path P without any of its nodes having failed either of these tests, then by definition we have found a copy of H in G . (As we note later, the degree test is not necessary either for the correctness of the algorithm or the bound on the worst-case running time, but it is extremely useful in practice.)

There will typically be an extremely large number of distinct k -node paths in G , so we need to organize the computation carefully in order for the search algorithm to run efficiently. We do this as follows:

- First, we loop over all nodes v of G , trying each as the candidate starting point y_1 for the path P (the node that will correspond to x_1 in H). If the degree of v is not equal to Δ'_1 , then we skip v in this process, since it cannot correspond to the node x_1 in H .
- For each node v of degree Δ'_1 in G , we will organize all paths originating at $y_1 = v$ into a search tree $\mathcal{T}_v$ in the natural way: each node α in $\mathcal{T}_v$, at depth l , will correspond to an l -node path in G , starting at $y_1 = v$, that has not yet failed any of the degree or internal structure tests.
- We grow $\mathcal{T}_v$ one level at a time. For each node α of $\mathcal{T}_v$, at depth l , corresponding to an l -node path $P = \{v = y_1, y_2, \dots, y_l\}$ in G , we first check whether y_l passes the degree and internal structure tests. If it does not, we declare α to be a leaf of $\mathcal{T}_v$. If it does pass, then we create a new child α' of α in $\mathcal{T}_v$ for each way of extending P by adjoining a neighbor of y_l that does not already appear on P .

If $\mathcal{T}_v$ ever acquires a node at depth k , then this corresponds to a k -node path in G that has passed all of our tests, and hence is a copy of H . Conversely, if there is such a path P originating at v , then our tree-growing procedure will continue adding nodes to $\mathcal{T}_v$ until it produces a node at depth k corresponding to P .

Note that the total running time of this algorithm is only a small factor larger than the total number of nodes in all search trees $\mathcal{T}_v$ (summed over all nodes v in G), and so a key issue in the analysis is to show that with high probability, the total number of nodes in all $\mathcal{T}_v$ is not too large.

3.2. Analysis

To prove the correctness and efficiency of the attack, we show two things: with high probability, the construction produces a unique copy of H in G , and with high probability, the total number of nodes in all search trees $\mathcal{T}_v$ in the recovery algorithm does not grow too large.

The formal statements of these two claims are as follows.

- **Uniqueness.** Let $k \geq (2 + \delta) \log_2 n$ for an arbitrary positive constant $\delta > 0$, and suppose we use the following process to construct an n -node graph G :

- (i) We start with an arbitrary graph G' on $n - k$ nodes, and we attach new nodes $X = \{x_1, \dots, x_k\}$ arbitrarily to nodes in G' .
- (ii) We build a random subgraph H on X by including each edge (x_i, x_{i+1}) for $i = 1, \dots, k-1$, and including each other edge (x_i, x_j) independently with probability $1/2$.

Then with high probability, there is no subset of nodes $S \neq X$ in G such that $G[S]$ is isomorphic to $H = G[X]$.

- **Efficiency.** For every $\epsilon > 0$, with high probability, the total number of nodes appearing in all the search trees $\mathcal{T}_v$ (over all v in G) is $O(n^{1+\epsilon})$.

While the proofs of these claims are somewhat involved, the basic idea underlying them is rooted in an argument formulated by Paul Erdős to prove a result in Ramsey Theory.^{5,11} In the simplest form of the argument, let us suppose we have an n -node graph G and a random k -node graph H on nodes $\{x_1, x_2, \dots, x_k\}$, with each edge (x_i, x_j) present independently with probability $1/2$. What is the probability that G contains a k -node subgraph that is isomorphic to H ? For any k -tuple of nodes $v_1, v_2, \dots, v_k$ in G , the probability that the subgraph of G on this k -tuple is isomorphic to H , under the mapping sending v_i to x_i , is precisely $2^{-\binom{k}{2}} = 2^{-k(k-1)/2}$, since the presence or absence of the random edge (x_i, x_j) has to match the presence or absence of (v_i, v_j) for each (i, j) pair. But there are fewer than n^k such k -tuples of nodes in G , and so the probability that *any* of them yields such an isomorphism is less than $n^k 2^{-k(k-1)/2}$. Now a direct calculation shows that once k exceeds $2 \log_2 n$, this probability shrinks rapidly to 0, and hence it is likely that there is no isomorphic copy of H in G .

This gives the central idea of the proofs, but the details become more complicated because the graph H in the active attack is necessarily being attached by edges to the graph G —and this creates the possibility of isomorphisms that create a second copy of H out of parts of the original H together with parts of the rest of G . Showing that this is unlikely to happen requires a more intricate argument.

It is important to stress, however, that the intricacy of the proofs is an aspect of the analysis, not of the algorithms themselves. The construction of H and the recovery algorithm have already been fully specified in the previous subsection, and they are quite simple to implement.

We conclude with some comments on the tests used in the recovery algorithm. Recall that as we build $\mathcal{T}_v$, we eliminate paths based on an internal structure check (do the edges among path nodes match those in H ?) and a degree check (do the nodes on the path have the same degree sequence as H ?). The proofs of our two main claims require just the internal structure check to prove uniqueness and to bound the size of $\mathcal{T}_v$, respectively, but it is important in practice that the algorithm use both checks: as the experiments in the next subsection will show, one can get unique subgraphs at smaller values of k , and with much smaller search trees $\mathcal{T}_v$, by including the degree tests. But it is interesting to note that since these theorems can be proved using only internal structure tests, the attack is robust at a theoretical level provided only that the attacker has control over the internal structure of X , even in scenarios where nodes elsewhere in the graph may link to nodes in X without the knowledge of the attacker. (In this case, we still require that the targeted nodes $w_j \in W$ are uniquely identifiable via the sets N_j and that all degrees in X remain logarithmic.)

3.3. Computational experiments

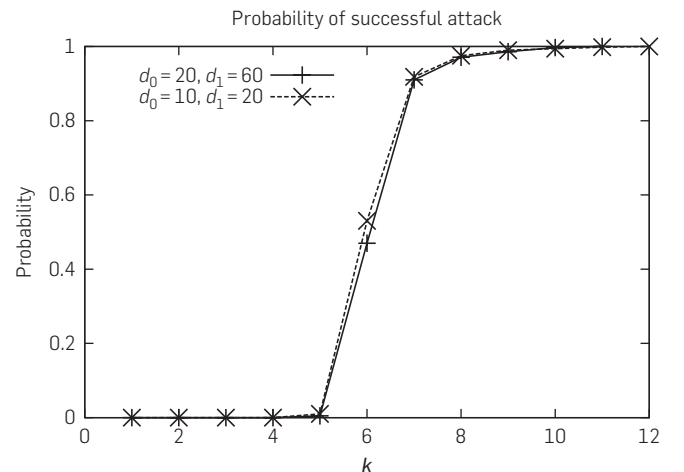
Social Network Data. We now describe computational experiments with the algorithm on real social network data drawn from an online setting. We find that the algorithm scales easily to several million nodes and produces efficiently findable unique subgraphs for values of k significantly smaller than the upper bounds in the previous subsections.

As data, we use the network of friendship links on the blogging site LiveJournal, constructed from a crawl of this site performed in February 2006. Each node in LiveJournal corresponds to a user who has made his or her blog public through the site; each user can also declare friendship links to other users. These links provide the edges of the social network we construct; they are directed, but we follow the principle of the previous subsections and convert them to undirected edges for purposes of the experiments. The LiveJournal data thus works well as a testbed; it has 4.4 million nodes and 77 million edges in the giant component of its undirected social network, and it exhibits many of the global structural features of other large online social networks. Finally, we emphasize that while LiveJournal has the right structure for our tests, it is not in reality an anonymous network—all the nodes in the network represent users who have chosen to publish their information on the Web.

We simulate anonymization by removing all the user names from the nodes; we then run our attack and investigate the ranges of parameters in which it successfully identifies targeted nodes. As a first question, we examine how often H can be found uniquely for specific choices of d_0, d_1 , and k . In our construction, we generate a random external degree Δ_i for each node x_i uniformly from $[d_0, d_1]$. We then create links to targeted nodes sequentially. Specifically, in iteration i we choose a new user w_i in $G - H$ to target; we then pick a minimal subset $X' \subseteq X$ that has not been used for any w_j for $j < i$, and where the degrees of nodes in X' are less than their randomly selected target degrees. We add an edge between w_i and each user in X' . We repeat this process until no such X' can be found. If, at the end of the process, some nodes in X have not yet reached their target degrees, we add edges to random nodes in G (and remove nodes from W so that no two nodes are connected to the same subset of X).

Uniqueness. We say the construction *succeeds* if H can be recovered uniquely. Figure 1 shows the success frequency

Figure 1. For two different choices of d_0 and d_1 , the value $k = 7$ gives the attack on the Live Journal graph a high probability of success. Both of these choices for d_0 and d_1 fall well within the degrees typically found in G .



for two different choices of d_0 and d_1 (the intervals $[10, 20]$ and $[20, 60]$), and varying values of k . We see that the success frequency is not significantly different for our two choices. In both cases the number of nodes we need to add to achieve a high success rate is very small—only 7. With 7 nodes, we can attack an average of 34 and 70 nodes for the smaller and larger degree choices, respectively.

We also note that the degree tests are essential for producing unique identifiability of H at such a small value of k . In fact, each of the 734 possible Hamiltonian graphs on 7 nodes actually occurs in the LiveJournal social network, so it is only because of its degree sequence in G that our constructed subgraph H is unique. (Our Uniqueness result does guarantee that a large enough H will be unique purely based on its internal structure; this is compatible with our findings since the analyzed bound of $(2 + \delta) \log_2 n$ is larger than the value $k = 7$ with which we are succeeding in the experiments.)

Efficient Recovery. In addition to being able to find H reliably, we must be able to find H quickly. We argued above that the total number of nodes in all search trees $\mathcal{T}_v$ would be sufficiently small that our search algorithm would be near-linear. In our experiments on the LiveJournal friendship graph, we find that, in practice, the total number of nodes in all $\mathcal{T}_v$ is not much larger than the number of nodes v whose degree in G is equal to Δ'_v . (Recall that we only build search trees for those v that have this degree.) For instance, when $d_0 = 10$ and $d_1 = 20$, there are an average of 70,000 nodes that have degree Δ'_v , while the total number of nodes in all search trees $\mathcal{T}_v$ is typically about 90,000.

Detectability. Our simple attack shows that simple anonymization does not preserve privacy of links. One might wonder about the *detectability* of the attack: can the curator of the data, who is releasing the anonymized version, not be able to discover and remove H ? The curator does not have access to the secret degree sequence or the edges within H and so cannot employ the same algorithm the attacker uses to discover H . However, if H were to stand out significantly in some other way, there might be an alternate means for finding it.

This subtle issue is worthy of more rigorous treatment; here, we provide the following indications that the subgraph H may be hard to discover. First is the simple fact that H has only 7 nodes, so it is difficult for any of its graph-theoretic properties to stand out with much statistical significance. Second, we describe some particular ways in which H does *not* stand out. To begin with, the internal structure of H is consistent with what is present in the network. For example, we have already mentioned that every 7-node Hamiltonian graph already occurs in LiveJournal, so this means that there are already subgraphs that exactly match the internal structure of H as an induced 7-node subgraph. (We are still able to find H because of the pattern of edges that connect nodes of H to nodes of $G - H$.) More generally, almost all nodes in LiveJournal are part of a very dense 7-node subgraph: If we look at all the nodes with degree at least 7, and consider the subgraph formed by those nodes and their 6 highest-degree neighbors, over 90% of such subgraphs have at least $11 > \frac{1}{2} \binom{7}{2}$ edges. These subgraphs are also almost all comparably well connected

to the rest of G .

4. THE CUT-BASED ATTACK

In the walk-based attack just presented, one needs to construct a logarithmic number of nodes in order to begin compromising privacy. On the other hand, we can show that at least $\Omega(\sqrt{\log n})$ nodes are needed in any active attack that requires a subgraph H to be uniquely identifiable with high probability, independent of both the structure of $G - H$ and the choice of which users to target.

It is therefore natural to try closing this gap between the $O(\log n)$ number of nodes used by the first attack and the $\Omega(\sqrt{\log n})$ lower bound required in any attack. With this in mind, we now describe our second active attack, the *cut-based attack*; it matches the lower bound by compromising privacy using a subgraph H constructed on only $O(\sqrt{\log n})$ nodes. While the bound for the cut-based attack is appealing from a theoretical perspective, there are several important respects in which the walk-based attack that we saw earlier is likely to be more effective in practice. First, the walk-based attack comes with a much more efficient recovery algorithm; and second, the walk-based attack appears to be harder for the curator of the data to detect (as the cut-based attack produces a densely connected component attached weakly to the rest of the graph, which is uncommon in many settings).

The Construction of H . We begin the description of the cut-based attack with the construction of the subgraph H .

- (1) Let b , the number of users we wish to target, be $\Theta(\sqrt{\log n})$, and let $w_1, w_2, \dots, w_b$ be these users. First, for $k = 3b + 3$, we construct a set X of k new user accounts, creating an (undirected) edge between each pair with probability $1/2$. This defines a subgraph H that will be in G .
- (2) Let $\delta(H)$ denote the minimum degree in H , and let $\gamma(H)$ denote the value of the minimum cut in H (i.e., the minimum number of edges whose deletion disconnects H). It is known that for a random graph H such as we have constructed, the following properties hold with probability going to 1 exponentially quickly in k : first, that $\gamma(H) = \delta(H)$; second, that $\delta(H) \geq (1/2 - \epsilon)k$ for any constant $\epsilon > 0$; and third, that H has no nontrivial automorphisms.⁵ In what follows, we will assume that all these properties hold: $\gamma(H) = \delta(H) \geq k/3 > b$, and H has no nontrivial automorphisms.
- (3) We choose b nodes $x_1, \dots, x_b$ in H arbitrarily. We create a link from x_i to w_i so that the edge (x_i, w_i) will appear in the anonymized graph G . Thus, b of the nodes of H each have a single edge to a node of $G - H$, while the other $k - b$ nodes of H have no edges to nodes of $G - H$.

A crucial property of H that we will use is the following: there are b edges in total that have one end in H and the other end in $G - H$; on the other hand, each node in H has more than b edges to other nodes of H .

Finally, we note that as with the walk-based attack in the

previous section, we can also carry out multiple copies of the present construction simultaneously, if desired, so as to attack multiple sets of targeted users $W_1, W_2, \dots, W_t$.

Efficiently Recovering H Given G . Now, when G is released, we identify the subgraph H and the targeted users $w_1, \dots, w_b$ using the following recovery algorithm.

- (1) We first compute the Gomory–Hu tree of G —this is an edge-weighted tree T on the node set V of G , such that for any $v, w \in V$, the value of the minimum $v - w$ cut in G is equal to the minimum edge weight on the $v - w$ path in T .¹³

Computing T is the most expensive step of the recovery algorithm, computationally. The best running time known for constructing a Gomory–Hu tree in a graph with n nodes and m edges is $O(mn)$ times a factor that is polynomial in $\log(m + n)$.³ This is a much larger worst-case bound than we have for the walk-based attack. On the other hand, computational experiments in Web graph analysis indicate that Gomory–Hu tree computations can in fact be made to scale to very large graphs in practice.¹²

- (2) We delete all edges of weight at most b from T , producing a forest T' . To find the set of nodes X we constructed, we iterate through all components of T' of size exactly k —let them consist of node sets $S_1, S_2, \dots, S_r$ —and for each such S_i we test whether $G[S_i]$ is isomorphic to H . These isomorphism tests can be done efficiently, even by brute force, since $k! = o(n)$. By adapting our proof of Uniqueness from the walk-based attack, we can show a form of uniqueness for H here too:

- *With high probability, there will be a single i such that $G[S_i]$ is isomorphic to H , and that S_i is equal to our set X of new nodes.*

- (3) Since H has no non-trivial automorphisms, from knowledge of S_i we can identify the nodes $x_1, \dots, x_b$ that we linked to the targeted users $w_1, \dots, w_b$, respectively. Hence we can identify the targeted users as well, which was the goal.

Some Specific Numbers for the Cut-Based Attack. It is useful to supplement the asymptotic results for the cut-based attack with some specific numbers. If the network G has 100 million nodes, then by creating 12 new user accounts we can succeed in identifying 3 chosen users in the system with probability at least 0.99. Creating 15 new user accounts leads to a microscopically small failure probability.

The calculation is as follows. We first generate 100 random 12-node graphs $H_1, \dots, H_{100}$, and see if any of them lacks nontrivial automorphisms and has a minimum cut of size at least 4. If any of them does, we choose one as our 12-node subgraph H . Computational experiments show that a random 12-node graph will have no nontrivial automorphism and $\gamma(H) \geq 4$ with probability roughly 0.25. Thus, with probability well over 0.999, one of the 100 graphs H_i will have this pair of properties. Now, if we use the i th of these random graphs in the construction, for a fixed i , then, applying the notation from the description of the attack above, there are at most 8333333 possible components S_j of size 12 in the

forest T' , one of which, say S^* , is our subgraph H . The probability that there even *exists* an $S_j \neq S^*$ that is isomorphic to H is bounded by $8333333 \cdot 12! < 2^{-66} < 6 \cdot 10^{-5}$. Hence the probability that *any* H_i will lead to non-uniqueness when attached to G is at most 0.006, and so in particular this holds for the H_i that we choose as H .

By way of comparison, the provable bounds for the walk-based attack require a number of new user accounts that is at least $2 \log_2 n$, which is approximately 53 when n is 100 million. On the other hand, as we have seen in our computational experiments, the walk-based attack appears to require fewer nodes in practice than the provable guarantees suggest, suggesting that further empirical comparison of these two attacks would be an interesting open question.

5. PASSIVE ATTACKS

In a passive attack, regular users are able to discover their locations in G using their knowledge of the local structure of the network around them. While there are a number of different types of passive attacks that could be implemented, here we imagine that a small coalition of passive attackers collude to discover their location. By doing so, they compromise the privacy of some of their neighbors: those connected to a unique subset of the coalition, and hence unambiguously recognizable once the coalition is found.

Here, we imagine that a coalition X of size k is initiated by one user who recruits $k - 1$ of his or her neighbors to join the coalition. (Other structures could lead to analogous attacks.) We assume that the users in the coalition know the edges among themselves—the internal structure of $H = G[X]$, using the terminology from the active attacks. We also assume that they know the names of their neighbors outside X . This latter assumption is reasonable in many cases: for example, if G is an undirected graph built from messages sent and received, then each user in X knows its incident edges.

The attack itself is analogous to the walk-based attack, except that the structure of H arises organically from the behavior of individuals using the system. A user x_1 selects $k - 1$ neighbors to form a coalition $X = \{x_1, x_2, \dots, x_k\}$. The coalition knows which edges (x_i, x_j) are in G and also the neighbors of each x_i in $G - X$. Once G is released, the coalition runs the search algorithm from the walk-based attack, with a minor modification due to the fact that H need not have a Hamiltonian path but instead has a single node connected to all others.

To help the passive attack succeed, we can incorporate a further optimization that was not explicitly used for the walk-based active experiments. For each nonempty set $S \subseteq \{1, 2, \dots, k\}$, let $g(S)$ denote the number of nodes in G that have edges to all the element of $\{x_i: i \in S\}$ and none of the elements of $\{x_i: i \notin S\}$. (In some places, we will abuse the notation for $g(\cdot)$ as follows: if U is a set of nodes in X rather than a set of indices, we will use $g(U)$ to denote the number of nodes in G that have edges to all elements of U and no elements of $X - U$.) Now, suppose we have a node a in a search tree T_v , corresponding to a path $y_1, y_2, \dots, y_l$ in G . For each $S \subseteq \{1, 2, \dots, l\}$, it should be the case that exactly $g(S)$

nodes of G are connected to all members of $\{y_i: i \in S\}$ and none of $\{y_i: i \notin S\}$; otherwise, $\{y_1, \dots, y_l\}$ cannot be the first l nodes of the copy of H in G .

Finally, once the coalition of users X finds itself, it can determine the identity of any user $w \notin X$ whose neighbor set S in X satisfies $g(S)=1$. (In this case, w is uniquely identified by the identities of its neighbors in X .)

Since the structure of H is not randomly generated, there is no *a priori* reason to believe that it will be uniquely findable or that the above algorithm will run efficiently. Indeed, for pathological cases of G and H , the problem is NP-Hard. However, we find on real social network data that the instances are not pathological and that subgraphs on small coalitions tend to be unique and efficiently findable.

The primary disadvantage of this attack in practice, as compared to the active attack, is that it does not allow one to compromise the privacy of arbitrary users. However, a natural extension is a *semi-passive attack* whereby a coalition of existing users colludes to attack specific users. To do this, the coalition X forms as described above with x_1 recruiting $k-1$ neighbors. Next, the coalition compares neighbor sets to find some set $S \subseteq X$ such that $g(S)=0$. Then, to attack a specific user w , each user in $\{x_i: i \in S\}$ adds an edge to w . Then, assuming that the coalition can uniquely find H , they will certainly find w as well.

Computational Experiments. Here, we consider the passive attack on the undirected version of the LiveJournal graph. For varying k , we consider a coalition of a user x_1 and his or her $k-1$ highest-degree neighbors. (We also consider the case where x_1 selects $k-1$ neighbors at random; the success rate here is similar.) We analyze the attack described above for a randomly chosen sample of users x_1 whose degree is at least $k-1$.

We find that even coalitions as small as three or four users can often find themselves uniquely, particularly when using the refined version of the algorithm. Figure 2 summarizes the success rates for different-sized coalitions based on both the “simple” algorithm using the internal structure of H and the degree sequence, as well as the “refined” algorithm that incorporates the function $g(S)$. With minimal preprocessing, G can be searched for a particular coalition almost immediately: On a standard desktop, it takes less than a tenth of a second, on average, to find a coalition of size 6.

At first glance, these results seem at odds with the results for the active attack in Figure 1, as the passive attack is producing a higher chance of success with fewer nodes. However, in the active attack, we limited the degrees of the users created so that H would be inconspicuous. In the passive attack, there is no such limit, and many users’ highest-degree neighbor has degree well over the limit of 60 that we imposed on the active attack; this makes it easier to find the resulting subgraph H . When we consider only those coalitions whose members all have degrees analogous to those in the active attack, the results are similar to the active attack.

As Figure 3 shows, the passive attack identifies relatively few nodes outside the coalition, compared to the active attack. However, with a semi-passive attack, we can greatly increase the number of users compromised, as indicated by Figure 3 (and recall that these users can be chosen arbitrarily

Figure 2. Probability of success for different coalition sizes in the LiveJournal graph, comparing a simple algorithm using only the degrees and internal structure of the coalition, and a more refined algorithm using the edges connecting H to $G-H$.

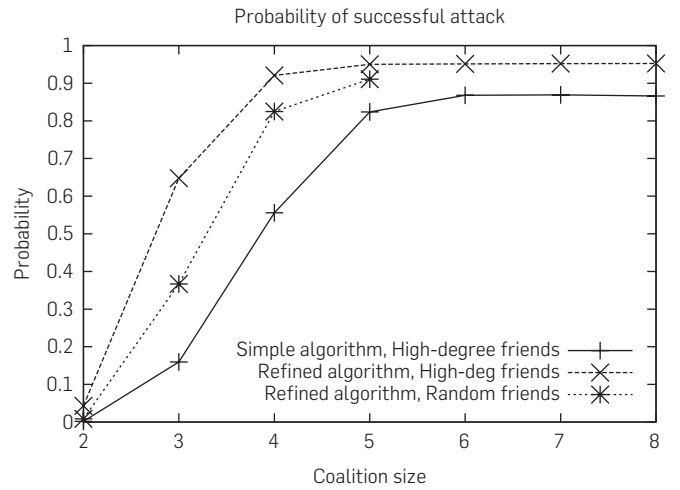
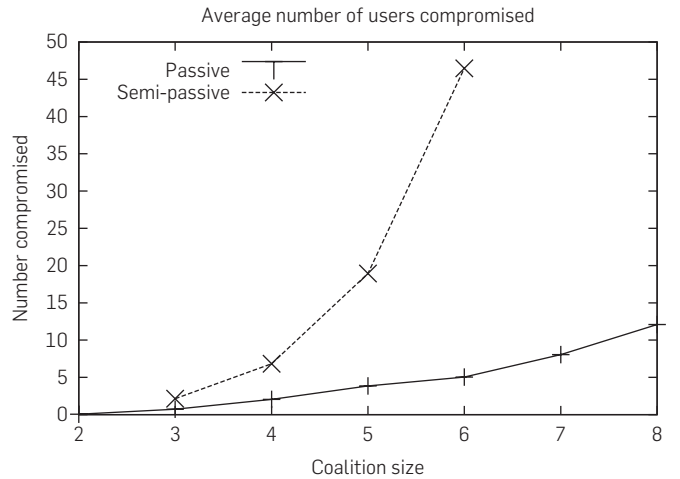


Figure 3. As the size of the coalition increases, the number of users in the LiveJournal graph compromised under the passive attack when the coalition successfully finds itself increases superlinearly. The number of users the semi-passive attack compromises increases exponentially.



by the coalition). Moreover, when the coalition is compromising as many users as possible, the semi-passive attack tends to have a higher success rate.

6. DISCUSSION

It is natural to ask what conclusions about private analysis of social network data should be drawn from this work. As noted at the outset, our work is not directly relevant to all settings in which social network data is used. For example, much of the research into online social networks is conducted on data collected from Web crawls, where users have chosen to make their network links public. There are also natural scenarios in which individuals work with

social network data under safeguards that are primarily legal or contractual, rather than computational, in nature—although even in such cases, there are compelling reasons why researchers covered by contractual relationships with a curator of sensitive data should still only publicly release the results of analyses carried out through a privacy mechanism to prevent the information in these analyses from implicitly compromising privacy. In cases such as these, where computational safeguards are not the primary focus, important questions of data utility versus privacy still arise, but these questions are not something our results directly address.

What our results do show is that one cannot rely on anonymization to ensure individual privacy in social network data, in the presence of parties who may be trying to compromise this privacy. And while one natural reaction to these results is to try inventing methods of thwarting the particular attacks we describe, we think this misses the broader point of our work: true safeguarding of privacy requires mathematical rigor, beginning with a clear description of what it means to compromise privacy, what are the computational and behavioral capabilities of the adversary, and to what information might it have access, now or in the future.

There is a growing literature to which we can turn for thinking about ensuring privacy in settings such as these. There has been extensive recent work on privacy-preserving data mining, beginning with Agrawal et al., Samarati, and Sweeney^{1,2,23,24} which rekindled interest in a field quiescent since the 1980s, and increasingly incorporating approaches from modern cryptography for describing and reasoning about information leakage.^{4,7,10,18} The notion of *e*-differential privacy gives very strong guarantees, independent of the auxiliary information and computational powers of the adversary (see Dwork et al.^{8,9,10}). This notion departs from previous ones by shifting away from comparing what can be learned about an individual with versus without the database, instead concentrating on how the database behaves with versus without the data of an individual.

A simple and general *interactive* mechanism for ensuring differential privacy is given in Dwork et al.¹⁰ In this mechanism, a question is posed, the exact answer is computed by the curator, and then a noisy version of the true answer is returned to the user. The advantage of interaction lies in the fact that accuracy must deteriorate with the number and complexity of questions asked (see Dinur and Nissim,⁷ *et sequelae*). In a noninteractive solution, the curator must produce an object that answers all *potential* future questions; interactive approaches answer only those questions actually asked.

A lively literature (see, e.g., Hardt and Rothblum¹⁴ and the references therein) explores the tradeoffs between accuracy, computation, and degree of differential privacy in answering very large numbers of *counting queries*, that is, questions of the form “How many people in the database satisfy property *P*?” In the context of a social network in which the goal is to protect the privacy of individual friendships, this captures questions of the form “How many edges (friendships) connect people with property *P* to people with property *Q*?” such as, “How many friendships are there between people who went to Princeton High School and Cornell graduates?”

The only privacy definition of which we are aware that

protects against arbitrary auxiliary information is differential privacy. Further progress on differentially private analysis of social networks awaits compelling and precise analytical goals.

Acknowledgments

This work has been supported in part by NSF grants CCF-0325453, IIS-0329064, CNS-0403340, and BCS-0537606, by the Institute for the Social Sciences at Cornell, and by the John D. and Catherine T. MacArthur Foundation. 

References

1. Agrawal, D., Aggarwal, C. On the design and quantification of privacy preserving data mining algorithms. In *ACM Symposium on Principles of Database System* (2001).
2. Agrawal, R., Srikant, R. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD* (2000).
3. Bhalgat, A., Hariharan, R., Kavitha, T., Panigrahi, D. An $\tilde{O}(mn)$ Gomory-Hutree construction algorithm for unweighted graphs. In *Proceedings of ACM Symposium on Theory of Computing* (2007).
4. Blum, A., Dwork, C., McSherry, F., Nissim, K. Practical privacy: the SuLQ framework. In *ACM PODS* (2005).
5. Bollobás, B. *Random Graphs*. Cambridge University Press, Cambridge, U.K., 2001.
6. Crandall, D., Backstrom, L., Cosley, D., Suri, S., Huttenlocher, D., Kleinberg, J. Inferring social ties from geographic coincidences. *Proc. Natl. Acad. Sci.*, 107 (2010).
7. Dinur, I., Nissim, K. Revealing information while preserving privacy. In *Symposium on Principles of Database System* (2003).
8. Dwork, C. Differential privacy. *Proceedings of International Colloquium on Automata, Languages and Programming* (2006).
9. Dwork, C. A Firm Foundation for Private Data Analysis. *CACM* 54, 1 (2011).
10. Dwork, C., McSherry, F., Nissim, K., Smith, A. Calibrating noise to sensitivity in private data analysis. In *Proceedings of Theory of Cryptography Conference* (2006).
11. Erdős, P. Some remarks on the theory of graphs. *Bull. AMS* 53 (1947).
12. Flake, G., Tarjan, R., Tsioutsoulakis, K. Graph clustering and min cut trees. *Internet Math.* 1 (2003).
13. Gomory, R., Hu, T.C. Multi-terminal network flows. *J. Soc. Ind. Appl. Math.* 9 (1961).
14. Hardt, M., Rothblum, G. A multiplicative weights mechanism for privacy-preserving data analysis. In *Proceedings of FOGS* (2010).
15. Hay, M., Miklau, G., Jensen, D., Towsley, D., Weis, P. Resisting structural re-identification in anonymized social networks. In *Proceedings of the VLDB Endowment*, 1 (2008).
16. Jernigan, C., Mistree, B. Gaydar: Facebook friendships expose sexual orientation. *First Monday* 14 (2009).
17. Kumar, R., Novak, J., Pang, B., Tomkins, A. On anonymizing query logs via token-based hashing. In *Proceedings of the 16th International World Wide Web Conference* (2007).
18. Mishra, N., Sandier, M. Privacy via pseudorandom sketches. In *ACM Symposium on Principles of Database System* (2006).
19. Mislove, A., Viswanath, B., Gummadi, P.K., Druschel, P. You are who you know: inferring user profiles in online social networks. In *ACM WSDM* (2010).
20. Narayanan, A., Shmatikov, V. Robust de-anonymization of large sparse datasets (How to break anonymity of the Netflix prize dataset). In *Proceedings of the IEEE Symposium on Security and Privacy* (2008).
21. Narayanan, A., Shmatikov, V. De-anonymizing social networks. In *Proceedings of the IEEE Symposium on Security and Privacy* (2009).
22. Novak, J., Raghavan, P., Tomkins, A. Anti-aliasing on the web. *Proceedings of the 13th International World Wide Web Conference* (2004).
23. Samarati, P. Protecting respondents' identities in microdata release. *IEEE TKDE* 13 (2001).
24. Sweeney, L. *k*-anonymity: a model for protecting privacy. *Intl. J. Uncertainty Fuzziness Knowledge-Based Systems* 10 (2002).
25. Zheleva, E., Getoor, L. The illusion of privacy in social networks with mixed public and private user profiles. *Proceedings of the 18th International World Wide Web Conference* (2009).

Lars Backstrom (lars@fb.com),
Facebook, Palo Alto, CA.

Cynthia Dwork (dwork@microsoft.com),
Microsoft Research, Silicon Valley
Campus, Mountain View, CA.

Jon Kleinberg (kleinber@cs.cornell.edu),
Cornell University, Ithaca, NY.



DOI:10.1145/2043174.2043200

Peter Winkler

Puzzled

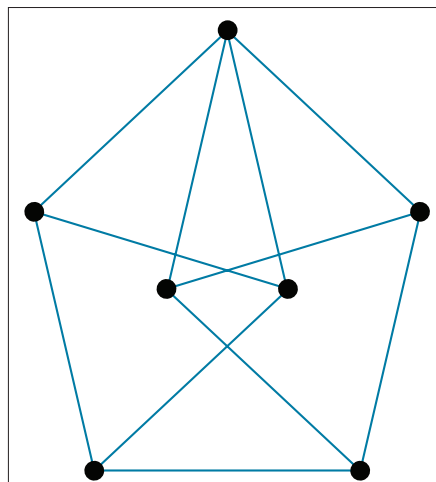
Solutions and Sources

Last month (Nov. 2011, p. 120) we posted a trio of brainteasers, including one as yet famously unsolved, concerning distances between points on the plane. Here, we offer solutions to two of them. How did you do?

1. Cities of gold.

Solution. You were asked to determine whether it is possible to place seven points (cities of gold) on the plane in such a way that among any three, at least two are a specified distance—10 leagues—apart. It turns out there is.

We can assume that the specified distance is 1. Two unit-side equilateral triangles sharing a side make what we call a “lozenge” with two sharp endpoints. Take two lozenges with a common sharp endpoint P , and swing them with P fixed in such a way that their other endpoints are unit distance apart (see the figure here). Together, the two lozenges have seven vertices. To see that they satisfy the condition, suppose there were three points among the seven that do not include a pair at distance 1. This threesome cannot contain the “fulcrum” point



Locations of the seven cities of gold, with each line representing a distance of 10 leagues; the top point is the fulcrum P .

P , because all but two of the remaining six points are at unit distance from the fulcrum, and these two—the other sharp lozenge endpoints—are unit distance from each other. So forget the fulcrum, but the other six points lie on two equilateral triangles, and any three must include at least two vertices of one of the triangles.

This cute problem was passed to me (without the spurious history) by mathematical wizard Frank Morgan of Williams College.

2. Frisbee players.

Solution. If the Frisbee players are arranged in a regular nonagon with longest diagonals of length 100 yards, then nine pairs of players will be at this distance, with none farther.

In fact, for any n , you cannot get more than n “maxpairs,” or pairs of points at maximum distance, among n points in the plane. To prove this, assume it is false, and let the points $A, B, C, \dots$ constitute a counterexample of the smallest possible size n .

Note first that any two maxpairs AB, CD must “cross”; that is, the line segment between A and B crosses the segment between C and D ; otherwise one of the diagonals of the quadrilateral $ABDC$ would exceed the supposed maximum length.

Now if, in our purported counterexample, no point was involved in more than two maxpairs, then there would be only n maxpairs in total. So there must be some point P that is in three maxpairs, say, PA, PB , and PC , in

that order clockwise around P . Note that the angle between PA and PC cannot be more than 60 degrees or else the third side AC of the triangle would be too long.

Observe now that the point B cannot be involved in any other maxpairs, because such a pair would cross both PA and PC , an impossibility. Dropping B out of our configuration yields a smaller configuration with one fewer point and one fewer maxpair, reaching a contradiction.

This puzzle appeared in 1957 on the William Lowell Putnam Exam, an annual contest for college students (<http://math.scu.edu/putnam/>), which is a great source for challenging mathematical puzzles.

3. Three colors, seven points.

Solution. To see how the layout of the seven points in the first puzzle gives us information about painting the plane, consider the colors these seven points would have to be if you could paint with only three colors. By the pigeonhole principle (used several times already in this column) at least three of the seven points must then get the same color, but we know these three contain two points at unit distance, and points at unit distance are not allowed to have the same color. Voila!

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.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

[CONTINUED FROM P. 144] an operating system from scratch to figure it out, which we did. Then we looked at our findings and realized they should be applicable to any standard operating system. So, with a few of my colleagues and students, we did a study to see how much work would be necessary to scale the Linux kernel to a large number of cores. If you have enough manpower, it's certainly doable.

This is the system you built in which eight six-core chips were used to simulate the performance of a 48-core chip.

Yes, indeed. There are a lot of interesting problems to be solved, but my general sense is that things are going to evolve in the right direction, and that there won't be a point in time where we have to throw everything away and start over again.

Another insight to come out of that work was that it can be difficult to identify the root cause of performance issues. Is that what inspired your work on MOSBENCH, a set of application benchmarks designed to measure the scalability of operating systems?

Yes, MOSBENCH came out of that project. Typical benchmarks are just application benchmarks, where all the action is in the application itself. But we needed a benchmark that included a lot of system-intensive applications. Otherwise, you don't stress the operating system, and if you don't stress the operating system, it isn't scalable by default. So we collected several applications to stress different parts of the operating system—essentially, it's a workload generator.

What conclusions has it led to so far?

The Linux kernel scales pretty well. But there might be interesting future problems. One direction is having the operating system give you more control over the caches in which the data lives. The traditional view is that the cache is hidden from the operating system and the hardware just does its job of caching. In multicore, caches are spread all around the chip, some close by and others that are far away. There are cases where you want control over where the data is placed so you can get better performance. Something else we're looking at are abstractions that allow

“One of the big advantages of academia is that if you decide the problem's not interesting, you can change. That's a hard thing to do in a startup.”

you to build operating systems that are scalable by design, as opposed to scaling every subsystem one by one. New concurrent data structures that exploit weak consistency semantics are another direction.

You have also done work on systems security, using information flow control to prevent the unauthorized disclosure of data.

The idea is simple. Typically when you build an application, and you want to make it secure, you put a check before every operation that might be sensitive. The risk is that you can easily forget a check, which can then be exploited as a security vulnerability. We tried to structure the operating system in such a way that even if you forget some of these checks, security is not immediately compromised. The way we do it is to draw a box around the operating system and label all data. Then we have a guard that checks whenever data is being sent across the border to make sure it's going to the right place, based on the data's label.

Some of your other security research focuses on making it easier to restore system integrity after an intrusion. So-called “undo computing,” for instance, seeks to undo any changes made by an adversary during the attack while preserving legitimate user actions.

Let's say you have a desktop, and you discover it was compromised a couple weeks after an attack. Then the question is, How do you restore its integrity? You could go back to a backup from

three weeks ago, when you know it's clean, and reinstall some pieces. But that's clearly a labor-intensive project. Or you try to find all the bad code and files, and remove them, which of course is also labor intensive. There are some automatic virus removers, but they're very specific to a particular virus.

What is your approach?

Here's one direction my colleague Nickolai Zeldovich and our students are exploring: Once you've determined that an adversary sent bad packets to your Web server, you know everything that could be influenced by those packets is suspicious, and all the influenced actions must be undone. We roll the system back to before the attack happened, and roll forward all the actions that were not influenced by the adversary's actions. If everything works out correctly, you will end up in a clean state, but you will still have all the work that you did in the last three weeks.

What if the actions of the adversary are intermingled with the actions of the user?

Undoing that intermingling and keeping track of the dependencies requires some reasonably sophisticated techniques. Another aspect of the problem is that you really don't want to replay or redo every operation. So we have a bunch of clever observations saying, well, this work or this operation could never have been influenced by the attacker's actions, so therefore we don't have to redo them. We have some encouraging results, but we're still trying to figure out whether we can make this work in practice for heavily used complex systems.

Do you have plans to do another startup?

I'm going to wait and see. It's not until the later stages of a project that I think about whether it solves a real problem that people have and, if so, would it be worthwhile to start a company around it. One of the big advantages of academia is that if you decide the problem's not interesting, you can change. That's a hard thing to do in a startup. 

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

© 2011 ACM 0001-0782/11/12 \$10.00

Q&A

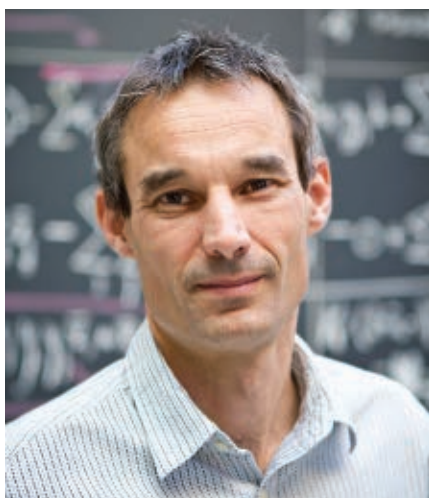
Scaling Up

M. Frans Kaashoek talks about multicore computing, security, and operating system design.

M. FRANS KAASHOEK'S interest in computing was sparked, like many others in the field, by an early love for programming. At Vrije Universiteit, he discovered he could turn his hobby into a career, and studied with MINIX creator Andrew S. Tanenbaum before accepting a professorship at Massachusetts Institute of Technology's Department of Electrical Engineering and Computer Science. Kaashoek has since conducted wide-ranging research in computer systems, including operating system design, software-based network routing, and distributed hash tables, which revolutionized the storage and retrieval of data in decentralized information systems. He also helped found two startups: Sightpath, a video broadcast software provider that was acquired by Cisco Systems in 2000, and Mazu Networks, which was acquired by Riverbed Technology in 2009. Kaashoek was named an ACM fellow in 2004 and elected to the National Academy of Engineering in 2006. Last year his work was recognized with an ACM-Infosys Foundation Award (see "Unlimited Possibilities" in the June 2011 issue of *Communications*).

You have said that your work on the exokernel operating system, which enables application developers to specify how the hardware should execute their code, was driven by intellectual curiosity. Can you elaborate?

We wanted to explore whether we could build a kernel interface that defines no abstractions other than what the hardware already provides, and that exports the hardware abstractions



directly to applications. Traditionally, the kernel provides a fixed set of unchangeable abstractions. For example, you have a very complex, unchangeable kernel interface like traditional Unix systems, or you have a small, unchangeable microkernel interface, which defines a few carefully chosen abstractions. An exokernel design allows the programmer to define its own operating system abstractions.

For its minimalism, it sounds almost like an extreme version of microkernel design.

The main goal with a microkernel is to make the kernel small. That was not necessarily our goal. So, for example, we would have been perfectly happy to put a device driver inside the kernel if we thought it was the right thing to do.

How did the project evolve?

We were able to build a prototype that demonstrated the approach could

work in practice. But I don't think there's any direct technology transfer from our ideas into products although there was one startup that used our code. The impact has been more indirect. Academically, it influenced other systems that were built afterward. On the more commercial side, it also has been credited in work on machine monitors for handheld devices.

Operating systems design has become such a partisan issue. What is your take on it?

I have a pragmatic view. In research, taking an extreme position is interesting because it forces you to clarify your thinking and solve the hard case. In practice, I think people are going to do whatever helps solve the particular problems they have. If you look at a monolithic kernel like Linux—I know you can't call it a microkernel system, but some of the servers run as applications in user space and some run in the kernel, and it really becomes shades of gray. And some people draw this line slightly differently than others. But if the kernel is already working fine, why change it?

Since your work on exokernels, you have done several other projects on operating systems design, in particular as it relates to multicore computing.

You might say that multicore has nothing to do with the operating system because it is, in many ways, already inherently parallel; it provides processes that can run on different cores in parallel. But many applications rely heavily on operating system services, particularly systems applications like email and Web servers. So if the operating system services don't scale well, those applications can't scale well, either.

So your work is focused on building scalable operating systems.

Originally, we thought we would have to write [CONTINUED ON P. 143]



Distinguished Speakers Program *talks by and with technology leaders and innovators*

Chapters • Colleges & Universities • Corporations • Agencies • Event Planners

Need a Great Technical Speaker for Your Next Event?

The Association for Computing Machinery (ACM), the world's largest educational and scientific computing society, now provides colleges and universities, corporations, event and conference planners, and agencies – in addition to ACM local Chapters – with direct access to top technology leaders and innovators from nearly every sector of the computing industry.

Book the speaker for your next event through the ACM Distinguished Speaker Program (DSP) and deliver compelling and insightful content to your audience at a remarkably reasonable price. Our program features renowned thought leaders in academia, industry and government, speaking about the topics that matter most in the computing and IT world today. Our booking process is simple and convenient, please visit us at: www.dsp.acm.org.

The ACM Distinguished Speaker Program is an excellent solution for:

Corporations Educate your technical staff, ramp up the knowledge of your team, and give your employees the opportunity to have their questions answered by experts in their field.

Colleges & Universities Expand the knowledge base of your students with exciting lectures and the chance to engage with a computing professional in their desired field of expertise.

Event & Conference Planners Use the ACM DSP to help find compelling speakers for your next conference and reduce your costs in the process.

ACM Local Chapters Boost attendance at your meetings with live talks by DSP speakers and keep your chapter members informed of the latest industry findings.

Captivating Speakers from Exceptional Companies, Colleges & Universities

DSP speakers represent a broad range of companies, colleges and universities, including: IBM, Microsoft, BBN Technologies, Raytheon, Sony Pictures Imageworks, National Institute of Standards and Technology, Lawrence Livermore National Laboratory, Siemens Information Systems Bangalore, Stanford University, University of Pennsylvania, University of British Columbia, Georgia Tech, Carnegie Mellon, UCLA, McGill University, Tsinghua University and many more.

Topics for Every Interest

Over 250 lectures are available from nearly 100 different speakers with topics covering Software Engineering, High Performance Computing, Human Computer Interaction, Artificial Intelligence, Gaming, Mobile Computing, and dozens more. Some of our most popular lectures include:

- Electronic Voting in the 21st Century
- Software Engineering Best Practices
- Software Under Siege: Viruses and Worms
- Spatial Databases and Geographic Information Systems
- Careers in Computing – How to Prepare and What to Expect

Quality is Our Standard

The same ACM you know from our world-class digital library, magazines and journals is now putting the affordable and flexible Distinguished Speaker Program within reach of the computing community.

To select a speaker for your next event, get started today at www.dsp.acm.org.

If you have questions, please send them to acmdsp@acm.org.

Microsoft
Research
The DSP is sponsored,
in part, by Microsoft Europe



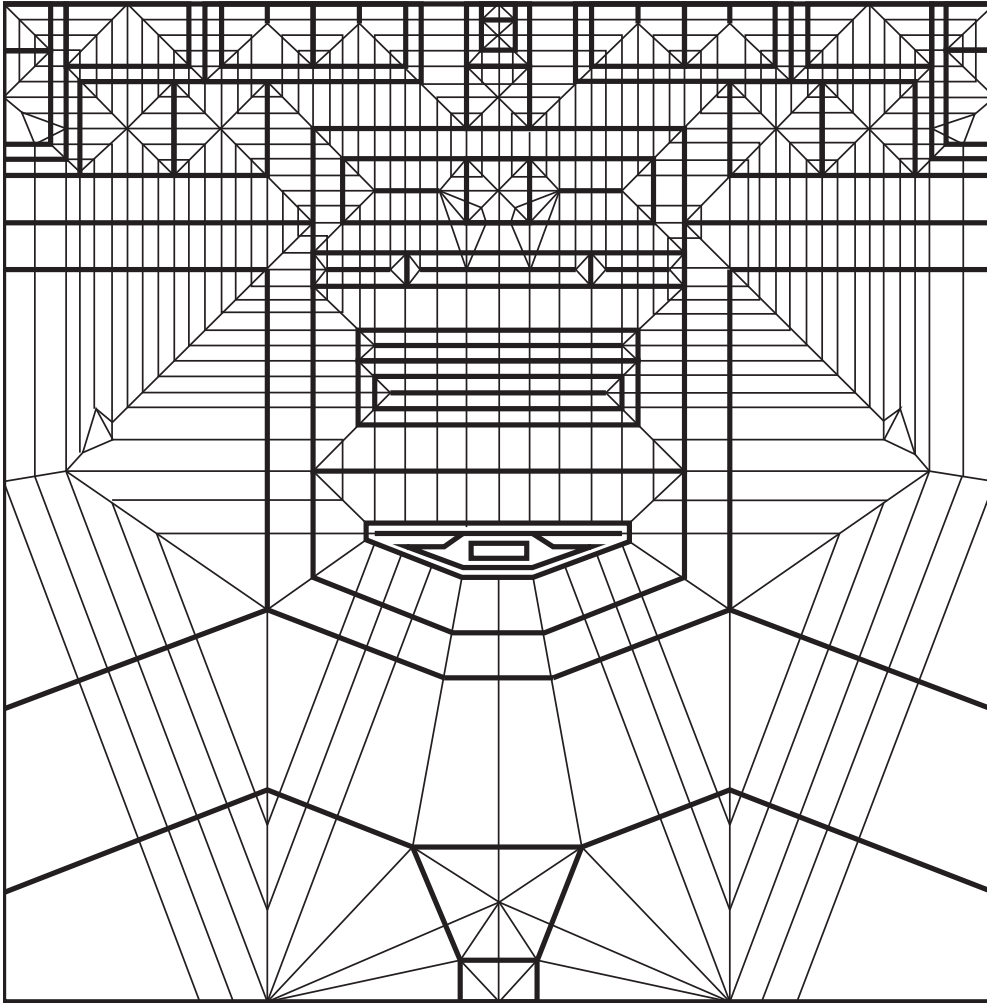
**Association for
Computing Machinery**

Advancing Computing as a Science & Profession

TEI2T



2



© Jason Ku



SIXTH INTERNATIONAL CONFERENCE ON TANGIBLE, EMBEDDED AND EMBODED INTERACTION

JANUARY 10, 2011 - EARLY REGISTRATION DEADLINE
FOR MORE INFORMATION: WWW.TEI-CONF.ORG/12

TEI2012 is a conference for researchers, artists and designers interested in creating user experiences that bridge the gap between bits and atoms through work in HCI, design, interactive arts, tools and technologies.

Themes this year include Organic User Interfaces and explorations of folding and unfolding in design.



SIGCHI
special interest group computer human interaction

February 19-22, 2012 | Queen's Human Media Lab | Kingston, ON Canada