COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

09/2013 VOL.56 NO.09

Uncomfortable User Experience

Association for Computing Machinery

UIST 2013

26th ACM Symposium on

User Interface Software & Technology

October 8th - 11th, 2013

St Andrews, Scotland, UK

Paper Deadline: April 5

Co-located with ITS 2013: ACM Interactive Tabletops and Surfaces (Oct 6th - 9th)

acm

http://acm.org/uist

Poster designed by rareloop.com

IPDPS 2014 PHOENIX

www.ipdps.org

19-23 May 2014
Arizona Grand Resort
Phoenix, Arizona USA

- **KEYNOTES & TECHNICAL SESSIONS**
  - *Algorithms*
  - *Applications*
  - *Architecture*
  - *Software*
- **WORKSHOPS & PhD FORUM**
  - *Submission Dates Vary*
- **COMMERCIAL PARTICIPATION**

GENERAL CHAIR
Manish PARASHAR (Rutgers University, USA)

PROGRAM CHAIR
David A. BADER (Georgia Institute of Technology, USA)

PROGRAM VICE-CHAIRS
• *Algorithms*
Anne BENOIT (ENS Lyon, France)

• *Applications*
Padma RAGHAVAN (Pennsylvania State University, USA

• *Architecture*
Yan SOLIHIN (North Carolina State University, USA)

• *Software*
Ron BRIGHTWELL (Sandia National Laboratories, USA)

WORKSHOPS CHAIR
Ümit V. ÇATALYÜREK (Ohio State University, USA)

WORKSHOPS VICE-CHAIR
Yogesh SIMMHAN (University of Southern California, USA)

PhD FORUM CO-CHAIRS
Luc BOUGÉ (ENS Cachan, France)
Bo HONG (Georgia Institute of Technology, USA)

Sponsored By:

IEEE computer society
Technical Committee on Parallel Processing

In Cooperation With:

ACM SIGARCH

IEEE Computer Society Technical Committee on Computer Architecture

IEEE Computer Society Technical Committee on Distributed Processing

28th
IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM

IPDPS 2014 CALL FOR PARTICIPATION
The 28th IEEE-IPDPS will be held 19-23 May in Phoenix at the Arizona Grand Resort, an all-suites family-friendly resort and meeting place. Phoenix has an international airport and is the hub for several US airlines. It is only a six hour drive from Los Angeles and Las Vegas and the starting point for one day tours of the Grand Canyon as well as other historic communities in the area. For those who come early or stay late, there are boundless Southwest tourist attractions and activities. For shorter trip times, AGR has a water-world, golf course, and hiking and riding trails on the premises and an indoor athletic club and spa for fitness training. Check the IPDPS Web pages for updates and information on workshops, the PhD Forum, and other events as the full program develops and see the full call for papers for details on submitting papers for 2014.

IPDPS 2014 CALL FOR PAPERS
Scope: Authors are invited to submit manuscripts that present original unpublished research in all areas of parallel and distributed processing, including the development of experimental or commercial systems. Work focusing on emerging technologies is especially welcome. Topics of interest include, but are not limited to:

•**Parallel and distributed algorithms**, focusing on topics such as: numerical, combinatorial, and data-intensive parallel algorithms, scalability of algorithms and data structures for parallel and distributed systems, communication and synchronization protocols, network algorithms, scheduling, and load balancing.

•**Applications of parallel and distributed computing**, including computational and data-enabled science and engineering, big data applications, parallel crowd sourcing, large-scale social network analysis, management of big data, cloud and grid computing, scientific and biomedical applications, and mobile computing. Papers focusing on applications using novel commercial or research architectures, big data approaches, or discussing scalability toward the exascale level are encouraged.

•**Parallel and distributed architectures**, including architectures for instruction-level and thread-level parallelism; petascale and exascale systems designs; novel big data architectures; special-purpose architectures, including graphics processors, signal processors, network processors, media accelerators, and other special purpose processors and accelerators; impact of technology on architecture; network and interconnect architectures; parallel I/O and storage systems; architecture of the memory hierarchy; power-efficient and green computing architectures; dependable architectures; and performance modeling and evaluation.

•**Parallel and distributed software**, including parallel and multicore programming languages and compilers, runtime systems, operating systems, resource management including green computing, middleware for grids, clouds, and data centers, libraries, performance modeling and evaluation, parallel programming paradigms, and programming environments and tools. Papers focusing on novel software systems for big data and exascale systems are encouraged.

What/Where to Submit: More details on submissions and instructions for submitting manuscripts are available at www.ipdps.org. All submitted manuscripts will be reviewed. Submitted papers should NOT have appeared in or be under consideration for another conference, workshop or journal.

Important Dates:

October 11, 2013	Abstracts due
October 18, 2013	**Submission deadline**
December 09, 2013	Author notification
February 01, 2014	Camera ready due

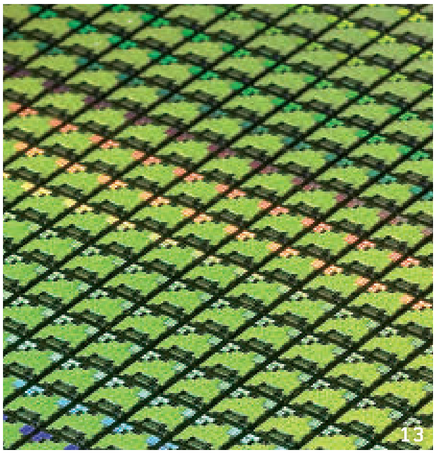
COMMUNICATIONS OF THE ACM

Departments

Last Byte

Association for Computing Machinery
Advancing Computing as a Science & Profession

News



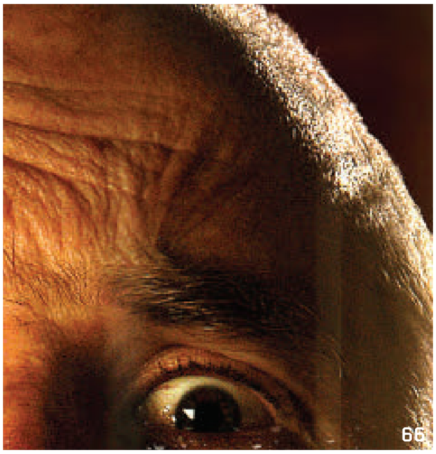
Viewpoints

PHOTOGRAPH BY JOAN CHANG

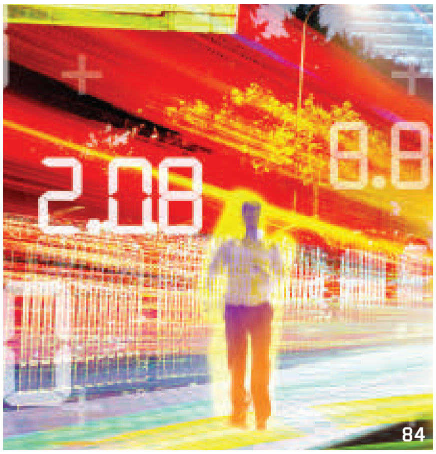
Practice

Articles' development led by acmqueue
queue.acm.org

Contributed Articles



Review Articles



Research Highlights

About the Cover: While art and cinema have long been known for exploring ways to make the viewer feel anxious or frightened, this month's cover story (p. 66) focuses on the benefits of designing computer games, apps, amusement rides, and other online experiences with the deliberate purpose of making the user feel uncomfortable. Cover photo illustration by Barry Downard.

IMAGE FROM SHUTTERSTOCK.COM; PHOTO ILLUSTRATION BY BARRY DOWNARD; ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

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

Executive Director and CEO
John White
Deputy Executive Director and COO
Patricia Ryan
Director, Office of Information Systems
Wayne Graves
Director, Office of Financial Services
Russell Harris
Director, Office of SIG Services
Donna Cappo
Director, Office of Publications
Bernard Rous
Director, Office of Group Publishing
Scott E. Delman

ACM COUNCIL

President
Vinton G. Cerf
Vice-President
Alexander L. Wolf
Secretary/Treasurer
Vicki L. Hanson
Past President
Alain Chesnais
Chair, SGB Board
Erik Altman
Co-Chairs, Publications Board
Ronald Boisvert and Jack Davidson
Members-at-Large
Eric Allman; Ricardo Baeza-Yates; Radia Perlman; Mary Lou Soffa; Eugene Spafford
SGB Council Representatives
Brent Hailpern; Andrew Sears; David Wood

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
Jiaguang Sun

PUBLICATIONS BOARD

Co-Chairs
Jack Davidson; Joseph Konstan
Board Members
Ronald F. Boisvert; Marie-Paule Cani; Nikil Dutt; Carol Hutchins; Ee-Peng Lim; Patrick Madden; 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

STAFF

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

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

Art Director
Andrij Borys
Associate Art Director
Margaret Gray
Assistant Art 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
acmhelp@acm.org
Letters to the Editor
letters@cacm.acm.org

WEBSITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ACM ADVERTISING DEPARTMENT

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

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

Media Kit acmmediasales@acm.org

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

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

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

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

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Association for Computing Machinery

Printed in the U.S.A.

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/2500468.2500469

Moshe Y. Vardi

Has The Innovation Cup Run Dry?

We like to think we have been surfing a tsunami of computing innovation over the past 70 years: mainframe computers, microprocessors, personal computers, the Internet, the World Wide Web, search, cloud computing, social media, smartphones, tablets, big data, and the like. The list goes on and on, and the future for continuing innovation is quite bright, according to the conventional wisdom.

Recently, however, several people have been questioning this techno-optimism. In a commencement address at Bard College at Simon's Rock, U.S. Federal Reserve chair Ben Bernanke compared life today to his life as a young boy in 1963, and his grandparents' lives in 1913. He argued that the period from 1913 to 1963 saw a dramatic improvement in the quality of daily life, driven by automobilization, electrification, sanitation, air travel, and mass communication. In contrast, life today does not seem that different than life in 1963, other than the fact we talk less to each other, and communicate more via email, text, and social postings.

In fact, the techno-pessimists argue the economic malaise that we seem unable to pull ourselves out of—the sluggish economic growth in the U.S., the rolling debt crisis in Europe, and the slowdown of the BRICS—is not just a result of the financial crisis but also an indication of an innovation deficit. Tyler Cowen has written about the "great stagnation," arguing we have reached a historical technological plateau and the factors that drove economic growth since the start of the Industrial Revolution are mostly spent. Robert Gordon contrasted the 2.33% annual productivity growth during 1891–1972 to the 1.55% growth rate during 1972–2012. Garry Kasparov argued that most of the science underlying modern computing was already settled in the 1970s.

The techno-optimists dismiss this pessimism. Andrew McAfee argued that new technologies take decades to achieve deep impact. The technologies of the Second Industrial Revolution (1875–1900) took almost a century to fully spread through the economies of the developed world, and have yet to become ubiquitous in the developing world. In fact, many predict we are on the cusp of the "Third Industrial Revolution." *The Economist* published a special report last year that described how digitization of manufacturing will transform the way goods are made and change the job market in a profound way.

So which way is it? Have we reached a plateau of innovation, dooming us to several decades of sluggish growth, or are we on the cusp of a new industrial revolution, with the promise of dramatic changes, analogous to those that took place in the first half of the 20th century?

From my perch as the editor-in-chief of *Communications of the ACM*, I find it practically impossible to be a pessimist (which is my natural inclination). The flow of exciting research and news articles that we publish monthly continues to be innovative and exciting. No stagnation here!

Last year ACM celebrated Alan Turing's centenary by assembling a historic gathering of almost all of the living ACM A.M. Turing Award Laureates for a two-day event in San Francisco. Over 1,000 participants attended the meeting and the buzz was incredible. Participants I talked to told me this was one of the most moving scientific meetings they have ever attended. We were celebrating not only Turing's centenary, but also 75 years of computing technology that has changed the world, as well as the people who pioneered that technology. Like McAfee, it is difficult for me to imagine this technology not broadening and deepening its impact on our lives.

Earlier this year the McKinsey Global Institute issued a report on "Disruptive Technologies: Advances that will transform life, business, and the global economy," in which they assert that many emerging technologies "truly do have the potential to disrupt the status quo, alter the way people live and work, and rearrange value pools." By 2025, the report predicted a $5–$7 trillion potential economic impact from automation of knowledge work and the prevention of 1.5 million driver-caused deaths from car accidents by automation of driving. The 12 potential economically disruptive technologies listed in the report are: mobile Internet; knowledge-work automation; the Internet of Things; cloud technology; advanced robotics; autonomous and near-autonomous vehicles; next-generation genomics; energy storage; 3D printing; advanced materials; advanced oil and gas exploration and recovery; and renewable energy.

"Predictions are difficult," goes the saying, "especially about the future." It will be about 25 years before we know who is right, the techno-pessimists or the techno-optimists. For now, however, count me an optimist, for a change!

Moshe Y. Vardi, EDITOR-IN-CHIEF

DOI:10.1145/2500468.2500470

Vinton G. Cerf

Freedom and the Social Contract

The last several weeks (as of this writing) have been filled with disclosures of intelligence practices in the U.S. and elsewhere. Edward Snowden's unauthorized release of highly classified information has stirred a great deal of debate about national security and the means used to preserve it.

In the midst of all this, I looked to Jean-Jacques Rousseau's well-known 18th-century writings on the Social Contract (*Du Contrat Social, Ou Principes du Droit Politique*) for insight. Distilled and interpreted through my perspective, I took away several notions. One is that in a society, to achieve a degree of safety and stability, we as individuals give up some absolute freedom of action to what Rousseau called the sovereign will of the people. He did not equate this to government, which he argued was distinct and derived its power from the sovereign people.

I think it may be fair to say that most of us would not want to live in a society that had no limits to individual behavior. In such a society, there would be no limit to the potential harm an individual could visit upon others. In exchange for some measure of stability and safety, we voluntarily give up absolute freedom in exchange for the rule of law. In Rousseau's terms, however, the laws must come from the sovereign people, not from the government. We approximate this in most modern societies creating representative government using public elections to populate the key parts of the government.

I think it is also likely to be widely agreed that a society in which there was no privacy and every action or plan was visible to everyone might not be a place in which most of us might like to live. I am reminded, however, of my life in a small village of about 3,000 people in Germany. In the 1960s, no one had phones at home (well, very few). You went to the post office to mail letters, pick up mail, and make or receive phone calls. In some sense, the Postmaster was the most well-informed person about the doings of the town. He saw who was calling or writing to whom. There was not a lot of privacy. The modern notion of privacy may in part have derived from the growth of large urban concentrations in which few people know one another.

In today's world, threats to our safety and threats to national security come from many directions and not all or even many of them originate from state actors. If I can use the term "cyber-safety" to suggest safety while making use of the content and tools of the Internet, World Wide Web, and computing devices in general, it seems fair to say the expansion of these services and systems has been accompanied by a growth in their abuse. Moreover, it has been frequently observed that there is an asymmetry in the degree of abuse and harm that individuals can perpetrate on citizens, and on the varied infrastructure of our society. Vast harm and damage may be inflicted with only modest investment in resources. Whether we speak of damage and harm using computer-based tools or damage from lethal, homemade explosives, the asymmetry is apparent. While there remain serious potential threats to the well-being of citizens from entities we call nation- states, there are similarly serious potential threats originating with individuals and small groups.

Presuming we have accepted the theory that safety is partly found through voluntarily following law, we must also recognize that there are parties domestic and otherwise who wish us individual and collective harm. The societal response to this is to provide for law enforcement and intelligence gathering (domestic and non-domestic) in an attempt to detect and thwart harmful plans from becoming harmful reality. We do not always succeed.

The tension we feel between preserving privacy and a desire to be protected from harm feeds the debate about the extent to which we are willing to trade one for the other. Not everyone, nor every culture, will find the same point of equilibrium. Moreover, as technology and society evolve, the equilibrium points may shift. It has been said that "security" is not found in apprehending a guilty party but in preventing the harm from occurring. While this notion can surely be overextended, it can also be understood to justify a certain degree of intelligence gathering in the service of safety and security.

There is some irony in the fact that our privacy is more difficult than ever to preserve, given the advent of smartphones, tablets, laptops, the Web and the Internet, but that the threats against our safety and security use the same infrastructure to achieve nefarious ends. Our discipline, computer science, is deeply involved in the many dimensions of this conundrum and we owe it to our fellow citizens to be thoughtful in response and to contribute to reasoned consideration of the balance our society needs between potential policy extremes.

Vinton G. Cerf, ACM PRESIDENT

DOI:10.1145/2500468.2500482

For Privacy and Security, Use Public Keys Everywhere

MANY ACM MEMBERS concerned about the recent disclosures of massive worldwide surveillance of civilians wonder how to respond. My recommendation is to use public keys[1] for all electronic communication and storage. Public key encryption, along with corresponding private key decryption, enables personal privacy and security because information cannot be snooped or altered between sender and receiver. Public keys involve no single point of failure because private keys are distributed among potentially billions of users. Moreover, technology is available to make public key encryption and decryption almost invisible to those users.

In 2010, Google increased the security of Gmail by enabling default encryption of communication between browsers and its servers. Unfortunately, Gmail's use of a single key set for all users can be a point of failure. If keys in the key set are stolen or otherwise obtained, the communication of millions of Gmail users risks compromise through dynamic "man-in-the-middle" attacks mounted in the communications infrastructure. Also, Google servers can be a point of failure for organizational and technical reasons; for example, intruders are reported to have successfully compromised Gmail servers and obtained the Gmail contact information and email contents of economic targets and political opponents. The lesson is that everyone should be using public keys to preserve privacy and prevent alteration of their email communications.

Apple uses individual user encryption keys in its iMessage service but does not offer public keys in its iCloud service. Consequently, users who back up their messages in iCloud risk losing their privacy and security. Private information should be stored in the cloud only after it is encrypted with a public key. Information to be shared can be encrypted through shared keys established for that purpose.

A privacy card (stored in, say, a user's pocket or purse) can maintain the security of the user's keys and encrypted biometric information, helping prevent a card from being used by someone other than its owner. Devices can use a combination of near-field and Bluetooth radio communication with a privacy card to develop temporary shared secrets for secure communication without revealing the user's keys or biometric information and to securely share public keys with the privacy cards of others. Extracting private information from a card lost or stolen is extraordinarily difficult, as information can be erased if the card is tampered with. Moreover, a card's keys can be revoked and a new card authenticated through commercial and/or community services.

Fortunately, solutions involving public keys are available, including for Gmail (https://www.penango.com/), Windows (http://www.gpg4win.org/), iOS (https://gpgtools.org/gpgmail/), and Linux (http://gnupg.org/), as well as for telephoning, videoconferencing, and instant messaging (https://jitsi.org/).

An important starting point is medical systems using privacy cards to establish confidentiality, as they increasingly rely on digital communications. But this is just one starting point. Universal use of public keys promises to be both a hallmark and a foundation of free and open societies.

Carl Hewitt, Palo Alto, CA

Reference

1. Levy, S. *Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age.* Penguin Books, New York, 2001.

Beware BYOD

A new independence movement known as Bring Your Own Device, or BYOD, has taken hold in computing, even in computer science classrooms. BYOD was coined by Ballagas et al.[1] in 2004 in the context of interacting with public displays but became popular after 2009 when Intel began supporting its employees who wanted to use their personal cellphones and tablets at work,[2] requiring permission to connect their private devices to workplace networks.

Companies allowing BYOD could view their acquiescence as being responsive to the needs of employees and as a way to cut the costs of corporate computing. BYOD also found a home in education, from K–12 to college. However, as more and more educational software became available and e-books replaced heavy textbooks, students also required more endpoint devices.

BYOD requires a major change in attitude no matter how applied. In many schools, teachers routinely confiscate student cellphones to be returned to parents at a later time. However, BYOD also potentially allows school districts to cut their IT budgets, possibly helping the movement make inroads there, but BYOD in computer science education represents a particularly dangerous trend for multiple reasons:

Distraction. Students in classes other than computer science can be told to use their devices selectively, with a teacher defining "listening time" and "device time" and keeping the two apart. In computer science education, students are likely to need their devices continuously, putting computer games and movie Web sites at their fingertips, with some students finding multitasking between, say, lecture and movie perfectly reasonable;

Cheating. Students working on programming problems with their own devices have access to communication tools that enable unauthorized collaboration; that is, they can make full use of social media and "messenger" programs to cheat;

More cheating. Students working on computing problems with their own devices may likewise have access to the Web where solutions to many problems are readily (and temptingly) available;

Still more cheating. Websites (such as those belonging to rent-a-coder brokers) make it easy to find programmers willing to solve homework problems for a fee; and

Incompatibility. Experience at the New Jersey Institute of Technology shows programs running successfully on one device (such as a student's) might not even compile on another device (such as a teacher's), even in nominally standardized environments, risking undeserved poor grades.

BYOD results in an educational environment beyond the control of instructors in a way that can make it impossible to apply consistent grading standards. BYOD in computer science education is thus harmful to achieving students' professional goals and instructors' educational objectives.

James Geller, Newark, NJ

References

1. Ballagas, R., Rohs, M., Sheridan, J.G., and Borchers, J. BYOD: Bring your own device. In *Proceedings of the Workshop on Ubiquitous Display Environment* (Nottingham, U.K., Sept. 7–10). Ubicomp, 2004.
2. Roman, J. *BYOD: Get Ahead of the Risk.* Govinfo Security, Jan. 11, 2012; http://www.govinfosecurity.com/byod-get-ahead-risk-a-4394

What a License Really Certifies

Vinton G. Cerf's "From the President" editorial "'But Officer, I was Only Programming at 100 Lines Per Hour!'" (July 2013) raised the issue of how to license "software designers and implementers." The state of Texas first licensed software engineers in 1998. ACM pulled out of the Texas software engineering licensing effort in 1999, while the IEEE continued on by itself. Texas and many other states now offer licensing through the "Principles and Practices Exam of Software Engineering" exam (http://ncees.org/exams). Cerf implied licensing consists solely of an exam. However, typical state licensing requirements include a degree from an accredited program, the fundamentals of engineering exam, four years of documented engineering practice, the professional engineer exam in the area of practice, and three letters of recommendation from licensed professional engineers familiar with the candidate's engineering practice. If such credentials are not sufficient to formally determine competency, what is?

Duncan M. (Hank) Walker,
College Station, TX

Author's Response:

I did not intend to imply that licensing consisted solely of an exam but rather assumed it would likely be in addition to any other evidence of training and competence. It would be of interest to know what has been the outcome of the Texas licensing program. In particular, under what circumstances might work opportunities be restricted to persons holding such a license. Can Walker cite other states with similar programs? Are there conditions under which the license is a "must have" as opposed to a "nice to have" insignia in the state of Texas?

Vinton G. Cerf, ACM President

Who's Right?

Looking to rebut Vinton G. Cerf's *New York Times* op-ed essay "Internet Access Is Not a Human Right" (Jan. 4, 2012), Stephen Wicker's and Stephanie M. Santoso's Viewpoint "Access to the Internet Is a Human Right" (June 2013) missed a few things; they said, for example, Internet access is intertwined with "human capabilities that are considered fundamental to a life worth living." Hmm. Does that mean, say, a monk who chooses to live in a community without Internet access is "diminished or denied"? The ordinary dictionary definition of rights (as in http://www.m-w.com "something to which one has a just claim, as the power or privilege to which one is justly entitled" or "something that one may properly claim as due") requires no action by anyone else. Could it be that a person who chooses to live in a remote place with no access to an ISP is likewise "denied his or her rights"? Moreover, who might be sued or arrested for violating those rights? We in the Western world live at the historical pinnacle of human luxury and comfort where many constantly try to expand "rights." Such attempts are misguided for many reasons, including the related diminishment of the fundamental rights inherent in just being a human. When everything is a right, nothing is.

Alexander Simonelis,
Montréal, Canada

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

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

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

DOI:10.1145/2500468.2500471 http://cacm.acm.org/blogs/blog-cacm

Big Data Is 'Buzzword du Jour;' CS Academics 'Have the Best Job'

Michael Stonebraker analyzes the different varieties of Big Data, while Judy Robertson considers the rewards of teaching computer science.

Michael Stonebraker "What Does 'Big Data' Mean?"

http://cacm.acm.org/blogs/blog-cacm/155468-what-does-big-data-mean/fulltext
September 21, 2012

It is interesting to note that a substantial subset of the computer science community has redefined their research agenda to fit under the marketing banner of "Big Data." As such, it is clearly the "buzzword du jour." As somebody who has been working on database problems for a very long time (which, by definition, deal with big data), I would like to explain what I think "big data" means, and discuss what I see as the research agenda.

In the community I travel in, big data can mean one of four things:

Big volumes of data, but "small analytics." Here the idea is to support SQL on very large datasets. Nobody runs "Select*" from something big, as this would overwhelm the recipient with terabytes of data. Instead, the focus is on running SQL analytics (count, sum, max, min, and avg with an optional group_by) on large amounts of data. I term this "small analytics" to distinguish this use case from the one that follows.

Big analytics on big volumes of data. By big analytics, I mean data clustering, regressions, machine learning, and other much more complex analytics on very large amounts of data. At the present time, users tend to run big analytics using statistical packages, such as R, SPSS and SAS. Alternately, they use linear algebra packages such as ScalaPack or Arpack. Lastly, there is a fair amount of custom code (roll your own) used here.

Big velocity. By this I mean being able to absorb and process a fire hose of incoming data for applications like electronic trading, real-time ad placement on Web pages, real-time customer targeting, and mobile social networking. This use case is most prevalent in large Web properties and on Wall Street, both of whom tend to roll their own.

Big variety. Many enterprises are faced with integrating a larger and larger number of data sources with diverse data (spreadsheets, Web sources, XML, traditional DBMSs). Many enterprises view this as their number-one headache. Historically, the extract, transform, and load (ETL) vendors serviced this market on modest numbers of data sources.

In summary, big data can mean big volume, big velocity, or big variety. In the remainder of this post, I talk about small analytics on big volumes of data.

Big Volume, Small Analytics

I am aware of more than five multi-petabyte data warehouses in production use running on three different commercial products. No doubt there are a couple of dozen more. All are running on "shared nothing" server farms with north of 100 usually "beefy" nodes, survive hardware node failures through failover to a backup replica, and perform a workload consisting of SQL analytics as defined previously. All report operational challenges in keeping a large configuration running, and would like new DBMS features. Number one on everybody's list is resource elasticity (i.e., add 50 more servers to a system of 100 servers, automatically repartitioning the data to include the extra servers, all without taking down time and without interrupting query processing). In addition, better resource management is also a common request. Here, multiple cost centers are sharing a common resource, and everybody wants to get their fair share. The pundits—for example, Curt Monash—often identify some of these data warehouses.

A second solution to this use case appears to be Hive/Hadoop. I know of a couple of multi-petabyte repositories using this technology, most notably Facebook. Again, there are probably a couple of dozen more, and I know of many IT shops that are prototyping this solution. There have been quite a few papers in the recent literature documenting the inefficiency of Hadoop, compared to parallel DBMSs. In general, you should expect at least an order of magnitude performance difference. This will translate into an order of magnitude worse response time on the same amount of hardware, or an order of magnitude more hardware to achieve the same performance. If the latter course is chosen, this is a decision to buy a lot of iron and use a lot of power. As detailed in my previous blog post with Jeremy Kepner, I am not a big fan of this solution.

In addition, Google and other large Web properties appear to be running large configurations with this sort of workload on home-brew software. Some of it looks much like commercial RDBMSs (e.g., F1) and some of it looks quite different (e.g., BigTable).

Off into the future, I see the main challenge in this world to be 100% uptime (i.e., never go down, no matter what). Of course, this is a challenging "ops" problem. In addition, this will require the installation of new hardware, the installation of patches, and the next iteration of a vendor's software, without ever taking down time. Harder still is schema migration without incurring downtime.

In addition, I predict the SQL vendors will all move to column stores, because they are wildly faster than row stores. In effect, all row store vendors will have to transition their products to column stores over time to be competitive. This will likely be a migration challenge to some of the legacy vendors.

Lastly, there is a major opportunity in this space for advanced storage ideas, including compression and encryption. Sampling to cut down query costs is also of interest.

Disclosure
In addition to being an adjunct professor at the Massachusetts Institute of Technology, Michael Stonebraker is associated with four startups that are either producers or consumers of database technology.

Judy Robertson "On the Pleasures of Teaching Computer Science Students"

http://cacm.acm.org/blogs/blog-cacm/164619-on-the-pleasures-of-teaching-computer-science-students/fulltext
May 23, 2013

Yesterday was the most important day of my work calendar. We awarded degrees to 50 computer science students, thus fulfilling one of our main purposes as academics. I had the pleasure of telling some of the top students their marks in person. I've been doing this job for a good few years now, but I still can't quite get used to the buzz I get when students hear that they have succeeded. Their faces are pictures of incredulity, joy, but mostly relief that they made it. I am proud to have played a small part in their learning journeys.

We often interview students as part of our selection process into the first year, at the start of their journey. That can be extremely revealing. It's quite a daunting situation for some teenagers to find themselves in a room with an unknown adult and try to talk their way into a university course. From time to time, though, the young person's sheer passion for computer science shines out through the shyness. One chap this year was carefully cultivating an air of teenage boredom until we stared talking about computer games development, when he revealed his awe and reverence of his game development heroes (who no doubt were bored teenagers themselves once). Another candidate, the first member of his family to apply to university, spoke fondly of how he put his first computer together with his granddad when he was eight years old. School students at our Turing birthday party last year were delighted to talk to our students about their programming projects, as they said their teachers didn't understand what they were working on. I strongly remember one of our current Ph.D. students almost dancing with excitement when he got to talk with one of the professors about Open GL. He had been teaching himself for years but now he had someone else to talk to about his favorite topic. He had come home.

As academics, our role is to teach the foundations of computer science while fueling—rather than dampening—this passionate geekery. We try to fan the flames of geekery in those who have never had the good fortune to experience it before. It is hard for us to do this, and even harder for the students to keep motivated throughout the long journey to graduation. To get a CS degree at my university, you need to pass 32 different courses, picking up 480 credits on the way. On each of these 32 courses, there are possible ways to slip up: course work whose spec you cannot fathom, compilers that hate you, unit tests that spontaneously fail just before the deadline, exams in which your mind goes inexplicably blank. Many students also have the hurdles of young adulthood to deal with too—a potential mixture of financial hardship, leaving home, relationship break-ups, bereavement, or mental health difficulties.

In spite of all this, the students get through it. They learn where to put their semicolons. They grasp how Quicksort works. They sort out their matrices and master the halting problem. They fall in love with APIs and engrave comms protocols on their hearts. They learn how to write, how to present their ideas, how to think. This is a privilege to witness. Academics really do have the best job.

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology. **Judy Robertson** is senior lecturer at Heriot-Watt University.

Call for Nominations

The ACM Doctoral Dissertation Competition

Rules of the Competition

ACM established the Doctoral Dissertation Award program to recognize and encourage superior research and writing by doctoral candidates in computer science and engineering. These awards are presented annually at the ACM Awards Banquet.

Submissions

Nominations are limited to one per university or college, from any country, unless more than 10 Ph.D.'s are granted in one year, in which case two may be nominated.

Eligibility

Please see our website for exact eligibility rules. Only English language versions will be accepted. Please send a copy of the thesis in PDF format to emily.eng@acm.org.

Sponsorship

Each nomination shall be forwarded by the thesis advisor and must include the endorsement of the department head. A one-page summary of the significance of the dissertation written by the advisor must accompany the transmittal.

Deadline

Submissions must be received by **October 31, 2013** to qualify for consideration.

Publication Rights

Each nomination must be accompanied by an assignment to ACM by the author of exclusive publication rights. (Copyright reverts to author if not selected for publication.)

Publication

Winning dissertations will be published by ACM in the ACM Digital Library.

Selection Procedure

Dissertations will be reviewed for technical depth and significance of the research contribution, potential impact on theory and practice, and quality of presentation. A committee of individuals serving staggered five-year terms performs an initial screening to generate a short list, followed by an in-depth evaluation to determine the winning dissertation.

The selection committee will select the winning dissertation in early 2014.

Award

The Doctoral Dissertation Award is accompanied by a prize of $20,000 and the Honorable Mention Award is accompanied by a prize of $10,000. Financial sponsorship of the award is provided by Google.

For Submission Procedure

See http://awards.acm.org/doctoral_dissertation/

acm Association for Computing Machinery

Science | DOI:10.1145/2500468.2500472

Chris Edwards

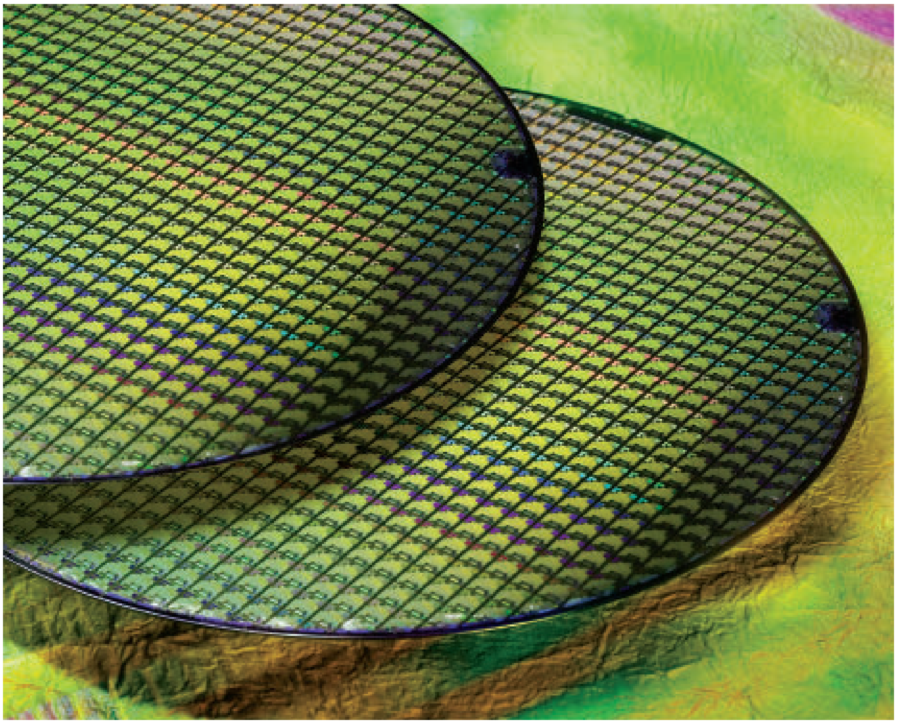
Magnetic Diversion for Electronic Switches

'Chameleon processors' could function as programmable logic or nonvolatile memory.

A PROGRAMMABLE LOGIC GATE that uses changes in magnetic fields to alter its behavior could provide a low-energy alternative to circuits based on the transistors that are used in practically all computers today.

Rather than take the approach of pure 'spintronic' computing devices [see "Computing with Magnets," *Communications of the ACM*, March 2012], the device developed at the Spin Convergence Center in Seoul, South Korea, uses magnetism to enhance the operation of semiconductor-based electronics.

Jinki Hong, professor at the Korea Institute of Science and Technology, who led the team at the Spin Convergence Center, says the aim of the work there is to develop 'chameleon processors.' "These are devices that function as both programmable logic and nonvolatile memory—retaining information even when not powered," says Hong.



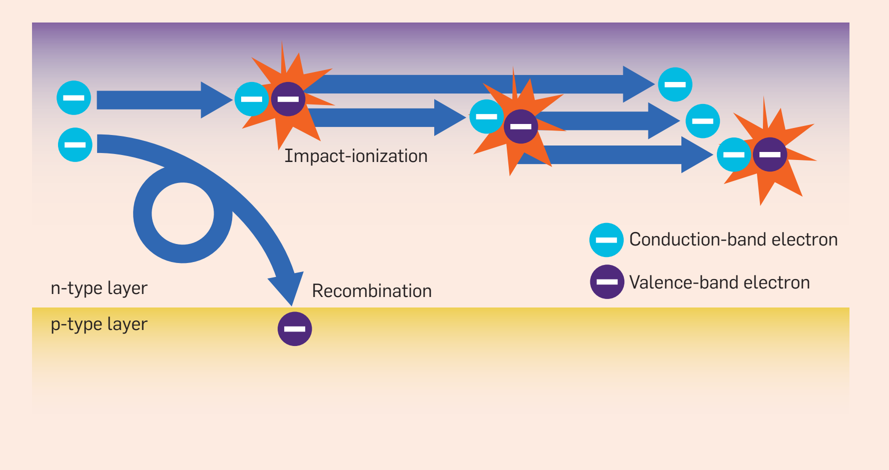
PHOTOGRAPH BY JOAN CHANG

By incorporating nonvolatile memory into their structure, devices like those developed in Seoul could slash the amount of energy that computers need to process data. Although it uses current to pass information between gates, switching the polarity of magnetic fields around the device performs logic functions. Two of these devices wired in series provide the same kinds of logic operation for which 10 or more transistors are needed today.

Sayeef Salahuddin, assistant professor at the University of California, Berkeley, says the motivation for developing spintronic and hybrid devices such as Hong's "is the realization that transistors are becoming excessively power-hungry. A more energy-efficient solution, even if low-performance, could make a difference."

The problem with the transistors used in all computers today is that they leak even when turned off. Planar

When magnetic fields are aligned to turn on the programmable logic gate, the flow of electrons results in an avalanche. In the off-state, conduction-band electrons are pulled toward the lower p-type layer and recombine with nuclei, moving them out of the conduction band.



transistors made on a 22nm process—which is currently the most advanced in production—have a ratio of on-state to off-state current flow of just two orders of magnitude. A decade ago, this ratio was close to a million. Designing ICs to support low leakage levels calls for devices that operate with a low on-state current, which generally results in low performance.

In 2002, Kailash Gopalakrishnan and colleagues at Stanford University proposed the impact-ionization switch as an alternative to the conventional transistor that could boost the current ratio to 10 million. The heart of the impact-ionization switch is a diode, rather than a transistor.

Diodes pass current in one direction, but almost completely block it in the reverse direction. A common application for this is in power supplies, where they are used to help convert alternating current (AC) into direct current (DC). However, there is a limit to how effectively they can block current.

Even when reverse-biased, a diode will pass very low levels of current that are largely the result of electrons absorbing enough thermal energy to break free from the bond to a nucleus and then being swept through the device in the direction of the applied voltage. These carriers are often involved in collisions with other electrons; if energetic enough, these electrons, in turn, can break loose.

A high voltage can generate an electric field that will accelerate electrons to high speeds, enough to free any electrons they strike. In turn, these cause even more impact ionization. The result is an avalanche that causes the diode to start passing high levels of current in the wrong direction.

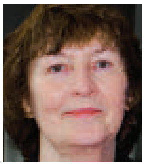
"Impact ionization is widely used in semiconductor devices, and many kinds of devices based on this technology have been already commercialized. For example, the avalanche photodiode is used for optical communication," says Hong. "Our device can be considered to be a magnetic-field version of an electrical diode."

The key to the operation of the Seoul device lies in its application of magnetoresistance, a property that revolutionized hard-drive technology 20 years ago. Magnetoresistance is the tendency for the resistance to electrical current to increase with changes in the strength and direction of a magnetic field. Traditionally, the effect was found strongly only in magnetic materials.

Five years ago, Bert Koopmans and

ACM Member News

DISTINGUISHED EDUCATOR DANN SEES THE ALICE PROJECT EMPOWERING PROGRAMMERS



An empty nest inspired former science and chemistry teacher Wanda Dann to switch her subject focus in midlife to computer science and visualization programming.

Dann, now an associate professor at the Carnegie Mellon University School of Computer Science in Pittsburgh, PA, in 2012 was named by ACM as one of six Distinguished Educators.

Today, Dann directs *The Alice Project*, a National Science Foundation-funded project that distributes the Alice 3D animation tool to programming novices to help them create visualizations and animations.

The Alice tool and its programming environment borrow from Lewis B. Carroll's classic *Alice's Adventures in Wonderland*. "Carroll told the tale with logic and visuals, and that's what we do with the Alice Project; we have 3D models of a dormouse, the Red Queen, dragons, mice, rabbits, and people," Dann says.

She observes that students learn very well, for example, with visual representations of molecular structures representing chemical bonding. "I wanted an effective way for novices to use visualization in programming," Dann says. The Alice Project provides software, support, textbooks and curriculum material, and workshops that incorporate technology and animation, "so teachers modify the way they teach computer science and programming while still maintaining the rigor of the programming concept and content for their students," Dann says.

The Alice Project has effectively improved retention rates among college freshmen using the software from 50% to about 80%, and it's now used in high schools and middle schools. The software also has a global reach, with one million downloads annually. Carnegie Mellon has expanded The Alice Project via summer faculty development workshops in the U.S. and abroad.

The most satisfying aspect for Dann is that she is empowering both teachers of programming and their students. She recalls teaching a class in random motion animation involving moving particles in a cloud of smoke; the animation required different programming segments for each motion. A student suggested generating random negative/positive, which cut the number of motion segments in half. "When a student comes back and one-ups me, that's a real achievement."

—*Laura DiDio*

colleagues at the Eindhoven University of Technology stumbled across a large magnetoresistive effect in the non-magnetic material silicon. "We saw very large effects that resulted from impact ionization."

The Seoul team used the non-magnetic material indium antimonide (InSb)—which has been proposed as a possible high-speed successor to silicon for future transistors—because its properties "lead to easy control of impact ionization by magnetic fields," Hong says.

The experiment, reported in the science journal *Nature* earlier this year, used externally applied magnetic fields to control the motion of electrons along a channel of InSb. Underneath this layer was a thin strip of InSb doped to have a low electron concentration—if electrons moved into this region, they would have a high probability of forming bonds with nuclei there and drop out of the free-carrier pool, preventing them from ionizing any more electrons. A magnetic field applied in one direction provided the necessary force, with the result of turning the device off. When the field reversed, the electrons were able to move from one end of the device to the other more freely and take part in impact ionization.

Building magnetic-field control into the device itself is a major stumbling block, but Hong envisages tiny ferromagnets being deposited alongside the InSb diode channels, pointing to work performed in the lab five years ago. "We demonstrated the fabrication of 1µm magnets," he says, noting that the techniques used in magnetic memories that are just beginning to move into production could be used to switch field directions as needed.

Michael Delmo, a postdoctoral fellow at Osaka University who has studied transport mechanisms that lead to magnetically controlled impact ionization, says incorporating magnetic elements into what is primarily an electronic device is a realistic proposition. "But this technology is in its infancy, and many things about it are still unknown," he adds.

The magnetoresistive device may lend itself to plastic electronics, in which organic polymers are printed onto a surface to form circuits.

In addition, the magnetically controlled impact-ionization device is large compared with today's logic transistors, and its current ratio is less than one order of magnitude. The question is whether the device will scale down easily and fulfill the promise of impact ionization devices to provide significant energy advantages over conventional transistors.

Koopmans says when the Eindhoven team found silicon could display large magnetoresistance, "It was a new and very interesting effect. But, at the time, I was not sure whether it would be easy to handle in miniaturized nanodevices. We had been working on rather large devices."

Chiara Ciccarelli, a researcher at the University of Cambridge who has investigated magnetoresistance in silicon, says as devices get smaller, "although there is no fundamental limit to room-temperature magnetoresistance, its magnitude decreases."

Ciccarelli points out that high-mobility materials, of which InSb is an example, show stronger magnetoresistance than silicon, which should help in smaller devices. Scaling could compensate in other ways. "As the length of the channel decreases, the electric field increases," she says. "That suggests miniaturization could play a positive role in the magnetoresistance of these devices. However, this is not what has been observed in the devices studied so far."

Even if magnetoresistance can be maintained at a high-enough level, there is a limit to how small the device can be made before impact ionization itself ceases.

"The minimum length of the channel is determined by a 'dead space,' the distance a carrier travels before acquiring enough energy from the electric field to participate in impact ionization," says Hong. This dead space may not shrink past 20nm—the length of the channel in today's most advanced transistors.

Even if it cannot be made as small as a logic transistor, the magnetically controlled device could still have a future. The nonvolatile nature of spintronic devices also means that for systems that are active only intermittently—such as smart sensors—the energy savings achieved by only having to power a circuit while it is processing could be immense, and outweigh silicon's likely advantage in size.

The magnetoresistive device may lend itself to plastic electronics, in which organic polymers are printed onto a surface to form circuits. These processes can be performed at very low temperatures and with much cheaper equipment than that used in conventional semiconductor fabs.

Delmo says the electron dynamics are different in organic semiconductors from those in silicon or InSb: "If these mechanisms do not depend on the device size, organic semiconductors could be strong candidates for magnetoresistive-based semiconductor logic technology."

Further Reading

Joo S., Kim, T., Shin, S.H., Lim J.Y., Hong, J., Song J.D., Chang J., Lee, H.W., Rhie, K., Han, S.H., Shin, K.H. and Johnson, M.
Magnetic-field-controlled reconfigurable semiconductor logic, *Nature 494*, 72, Feb. 7, 2013

Schoonus, J.J.H., Bloom, F.L., Wagemans, W., Swagten, H.J.M., Koopmans, B.
Extremely large magnetoresistance in boron-doped silicon. *Physics Review Letters 100*, 127202, Mar. 27, 2008

Ciccarelli, C., Park, B.G., Ogawa, S., Ferguson, A.J., and Wunderlich, J.
Gate-controlled magnetoresistance in a silicon metal-oxide-semiconductor field-effect transistor. *Applied Physics Letters 97*, 082106, Aug. 25, 2010.

Datta, S., Behin-Aein B., and Salahuddin, S.
Non-volatile spin switch for Boolean and non-Boolean Logic, Applied Physics Letters 101, 252411, Dec. 20, 2012. *Applied Physics Letters 95*, 132106, Sep. 30, 2009.

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

Technology | DOI:10.1145/2500468.2500473

Keith Kirkpatrick

Software-Defined Networking

Novel architecture allows programmers to quickly reconfigure network resource usage.

The demands of networked computer systems have changed dramatically since the early days of basic file sharing, peripheral sharing, or the hosting of companywide applications on a server. Today, organizations increasingly are using significantly more advanced computing environments to meet their needs, including cloud-based networks, virtualized desktops and servers, and remote data-storage devices, technologies that require significantly more computing resources, labor, and planning to properly deploy and maintain.

Enter software-defined networking (SDN), a new networking architecture that is designed to use standardized application programming interfaces (APIs) to quickly allow network programmers to define and reconfigure the way data or resources are handled within a network. The use of an API allows network applications (such as e-mail systems, cloud computing services, or telephony applications) to easily interface and reconfigure the network and its components (such as switches, racks of servers, virtual machines, and other end devices), or pull specific data, based on their particular requirements.

While SDN is not yet a household concept, it has garnered significant attention from major players in the virtualization and cloud computing space, another burgeoning segment of the computing world. Indeed, just over a year ago in July 2012, VMware Inc. agreed to acquire Nicira Networks, a Silicon Valley-based SDN startup that had flown under the radar for nearly five years, for $1.26 billion.

"Networking has remained stuck in the mainframe era for 15 years," says Andrew Harding, senior director of product marketing for Big Switch Networks, an SDN controller vendor. Harding notes that, while significant advances in other areas of technology have occurred (such as the shift from basic mobile phones to the world of smartphones using open APIs, like those found in the Android ecosystem), networking architecture and protocols have not kept pace, until now.

Configuring Networks

The promise of SDN can be thought of as being somewhat analogous to how mobile applications are built to interact with each other. In the mobile world, an application can make an API available to other developers for use in their own applications, without permanently modifying the first application. For example, Google Maps offers an API that allows applications to layer specific data on top of Google Maps data, such as a restaurant's location, by pulling in the relevant content. In essence, these location-based service applications are configuring Google Maps data for their own use, without requiring any change in the way Google Maps configures its data.

Similarly, in an SDN environment, applications can "reach through" to the network switches via an API and reconfigure the resources of the network to suit their needs. This ability to quickly reprogram or provision the network is achieved due to the way routers or switches are deployed.

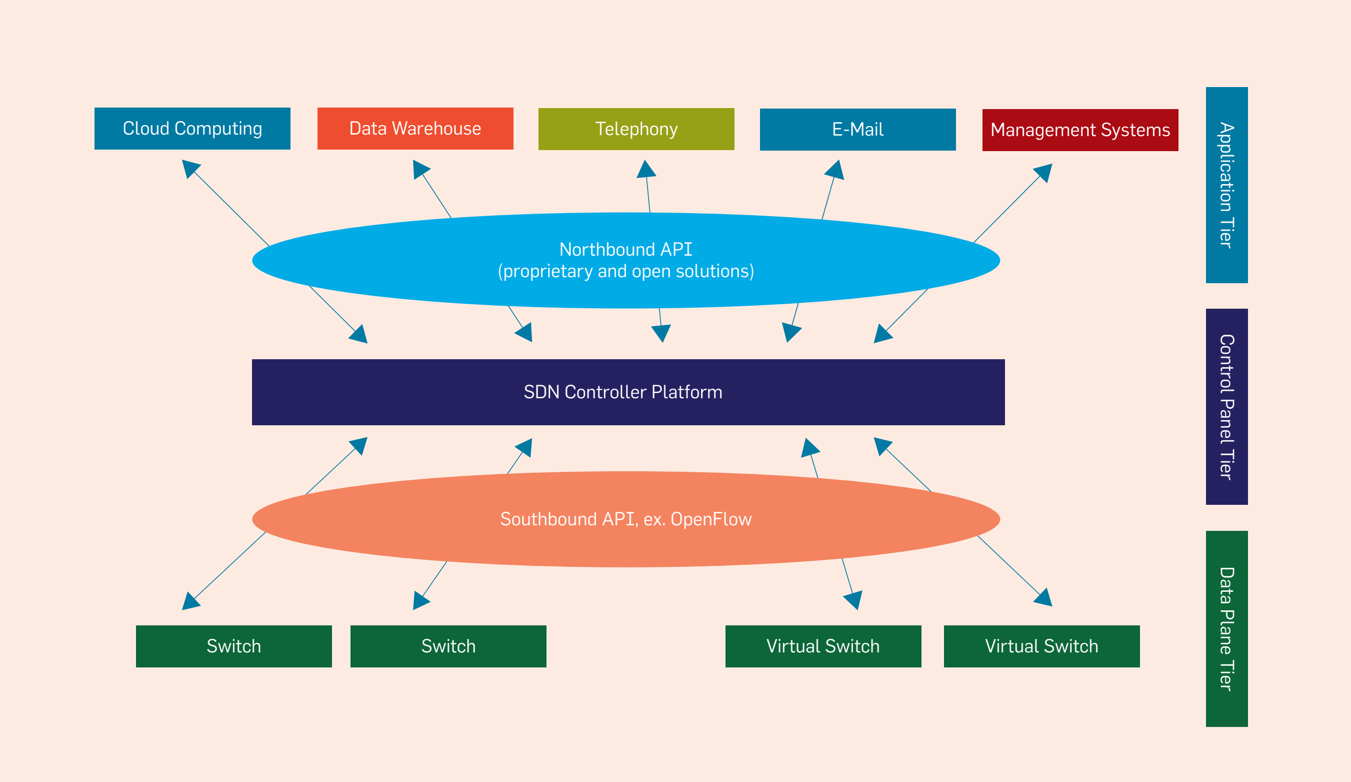
Historically, the transmission of data has been dominated by the use of dedicated switches or routers to direct

Network Virtualization:
Delivering on the Promises of SDN
Bruce Davie, Principal Engineer
vmware
ONS

Bruce Davie, principal engineer at VMware, notes SDN is "a mechanism, not a panacea."

COURTESY OF OPEN NETWORKING SUMMIT

An example of SDN architecture.



packets between servers or other connected devices. These switches consist of two "planes," or layers of the router interface: the data or forwarding plane, which handles the routing of data packets to its network destination; and the control plane, which creates the routing tables that determine how packets are sent to a destination. The control plane is also responsible for managing the connections between switches, handling errors and exceptions, and defining quality of service for different types of packets.

SDN decouples the link between the switch itself and the data-routing instructions, while adding an application programming interface between the two. These now "virtual" switches are not tied to any single piece of hardware or groups of devices, and thus can allow higher-level applications to pull data or reconfigure network resources from any connected device on the network.

"We're still stuck in this manual configuration era," Harding says. "But [now] you can program the network, [there are] a million applications you can pursue."

Network Interfaces

SDN can be defined as a three-tiered "stack" architecture, in which applications and high-level instructions sit in the top tier, a controller sits in the middle directing data traffic, and a third tier resides at the bottom, containing physical and virtual switches.

Each control device within a network is equipped with one or more interfaces, which enables the device to communicate with other components. In networking parlance, these interfaces are described directionally, with each direction related to the relationship between devices. For example, a northbound interface describes the communication with a higher-level component, while a southbound interface allows a particular network component to communicate with a lower-level component.

Northbound APIs

In SDN, the northbound API interface on the controller enables applications and the overall management system to program the network and request services from it. This application tier often includes global automation and data management applications, as well as providing basic network functions such as data path computation, routing, and security.

Currently, no formalized standards have been ratified for northbound APIs, with several dozen open and proprietary protocols being developed using different northbound APIs. The lack of a standard API is likely due to the varied nature of applications sitting above the controller, which can include managing cloud computing systems, network virtualization schemes, and other disparate or specialized functions.

Nevertheless, work on open northbound APIs is being done for specific vertical applications. OpenStack, a cloud computing effort backed by Arista Networks, Big Switch Networks, Brocade, VMware, and other SDN vendors, has developed the Quantum API, which is a vendor-agnostic API for defining logical networks and related network-based services for cloud-based systems. Several vendors have developed plug-ins for Quantum, which has helped it to become the default networking API for OpenStack, one of the largest open source cloud management platforms.

Southbound APIs

Though not explicitly required by SDN, OpenFlow is a protocol often used as the southbound API that defines a set of open commands for data forwarding. These commands allow routers to discover the network's topology and define the behavior of physical and virtual switches, based on application requests sent via the northbound APIs. Note, however, that while commonly used in SDN architectures, OpenFlow is not a requirement of SDN, and organizations may opt to use other types of southbound APIs for the control of switches and devices.

According to Dan Pitt, executive director of the Open Networking Foundation, a trade organization working to promote software-defined networking and the use of the OpenFlow protocol, the open protocol can assist organizations in scaling and reconfiguring their networks, while supporting the growing trend of network virtualization.

"The perpetuation of manual configuration through command-line interfaces has long held networking back from the advances in virtualization enjoyed by the computing world, and has led to high operating costs, long delays in updating networks to meet business needs, and the introduction of errors," Pitt says. "Eliminating the need to tie applications to specific network details like ports and addresses makes it possible to evolve the network's physical aspects without the delay and cost of both rewriting the applications and manually configuring the network devices."

The early benefits of SDN are largely going to stem from the use of network virtualization, which allows for more dynamic network segmentation and utilization.

Faster Hardware

Another key benefit with SDN, and its architectural configuration of splitting the data control plane from the routing tables, is the ability to incorporate faster, more powerful hardware. For example, a network switch can be used to handle the forwarding of data packets, while a separate virtual server can be configured to run the network control plane. This split configuration permits the network development and runtime environment to be located on a more advanced, speedier platform, rather than being relegated to the lower-end, slower management processors used on hardware switches and routers.

Benefits and Challenges

The early benefits of SDN are largely going to stem from the use of network virtualization, which allows for more dynamic network segmentation and utilization. SDN permits a more efficient use of network resources to support virtual machines (controlled by hypervisors, or the software used to support virtualization), as well as greater flexibility, via OpenFlow virtual switches. In essence, virtual machines that normally would be dedicated to static IP addresses can now be dynamically shared across virtual switches, allowing greater flexibility, as well as reduced operational expenses due to improved data efficiency and density.

While current real-world implementations are essentially pilot programs, the Open Networking Foundation is touting early efficiency gains and cost reductions. "When Genesis Hosting adopted SDN in their hosting facility, they gained a reduction in network administration costs of 50% and a reduction in IP address usage of 60%," Pitt says. "When Google converted its G-Scale WAN that interconnects its data centers worldwide to a 100% OpenFlow network, they reduced over-provisioning [to achieve] utilization levels above 95% with zero loss."

That said, not everyone is sold on the benefits of OpenFlow to drive SDN and other advanced networking appli-

Milestones

BCS Recognizes Two Oxford CS Lecturers

BCS, The Chartered Institute for IT, has recognized two University of Oxford computer science lecturers for outstanding contributions to the field.

Boris Motik has been named the recipient of the BCS Roger Needham Award, while Samson Abramsky F.R.S. is being awarded the BCS Lovelace Medal.

The Roger Needham Award, sponsored by Microsoft Research Cambridge, is awarded for a distinguished research contribution in computer science by a U.K.-based researcher within 10 years of receiving their Ph.D.

Motik was recognized for making major contributions to the design and standardization of the OWL2 ontology language, widely used in industrial research and applications. His research, on the development of techniques for intelligent management of large amounts of data, involves building advanced data management systems that can exploit background knowledge about an application domain in order to improve common tasks such as information production and search. Such systems are used in a broad range of applications, such as annotation of healthcare records, intelligent information management and retrieval in tourism, and provision of context-enabled services in consumer devices such as smartphones.

The BCS Lovelace Medal was established in 1998 in honor of Lady Augusta Ada Byron, Countess of Lovelace and daughter of Lord Byron. The medal is presented annually to individuals who have made a significant contribution to the advancement of information systems.

Since the 1980s, Abramsky has helped to set the modern research agenda for computer science. His contributions in each of the past three decades had a major impact on the field; they include domain theory in logical form, game semantics, and categorical quantum mechanics. He has shown the ability to change research fields and to establish new interdisciplinary approaches. His work over the past decade has shown that methods and concepts developed in theoretical computer science can be applied very directly in quantum information, and to the foundations of quantum mechanics.

Both Motik and Abramsky are based at the University of Oxford's Department of Computer Science. Motik will present the 2013 BCS Needham lecture in November, and Abramsky will deliver his Lovelace lecture in 2014.

For further information see: www.bcs.org/academy-awards.

cations. Pere Monclus, co-founder and CTO of PLUMgrid, a networking platform vendor, asks whether "OpenFlow is enough" to support the applications of today and tomorrow.

Monclus argues that the current rate of new versions of APIs (such as OpenFlow) being released is perhaps occurring too quickly, without allowing for innovations to drive the market. "If you want to go create an environment that you can develop applications that [will extend the functions of] the data plane, then OpenFlow is like a closed environment, because it already assumes that I know what you're going to need in terms of a data plane," Monclus says.

Furthermore, SDN is faced with a major challenge from traditional networking vendors, which have long relied on sales of their proprietary networking hardware and software, and could see a significant decline in the sales if SDN is rapidly adopted. Additionally, many organizations simply do not have the time, expertise, or capital to invest in a completely new networking architecture, particularly smaller organizations with limited IT staff and budgets.

That said, networking giants including Cisco, Microsoft, Hewlett-Packard, and IBM (among many others) joined in April 2013 with SDN vendors such as Big Switch Networks, Brocade, VMware, Arista, and PLUMgrid, to announce OpenDaylight, a new open source software project designed to create a collection of software for building networks that is largely based around the concepts of SDN. Despite the stated commitment to creating an industrywide, open source effort to promote SDN, there has been legitimate criticism that the larger networking companies have joined the consortium to push their own proprietary APIs to retain control over some of the hardware and software that may be used in future SDN implementations.

Many organizations simply do not have the time, expertise, or capital to invest in a completely new networking architecture.

Still, as a result of the industry's activities surrounding SDN, there is plenty of positive buzz about the technology's ability to reduce the time and cost of rolling out new applications, routing and transferring data, and reconfiguring networks virtually. This enthusiasm could result in gains in a decidedly non-virtual way: in March 2013, IDC projected the SDN market would reach $3.7 billion by 2016, capturing a 35% share of the switching market, up from what the market tracker quantified as "negligible" penetration in 2012.

Further Reading

SDN Inventor Martin Casado's 2005 Thesis on SDN: http://yuba.stanford.edu/~casado/vns_sigcse.pdf

"Ending The Confusion Around Software Defined Networking (SND): A Taxonomy," Gartner, http://www.gartner.com/resId=2367616.

Open Networking Foundation, October 2012 Interoperability Event Technical Paper,v0.4: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-test/onf-testing-interop-oct-2012-tech-doc-v0-4.pdf

Das, S., Parulkar, G., McKeown, N.
Why OpenFlow/SDN Can Succeed Where GMPLS Failed, *ECOC Technical Digest, 2012 OSA*, http://yuba.stanford.edu/~nickm/papers/ECEOC-2012-Tu.1.D.1.pdf

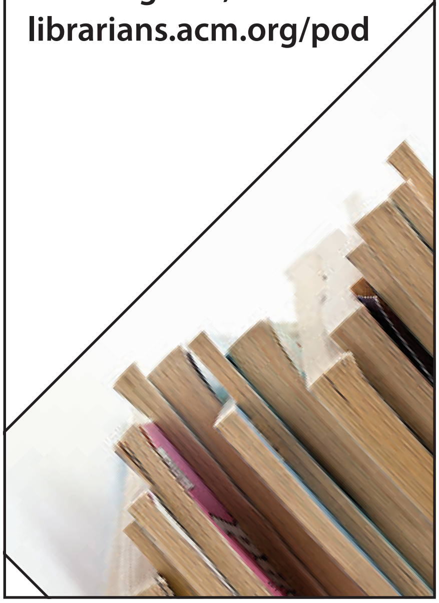
Kobayashi, M., Seetharaman, S., Parulkar, G., Appenzeller, G., Little, G., van Reijendam, J., Weissmann, P, McKeown, N.
Maturing of OpenFlow and Software Defined Networking through Deployments, *Elsevier August 14, 2012*, http://yuba.stanford.edu/~nickm/papers/openflow_deployment_journal_paper_aug2012.pdf

Video Content

Open Networking Summit 2013: Day One Highlights: https://www.youtube.com/watch?v=pDY9pjVhMew

Origins and Evolution of OpenFlow/SDN - Martin Casado, Open Networking Summit 2011: https://www.youtube.com/watch?v=4Cb91JT-Xb4

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



Society | DOI:10.1145/2500468.2500474

Esther Shein

Ephemeral Data

Privacy issues can evaporate when embarrassing content does likewise.

As people have discovered on social media sites, the past sometimes comes back to haunt you—and can destroy lives in the process. Recently, for example, a Florida woman made headlines by filing what is believed to be the first "revenge porn" suit in that state against an ex-boyfriend for posting naked photos of her years after their breakup. The woman is now trying to get a law passed in Florida that would criminalize "cyber stalking."

Such measures may be prompting some people to think twice as they send photos, email, videos, and texts they assume are between themselves and another party only. However, not everyone is that circumspect. Regardless of where you fall, the introduction of a new class of apps means users no longer have to wonder what gets saved by the recipient and what doesn't; these apps are designed to protect your privacy by making content disappear after a defined period. So-called ephemeral data is being made possible through apps provided by companies including Snapchat, Gryphn, Wickr, and Silent Circle.

"People are becoming increasingly aware that some social media platforms keep a permanent record of all that they post and all they comment on or rank or like or otherwise display in their feeds and profiles, so the market is responding with new services that are more evanescent,'' observes Lee Raine, director of the Pew Research Center's Internet & American Life Project, a non-profit, non-partisan "fact tank" that studies the social impact of the Internet. Apps that make data ephemeral are going to be used with increasing frequency, Raine believes, because people are beginning to recognize that the context of a post can be misunderstood, and it is better to have more control over exchanges and not risk the possibility that content stays out there and causes problems down the road.

PHOTOGRAPH BY KONSTANTIN CHRISTIAN

The content on social media sites is typically assumed to be persistent because most technical systems are architected in ways that prioritize keeping data, notes danah boyd, a senior researcher at Microsoft Research and a research assistant professor in Media, Culture, and Communication at New York University (NYU). They are designed this way in order to enable asynchronous interactions—and also simply because it's possible, she adds. It wasn't that long ago that storage was expensive and people were forced to toss data; the fact that we do not have to do so now is often relished, boyd says. Business interests also push designers to focus on persistence because of the possibility of increased traffic and, thus, increased ads, she says.

Yet, people are used to the concept of ephemerality in the real world, and—to a lesser extent—in the early years of the digital world, which was full of hard disk crashes, incompatible and changing document formats, and unreadable floppy disks, maintains Viktor Mayer-Schönberger, a lawyer and professor at the University of Oxford whose research focuses on the role of information in a networked economy. Concurring with Raine, he says, "As users of digital tools discover that the ephemerality they are so used to…is largely gone, they will become wary of the comprehensive digital memory that has replaced it—and look for market-based solutions to reintroduce it. That's just what this breed of new apps provides."

The apps provide the ability for a user to designate how long a recipient can view the content before it "evaporates.” While still in a nascent stage, such apps are proving invaluable to consumers and businesses alike.

“Ephemeral messages are incredibly freeing and make people communicate more authentically and freely with their friends,’’ says Jeremy Liew, managing director at Lightspeed Venture Partners, which has invested about $500,000 in Snapchat. The Snapchat app lets users set a duration of between three and 10 seconds in which the receiver can view sent content, before the data disappears. “They can be their real selves…because it’s not there forever.”

Apps like Snapchat also raise societal implications about people’s motivations for wanting data to be ephemeral, according to boyd. They “challenge the assumption that persistence is the right default and raises questions about when and where people don’t want persistence,” she says. “Given the privacy rhetoric…a persistence-by-default-minded assumption is that anyone who doesn’t want their data to be persistent has something to hide.” Yet most users do not see their use of these apps in that light, she says. boyd argues that, more often than not, people turn to data-disappearing apps “because they see no reason to add a particular artifact to their digital detritus because everything is already so cluttered.”

A classic case for using Snapchat is when someone takes a picture of something they think a friend will find funny in the moment because of a private joke; they send it off to the friend to make them laugh. However, boyd says, the goal was never for that photo to stick around; it was meant to be an ephemeral act.

“There are users who are trying to keep things secretive, and there are users who are trying to minimize how much data companies have about them,’’ she adds. “But the primary use case is more about challenging the status quo by creating a space for ephemeral media sharing.”

In his book *Delete: The Virtue of Forgetting in the Digital Age*, Mayer-Schönberger says that living in a world where everything shared online is saved forever creates problems for people and societies that need to forget in order to move ahead. When people forget details, they instead abstract and generalize, which keeps us from being “stuck in the past,” he says.

“Ephemeral messages are incredibly freeing and make people communicate more authentically and freely with their friends.”

“Human forgetting does not only ensure a modicum of real-world privacy, as we forget private details of what we experience over time,’’ he explains. “Human forgetting also helps us to live and act in the present and look toward the future, rather being tethered to an ever-more-comprehensive, but also stifling, memory of our past. A central element of being human is to generalize, to abstract, to see the forest rather than just the trees.”

Mayer-Schönberger observes, “Comprehensive digital memory constrains seeing ourselves and each other as evolving over time and changing. We are constantly reminded of who we were, not who we are.” He recalls the story of Canadian psychotherapist Andrew Feldmar, who wrote an academic article in 2001 in which he mentioned he had taken LSD in the 1960s. In 2006, while going to meet a friend in Seattle, an immigration officer at the U.S. border Googled Feldmar as part of a random search, and discovered the article. Because Feldmar had not disclosed that he had taken drugs some 40 years earlier—although when confronted he didn’t deny it—he was interrogated for hours and fingerprinted before being barred from entering the U.S., effectively forever. “So his past has come to haunt him—through comprehensive digital memory,” says Mayer-Schönberger. “And he was assessed not on who he was, but who he had been 40 years earlier.”

Ephemeral Data in the Enterprise

While ephemeral data apps are gaining popularity with consumers, they are also finding a niche in the enterprise. Less than a year ago, Silent Circle came out with a suite of products for phone, email, text, and video that use end-to-end encryption and erase the session keys from a device once a call or text is ended. The company’s servers do not keep the keys and the encryption keeps the unauthorized from understanding people’s transmissions.

“The goal is to provide secure communications for people who travel who believe their communications are being compromised, and once people have evidence this is going on, there is a need for it,” says Jon Callas, Silent Circle cofounder and CTO. He says the company has been gearing the apps more toward businesses because they were surprised by the demand for them from that sector.

For example, there is the Asian law firm that became a client because in the country in which the firm is based, “It is not only legal but part of the cultural milieu that lawyers spy on each other,’’ Callas says. “It’s part of the game; if you can find out something about the other party, then you can use it to your advantage.” The firm bought a subscription to Silent Phone, so its employees can make phone calls and feel secure that someone is not eavesdropping. Silent Circle’s apps offer a “burn” feature that lets them set a timer on how long information lasts before it is deleted.

Mayer-Schönberger predicts that if ephemeral apps continue to gain momentum, we will see more of them – not just in the form of innovative apps from startups, but also incorporated into mainstream offerings. “Ephemerality may thus become a market differentiator that gets incorporated into more and more offerings online,’’ he says. “And there are a lot of startups now working on establishing themselves as intermediaries helping people who want to ‘market’ their personal data to data users.”

Is Data Really Ephemeral?

While apps like Snapchat, Gryphn, and Wickr tout the fact that they make data disappear, security experts say that’s a slippery slope. Moxie Marlinspike, a contributor to Open Whisper Systems, which provides open source security for Android mobile devices, and soon for iOS, maintains there is no real way to securely enforce ephemerality.

“It’s fundamentally not possible in terms of how technology works, so all solutions are basically sort of faking it,’’ Marlinspike says. “The data itself is just data, and there’s an app that decides at some point, ‘I’m going to delete this.’ If an app decides to make data disappear, what’s to keep someone else from writing software that says ‘don’t delete it?’” he asks. “There’s no way to securely enforce that, and I think that’s OK in the case of Snapchat, because [it] isn’t really about security; it’s about this fun, ephemeral communication.”

Whisper Systems’ technology enables people to communicate privately and securely, he says, which has a different purpose than apps that make data disappear. “Ephemeral data can be so easily circumvented. There are ways to stymie the ephemerality of the communication. But with Snapchat, I’m not sure people are incentivized to do that,” Marlinspike says. “With security apps, the consequences of people [intercepting conversations] are generally more severe.”

Liew acknowledges there are always ways for people to capture data. “Even with Snapchat, you can take a screenshot of a picture.” In that instance, Snapchat informs the sender that a screenshot was taken. But Liew believes there is an unspoken etiquette among Snapchat users that this would be “considered extremely bad manners.” He likens this to the expectation when two people are having a phone conversation that no one is recording it, since that would be a breach of trust. “Yet that’s not the way most texting works—everything is archived forever. Snapchat is trying to get back to that traditional phone conversation model.”

Rainie agrees that people often assume things are private and safe, when oftentimes they are not. “Partly that’s a technical story and partly it’s a social story about what people will do when they find out about other people’s secrets.” He points out that companies and people with access to companies and people who are friends of people displaying information “can do things that compromise the context in which the picture was originally shared. That’s an artful way of saying there are ways this ephemeral data becomes unstuck.”

“Ephemeral data isn’t new—what’s new is we have enough compute power and enough storage to record everything.”

Liew also notes that Snapchat cannot guarantee impermanence, since there are ways around ephemeral data. For example, someone could use another device to take a photo of the screen. Yet making ephemerality the default does return to older norms of behavior in the same way that people do not expect their telephone calls to be recorded—but it is always possible for someone to do so.

He adds that there is an incorrect perception that Snapchat is primarily used for sexting. He notes that “If people want to send naked pictures, they can do it through any number of mechanisms and have done so via email or texting or via AOL chatrooms or the U.S. Postal Service. I don’t think ephemeral data makes that more or less likely.”

Echoing boyd, Callas points out that “Ephemeral data isn’t new — what’s new is we have enough compute power and enough storage to record everything. That’s relatively new in human history…It’s simultaneously intriguing and creepy.”

He believes broad social norms surrounding privacy, tolerance, and data will evolve in two directions—that a lot of things that are shocking right now will not be shocking down the road. The sexy photos that someone posted in their twenties will have less impact, Callas claims, because many people will be in similar situations. “It’s going to be common that people have embarrassing pictures, and it’s going to be that things that were embarrassing won’t be embarrassing…because of cultural norms. We will develop a new set of manners and etiquette.” C

Further Reading

Albrecht Schmidt and
Christian Søndergaard Jensen
Proceedings of the Eleventh international Conference on Database Systems for Advanced Applications. Springer, 2006. Pgs. 141-155 (http://vbn.aau.dk/en/publications/efficient-maintenance-of-ephemeral-data%2805d81e40-9fdd-11db-8ed6-000ea68e967b%29.html)

“Ephemeral Images, Indelible Data: The Promises and Challenges of Digital Imaging as Applied to Pathology,” NCI Biomedical Informatics Blog, Sept. 12, 2012. (http://ncip.nci.nih.gov/blog/2012/08/15/ephemeral-images-indelible-data-the-promises-and-challenges-of-digital-imaging-as-applied-to-pathology/)

“Courts Weigh in on Accessing Ephemeral Data for E-Discovery,” Information Management Journal, Vol. 46, No. 6, November-December 2012. (http://www.questia.com/library/1G1-321579846/courts-weigh-in-on-accessing-ephemeral-data-for-e-discovery)

boyd, danah.
2010. “Privacy and Publicity in the Context of Big Data.” WWW. Raleigh, North Carolina, April 29. http://www.danah.org/papers/talks/2010/WWW2010.html

Viktor Mayer-Schönberger
“Delete: The Virtue of Forgetting in the Digital Age.” Princeton University Press. July 5, 2011. (http://www.amazon.com/Delete-Virtue-Forgetting-Digital-Paper/dp/0691150362/ref=sr_1_1?s=books&ie=UTF8&qid=1368132105&sr=1-1&keywords=Delete%3A+The+Virtue+of+Forgetting+in+the+Digital+Age.)

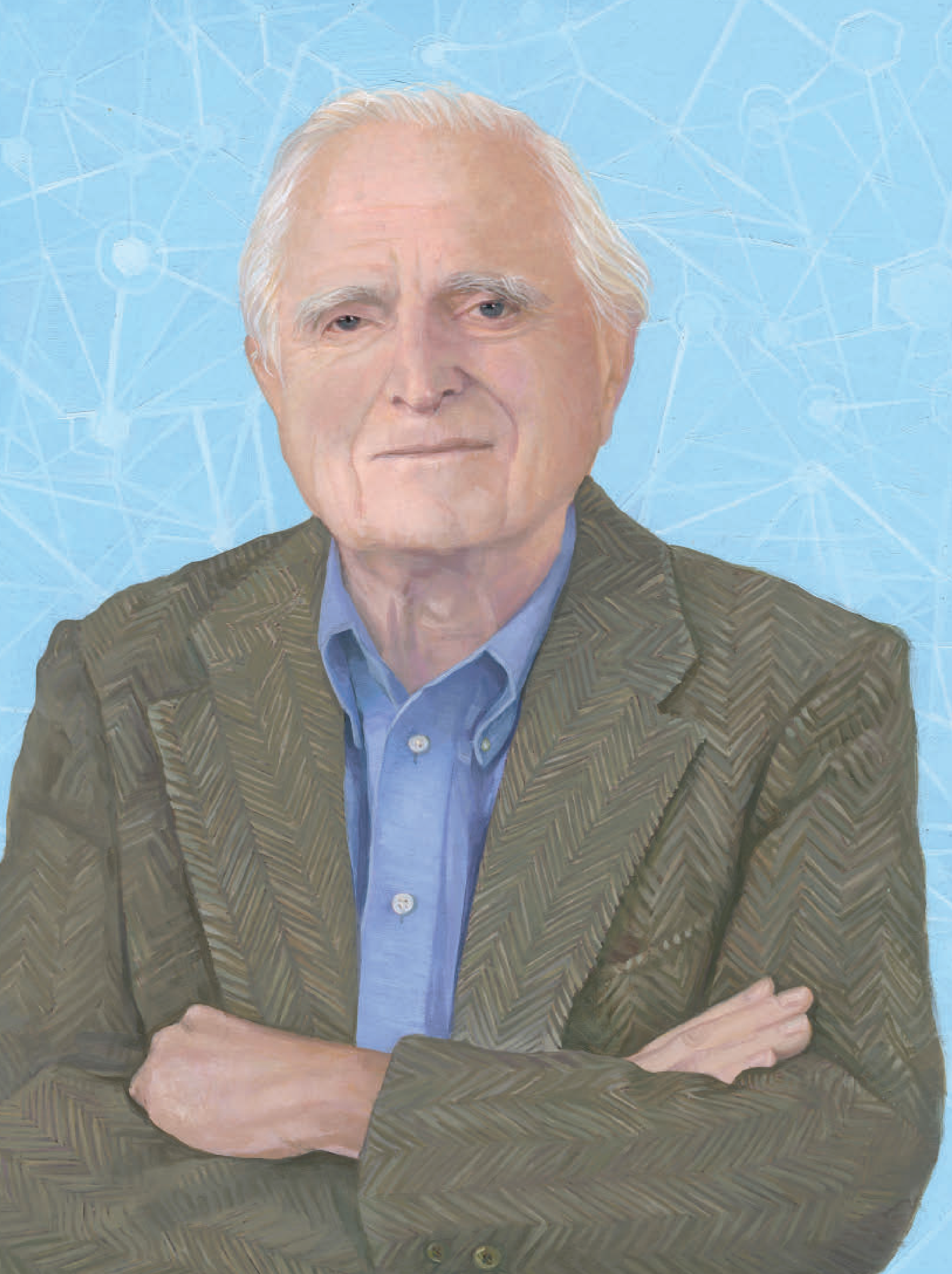
Evgeny Morozov
“The Net Delusion: The Dark Side of Internet Freedom.” Reprint edition Feb. 28, 2012 (http://www.amazon.com/The-Net-Delusion-Internet-Freedom/dp/1610391063/ref=sr_1_1?ie=UTF8&qid=1368131845&sr=8-1&keywords=The+Net+Delusion%3A+The+Dark+Side+of+Internet+Freedom.)

Videos

“A Peek Into the Circle” http://www.youtube.com/watch?v=yJpCW3DOmiY

“Ephemeral Images, Indelible Data: The Promises and Challenges of Digital Imaging as Applied to Pathology.” http://www.youtube.com/watch?v=j1yh-Oak528

Esther Shein is a freelance technology and business writer based in the Boston area.



In Memoriam | DOI:10.1145/2500468.2500481

Laura DiDio

Remembering Douglas Engelbart

“INSPIRING,” “VISIONARY,” “HUMBLE,” “honest,” “impeccable integrity,” “passionate and stubborn about his work.”

Tributes poured in for Douglas Engelbart, inventor of the computer mouse and an Internet pioneer, following his death from kidney failure July 2 at his home in Atherton, CA. People from all echelons of the technology community extolled Engelbart’s enduring legacy and hailed his more than six decades of work in computer research and inventions as not just innovative technology, but as mechanisms that could transform the world and solve society’s problems.

“Doug once said the least-important things he did were his inventions, such as the mouse,” recalls Alan Kay, president of the Viewpoints Research Institute (VRI), a nonprofit public benefit organization on whose Board of Advisors Englebart served. “What was most important [to him] was what he was trying to improve, and how he was trying to improve it,” Kay said.

Kay said many of Engelbart’s key contributions had to do with how ideas and actions can be synergistically intercommunicated and constructed into better forms and processes with the aid of technological amplifiers.

Engelbart’s technology vision attracted a devoted coterie of engineers who shared his philosophy. “Doug had more technical groupies than anyone I have ever known. He touched and inspired people way beyond technology,” recalled Curtis Carlson, chief executive officer at SRI International, where Engelbart held one of his first technology jobs. “His impact and legacy are still felt at SRI today.”

“No one before Doug thought about the customer experience and the impact it would have on corporations and hundreds of millions of people worldwide,” Carlson said. “There would be “no Apple or Xerox PARC without Doug Engelbart. ...It was a magic moment when Steve Jobs intercepted Doug Engelbart at exactly the right time and started using the computer mouse.”

Kay, the 2003 A.M. Turing Award recipient, said Engelbart—the man and his work—had a profound influence on him. “Doug had some of the most significant insights about humans, and about how solving some of our most important human problems could be aided/augmented by the visionary use of computers.”

In the 1960s, Kay witnessed Engelbart’s now-famous “Mother of All Demos” presentation of interactive computing designed to support collaborative work groups. “Doug inspired funders and computer people to combine to make a series of ‘living lab’ systems that both could demo some of the ideas, and were also engineered and fleshed out well enough to be put into daily use,” Kay said.

Logitech CEO Guerrino de Luca considered Engelbart “one of the most under-recognized geniuses of our times,” even though he received the ACM A.M. Turing Award in 1997, and received the National Medal of Technology and Innovation from President Bill Clinton in 2000. Despite these awards and any measure of recognition he managed to achieve, “Doug stayed humble, curious, and accessible to ideas and people all his life,” de Luca said.

Distinguished Career

Engelbart joined SRI (then known as Stanford Research Institute) in the 1960s. His first assignment was working on magnetic devices and the miniaturization of electronics; he eventually received more than a dozen patents for his work. At SRI, Engelbart founded the Augmentation Research Center (ARC), where he and a team of about a dozen engineers focused on developing new collaboration and information processing tools. He and his team, which included lead engineer Bill English, developed bitmapped screens, the computer mouse, hypertext, and initial steps toward the graphical user interface.

Engelbart and his team became known for their intensity, devotion, and committed work ethic. They had endless discussions and debates and were willing to work 12- and 15-hour days, Carlson said. “In 2013, when we have the Internet and ubiquitous connectivity, it is not uncommon, but in the context of the 1960s work ethos where a 9-to-5 workday was the accepted norm, it was unheard of,” he noted.

Engelbart was well liked and respected by executives, colleagues, and those who worked for him. “He was soft-spoken, friendly, and approachable,” Carlson recalls.

Transformative Power

Everyone interviewed for this article said Engelbart was extremely tenacious, which worked both for and against him. Early in his career, Englebart’s passion for his work and ideas had investors flocking to fund him; by the 1980s, however, he struggled to find backers.

“His biggest quirk was being stubborn and trying to stay 100 years ahead of everyone,” Carlson said.

Kay said one of the great ironies of Engelbart’s later career was that “He had a harder and harder time getting heard, because he stayed with his message.”

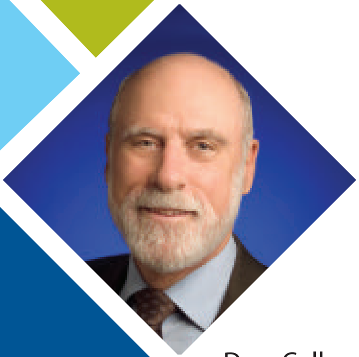
Ultimately, “visionary” was the word used most often to describe Engelbart.

Carlson claimed that in addition to his renowned “Mother of All Demos,” Engelbart originated the concept that technology speed and performance double every two years, famously known as Moore’s Law after Intel Corp. co-founder Gordon Moore. “Doug was the first person who gave a talk about the performance scaling law for integrated systems,” Carlson asserted.

De Luca said Engelbart’s vision “of bootstrapping Collective IQ, his passion in life, was a 50-year-plus vision. His belief that technology is the key to mankind’s ability to solve difficult problems collectively has the transformative power that few others share.”

Laura DiDio is Principal Analyst at Information Technology Intelligence Consulting (ITIC).

ILLUSTRATION BY LOUISE WEIR, BASED ON PHOTOGRAPH BY SCOTT BRAMWELL



Association for Computing Machinery

Global Reach for Global Opportunities in Computing

Dear Colleague,

Today's computing professionals are at the forefront of the technologies that drive innovation across diverse disciplines and international boundaries with increasing speed. In this environment, ACM offers advantages to computing researchers, practitioners, educators and students who are committed to self-improvement and success in their chosen fields.

ACM members benefit from a broad spectrum of state-of-the-art resources. From Special Interest Group conferences to world-class publications and peer-reviewed journals, from online lifelong learning resources to mentoring opportunities, from recognition programs to leadership opportunities, ACM helps computing professionals stay connected with academic research, emerging trends, and the technology trailblazers who are leading the way. These benefits include:

Timely access to relevant information

- ***Communications of the ACM*** magazine
- ***ACM Queue*** website for practitioners
- Option to subscribe to the ***ACM Digital Library***
- ACM's ***50+ journals and magazines*** at member-only rates
- ***TechNews***, tri-weekly email digest
- ***ACM SIG conference*** proceedings and discounts

Resources to enhance your career

- **ACM Tech Packs**, exclusive annotated reading lists compiled by experts
- **Learning Center** books, courses, webinars and resources for lifelong learning
- Option to join **36 Special Interest Groups** (SIGs) and **hundreds of local chapters**
- **ACM Career & Job Center** for career-enhancing benefits
- ***CareerNews***, email digest
- **Recognition of achievement** through Fellows and Distinguished Member Programs

As an ACM member, you gain access to ACM's worldwide network of more than 100,000 members from nearly 200 countries. ACM's global reach includes councils in Europe, India, and China to expand high-quality member activities and initiatives. By participating in ACM's multi-faceted global resources, you have the opportunity to develop friendships and relationships with colleagues and mentors that can advance your knowledge and skills in unforeseen ways.

ACM welcomes computing professionals and students from all backgrounds, interests, and pursuits. Please take a moment to consider the value of an ACM membership for your career and for your future in the dynamic computing profession.

Sincerely,

Vint Cerf

President
Association for Computing Machinery

acm
Association for
Computing Machinery

Advancing Computing as a Science & Profession

Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: AD13

You can join ACM in several easy ways:

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

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

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

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

Please print clearly

Name

Address

City State/Province Postal code/Zip

Country E-mail address

Area code & Daytime phone Fax Member number, if applicable

Purposes of ACM

ACM is dedicated to:
1) advancing the art, science, engineering, and application of information technology
2) fostering the open interchange of information to serve both professionals and the public
3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/about/code-of-ethics

choose one membership option:

PROFESSIONAL MEMBERSHIP:

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

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

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

STUDENT MEMBERSHIP:

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

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

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

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

**All new professional 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.

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

o Professional Member Dues ($99 or $198)	$ ____________________
o ACM Digital Library ($99)	$ ____________________
o Student Member Dues ($19, $42, or $62)	$ ____________________
Total Amount Due	$ ____________________

Card # Expiration date

Signature

DOI:10.1145/2500129

James Grimmelmann

Law and Technology

What to Do About Google?

Whether it is acting as a conduit, an editor, or an advisor, the search engine should put user interests first.

GOOGLE IS A NOUN, a verb, and a controversy. It receives two-thirds of all searches in the U.S., and more than 90% in many European countries. It has dipped its toes—or perhaps its tentacles—into local listings, news, books, videos, flights, patents, and prices, to name just a few. If it exists, Google wants to index it.

Unsurprisingly, this modern octopus has its critics. There are, among others, newspapers upset about having their headlines scraped and aggregated, trademark owners upset about keyword ads for their competitors, introverts upset that searches on their names resurrect painful and humiliating memories, governments upset at the subversive and scandalous things citizens can find with a quick search, and privacy advocates upset at Google's immense stockpiles of personal data. And that is just the search engine; if one were to add in the concerns about Android rootkits, Google Glass creep shots, driverless car crashes and the rest, this column would not be long enough to list them all.)

The oldest and most persistent critique of Google's power, known as "search bias," is the fear that search rankings create reality rather than reflecting it. If Google demotes the restaurant Le Snoot from being the first result for "restaurant near 54321" to the hundreth, many gourmets will make their reservations elsewhere. If Dave's Diner takes its place as the number-one result, diners will go there instead. Google can literally pick winners and losers in the game of the Internet.

The most explosive search bias allegations against Google involve its vertical search engines, like Google+ Local (to find nearby businesses and restaurants) and Google Flights (to find and book airline tickets). Google gives these specialized search results prominent placement at the top of its results pages. Competitors like Yelp and Expedia have charged that this gives Google's vertical offerings an unfair advantage over their competing vertical search engines. For years, they and other Google critics have been pressing regulators in the U.S. and the European Union to curb Google's allegedly abusive practices.

Google can literally pick winners and losers in the game of the Internet.

But they have been sorely disappointed, on both sides of the Atlantic. In January, the Federal Trade Commission dropped its search bias investigation, concluding the changes competitors complained about "could plausibly be viewed as an improvement in the overall quality of Google's search results."[5] In April, the European Commission went a bit further, but not much. In a proposed settlement, Google agreed to label its own vertical search results more prominently, and to add a few, not particularly conspicuous, links to rival vertical search en-

ILLUSTRATION BY ANTHONY FREDA

gines.[6] Both regulators left untouched the core practice responsible for so much criticism: top-ranked placement for Google's own news, flight information, and local results.

Did the authorities shirk their responsibilities to rein in an unruly titan? Or did they show admirable restraint in refusing the gum up the gears of an innovative technology? It is impossible to answer these and other policy questions about Google without some theory of what search engines are good for and what society ought to expect of them.

Fortunately, we have such a theory—or rather, we have three such theories. Some observers have compared Google to a traditional telecommunications *conduit* like a radio station.[2] Some have compared it to an *editor* deciding what stories to put in a magazine.[8] And some have compared it to an *advisor*, like the concierge in a hotel who answers questions about local attractions.[7] Each theory offers its own insights.

Calling search engines conduits emphasizes that they have become one of the new bottlenecks on the Internet. If Le Snoot's ISP decides to unplug its connection, no one will be able to reach lesnoot.com. The same will be true if the DNS records for lesnoot.com are deleted, or if search engines drop lesnoot.com from their indexes. And so, if the parallel holds, just as the Bell telephone network was regulated to ensure nondiscriminatory access for everyone, search engines should be too.

When people talk about "search neutrality"—by analogy to "network neutrality"—they are making an argument for treating Google as a conduit. Of course, Google could not simply rank all websites identically, because only one result can be first, but it ought to treat them all fairly. The opposite of a "neutral" search engine is a "biased" search engine; rather than listing websites in the order they deserve to be ranked, it injects its own discriminatory distortions. The claim that Google is doing something wrong when it puts its own flight search results higher on the page than Expedia's is a claim that Google should be acting as a neutral conduit but is not.

The conduit theory's natural enemy is the editor theory, which says that making distinctions among webpages is an act of judicious discretion rather than dangerous discrimination. The editors of *Communications* make countless decisions about which developments in computing are worth covering, which articles are most informative, and where to put them in the magazine. These decisions are not "right" or "wrong"; they simply reflect the judgment of its editorial board and staff. Google sees itself the same way. True, Google's editorial cycle is measured in milliseconds rather than in months. But when its search quality team meets to discuss algorithmic tweaks, it resembles a newspaper staff debating which stories to put on the front page of the metro section. And, continues the argument, just like the government cannot tell the *New York Times* to spike an unflattering story about Guantanamo Bay, it cannot tell Google which search results to show.

Finally, one could view Google as an advisor, helping users find what they are looking for. If so, the best search engine is one that is most useful to users, rather than the one that is least biased, or most reflects its programmer's point of view. Le Snoot may be the "best" restaurant in town, as judged by professional food critics. But some people do not like heavy French cuisine, others are vegans, and even cassoulet lovers would rather just have a slice of pizza now and then. Whether Le Snoot or Dave's Diner is more relevant to a user depends on what she intends as she types her query.

Calling Google an advisor cuts both ways: it gives Google both rights and duties. It gives a powerful argument against search neutrality: a law that puts Le Snoot back on top makes it more difficult for the user who wants a grilled cheese sandwich to get a decent meal. But just as readers would rightly be furious to discover the hotel concierge only recommended Le Snoot because the head chef slipped him an envelope stuffed with cash, search users would also have cause to complain if payola determined search rankings. More than a decade ago, the FTC strongly warned search engines against displaying undisclosed paid listings.[4]

All three theories capture something important about how search engines work. Each of them celebrates the contributions of one of the essential parties to a search. The conduit theory is all about websites with something to say, the advisor theory is all about the users who are interested in listening, and the editor theory is all about the search engine that connects them.

But when it comes to crafting sensible law for search engines, our sympathies should lie with users. The Internet has made it easier to speak to worldwide audiences than ever before, but at the cost of massively increasing the cacophony confronting those audiences. Since users' interests are as diverse as human thought, they need highly personalized help in picking through the treasures in the Internet's vast but utterly disorganized storehouse. The search engine is the only technology known to humanity capable of solving this problem at Internet scale.

Some familiar controversies about Google look rather different from this point of view. Take search bias. If Google is a conduit, bias is a serious problem; Google is setting up orange cones to block the highway and divert Internet users to the Google exit. If Google is an editor, bias is just as much a non-issue as when the front page of the *Daily News* promotes its own sports coverage rather than the *Post*'s.

When it comes to crafting sensible law for search engines, our sympathies should lie with users.

If Google is an advisor, though, the answer lies somewhere between "always wrong" and "always fine." The key question is not whether Google is helping itself or whether it is hurting websites, but whether it is helping users find what they want. Sometimes, for some queries, Google can quite reasonably think that users will be grateful if it lists its own services first. Flight search is a good example: Google's interactive OneBox helps users dive right into the flight-picking process.

At other times, for other queries, Google may have strong evidence that users prefer particular sites. If Google demotes them to insert its own pages that it knows users would rather not see, that could be problematic. It is a form of deception: Google is telling the user, "This is the best I can do for you" when it knows full well it could do better.

The FTC properly recognized that deception was the real issue in the Google case. The FTC's decision to drop its search bias investigation hinged on a conclusion that Google had not underplayed its hand. Some, like Expedia and Yelp, criticized the outcome. But there is a difference between *disagreeing* with Google's ranking decisions—everyone wants to be king of the results page—and showing that those decisions were made in bad faith.

Another advantage of treating search engines as advisors is that it helps put user privacy at the center of the conversation. We depend on advisors to keep confidential what we tell them: doctors and investment advisers are legally obligated to secure their records; so too for search engines. Our query histories are some of the most personal and potentially embarrassing data trails we leave behind us. They have even been used as evidence in murder trials. Strong privacy protections for user search data are essential.

Some of these points apply beyond search engines; some do not. The anti-payola principle is a general one; the FTC has warned advertisers that they must disclose sponsored blog posts, and even sponsored tweets.[3] So is the idea that the government should not make users' choices for them; Tulsa cannot tell Yelp that the Holiday Inn deserves an extra star and the Ramada does not. But the duty of loyalty is weaker where advice is not personalized; consumers can continue to leave humorous reviews of the Three Wolf Moon T-shirt at Amazon, even though the reviews may not be especially helpful for shoppers.[1]

Google is not the Eye of Sauron, finding all that is good on the Internet and corrupting it. Nor, despite its mission "to organize the world's information and make it universally accessible and useful," is it humanity's informational savior. Google is a company that provides an enormously significant online service. When that service raises serious legal questions, we should ask whether it is good for the users or bad for the users.

References

1. Amazon.com. The mountain three wolf moon short sleeve tee; http://www.amazon.com/The-Mountain-Three-Short-Sleeve/dp/B002HJ377A
2. Chandler, J.A. A right to reach an audience. Hofstra *Law Review 35*, 3 (2007), 1095–1137.
3. Federal Trade Commission. *.Com Disclosures*, 2013; http://www.ftc.gov/os/2013/03/130312dotcomdisclosures.pdf.
4. Federal Trade Commission. Letter to Gary Ruskin. *Re: Complaint Requesting Investigation of Various Internet Search Engine Companies for Paid Placement and Paid Inclusion Programs*, (June 27, 2002).
5. Federal Trade Commission. Statement Regarding Google's Search Practices. *In the Matter of Google Inc.*, FTC File No. 111-0163 (Jan. 3, 2013).
6. Google Inc. Commitments. *Foundem and Others*, Case COMP/C-3/39.740 (Apr. 3, 2013).
7. Grimmelmann, J. Speech engines. *Minnesota Law Review* (2014), in press.
8. Volokh, E. and Falk. D.M. Google first amendment protection for search engine search results. *Journal of Law, Economics, and Policy 8*, 4 (2012), 883–899.

James Grimmelmann (james@grimmelmann.net) is Professor of Law at the University of Maryland.

DOI:10.1145/2500131

Thomas Haigh

Historical Reflections
Software and Souls; Programs and Packages

How can historians tell stories about software without focusing solely on the code itself?

IMAGE FROM SHUTTERSTOCK.COM

Despite his central place in the development of Western philosophical thought, René Descartes is only occasionally cited in computing literature. However, the way we think about hardware and software tends to follow the template established by centuries of Dualist philosophy and thousands of years of human spiritual belief. Hardware is the body. Encumbered by the material world it is constrained by physical laws and will eventually succumb to the ravages of time. Software is the soul. The essence of code is its immateriality. An invisible spark of life, it controls the operation of the machine and can transmigrate from one host to another. It is bound not by the laws of this world but by those of another. The distinction between hardware and software partitions the careers, journals, conferences, interest groups, and businesses of computing into separate camps. In recent years it has also shaped the work of historians of computing, as software history has become an increasingly popular area of research.

This column explores some of the fundamental challenges software poses to historians and draws upon my own historical research in the area to argue that software never really exists in a pure, non-material form. Instead, software has always been treated historically as a "package" including more than just computer code—though what has been meant by this has changed significantly over time.

However, as you are reading *Communications* you probably know a lot about the insides of computers and may very well be an academic computer scientist. So let me admit two objections to my opening generalization before I move on with the historical reflections. First, the lines between hardware and software have become less clear over the years. Technologies such as FPGAs, virtual machines, APIs, and microcoded instruction sets complicate the simple picture of programs directly manipulating hardware. Second, the recent Turing Centenary has reminded us that the founding insight of theoretical computer science is that hardware and software are, from the viewpoint of computability, almost entirely interchangeable. Still, a primary consequence of this insight has been to allow theorists to ignore issues of platforms and architectures entirely, further reinforcing our sense of programs and algorithms as creatures of pure logic unsullied by materiality.

Software in Museums and Archives

Museum curators and exhibit planners are the people challenged most directly by the special nature of software. Exhibiting computer hardware is not so different from exhibiting a stone axe head or a stuffed Dodo. You put the object in a glass display case, next to an identifying label. Visitors peer briefly at it as they walk by, and a few stop to glance at the text. The objects are arranged to tell some kind of story as the visitor walks. Usually it is a story of progress over time, and so the visitor notices objects in the cases becoming prettier or more complicated the closer he or she gets to the gift shop. These days there tends to be more focus on the story and less on the clutter, while interactive screens are supplementing wordy explanations. The stuffed Dodo and the mainframe would both benefit, should space and funding permit, from being placed in a diorama representing their natural habitat.

You cannot put a soul in a display case. Curators at leading museums such as the Science Museum in London and the Deutsches Museum in Munich have long been aware of the importance of software and have been grappling for a while with the question of how to collect and display it. Traditional approaches are not very satisfactory. One could line up cases full of disks, tapes, and shrinkwrapped boxes to represent the mass-market products of the late-1970s and 1980s, but this would not tell us much about the software itself and would not work at all for early software, enterprise software, internally produced software, or today's downloaded applications.

The challenge is daunting, which is why despite years of discussion no major museum has attempted a full-scale exhibit on the history of software. Even the Computer History Museum in Silicon Valley included relatively little on software in its recently unveiled permanent exhibit. To remedy this the museum is now working on a supplemental exhibit focused on software, and it will be interesting to see what solutions its curators come up with.

Archiving software presents its own challenges. Even if the medium holding the code is preserved then it is far from certain the bits will still be readable decades from now (a particular problem for magnetic media) or that any system able to run the software will still exist. A growing community of enthusiasts has developed online repositories, emulators, and physical archives to address these issues systematically. Coverage is pretty good in areas where collection is easy and nostalgia rampant, such as video games where Henry Lowood at Stanford University has built a substantial archive and most classic titles can easily be downloaded and played from amateur collections of dubious legality. Things like accounting software or online systems are much less likely to survive.

Conceptual Challenges

Thinking about the challenges involved in displaying and archiving software makes me glad that, as a historian who works on books and articles rather than exhibits, I can dismiss them as fascinating problems for someone else to solve. We historians of technology like to think of ourselves as storytellers, writing narratives in which technology, in one way or another, serves as our protagonist or plays a major role. This sidesteps some of the practical problems faced by curators and archivists but, alas, raises a whole set of new problems. The most important of which is: What is software anyway?

Existing historical writing on software has focused on just a handful of topics. One has been the early history of software engineering, particularly the seminal NATO Software Engineering conference held in the Bavarian Alps in 1968. Another has been the history of the software industry, given a thorough overview by Martin Campbell-Kelly in his book *From Airline Reservations to Sonic the Hedgehog*. There has also been a significant body of work about programmers, the people who produce software. Finally we have a significant number of historical accounts of particular software technologies, such as programming languages, mostly written by the people responsible for creating them.

What we do not have, as yet, is a history of software itself; a history of software as a thing or—as a museum curator might call it—an artifact. Such a history might appeal far beyond the (rather small) group of people who think of themselves primarily as historians of information technology. Enthusiasm among humanities and media studies scholars to come to grips with software has recently produced a somewhat dizzying range of self-proclaimed fields and movements, including software studies, critical code studies, video game studies, and demoscene studies. While these identities are still in flux, and media theorists can be rather fickle in chasing after hot new topics, they indicate a broad interest in understanding software as a new and complicated kind of artifact.

Perhaps the most relevant new agenda comes from "platform studies," launched by Ian Bogost and Nick Montfort with their beautifully crafted book *Racing the Beam*, an analysis of Atari's VCS game console that was a fixture in American basements and living rooms during the late 1970s and early 1980s. The focus on platforms recognizes the fact that software without hardware is nothing; as is hardware without software. Each gives meaning to the other. This is particularly apparent to programmers of the VCS, whose 128 bytes of memory could not accommodate a bitmapped screen. Game code was timed precisely against the speed at which the television's electron beam moved across its screen. Thus a program would structure all its activity around the need to place the bits representing the next screen line into the register controlling video output before the TV began to draw that line. This is an extreme case, but it reminds us that platforms are the most enduring technological systems in the world of computing, and that all software is shaped by the constraints of one platform or another. The platform approach also holds out the prospect for meaningful engagement between historians and other kinds of humanities and media scholars.

Existing historical writing on software has focused on just a handful of topics.

Software as Package

Back to software itself. The challenge facing historians is to find ways to tell stories about software that illuminate the fascinating and mysterious nature of software artifacts without falling into the opposite trap of narrowing our focus to look exclusively at the code itself. Our preference is generally to reconstruct the ways of seeing the world that made sense to people in the times and places we are writing about, rather than to impose alien perspectives such as those based on present-day concerns or on something more esoteric like postmodern literary theory.

Fortunately the history of software holds just such a concept, hidden in plain sight. That is the idea of software as a package. We still speak of software packages, yet we rarely stop to consider the implications of the idea. It has a long history: people started talking about programs as packages a couple of years before they started calling them software. So while the idea of packaged software has recently been associated with the fading practice of literally putting programs into shrink-wrapped boxes it was around for decades before computer programs became retail goods. In fact the very idea of programs as software is bound up with the idea of packaging, and goes back to earliest occasions on which people started to think about how programs developed with one computer center could be used by another one.

The practice of sharing programs is as old as the practice of writing programs (and older than the practice of executing them—some have claimed as the first computer programs material published in the 1840s to promote Babbage's never-finished Analytical Engine). The very first computers, built during the 1940s, were experimental one-off machines with their own unique instruction sets. That did not stop the creators of EDSAC, the first useful computer designed from scratch around the modern "stored program" paradigm, from building up a library of reusable machine code subroutines, or in 1951 from filling much of the world's first textbook on computer programming with code taken from this library.[a] However, it did mean the code would have to be reimplemented to work elsewhere.

That soon changed. By 1953 scientists and engineers at more than a dozen different sites were programming identical IBM machines. They began to exchange code and collaborate to develop packages jointly, a relationship formalized in 1955 with the creation of the SHARE user group and the development of new social and technical practices around its library of user-contributed programs. It appears to have been within SHARE that computer users began, by 1958, to refer to "packages." Its projects to jointly develop new program suites addressing high-priority areas were often given code names incorporating the words PAC, for example 9PAC for file maintenance and report generation. (A similar convention survived for decades in the world of mathematical software, as evidenced by the famous LINPACK benchmark for supercomputer ranking and many other PACKs for different specialized areas). Programs within the SHARE library followed specific social and technical conventions so they could be combined as needed by the user group's members. These included wiring control boards in a particular way, standardizing operational procedures, adopting common programming tools, and establishing shared coding conventions. So in this case the work of packaging meant transporting not just the code itself but also the tacit human knowledge, machine configurations, programming conventions, and operating practices it relied upon.

The word software gained favor around 1960, initially as a playful

[a] Wilkes, M., Wheeler, D., and Gill, S. *The Preparation of Programs for an Electronic Digital Computer.* Addison-Wesley, Reading, MA, 1951.

complement to hardware, already well known as a colloquial term for computer equipment.[b] It was sometimes used to describe everything else the computer manufacturer bundled with the computer hardware—perhaps including services, documentation, and other intangibles. In that sense it has its roots in packaging practice. Programs became software when they were packaged, and not everything in the package was code. For much of the 1960s the most commonly accepted definition of software therefore included only systems programs rather than applications, which were usually produced or heavily customized with the organizations where they were used. For example, a glossy 1962 pullout inserted into *Datamation*, a leading computer industry magazine, promoted Honeywell's expertise in software. This was defined as "the automatic programming aids that simplify the task of telling the computer 'hardware' to do its job." According to Honeywell the "three basic categories of software," were assembly systems, compilers, and operating systems.

When computer manufacturers eventually began to "unbundle" their software offerings, that is, charge separately for them, this was part of a broader trend toward packaging code as a good in its own right—literally as a "ware" for sale. Over the 1970s the mainframe packaged software industry developed from a curiosity to a significant market. This growth relied on a legal framework in which the rights of producers are protected, on the acceptance of banks and investors of software as a viable business, on the willingness of accountants to value packages as assets on a software company's balance sheet, on the willingness of customers to purchase something that may contain flaws they are unable to fix, and on the creation of a set of shared cultural understandings such as the difference between a bug fix (free) and an upgrade (usually paid for). None of these things were initially obvious, and each involved a process of collective learning and negotiation between producers and suppliers during which a variety of practices were experimented with to figure out a viable new way of packaging software.

The history of software is much more than just the history of code.

This is a column rather than a book, and there is insufficient space here to explore the many other chapters in the life of the software package such as the first high-quality manufacturer-supported packages (IBM's FORTRAN seems to have been a model), commercial software libraries, the shrink-wrapped model developed by the personal computer industry, online app stores, and subscription services. The shape and size of the package varied, and the bundle of code, documentation, services, support, and tacit knowledge assembled to make an enterprise product like SAP ERP into a salable commodity are clearly quite different from those packaged as Angry Birds. Still, airmail envelopes and modern intermodal shipping containers are both packaging technologies functioning on very different scales.

The point remains that the history of software is much more than just the history of code. Despite its apparent immateriality, software has always been tied to a platform and has always been physically embodied in one form or another. What turned programs into software was the work of packaging needed to transport them effectively from one group of users to another. To understand software we cannot just look at the bits. We need to examine the whole package. C

[b] Shapiro, F.R. Origin of the term software: Evidence from the JSTOR electronic archive. *IEEE Annals of the History of Computing 22*, 2 (2000) located an initial usage in 1958 by mathematician John W. Tukey to describe automatic programming aids.

Thomas Haigh (thaigh@computer.org) is an associate professor of information studies at the University of Wisconsin, Milwaukee, and chair of the SIGCIS group for historians of computing.

Further Reading

Akera, A.
Voluntarism and the fruits of collaboration. *Technology and Culture 42* (2001), 710–736. Examines the early history of the SHARE user group mentioned in this column.

Blanchette, J.-F.
A material history of bits. *Journal of the American Society for Information Science and Technology 62*, 6 (2011), 1042–1057. Begins with an interesting critique of the idea of information as an immaterial digital substance, as sometimes favored by media theorists, then surveys key modular design techniques used to produce the computer systems that support digital representations of information.

Campbell-Kelly, M.
The history of the history of software. *IEEE Annals of the History of Computing 29*, 4 (2007), 40–51. Explores the development of software history over its first few decades, identifying trends and key works.

Haigh, T.
Software in the 1960s as concept, service, and product. *IEEE Annals of the History of Computing 24*, 5 (2002), 5–13. Examines the origins of the software concept and reconstructs the challenges and opportunities early software packages provided to data processing users.

Hashagen, U., Keil-Slawik, R., and Norberg, A.L., Eds.
***Mapping the History of Computing: Software Issues.* Springer-Verlag, New York (2002). Papers and discussion from leading scholars chosen to reflect different approaches to software history. Includes several insightful papers from museum specialists on the challenges of collecting and displaying software.**

Lowood, H.E.
The hard work of software history. *RBM: A Journal of Rare Books, Manuscripts, and Cultural Heritage 2* (2001), 41–161. Overview of the challenges of software history, including those posed to curators.

Mahoney, M.S.
What makes the history of software hard. *IEEE Annals of the History of Computing 30*, 3 (2008), 8–18. A thoughtful attempt to position software history within an approach to what Mahoney called the "histories of computing(s)" structured around user communities and practices rather than hardware.

Williams, R. and Pollock, N.
***Software and Organizations: The Biography of the Enterprise-Wide System or How SAP Conquered the World.* Routledge, London, 2009. Applies perspectives from science studies to enterprise software packages, thus making a case for examining their "biographies" to understand them.**

DOI:10.1145/2500132

Peter J. Denning

The Profession of IT
The Other Side of Language

The conversation for action gives a framework for completing professional actions effectively.

Nearly four decades ago, Fernando Flores had the first ideas that led to his formulation of the conversation for action, which has become so influential in networked business and professional communities. The question of effective communication in organizations first came to him while he was a cabinet minister in the Chilean government.

Flores came to the U.S. and in 1980 completed a Ph.D. thesis at UC Berkeley on a new theory of communication for organizations. In the mid-1980s he wrote a series of unpublished essays on his theory, beginning with the conversation for action. Many of these essays have long circulated in an underground of his students and business clients. They have recently been published as a book edited by his daughter, Maria Letelier.[2] They are a treasure trove of timeless insights into professional issues we encounter today.

The core of Flores's theory is that action depends on commitments, and conversations are the sources of commitments. He argued that the elemental building block of coordination is the conversation for action, in which two parties commit to producing a valued outcome together. He viewed organizations as networks of commitments, enacted by recurring conversations for action. Effective managers tend conversations rather than direct and optimize the movements of workers. The network of commitments idea fit the Internet much better than previous management theories, and resonated with the knowledge-work idea promoted by Peter Drucker.

PHOTOGRAPH BY WAVEBREAK MEDIA LTD

In the early 1980s, Flores founded Action Technologies, a company to build a distributed laptop-to-laptop email service called The Coordinator, based on his theory. By 1990, nearly half a million copies of The Coordinator were in use in organizations around the world. Action Technologies extended the technology to a workflow management system for organizations. Their system mapped the network of commitments, managed assignments of people to roles in the network, and tracked the progress of work. They won several awards for pioneering the workflow industry.

Flores's communication theory energized a research community for computer-supported cooperative work (CSCW). It also attracted the ire of skeptics who regarded machines that tracked promises as a form of unwelcome workplace surveillance. His theory also energized a community of language-action software designers who focused on user practices around artifacts rather than artifacts themselves.[3] Their language action perspective has been more influential in designing apps for mobile devices and social networks than in traditional software engineering.

In the mid 1980s I was encountering management breakdowns in a research institute I was leading. My three dozen research scientists believed they were responsible for serendipitous discoveries in no particular time frame, while their funding sponsors believed they were responsible for deliverables with definite due dates. I treated the sponsor dissatisfaction as a communication problem and stepped up the flow of information about what our scientists were doing—brochures, pamphlets, tutorials, presentations, reports, and research papers. Unfortunately, this approach produced few results. Noting my quandary, a colleague recommended I contact Flores, which I did, and soon found myself reading his essays on conversations for action. His insights hit me like a lightning bolt. My management breakdowns were the result of scientists and sponsors having different commitments. I had been powerless to resolve them because I was oblivious to the language of commitments. After I began hosting scientists and sponsors in the missing conversations for action, most of the breakdowns disappeared.

Flores's essays gave me new insights into why other things important to me as a practicing professional did not work as well as I wanted. I learned how to influence moods and alter my timing when moods were bad. I learned that the practice of publicly sharing grounded performance assessments in teams makes it possible for team members to learn constructively from each other. I learned that the conversation for action and the network of commitments were not plans for machines to run organizations, but were tools for observation, helping to see how others in the network were responding and what their unspoken concerns were. Another essay (not in this collection) interpreted education as acquisition of capabilities for action at various skill levels, inspiring me to map out a program of reform for engineering education.[1]

Figure 1. Structure of a conversation for action.

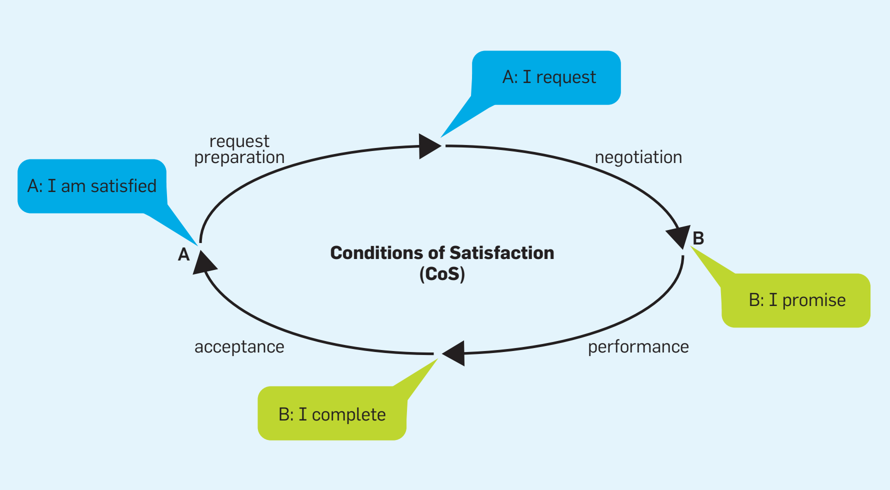
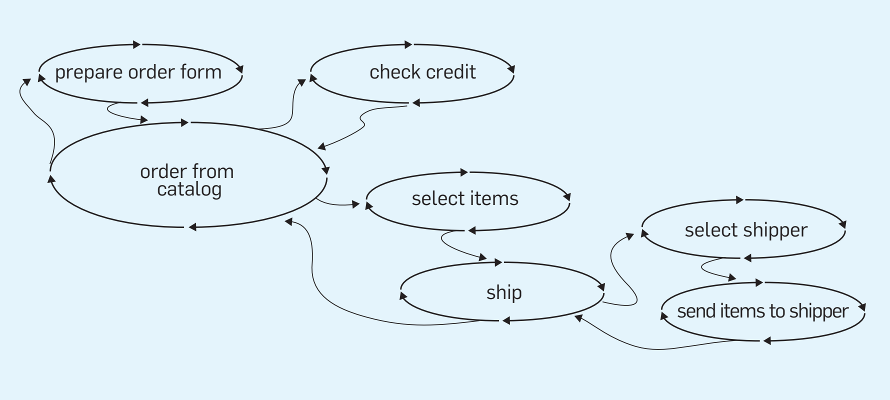


Figure 2. A network of commitments.



Two Sides of Language

Language has two sides. The familiar information side interprets language as a means to communicate messages containing information. Through these messages, we communicate facts, desires, intentions, and models of the world. This side places a great emphasis on facts, how we represent them with expressions in language, how we build models to explain related sets of facts, and how we communicate with each other about the truth of claims.

The less familiar commitment side interprets language as emotional, social, and historical. In our conversations with each other, we invent new realities, we negotiate, and we make history happen. We perform actions with requests, offers, promises, and declarations. We evaluate actions with assessments, and we make assertions about what is true.

The most common breakdowns in getting our work done come from four sources: misunderstandings, even when speakers believe they have been clear; miscoordinations, when different persons have different expectations of the intended deliverables; negative moods, which dispose people to be uncooperative; and distrust that builds with repeated misunderstandings, miscoordinations, and failed deliveries. None of these can be explained as failed information flows. They are all traceable to differences of listening and commitment. We depend on our skills with the commitment side to deal with them. Flores's essays are powerful exposés on this other side of language and the powers available to those who master it.

Anatomy of Conversation for Action

The original conversation for action (CFA) paper (and summary in the Winograd-Flores book[4]) made clear that the structure of coordination can be precisely described and accurately observed, and it can effectively guide actions. The CFA structure is something that anyone can master with practice. I will review it to remind us of its precision. Then I will discuss an important pitfall that arises paradoxically because the structure is so precise.

The CFA has a loop structure (see Figure 1) that sequences four commitments between two parties Alice (A) and Bob (B):

- Request or offer
- Promise or acceptance
- Declaration of completion, and
- Declaration of satisfaction

Alice is the "customer" and Bob is the "performer" in their loop.

The purpose of the loop is to cause a mutually agreed condition of satisfaction (COS) to become true. Alice proposes the condition with a request, and she and Bob may change it in negotiations before Bob accepts the request. Each segment of the loop represents a state of the conversation, and transitions between them are marked by observable "speech acts" of Alice and Bob in their conversation. After Alice declares satisfaction, the conversation is complete—at that point, the COS is fulfilled and the parties have no further commitments to each other. To complete the loop, the parties must coordinate smoothly during its performance.

The CFA diagram and structure are tools for observation. All the commitments, including the COS, are plainly visible to the parties and to observers of the conversation. Both parties become accountable for their own commitments, and each can assist if necessary to help the other person fulfill theirs.

Organizations set up recurrent CFAs between people filling various roles. We can draw maps like Figure 2 that show the organization as a network of commitments, in which subsidiary requests are linked to the segments of other requests that initiate them. The network is activated every time a customer initiates a request to the organization.

There are numerous ways to break a loop. Sometimes one of the four commitments is missing. For example, Alice might have thought dropping a hint was sufficient but Bob did not hear the hint as a request; or Bob might insincerely make a promise but has no intention of carrying it out. Sometimes the COS is ambiguous or understood differently by the two parties. Sometimes one of the two parties is missing, for example the customer is missing when a producer generates a result no one has asked for, or a producer is missing in an office where no one tends the inbox. Sometimes one of the parties is in an uncooperative or otherwise bad mood. Sometimes one party does not trust the other, perhaps because of a poor track record. The number of ways to break a loop is truly amazing. This is why it takes a skill to automatically recognize the structure, spot any missing elements, and take immediate corrective action. It is a way of observing and reacting to how the parties are *listening* to each other.

A Paradoxical Pitfall

A paradoxical pitfall arises because the CFA's precision invites mechanization. The Winograd-Flores book (page 65) unfolds the loop of Figure 1 into a nine-state machine diagram that includes additional states corresponding to other possible moves—for example, the four common responses to a request, namely accept, decline, defer, and negotiate. The state machine was embedded within The Coordinator software and was its tracking engine. The pitfall is that many people do not distinguish between the CFA as a machine and the CFA as a tool for observing and tracking commitments. The machine can detect speech acts, record state transitions, and measure the times spent in each state. However, the machine cannot make commitments. Only the human participants can. It is a mistake to equate the CFA with a machine.

The CFA was intended from the beginning as a guide for observing commitments and listening for concerns. With this guide, a skilled team leader could navigate around bad moods, distrust, and environmental distractions. The skill of performing in a CFA this way is not difficult to learn once you understand the structure and its purpose.

Other Conversation Types

Conversations for action do not happen in isolation. They are almost always preceded by one or both of

- Conversations for possibilities
- Conversations for context.

A conversation for possibilities identifies possible actions, without committing to any one. It is done in a mood of speculation. For example, we could invent possible ways to solve a problem or respond to an opportunity. Some of the possibilities can become action when they become requests or offers in a CFA. A conversation for context frames the purpose and meaning of a team or project so that conversations for possibilities and for action can meaningfully follow.

If as team leader you leave either of these out, you are likely to have coordination problems because your team does not understand the purpose or cannot make sense of the proposed actions.

Conclusion

The conversation for action interprets basic human coordination as a loop cycle of four commitments progressing toward a mutually agreed goal. It creates a precise framework for observing commitments and allowing the parties to adjust should a conversation veer off track. This conversation exists in the commitment side of language rather than the information side.

It is remarkable this simple linguistic structure for coordination is universal. It is observable in every language.

Professionals who master the skill of completing their loops will reap benefits including increased productivity because of reduction of wasted steps, delivery of more value to customers, fewer coordination breakdowns with teams and clients, and significantly improved reputation for integrity and reliability.

Now that the collection of seminal essays on these topics is available, you have the opportunity to use them to help you reflect on the breakdowns you are experiencing with your customers, clients, and teams. Maybe a lightning bolt of insight will strike you, too. C

References

1. Denning, P. Educating a new engineer. *Commun. ACM 35*, 12 (Dec. 1992), 83–97.
2. Flores, F. *Conversations for Action and Other Collected Essays.* 2013; http://conversationsforaction.com.
3. Weigand, H. Two decades of the language action perspective. *Commun. ACM 49*, 5 (May 2006), 44–46.
4. Winograd, T. and Flores, F. *Understanding Computers and Cognition: A New Foundation for Design.* Addison-Wesley, 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, is Editor of ACM *Ubiquity*, and is a past president of ACM. The author's views expressed here are not necessarily those of his employer or the U.S. federal government.

DOI:10.1145/2500138

Željko Obrenović

Viewpoint
Research and Practice: The Curious Case of 'Small' Researchers-Practitioners

Seeking a more efficient combination of the best elements of the research and practice communities in small organizations.

IMAGE FROM SHUTTERSTOCK.COM

In recent years we have witnessed more attempts at bridging the practice-research gap in computer science.[5] ACM and IEEE Computer Society, for example, seem to be increasingly more open to "the voice of practice."[1] *Communications* now includes the Practice section. *ACM Queue* promotes itself as an online magazine for practicing software engineers, "written by engineers for engineers." *ACM interactions* describes its goal as to "lay between practice and research…making…research accessible to practitioners and making practitioners voices heard by researchers." *IEEE Software* defines its mission as to build the community of leading software practitioners. The International Conference on Software Engineering (ICSE) has the Software Engineering in Practice track, and, similarly, the ACM SIGCHI conference accepts case studies intended to "specifically reach out to the practitioner communities."

While the research-practice symbiosis seems to be flourishing, doing research as a practitioner is still not easy. It is even more difficult if research is not conducted in big companies or in collaboration with universities. Many of us are researchers-practitioners working in relatively small companies. By researchers-practitioners, I mean practitioners with clear practical tasks in their job, but who have background or skills of a researcher, obtained, for example by getting a Ph.D. or working as a postdoctoral researcher. And I call these practitioners "small" because they usually do research independently or in small teams, and cannot associate with their work research reputation and influence of their institution or companies. In small companies, we may not have a number of things that researchers in universities or big companies take for granted,[8] such as an explicit research department, budget for conferences, freedom, or even the job description and status of a researcher. But we can bring to practice the benefits of research approach, rigor, and discipline. And we can make accessible to the research community valuable insights and unique lessons from practice.

Contributions of small practitioners-researchers, however, are not always recognized and valued. Furthermore, they face a number of challenges and obstacles that researchers in big companies or in universities do not. In this Viewpoint, I want to call attention to the value of doing "small" research in small companies, and point out some of the main obstacles that such work faces.

While the research-practice symbiosis seems to be flourishing, doing research as a practitioner is still not easy.

Recognizing the Value of "Small" Research in "Small" Companies

Researchers are, in general, good in critical thinking, analysis, and dissemination of their findings. These skills, combined with practical work, can bring to their companies and the research community several benefits. Here, I discuss two characteristics of research work I find particularly relevant for small researchers: generalization and publishing.

Generalization. Normally, the goal of practice is to create a successful product, and lessons learned in this activity are restricted to the particular solution and the people involved in it. To be acceptable as research contributions, however, these lessons need to be generalized, applicable beyond original context, and useful to others (see Obrenović[6] for more details about such generalized knowledge).

Generalization is not only an abstract academic goal, but it can be valuable for practice. In my previous position I worked in a relatively small company in a department called "best practices." The primary goal of our department (one engineer, one architect, and one researcher) was to collect, generalize, and share best software development practices related to our software products. Being a relatively small company meant we did not have the luxury to repeat errors, and our department was built with the aim of maximally leveraging the lessons learned in our projects. Our task was not to simply collect these lessons, but to generalize them and make them usable and understandable to the broader audience, within and outside our company. Applying research approaches, such as using analytic generalizations, evaluations, and connecting our findings to existing work, helps significantly. Good generalizations can also help avoiding low-level technical jargon. Consequently, our work has been valuable not only for our architects and developers but also to our sales team, who were able to use some of our analyses as arguments in discussion with demanding and critical clients. In contrast to research in big companies, small researchers are closer to the "battlefield" and can more directly contribute to the company's success.

For the research community, generalizations of practical solutions on a broader scale and across multiple projects are particularly valuable. For example, we recently published an article about security patterns of integrating authentication and personalization, generalizing security implementations in several of our projects.[7] I also see a potential value of having more smaller companies sharing their "best practices," combined with additional effort of the academic community to connect and further generalize these practices. I had an opportunity to witness the value of this approach firsthand, when I was one of the guest editors for the special issue of *ACM Multimedia Systems Journal on Canonical Processes of Media Production*.[4] This special issue was not only a collection of articles, but it presented a model of media production that was based on generalization of 10 companion articles describing different media production domains (each of which presented some specific media production system or project). Contributions included several media production companies, artists, and academic researchers. The resulting model significantly benefited from interaction and generalization of issues from our industrial contributors. Our industrial contributors also benefited from connecting their work to other solutions, as they were able to get new ideas and see that their issues are shared by others and that they can learn from each other's experiences. It would be interesting to see more such attempts in other domains, where small researchers would present their initial generalizations of their domains, and a broader research community would connect these generalizations to other industrial and academic work.

Publishing Results. Publishing findings from practice has obvious benefits for the research community as it enables it to obtain deeper insights about relevant practical issues, and gets more realistic overview of the state of the practice.[3] Stolterman, for example, argued that many research projects about theoretical approaches, methods, tools, and techniques for supporting interaction designers in their practice failed because they were not guided by a sufficient understanding of the nature of practice.[9]

Publishing can also significantly help a small company. One of the most important values of publishing in peer-reviewed venues is receiving knowledgeable and valuable criticism. By publishing your results, you also have to make the reasoning behind your generalized claims explicit, public, and open to critical reflection and discussion, which enables receiving feedback of experts and colleagues from different communities. Publishing results can also have positive influence on company's promotion and hiring of new employees. Small companies normally cannot sponsor huge events, but presenting a paper at a conference, combined with promotion of this event by the company, may give a company a fair share of visibility and promotion for much smaller price. Small companies may also have more difficulties attracting high-quality employees, and I received unexpected encouragement to actively participate in conferences from the Human Resources (HR) department. The HR department elaborated that such activities can help the company to demonstrate the quality of its work and its people, both to potential new clients and employees.

Main Obstacles

Doing research outside universities or big companies, even when conducted with rigor and discipline, comes with a number of challenges. Finding time and resources for research in small companies is always challenging. And practice does not always recognize the value of research contributions. It may require significant time and effort to convince relevant people in your company of the potential value of doing research. Practice also needs to understand that it is not enough to simply relabel "development" as "research," and that research cannot be done properly without individuals who are disciplined and objective enough to conduct it with scientific rigor.

Less obviously, and contrary to the recent trend of openness for "the voice of practice," a small researcher-practitioner may face even bigger barriers from the research community. Research work is difficult and incomplete if a researcher is not a part of a community of researchers. However, for researchers-practitioners coming from smaller or less-known companies, it may be difficult to become a part of such a community. First, it may be difficult to find a venue open for contributions of the practitioners. Reviewers also may be biased toward more academic contributions and methods. When you try to submit some of your work for publication in places that seem to promote strong practice orientation, you may find many of them are not open for your contributions. For example, the *Communications* Practice section publishes articles "by invitation only." Similarly, *ACM Queue* reviews articles only from authors who have been "specifically invited to submit manuscripts." This makes it practically impossible for people outside a relatively small group of elite practitioners to even try to contribute regardless of the quality of their contribution.

Educational institutions also need to think about how to educate researchers-practitioners.

Another barrier from the academic side comes from stereotypes about the research process. When working for my previous company, I tried to join the ResearchGate, as several of my papers have been uploaded there by other co-authors. However, when trying to register with my company email address, I received the following email message: "We've reviewed your request and regret to inform you that we cannot approve your ResearchGate account at present. As ResearchGate is a network intended for scientific and academic exchange, we ask that you sign up with an email address affiliated with your institution (e.g., university, organization, or company) or provide us with details of your independent research (e.g., research discipline and current project)."

My email address was affiliated with my institution (a company), in an obvious way (my name at my company domain). However, it seems a company is considered a research organization only if it is a well-known institution, and with a separate research department (for example, Google Labs, Microsoft Research, Yahoo Research, Philips Research...). This anecdote points to a problem of researchers from smaller companies who may be discriminated in their attempts to become part of the research community, and may have difficulties passing the threshold of being considered worthy of belonging to the research community. Also the notion of a research project seems to be closer to the academic environment where researchers work for several years on the same project. In practice, there may be a long-term research thread, but research contributions do not necessarily belong to an explicit project.

Conclusion

There is a potential value for both, practice and research, if we have more active "small" researchers-practitioners. With declining numbers of research positions in academia[2] we have increasing numbers of research-capable people entering small companies. Practice is rich and still hugely unexplored area, and researchers-practitioners may be in unique positions to witness or make important discoveries in many areas of computing. However, there are a number of barriers and challenges that "small" practitioners-researchers face. Practice needs to become more aware about the value of applying research rigor and discipline, and the research community must be more open for attempts of "small" researcher-practitioners to join them as equals. Educational institutions also need to think about how to educate researchers-practitioners, rather than researchers or practitioners. It also requires more continued efforts of small researchers-practitioners to do high-quality research, contribute to the research community, and call attention to their problems.

References

1. Bourne, S. and Cantrill, B. Communications and the practitioner. *Commun. ACM 52*, 8 (Aug. 2009), 5.
2. Briand, L. Embracing the engineering side of software engineering. *IEEE Software 29*, 4 (July–Aug. 2012), 96–96.
3. Glass, R.L. One man's quest for the state of software engineering's practice. *Commun. ACM 50*, 5 (May 2007), 21–23.
4. Hardman L., Obrenovic, Z., and Nack, F., guest eds. Special issue of *ACM Multimedia Systems Journal* on canonical processes of media production *14*, 6 (Dec. 2008), 327–433.
5. Norman, D.A. The research-practice gap: The need for translational developers. interactions 17, 4 (July 2010), 9–12.
6. Obrenović, Ž. Design-based research: What we learn when we engage in design of interactive systems. *interactions 18*, 5 (Sept. 2011), 56–59.
7. Obrenović, Ž. and den Haak, B. Integrating end-user customization and authentication: The identity crisis. *IEEE Security and Privacy 10*, 5 (Sept./Oct. 2012), 82–85.
8. Spector A., Norvig, P., and Petrov, S. Google's hybrid approach to research. *Commun. ACM 55*, 7 (July 2012), 34–37.
9. Stolterman E. The nature of design practice and implications for interaction design research. *International Journal of Design (IJDesign) 2*, 1 (2008).

Željko Obrenović (obren@acm.org) is a technical consultant with the Software Improvement Group in Amsterdam, The Netherlands.

DOI:10.1145/2500133

Lawrence Snyder

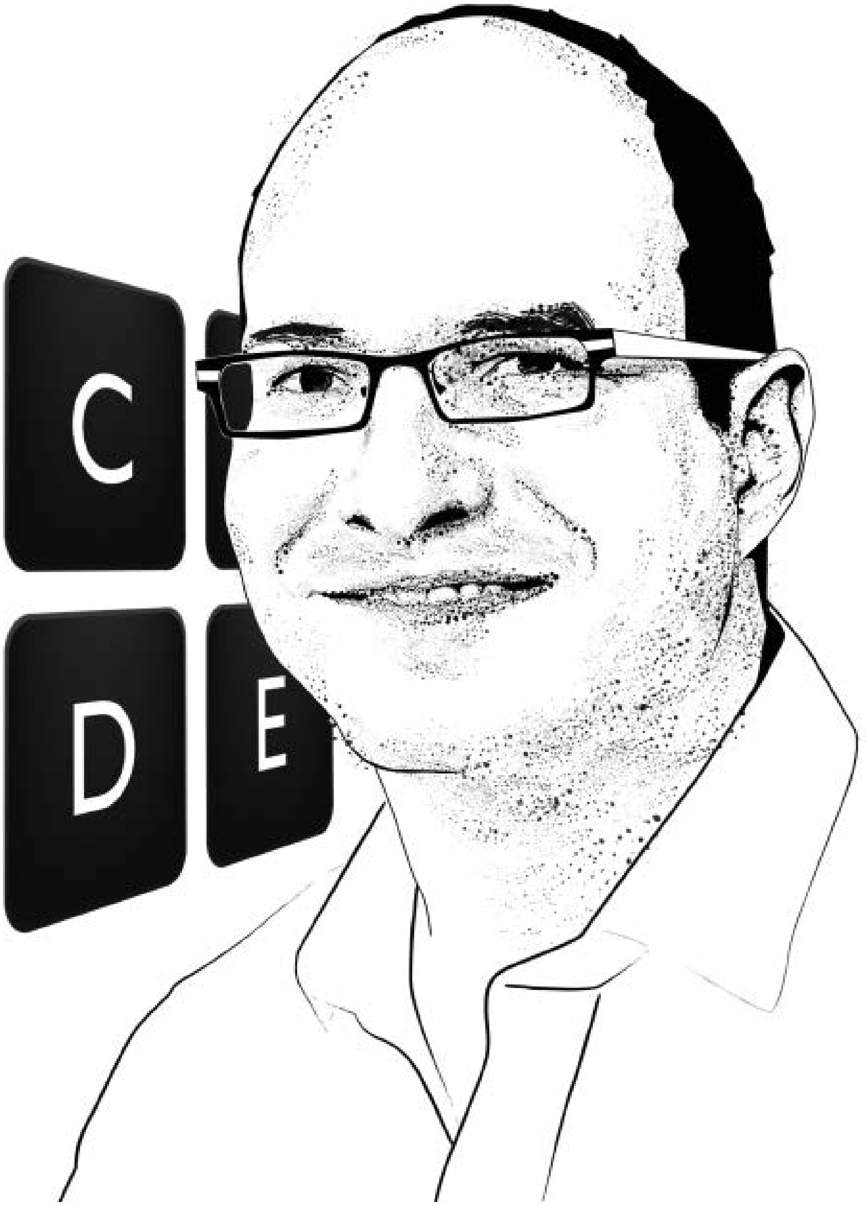
Interview

An Interview with Hadi Partovi

The Code.org founder discusses his first program, inspirations, and "seizing the day."

Ten days after the Code.org video "What most schools don't teach" went viral, University of Washington Emeritus Professor Lawrence Snyder joined Code.org's Hadi Partovi for espresso at Belle Pastries. "It's my office," Partovi joked of the confectionary shop that is near his Bellevue, WA, home, and where he had just wrapped up another hour-long meeting.

Hadi Partovi, the founder and CEO of Code.org, is the creator of videos[a] to inspire people to learn programming: Tech industry luminaries recall their first programs; NBA All-Star Chris Bosh says it's fun, and something anyone can learn; will.i.am of the Black Eyed Peas is incredulous that in 2013 most people cannot read and write code. All these programmers make the case that programming is fun and empowering. Drew Houston, the creator of Dropbox, says, "It's the closest thing we have to a superpower" and will.i.am calls programmers, "Rock Stars." Directed by Lesley Chilcott (*An Inconvenient Truth*), this crisp, intense video inspires action. On the Code.org site endorsers ranging from former U.S. president Bill Clinton to the performer Snoop Dogg call for expanded opportunities for people to learn programming and computer science.[b]



Code.org founder and CEO Hadi Partovi.

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

[a] See https://www.youtube.com/watch?v=nKIu9yen5nc.

[b] See http://www.code.org.

Partovi's intense passion for the Code.org mission derives in part from the success he—and his identical twin brother Ali—achieved from knowing programming and understanding computer science. The Partovis emigrated from Iran to New York in 1984. As fresh Harvard graduates in 1994 the brothers started LinkExchange, which they sold to Microsoft four years later for $225 million. They co-founded other companies including iLike, which they have also sold, and have become startup advisors and angel investors; they are pillars of the tech community. Earlier this year, the brothers answered a wide range of questions on Reddit.com's Ask Me Anything,[c] including their best "twins pranks."

As the interview was beginning, Snyder remarked, "I know you are an identical twin…am I interviewing the real Hadi Partovi?" With a smile and a chuckle, Partovi replied, "I am the real Hadi, yes."

Your video begins with Bill Gates, Jack Dorsey, Mark Zuckerberg, and others saying when they started programming. When did you start?

My first program was for a Texas Instruments calculator that my dad got for me when I was seven or eight years old. It was basically programming machine codes. My first real computer was a Commodore 64 that I got when I just turned 10. I started learning Basic on that.

That was in Iran?

Yes, we were living in Iran. Obviously, there were no CS classes there. Our family came to New York in 1984, so I grew up in New York. Until late high school I didn't have any computer classes. I was just learning programming on my own from a book. Then I went to Harvard.

And you taught CS there?

At most universities there is a shortage of CS teachers like everywhere else, and so at Harvard undergrads in the CS Department were regularly the section leaders. I started doing that the second semester of my freshmen year because I came in having already taken AP Computer Science; I think the AB test that was available back then. So, I'd passed out of the first year of CS, and started teaching. For all four years that was my main job during the school year.

Was that the start of your interest in education?

I wouldn't say it was the start, but it has certainly been the most common theme. I actually have education in my blood. My dad co-founded the main tech university in Iran, effectively the MIT of Iran, called Sharif University. My dad and uncle were chairs of the Physics Department. So, I've always had education in my background.

Who inspires you to do this kind of work with Code.org?

There are two inspirational things out there that actually gave me the kick in the pants to do it. Well, actually multiple.

In terms of someone who currently inspires me, I would say Bono. He doesn't do anything in education. But he's such a strong voice for saying, "Do something valuable for your life. Don't wait for other people to do something." Considering that a year ago I was unemployed, and basically am still unemployed and enjoying semi-retirement, hearing someone like Bono, who's obviously better off than me, and he's dedicating his life to solving worldwide poverty. It's not an easy problem to solve, but he's so clearly dedicated to it. He is going to see that problem through. That's been inspirational to me. "What is a problem that I could see through and solve?"

The point at which I said, "I have to do this," was the day that Steve Jobs died, because I had always envisioned Steve Jobs to play a role in the video. He had been vocal about computer science education. The opening line of the video is there because of that. The day he died, like everybody else, I had this pang of grief of the loss of a great man. But, I also thought, "He's 12 years older than me. What am I going to do in the next 12 years compared to how much he's done?"

I had this idea for the video for, like, three years and wasn't doing anything about it, because it is very easy to not do things. And that gave me the motivation. You know, I'm sitting here not doing anything, and people are out there dying. I need to go out and do stuff.

You say you have been thinking about the video for three years. Say a little more about what's happened in that time.

In March 2012 I was at a tech conference sitting around the fireplace with Jack Dorsey and Drew Houston, the founders of Twitter and Dropbox. This was just a few months after Steve Jobs' death, and I had decided I wanted to do the video, but I didn't know where to start. And I told them, "Hey, I have this idea for doing a video, what do you think? Would you guys be interested?" And both immediately said, "Yes, you should totally do that. It's a brilliant idea! We're in." It was such a strong positive "yes"…I never thought it would be so easy to recruit somebody. Obviously, I went next to see if I could get Bill Gates. That

Hadi Partovi: "My effort is to turbocharge computer science education."

IMAGE COURTESY OF CODE.ORG

[c] See http://www.reddit.com/r/IAmA/comments/19eqzm/iam_hadi_partovi_cofounder_of_codeorg_here_with/

wasn't "Well, let's think about it. No… et cetera." I sent him an email message—I had worked at Microsoft, so he knew who I was—and I had asked another guy at the same conference who is close to Bill, "Could you follow up on my email to put in a good word?" And I got a "yes" in days! So, if you have Bill Gates, things get easier.

Then, I asked Mark Zuckerberg. That was actually one of the hardest to get, but not because he was against the idea. I also know 'Zuck' from having been an early advisor to Facebook, so he knew who I was and I think interested in the idea, but this was a month and a half before the IPO. He was incredibly busy with the IPO. His assistant kept saying, "He can't make time for anything at all." And I said, "Can he make time for something one year from now?" She's like, "Yes, but this isn't a year from now." And I said, "I'll do this whenever he's ready, just ask him if he can do it a year from now." And he said, "yes," and it didn't take a year.

After having Bill and 'Zuck' say yes, it shifted more to letting people in rather than asking them in. After that, almost anyone you asked, said "yes." It was downhill from there in terms of getting people to participate.

There seem to be some notable omissions.

We wanted to make sure that everybody we got was a programmer.

How did you know Chris Bosh programmed?

Chris was all Leslie Chilcott's work, our director. I didn't even know he knew computer programming. She did the research. She wanted an athlete that teens really look up to. It's not only that he's an athlete, and he's African American, but also he's so articulate and so down-to-earth, and—you know—humble. He really comes across well. I give her all the credit for that.

My students reacted favorably to him.

I have shown the video at multiple high schools. You know, high school kids are kind of rowdy? They weren't in my high school. But the high schools I visited—average high schools with very diverse populations—and the kids are just sitting there, some of them are chatting among themselves. But when Chris Bosh comes up, people would jump up and down, and high five one another. Really excited, which is just great.

» ACM, Code.org Join Forces

ACM has partnered with Code.org in carrying out a critical new initiative of scaling K–12 computer science education to schools around the U.S. In addition, Computing in the Core, a non-partisan advocacy coalition, founded in 2010 by ACM with support from several other partners, including the Computer Science Teachers Association, Microsoft, Google, and the National Center for Women and Information Technology is merging with Code.org.

It seemed like your video knocked down a lot of stereotypes. There were no empty pizza boxes or Diet Coke cans sitting around. No one is sitting in a darkened room staring at a screen. Purposeful?

Yes, that's very intentional. Obviously, I know from personal experience that there is a lot of pizza and soda that goes into computer programming, but the research has shown that the number-one reason kids say they decide not to go into it is they don't want to work in a dark basement their whole life. It's not just about the jobs and the glitz. We had to show the workplace. There's a lot of sunlight and really cool perks that aren't just about working in a dark basement all your life. You know, most university computer labs are in the basement or other dark areas … just what you would envision.

How do you describe Code.org?

Code.org is an effort to bring computer science to every school and every student in America … actually, I would change it from "effort"—I would call it a movement. My goal really in making the video is to create a movement. I still am not completely sure what I do next with it versus what other people are going to be doing anyway. I am certain we have kick-started hundreds of efforts that either started because of what we have done, or just got a whole lot more fuel in the gas tank because of what we did. We may never hear from them. I just got an email message from a teacher in South Africa who said that they are going to bring computer science into

Calendar of Events

September 15–18
20th European MPI Users' Group Meeting,
Madrid, Spain,
Contact: Jack Dongarra
Email: dongarra@cs.utk.edu

September 24–28
Seventh ACM Conference on Recommender Systems,
Hong Kong,
Sponsored: SIGCHI,
Contact: Qiang Yang
Email: qyang@cse.ust.hk

September 25–27
ACM SIGPLAN International Conference on Functional Programming,
Boston, MA,
Sponsored: SIGPLAN,
Contact: Greg Morrisett
Email: greg@eecs.harvard.edu

September 29–October 4
ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems,
Miami, FL,
Sponsored: SIGSOFT,
Contact: Antonio Vallecillo
Email: av@lcc.uma.ed

September 29–October 4
Ninth Embedded System Week,
Montreal, Canada,
Sponsored: SIGMICRO,
Contact: Luca P. Carloni
Email: luca@cs.columbia.edu

September 30–October 1
The 31st ACM International Conference on Design of Communication,
Greenville, NC,
Sponsored: SIGDOC,
Contact: Michael J. Albers
Email: mephitis.skunk@gmail.com

October 1–3
ACM Symposium on Cloud Computing,
Santa Clara, CA,
Sponsored: SIGMOD, SIGOPS,
Contact: Guy M. Lohman
Email: lohman@almaden.ibm.com

October 1–4
Research in Adaptive and Convergent Systems,
Montreal, QC,
Sponsored: SIGAPP,
Contact: Jiman Hong
Email: jiman@ssu.ac.kr

their school and all of the neighboring schools. They got motivated and they're doing that. We're probably not going to talk to that person again.

People have asked what would you say to President Obama about computer science if you had his attention for 30 minutes?

At the AMA[d] I listed six points as reasons to support CS in schools. But, the most important thing for me to get the president's attention on would be this subtle difference between computer science and STEM. And that subtle difference is, I think, one of the greatest weaknesses of the computer science education effort. Whether in the tech industry or the CS Ed community everybody knows what computer science is. When you go outside, most people have never studied it, and so it is this vague thing they don't understand and they put it in the STEM bucket...and things get lost in multiple ways. If you then say, "things aren't going well," Americans are like, "Oh, yeah, we're bad at math and science." What's missing—they don't even realize—we *teach* math and science at every school to every student. We *try*. And now we're bad at it. Whereas, in CS we don't even teach it in 90% of the schools. We're not even trying. There's a big difference here.

And the other thing that gets lost is that 60% of all STEM jobs are in the computing arena, and only 2% of STEM education is. And most states don't even classify computer science as part of STEM. All those things get lost. So, explaining that subtle difference to him would be the most important thing.

On the issue of training teachers to teach all of these eager students, the CS 10K Project is targeted at developing those teachers. Do you have thoughts on that task?

Well, 10,000 teachers is a lot more than we have. It's not sufficient to get it into every school in the country. Personally, I think one of the things that helps us is, we are talking about computer science, which is something so many people like myself learned on their own with a computer. As a result, part of what can help is using technology, whether in the training of the teachers or the training of the students. So I constantly think about what we can do to make the computer not just the tool or the black slate, but bring it into the actual educational activity itself. And I don't think of this as a world where there is no teacher. I think there is some set of kids who can learn without a teacher, I just think the computer can make the teacher's job easier.

One thing that gives me a lot of hope is, I did a survey with the College Board revealing the fact that 75% of all AP STEM teachers have taken at least one computer science class in college. That's a huge percentage. Retraining those teachers should be much easier than finding tens of thousands of computer programmers who want to take a job as a teacher. These are people who are teachers; they have taken all of the course work and trained to manage a classroom. And, they have taken a semester of computer science. So, it should be an easier job than you would envision to get them excited and retrained, especially if you offered them a system where the computer is helping with the teaching.

If you look at it from the tech industry's standpoint or the computer science 'yield' standpoint, there is an interest in getting more computer scientists and more people to become software programmers as their jobs. But if you look at it from the standpoint of fifth-graders—just getting kids more excited about what they are learning—computer programming should be the single most fun class there is. You go to math class and you do addition and multiplication tables, and memorizing capitals of states. And in computer class you should be able to learn about gravity by shooting a cannonball up a hill, and seeing how it flies ... which should be much more fun than almost anything you do. It is an opportunity to make existing classes more engaging, and students—especially girls—feel more empowered and creative.

For Code.org, what do you see as the next goal. Where are you headed now?

Right now there are two things going on. One is, I am just trying to rest. I haven't had very much sleep for the

[d] See http://www.reddit.com/r/IAmA/comments/19eqzm/iam_hadi_partovi_cofounder_of_codeorg_here_with/

"Whether you want to make a lot of money or just change the world, computer programming is an incredibly empowering skill to learn."

last three months. We are a very, very lean organization. I am the only full-time member of the staff right now. So, just getting some sleep is my most important goal. The other thing going on now—aside from the hundreds of efforts that have just been kick-started…or thousands, perhaps, without us being involved—our website basically reaches out to teachers and educators, saying, if you want this at your school, let us know. And we have also reached out to engineers, saying, if you want to volunteer to help, let us know. One of our biggest challenges is—get this—we have 10,000 schools that want our help bringing CS to them. And we have 20,000 software engineers who say, we want to help. We have no capacity to deal with that yet. But, clearly, what's next for us is figuring out how…you know, it's not a requirement that we use those volunteers, but if we don't bring something to those 10,000 schools, it's a lost opportunity.

The CS Ed community seems to be behind you 100%…

Good.

The only sort of complaint I've heard is that you're talking about "code," and we want "computer science." You've heard that complaint?

It's not the only complaint I've heard. I have said this to many CS Ed people, "If 'computer science' were a four-letter word and 'code' was a 16-letter word, it would be called computer science dot org." The reason it is called Code.org is that it's a short domain name, it's inspirational and that's that. For me, whatever it takes to get people just getting started trying something, there's this whole field available to them, and the field isn't called "coding," the field is "computer science." But, if we can get kids interested in coding, if we can get every kid touching it a little bit, and then a small percentage decides to study it for their entire lives, that'll be great.

I am most interested in the top of the funnel, of touching the most kids with something, and I am sure that a lot of people will get attracted. The real issue is there are studies that show that some populations, especially women, are turned off by coding, and it's not the best way to get them interested. What makes much more sense to women is hearing about how computers can be used to change the world, to help people, and so on. And the name 'code' doesn't really suggest that. That said, I am very well aware of this issue. It's not by accident that my own quote in the video is, "Whether you want to make a lot of money or just change the world, computer programming is an incredibly empowering skill to learn." Then the woman who comes after me says she wished someone had told her early on that software is about humanity and helping people through software. Those messages clearly resonate, and we need to get them out. And the organization is called Code.org because it is a four-letter word.

That was Vanessa Hurst?

Yes, she actually runs an effort called Girl Develop It. One of the exciting things about them is that they have workshops in six or seven cities, and just the day we launched, they had 15 new cities say, "We want your workshops." And how did they get found? I have no idea. Did people look at the website footer where we have Meet The Cast? Did they find her there? Or did they just naturally look around for organizations that do this? I don't know.

Finally, I wonder what you would say if you could speak to all of the members of ACM?

I would say two things. One would be, to any member of the ACM, especially if you're one working on how to grow computer science education, and you have 20 years perhaps working on this problem, this next 12 months is the "seize the day" moment. This thing can get really big, fast, which is exciting. And I am sure lots of people have the sense of, "Who are these newcomers; I've been doing this for a long time." My effort isn't to co-opt anything anybody's doing. Rather, it's to turbocharge it. And I think there is a lot I have to learn from the efforts that have been done in this space.

The second thing I would say: I think there is an opportunity to take this field from something that tens of thousands of people study, to something that millions of people study a year. That's a 100x growth. And so we need to think about what are things we can do differently than in the past to get 100x growth. Just repeating everything the way we have done it times 100 is probably not going to work. And that is the question I ask. It is easy to see that we can double what we've got just by having twice as many people do the same thing. But 100x is much harder.

It does seem to be the "seize the day" moment.

One reason I would say this is the "seize the day" moment is that with our launch, we had quotes on our page from people like the President of the Fred Hutchinson Cancer Research Institute, astronauts, Dr. Oz tweeting about it, Snoop Dogg the rapper, politicians from both sides of the aisle, Enrique Iglesias tweeted about it, Arianna Huffington tweeted about it, the White House posted about it. The number of people not in computer science—I am mentioning people on the fringes, have said this is an interesting thing for them. Linkin Park, a band that has 15 million Facebook followers, has numerous times posted about it. People are coming out of the woodwork to make something happen. And that is definitely not happened before in this field, which is why we need to jump on this moment.

Lawrence Snyder (snyder@cs.washington.edu) is Emeritus Professor in the Department of Computer Science and Engineering at the University of Washington.

DOI:10.1145/2500468.2500476

Article development led by acmqueue
queue.acm.org

Design requirements of nonblocking systems.

BY MAGED M. MICHAEL

The Balancing Act of Choosing Nonblocking Features

WHAT IS NONBLOCKING progress? Consider the simple example of incrementing a counter C shared among multiple threads. One way to do so is by protecting the steps of incrementing C by a mutual exclusion lock L (such as, `acquire(L);old := C;C := old+1; release(L);`). If a thread P is holding L, then a different thread Q must wait for P to release L before Q can proceed to operate on C. That is, Q is blocked by P.

Now consider an implementation of the increment operation using the Compare-and-Swap (CAS) atomic primitive. CAS atomically reads a shared location and compares the read value with an expected value. If equal, it writes a new value to the location and returns an indicator of equality with the expected value. In the following implementation &C is the address of C:

```
do {old := C; } until CAS(&C, old, old+1);
```

This implementation is nonblocking because no thread can be blocked by the inaction of other threads (caused by, for example, preemption, a page fault, or even termination), regardless of where the other threads stop.

There are three established levels of nonblocking progress: obstruction-free, lock-free, and wait-free. The weakest is obstruction freedom. An operation is *obstruction-free* if it is guaranteed to complete in a finite number of steps when running alone.[9] An obstruction-free operation does not need to wait for actions by other threads, regardless of where they have stopped. An obstruction-free operation, however, may starve or end up in a livelock scenario with one or more concurrent operations where the actions of the threads prevent them all from making progress.

Lock-free progress combines obstruction-free progress with livelock freedom. That is, in a system with only lock-free operations, whenever an operation takes a finite number of steps, some operation (maybe a different one) must have completed during that period.

Finally, *wait-free* progress[7] combines lock-free progress with starvation-freedom. That is, a wait-free operation is guaranteed to complete in a finite number of its own steps, regardless of the actions or inaction of other operations.

It is worth noting these levels of progress guarantees require that primitive memory-access steps have at least the same level of progress guarantee at the hardware-arbitration and cache-coherence levels. For example, a cache-coherence protocol that may cause some primitive memory accesses to be retried indefinitely cannot support a wait-free algorithm that uses such primitives.

Uses of Nonblocking Operations

Nonblocking operations are often used for systems or interthread inter-

IMAGE BY NORMA CORNES

actions where having threads wait for actions by other threads make the system vulnerable to deadlock, livelock, and/or prolonged delays. Examples of such uses are:

- *Async signal safety.* This allows signal handlers of asynchronous signals to share data or services with the interrupted thread, with a guarantee that the signal handler never needs to wait for actions by the interrupted thread—which, as a result of being interrupted, cannot run until the signal handler completes. Nonblocking operations used in particular by the signal handlers can guarantee the absence of such waiting and provide async signal safety.
- *Kill-safe systems.* Such systems guarantee availability even if processes may be terminated at arbitrary points. This requirement arises in server systems, where the ability to terminate processes representing client requests allows high server throughput. Such systems must guarantee availability even when processes may be terminated at arbitrary points while operating on shared structures or services. When using nonblocking operations in such a system, the remaining processes never have to wait for action—that will never happen—by a terminated process.
- *Soft real-time applications on commodity systems.* Using nonblocking operations on such systems (for example, media players) helps avoid priority inversion and provides preemption tolerance. That is, it eliminates the possibility that an active thread—potentially executing a high-priority task—is blocked awaiting action by other threads that are delayed for a long time (say, because of page faults). This helps make long noticeable delays highly unlikely in such systems.

Selecting Nonblocking Features

Unlike the example of the shared counter, almost any nonblocking operation that involves more than one location presents multiple considerations that are often at odds. This article examines trade-offs and compromises that have to be considered in selecting features of nonblocking operations. These trade-offs are seldom straightforward and sometimes require compromises in order to achieve the bare minimum of system requirements. Balancing these trade-offs can be daunting if you start unaware of all the potential pitfalls.

The goals of this article are to walk the reader through the various issues and features of nonblocking operations and the various choices to be made; to understand the interactions among these options; and to develop a sense of the best path to take to disentangle the interdependencies among these choices and quickly prune options that are incompatible with the main objectives of using nonblocking operations in the first place.

Figure 1. Variant of the classic lock-free IBM LIFO free list algorithm.

```
Structures:
Header : <pointer, integer>  // Initially: <null, 0>

Push (b : pointer) : void
1: h = Header;            // single-width read of Header
2: *b = h;                // Make the pushed block point
                          // to the old header
3: if CAS(&Header, h, n) return ; else goto 1;
                          // use single width CAS to try to make
                          // the pushed block the new header
                          // if Header is still equal to h

Pop () : pointer
1: <h,t> = Header; if h == null return null;
                          // double-width read of Header
2: n = *h;           // read the pointer in the first block
3: if CAS(&Header, <h,t>, <n,t+1>) return n ; else goto 1;
                          // Use double-width CAS to try to make
                          // the pointer in the popped block
                          // become the new Header, if Header
                          // has not changed since it was read
                          // in line 1
```

For some uses of nonblocking operations, such as kill-safety, the need for nonblocking progress likely involves operations on multiple data structures and services. In these cases, one has to consider the totality of the interdependence of the features of the various nonblocking components to reach an acceptable solution.

A Running Example

Figure 1 presents a variant of the classic lock-free IBM LIFO (last-in first-out)-free list algorithm[10] as a running example. The variable `Header` consists of a packed pointer-integer pair that can be read and operated on atomically by a double-width CAS. The integer size is assumed to be large enough that it never overflows. The notation `*b` is used to indicate the location pointed to by a pointer `b`. Depending on the context, reads and CAS operations on `Header` are either single- or double-width.

Ignoring for now why an integer is packed with the pointer in `Header` (explained later), the reader can notice that the inaction of any number of threads stopped anywhere in the Push and Pop operations cannot block other threads from operating on the list. In fact, since Push and Pop are not only nonblocking but also lock-free, then as long as there are active threads attempting these operations, some operation will complete. Whenever any thread takes five steps in one of these operations, some operation must have completed (by the same or a different thread) during that period.

Key Issues in Selecting Nonblocking Features

The following are the key issues to consider in selecting the features of nonblocking operations.

Levels of progress guarantee. The appropriate level of nonblocking progress (for example, obstruction-free vs. lock-free) and the extent of use of nonblocking operations depend on the progress guarantees required by the system.

To achieve async signal safety using nonblocking operations, only the signal handler's operations need to be nonblocking, while operations by the main threads can be blocking. For example, in a case where signal handlers may search hash tables that may be updated by threads that may be interrupted, a fully nonblocking algorithm is not necessary, whereas an algorithm with nonblocking search operations and blocking add and remove operations (as described by Heller et al[6]) can be sufficient.

If the only concern is the interaction between the signal handler and the interrupted thread, then using obstruction-free operations is sufficient. Kill-safety tends to require broader use of nonblocking operations to access objects shared by processes that are subject to arbitrary termination. Similar to async signal safety, the use of obstruction-free operations suffices to provide availability. Using lock-free or wait-free operations guarantees livelock freedom or starvation freedom, respectively.

For avoiding priority inversion in soft real-time applications, it may suffice to make the high-priority operation obstruction-free; however, stronger progress guarantees (lock-free or wait-free) are clearly desirable in this context.

Wait-free algorithms tend to entail space at least linear in the number of threads that may operate concurrently.[1] Aside from the space overheads, some recent wait-free algorithms show reasonably competitive performance.[5,19] Lock-free algorithms can achieve competitive performance with blocking algorithms and do not have inherently high space requirements. There are hardly any custom obstruction-free algorithms for specific data structures that are not lock-free; however, obstruction-free algorithms for arbitrary atomic sections (or transactional memory) are evidently simpler than corresponding lock-free algorithms.

Accordingly, in choosing the appropriate level of nonblocking progress to use, you must take into account what is absolutely needed to achieve the primary requirement (for example, kill-safety) vs. what is desirable (for example, wait-free progress) and what this entails for the other issues.

If the number of threads that can concurrently execute certain operations is limited, then strong progress guarantees can be achieved by using simpler algorithms than when higher levels of concurrency are allowed (for example, single-producer or single-consumer queues vs. multiproducer multiconsumer queues).

To achieve async signal safety using nonblocking operations, only the signal handler's operations need to be nonblocking, while operations by the main threads can be blocking.

An issue that is sometimes missed is the effect of read operations on update operations. Consider two buffer implementations. Both support check (if empty) and add operations. In both implementations, operations are lock-free. The first implementation guarantees that check operations never prevent an add operation from completing. In the other, check operations may interfere with and prevent add operations from completing (for example, as a result of using reference counting, as discussed later).

You can see how problematic the latter implementation is when the act of checking if the buffer is not empty can prevent items from ever being added to the buffer. Therefore, in selecting the appropriate level of progress for a nonblocking operation, it is not enough just to require the implementation, for example, to be lock-free. It is also important to decide which operations must be immune from interference by others. In the buffer example, while add operations are lock-free with respect to each other in both implementations, it is desirable that an individual add operation is wait-free with respect to any number of concurrent check operations.

The **choice of data structures** is one of the most important decisions in designing a nonblocking environment. This step is often overlooked because data-structure choices may be inherited from sequential or blocking designs. In choosing data structures, designers need to consider the minimum requirements and the range of desirable characteristics.

For example, if lock-free FIFO lists are considered, one should ask whether FIFO order is indeed necessary or if a more relaxed order is acceptable. If it is the latter, then the design may be simplified and performance under certain conditions may be improved by using lock-free LIFO lists instead. The classic LIFO list algorithm has simpler memory safety requirements (as discussed later) and in general has shorter path length than FIFO algorithms.

Another example is that a search structure that may be a good choice in sequential code (for example, balanced trees) may not be the best choice in nonblocking systems compared with hash structures. Also, static structures such as hash tables with open addressing can be simpler to manage than dynamic structures such as hash tables with chaining.

Safe memory reclamation issues. Nonblocking operations, by definition, do not wait for actions by other nonblocking operations and cannot expect to prevent other nonblocking operations from taking actions. This poses a problem for nonblocking operations that dereference pointers to dynamic structures that could be removed and freed by other operations. Without proper precautions a nonblocking operation may be about to access a dynamic block when the block gets removed from the structure and freed by another operation. This could lead to problematic outcomes such as an access violation if the block were freed back to the operating system, corrupting a different structure that happens to allocate and use the memory of the freed block, or returning an incorrect result.

Using the LIFO-free list (in Figure 1) as an example, one can see that a Pop operation may lead to accessing free memory. For example, a thread `P` reads a pointer to address `A` from the variable `Header` in line 1 of Pop. Then, another thread `Q` pops the block at `A` from the list and frees it back to the operating system. Now `P` proceeds to line 2 in Pop and dereferences the pointer to `A`, potentially suffering an access violation.

Paul McKenney[13] offers a detailed discussion of safe memory reclamation issues and solutions. The following is a brief list of categories of memory-safety solutions and their implications:

- *Automatic GC (garbage collection).* On systems with GC, such as Java applications, memory safety is implicitly guaranteed. In the preceding example, as long as thread `P` holds a reference to block `A`, block `A` is guaranteed not to be freed. Of course, this raises the question of whether GC itself is nonblocking.
- *RCU (Read-Copy-Update) and epoch-based collectors.* Briefly, RCU-like solutions guarantee that blocks removed from certain data structures are not freed until it can be established that all threads have reached quiescence points where it is impossible that any threads can still be holding references to such blocks.[14]
- *Reference counting.* Blocks are associated with counters that are incremented and decremented as threads acquire and release references to such blocks. Typically, a block is freed only when its reference count goes to zero and it is guaranteed that no new references to it can be created.[20]
- *Hazard pointers.* Briefly, in this method each thread that may traverse blocks that may be removed and freed owns a number of hazard pointers. Before dereferencing a pointer, a thread sets one of its hazard pointers to the pointer value. Other threads that may remove the block guarantee that they will not free the block until no hazard pointers point to it.[8,16]

The choice of data structures is one of the most important decisions in designing a nonblocking environment.

Hardware transactional memory may simplify nonblocking safe memory reclamation.[4] As discussed later, however, most upcoming mainstream HTM implementations are best effort and require an alternate non-HTM path.

The options for safe memory reclamation in order of increasing flexibility (and difficulty of memory-safety solutions) are:

- Freed blocks will be reused but never freed for different uses.
- Freed blocks can be coalesced and reused for different types and sizes but not returned to the operating system.
- Freed blocks may be coalesced, reused arbitrarily, or returned to the operating system.

Note that for some algorithms (for example, the classic LIFO list), memory safety might not be a problem if the operating system supports nonfaulting loads. In the scenario just mentioned of thread `P` reading address `A` in line 2 of Pop after the block at `A` was freed to the operating system, a system with nonfaulting loads would allow such a read.

ABA prevention. The ABA problem is common in optimistic concurrency algorithms, including nonblocking ones. The term ABA refers to the change of the value of a shared variable from `A` to `B` and back to `A` again. Using the LIFO-list Pop operation as an example and ignoring the packed integer with Header for now, the following is an ABA problem scenario starting with a list that includes blocks A and B:

a. A thread P reads the value A from Header in line 1 and B from *A in line 2;

b. Other threads pop blocks A and B and push blocks C and A, leaving Header holding the value A again;

c. P performs a CAS operation on Header with parameters A and B in line 3, and the CAS succeeds.

Now the list is corrupted. Header points to B, which is not in the list anymore. What went wrong is that (without the packed integer) P's Pop algorithm cannot differentiate between the case where Header never changed between lines 1 and 3 and this scenario where the list changed but Header returned to its old value before the CAS in line 3.

The classic LIFO algorithm packs an integer with the pointer in the Header variable and is designed such that the counter will change if the list changes between lines 1 and 3 of Pop (assuming that the counter never overflows).

Packing an integer with the main content of variables vulnerable to the ABA problem is the classic ABA prevention solution.[2] It remained the only solution for decades, but it has its limitations. It requires wide atomic operations to allow space for a large enough integer that cannot overflow (at least cannot overflow in the lifetime of one operation, such as Pop in the example). Another disadvantage of this solution is that it requires the integer subfield to retain its semantics indefinitely. This can make it very difficult to free the memory of dynamic blocks that include variables packed with ABA-prevention counters, thus meaning memory cannot be coalesced or reused for different purposes.

Although the ABA problem can occur in algorithms that do not use dynamic memory, its solutions are intertwined with safe memory reclamation solutions. First, as already mentioned, the classic ABA solution can hinder memory reclamation. Second, any memory-safety solution (GC, RCU, reference counting, hazard pointers) can be adapted to construct an ABA solution, possibly with an added level of indirection.

An advantage of the RCU type of solution is that traversal operations have almost no overhead, while reference counting and hazard pointers have nontrivial overhead for traversal operations. On the other hand, unlike RCU, reference counting and hazard-pointer methods guarantee bounds on the number of removed blocks that are not ready for reuse.

A disadvantage of reference counting that can be significant in some cases is that it can cause a supposedly read-only operation (such as the check operation mentioned earlier) to actually write to shared data (reference counters) and prevent update operations from ever completing.

The operations LL (load-linked), SC (store-conditional), and VL (validate) are inherently immune to the ABA problem. LL(location) returns the value in a shared location. VL(location) returns a Boolean indicator of whether or not the shared location has not been written by another thread since the last LL to it by the current thread. SC(location, value) writes a new value to the location if and only if it has not been written by another thread since it was last LL'ed to it by the current thread, and it returns a Boolean indicator of the occurrence of such a write. If the read in line 1 of Pop is replaced with LL(&Header) and the CAS in line 3 of Pop is replaced with SC(&Header,n), then the Pop operation would be immune to the ABA problem, without the need for using a packed integer.

Actual architectures that support LL/SC (for example, IBM Power PC) do not support the full semantics of ideal LL/SC/VL. None supports the VL operation, and all impose restrictions on what can be executed between LL/SC and prohibit the nesting or interleaving of LL/SC pairs on different locations. So, while actual LL/SC support can help the lock-free LIFO algorithm avoid the ABA problem, it is limited in preventing the ABA problem in general.

While implementations of ideal LL/SC/VL present an absolute ABA prevention solution,[18] their strong semantics disallow many correct scenarios. For example, all ABA scenarios in the lock-free LIFO-list Push operation are benign. Therefore, it is advisable to consider algorithms expressed in terms of reads and CAS operations, and address only the harmful cases of ABA, such as Pop but not Push in the lock-free LIFO-list algorithm.

It is advisable to consider ABA prevention and safe memory reclamation solutions together to avoid unnecessary duplication of overhead or solutions that are contradictory or contrary to the overall system requirements.

Portability of required atomic operations. The range of hardware support for atomic operations needed for nonblocking algorithms and methods varies significantly. If portability is an important issue, designers need to take that into account in selecting data structures, algorithms, and supporting methods for safe memory reclamation and management, and ABA prevention.

Figure 2. Data structure requirements.

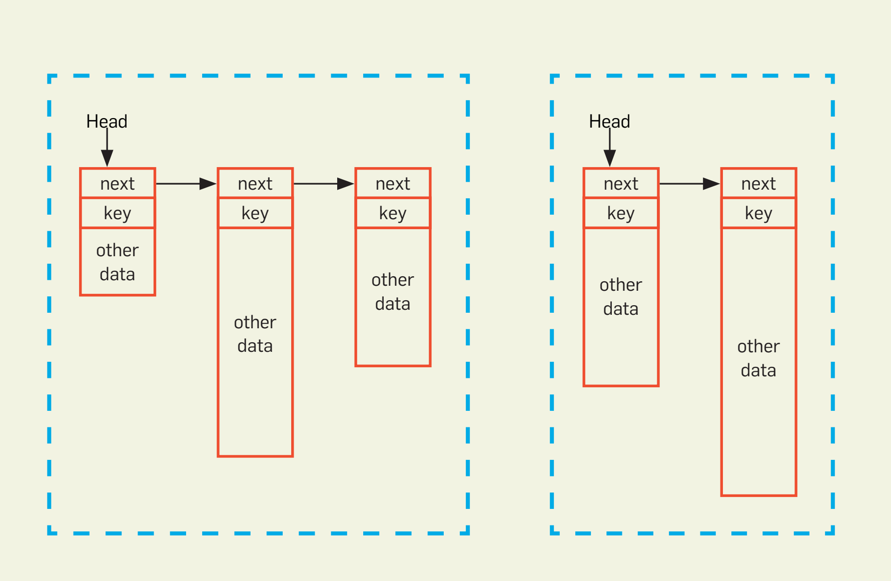
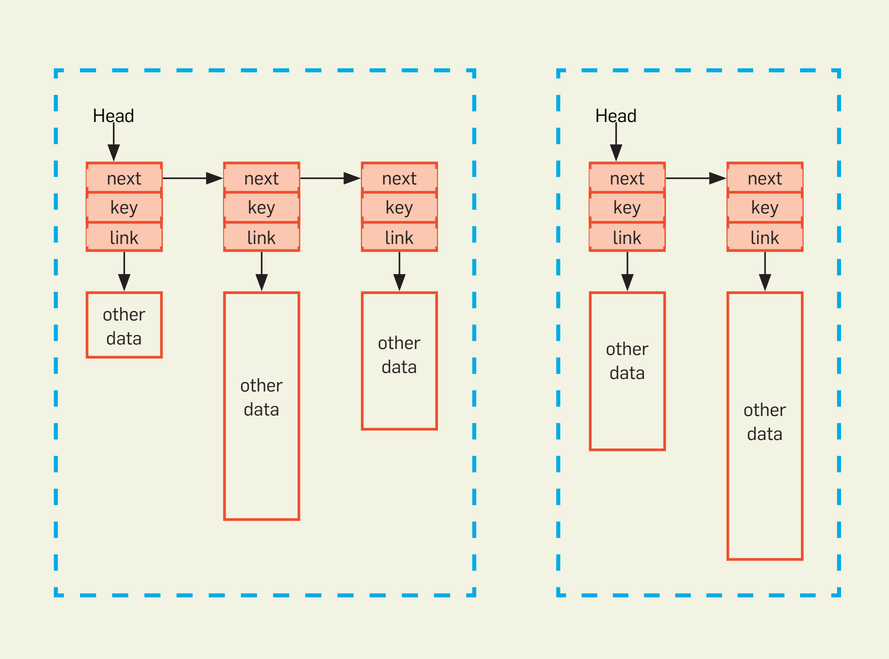


Figure 3. Design compromise.



Hardware support requirements include:

▸ *No support.* For example, the reading and writing of hazard pointers can be nonatomic.[16]

▸ *Nonuniversal atomic operations* (such as fetch-and-add, test-and-set, and atomic swap). Maurice Herlihy showed it is impossible to design wait-free (and lock-free) implementations of certain data types that can be operated on by an arbitrary number of concurrent threads using only such (nonuniversal) operations.[7]

▸ *Compare-and-swap.* Herlihy showed that CAS is a universal operation and can be used to design implementations of any data type with wait-free operations without limitation on the maximum number of threads that operate on it concurrently. Pointer-size CAS may suffer from the ABA problem. The classic solution to that problem requires the use of wider atomic operations (for example, double-width load and CAS primitives).

▸ *The PAIR LL and SC* (for example, `larx` and `stcx` on the IBM Power architecture). These were also shown by Herlihy to be universal operations. As already discussed, they are immune to the ABA problem in some cases in which CAS is susceptible. Therefore, pointer-size LL/SC may suffice or entail simpler code where double-width CAS is needed in the absence of LL/SC.

▸ *Hardware transaction memory.* Recently IBM (Blue Gene/Q,[21] System Z,[12] and Power[3]) and Intel[11] architectures are offering hardware support for arbitrary memory transactions. However, most of these HTM architectures (except IBM System Z) are best effort (that is, that they require programmers provide a nontransactional path in case the hardware transactions never succeed).

Note that if the number of threads that can concurrently execute certain operations is limited, nonuniversal atomic operations may suffice to design wait-free and lock-free implementations. For example, wait-free single-producer or single-consumer FIFO queue operations[15] (by skipping the appropriate locks in the two-lock algorithm) and single-updater sets with lock-free lookup by multiple threads[16] can be implemented with just atomic loads and stores.

Choice of language and memory ordering. In addition to the variety of requirements of atomic operations, nonblocking algorithms vary in their requirements of memory-ordering primitives. For example, in the Push operation of the running LIFO-list example (in Figure 1), the write in line 2 must be ordered before the write (in the case of a successful CAS) in line 3. Hardware platforms and high-level programming languages offer explicit primitives, as well as implicit guarantees of ordering among memory accesses, usually specified as the architecture or language memory consistency model.

Prior to Java 5 (2004) and C11/C++11, these languages could not be reliably used (as specified by their standards) to fully express the required memory ordering. Programmers and custom library writers relied on assembly language and machine binary codes to express such ordering.

Now with C11/C++11 and Java (5 and later), programmers can designate some variables as subject to special ordering (volatiles in Java and atomics in C11/C++11). In some cases, these designations can be heavy handed in imposing order around such variables even when not needed by the algorithms. Standard C11/C++11 offers finer levels of ordering that allow the programmer to specify the desired order.

The implementations of such languages have varying overheads on different hardware platforms. Designers of nonblocking implementations should take into account the choice of high-level languages and their memory-ordering performance on the targeted hardware platforms.

Choice of algorithms. The combined choice of data structures and algorithms is one issue where big compromises can be made to design a system that meets its overall requirements.

Nonblocking algorithms vary in their portability (for example, requirement of special hardware support), reliability (for example, whether or not they are widely used), complexity (for example, reasonable ease of implementation, maintenance, and modification), progress guarantees (for example, wait-free, and lock-free), and memory safety and ABA-prevention features (for example, compatibility or incompatibility with certain methods). The choice of algorithms is intertwined with most of the issues discussed here.

Case Study

The purpose of the following example is to demonstrate the interactions among nonblocking features and issues discussed in this article.

Consider a simplified example of a kill-safe system that requires processes to share large potentially variable-size records (each with a unique key value) that can be deleted or moved arbitrarily among various data structures. Operations on the records and data structures are search, add, delete, and update of certain fields. Figure 2 shows dynamic variable-size records that can be freed and moved among structures.

Considering that the records can be moved back and forth among data structures, any nonblocking algorithm will have to deal with the ABA problem. Since the records can be large and variably sized, limiting the reuse of their memory is not an acceptable option. The classic ABA solution (used in the LIFO-list Pop example) can be excluded, as it will limit arbitrary reuse of memory. Epoch-based solutions should also be excluded because they might get extremely complex under arbitrary termination. But then, the remaining solutions—reference counting and hazard pointers—are not acceptable options either. They depend on preventing the records from being reinserted in the same structures (which is a requirement) as long as there are active references to them. There is no acceptable solution.

This may be a good time to consider a compromise. How about adding a separate descriptor to each record? The descriptor holds the key value and any fields needed for traversing the data structures; the fields then may be changed in place without removing and adding the records to the structure. With this compromise the memory of the full records can still be freed, but it might be acceptable to keep descriptors from being freed for arbitrary reuse.

Now let's reconsider the ABA problem. Maybe the classic packed-integer solution can work. This has the advantage of allowing the descriptors (and the associated records) to be moved freely among data structures, but it adds a dependence on hardware support for wide atomic operations (for example, 128-bit CAS). The other options (hazard pointers and reference counting) do not require wide atomic operations but will require the use of a fresh descriptor (that is, one that does not have any old references lingering from prior operations) on every pair of moves of the records between data structures. Furthermore, both hazard pointers and reference counting add overhead (memory ordering and extra atomic operations, respectively). Finally, to deal with the actual allocation and deallocation of the variable-size records, the designer can use a lock-free allocation algorithm with coalescing and the ability to return free memory to the operating system.[17]

Now the designer is left with the option of balancing sequential performance and portability.

This case study goes through a few iterations over the various choices, starting with what is absolutely necessary, making a compromise (using descriptors and adding a level of indirection), and finally reaching a reasonable set of options for further consideration of their pros and cons.

Summary

This article has presented the key issues that can help designers of nonblocking systems make oft-needed compromises and balance trade-offs to reach a feasible design that satisfies all the requirements. In considering the totality of these issues, designers should go through a few passes on this list of features and issues. First, they should identify and separate the features that are absolutely necessary from those that are desirable but open to compromise. After that, they can start to consider the implications of using these various features.

Acknowledgments

Thanks to Samy Al Bahra and Paul McKenney for their insightful comments on earlier versions of this article.

Related articles on queue.acm.org

Structured Deferral: Synchronization via Procrastination
Paul E. McKenney
http://queue.acm.org/detail.cfm?id=2488549

Proving the Correctness of Nonblocking Data Structures
Mathieu Desnoyers
http://queue.acm.org/detail.cfm?id=2490873

Nonblocking Algorithms and Scalable Multicore Programming
Samy Al Bahra
http://queue.acm.org/detail.cfm?id=2492433

References

1. Attiya, H. and Hendler, D. Time and space lower bounds for implementations using k-CAS. In *Proceedings of the 19th International Conference on Distributed Computing* (2005), 169¬–183.
2. Brown, P.J., Smith, R.M. 1973. U.S. Patent 3,886,525. Shared data controlled by a plurality of users (filed June 1973).
3. Cain, H.W., Frey, B. Williams, D., Michael, M.M., May, C. and Le, H. Robust architectural support for transactional memory in the Power architecture. *ACM/IEEE 40th International Symposium on Computer Architecture* (2013).
4. Dragojevic, A., Herlihy, M., Lev, Y. and Moir, M. On the power of hardware transactional memory to simplify memory management. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (2011), 99–108.
5. Fatourou, P., Kallimanis, N.D. A highly efficient wait-free universal construction. *ACM Symposium on Parallelism in Algorithms and Architectures* (2011), 325–334.
6. Heller, S., Herlihy, M., Luchangco, V., Moir, M., Scherer III, W.N. and Shavit, N. A lazy concurrent list-based set algorithm. In *Proceedings of the Ninth International Conference on Principles of Distributed Systems* (2005): 3-16.
7. Herlihy, M. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems 13*, 1d (1991), 124–149.
8. Herlihy, M., Luchangco, V., Martin, P.A. and Moir, M. Nonblocking memory management support for dynamic-sized data structures. *ACM Transactions on Computer Systems 23*, 2 (2005), 146–196.
9. Herlihy, M., Luchangco, V. and Moir, M. Obstruction-free synchronization: double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems* (2003), 522–529.
10. IBM System/370 Principles of Operation. 1975. GA22-7000-4.
11. Intel Architecture Instruction Set Extensions Programming Reference. 2012.
12. Jacobi, C., Slegel, T.J., Greiner, D.F. Transactional Memory Architecture and Implementation for IBM System Z. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture* (2012), 25-36.
13. McKenney, P. Structured deferral: Synchronization via procrastination. *ACM Queue* (2013); http://queue.acm.org/detail.cfm?id=2488549
14. McKenney, P.E., Slingwine, J.D. Read-copy update: using execution history to solve concurrency problems. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems* (1998).
15. Michael, M.M. and Scott, M.L. Simple, fast, and practical nonblocking and blocking concurrent queue algorithms. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing* (1996); 267–275.
16. Michael, M.M. Hazard pointers: Safe memory reclamation for lock-free objects. *IEEE Transactions on Parallel Distributed Systems 15*, 6 (2004), 491–504.
17. Michael, M.M. Scalable lock-free dynamic memory allocation. In *Proceedings of the 2004 ACM SIGPLAN Conference on Programming Language Design and Implementation* (2004).
18. Michael, M.M. Practical lock-free and wait-free LL/SC/VL implementations using 64-bit CAS. In *Proceedings of the 18th International Conference on Distributed Computing* (2004); 144–158.
19. Timnat, S., Braginsky, A., Kogan, A. and Petrank, E. Wait-free linked-lists. In *Proceedings of the 16th International Conference on Principles of Distributed Systems* (2012), 330-344.
20. Valois, J.D. Lock-free linked lists using compare-and-swap. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing* (1995), 214–222.
21. Wang, A., Gaudet, M., Wu, P., Amaral, J.N., Ohmacht, M., Barton, C., Silvera, R. and Michael, M.M. Evaluation of Blue Gene/Q hardware support for transactional memories. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques* (2012), 127–136.

Maged M. Michael is a research staff member at the IBM Thomas J. Watson Research Center. He is an ACM Distinguished Scientist and a member of the Connecticut Academy of Science and Engineering.

DOI:10.1145/2500468.2500475

Article development led by acmqueue
queue.acm.org

Watch out for these pitfalls that can prevent Web application scaling.

BY SEAN HULL

20 Obstacles to Scalability

WEB APPLICATIONS CAN grow in fits and starts. Customer numbers can increase rapidly, and application usage patterns can vary seasonally. This unpredictability necessitates an application that is scalable. What is the best way of achieving such scalability?

This article reveals 20 of the biggest bottlenecks that reduce and slow down scalability. By ferreting these out in your environment and applications, and stamping out the worst offenders, you will be on your way to hyper growth.

10 Obstacles to Scaling Performance

Performance is key to Web scalability. If you add more customers, you want your application to continue servicing them all equally quickly. Too much latency will cause users to give up. You can keep your application's performance from degrading by knowing all the ways it can get clogged up and avoiding those bottlenecks.

1. Two-phase commit. Normally when data is changed in a database, it is written both to memory and to disk. When a commit happens, a relational database makes a commitment to freeze the data somewhere on real storage media. Remember, memory does not survive a crash or reboot. Even if the data is cached in memory, the database still has to write it to disk. MySQL binary logs or Oracle redo logs fit the bill.

With a MySQL cluster or distributed file system such as Distributed Replicated Block Device (DRBD) or Amazon Multi-AZ (Multi-Availability Zone), a commit occurs not only locally, but also at the remote end. A two-phase commit means waiting for an acknowledgment from the far end. Because of network and other latency, those commits can be slowed down by milliseconds, as though all the cars on a highway were slowed down by big loads. For those considering using Multi-AZ or read replicas, the Amazon RDS (Relational Database Service) use-case comparison at http://www.iheavy.com/2012/06/14/rds-or-mysql-ten-use-cases/ will be helpful.

Synchronous replication has these issues as well; hence, MySQL's solution is semi-synchronous, which makes some compromises in a real two-phase commit.

2. Insufficient Caching. Caching is very important at all layers, so where is the best place to cache: at the browser, the page, the object, or the database tier? Let's work through each of these.

Browser caching might seem out of reach, until you realize the browser takes directives from the Web server and the pages it renders. Therefore, if the objects contained therein have longer expire times, the browser will cache them and will not need to fetch them again. This is faster for not only the user, but also the servers hosting the website, as all returning visitors will weigh less.

Details about browser caching are available at http://www.iheavy.com/2011/11/01/5-tips-cache-websites-boost-speed/. Be sure to set expire headers and cache control.

Page caching requires using a technology such as Varnish (https://www.varnish-cache.org/). Think of this as a mini Web server with high speed and low overhead. It cannot handle complex pages as Apache can, but it can handle the very simple ones better. It therefore sits in front of Apache and reduces load, allowing Apache to handle the more complex pages. This is like a traffic cop letting the bikes go through an intersection before turning full attention to the more complex motorized vehicles.

Object caching is done by something like memcache. Think of it as a Post-it note for your application. Each database access first checks the object cache for current data and answers to its questions. If it finds the data it needs, it gets results 10 to 100 times faster, allowing it to construct the page faster and return everything to the user in the blink of an eye. If it does not find the data it needs, or finds only part of it, then it will make database requests and put those results in memcache for later sessions to enjoy and benefit from.

3. Slow Disk I/O, RAID 5, Multitenant Storage. Everything, everything, everything in a database is constrained by storage—not by the size or space of that storage but by how fast data can be written to those devices.

If you are using physical servers, watch out for RAID 5, a type of RAID (redundant array of independent disks) that uses one disk for both parity and protection. It comes with a huge write penalty, however, which you must always carry. What's more, if you lose a drive, these arrays are unusably slow during rebuild.

The solution is to start with RAID 10, which gives you striping over mirrored sets. This results in no parity calculation and no penalty during a rebuild.

Cloud environments may work with technology such as Amazon EBS (Elastic Block Store), a virtualized disk similar to a storage area network. Since it is network based, you must contend and compete with other tenants (aka customers) reading and writing to that storage. Further, those individual disk arrays can handle only so much reading and writing, so your neighbors will affect the response time of your website and application.

Recently Amazon rolled out a badly branded offering called Provisioned IOPS (I/O operations per second). That might sound like a great name to techies, but to everyone else it does not mean anything noteworthy. It is nonetheless important. It means you can lock in and guarantee the disk performance your database is thirsty for. If you are running a database on Amazon, then definitely take a look at this.

4. Serial Processing. When customers are waiting to check out in a grocery store with 10 cash registers open, that is working in parallel. If every cashier is taking a lunch break and only

PHOTOGRAPH BY ANDREW OLSCHER/SHUTTERSTOCK.COM

one register is open, that is serialization. Suddenly a huge line forms and snakes around the store, frustrating not only the customers checking out, but also those still shopping. It happens at bridge tollbooths when not enough are open, or in sports arenas when everyone leaves at the same time.

Web applications should definitely avoid serialization. Do you see a backup waiting for API calls to return, or are all your Web nodes working off one search server? Anywhere your application forms a line, that is serialization and should be avoided at all costs.

5. Missing Feature Flags. Developers normally build in features and functionality for business units and customers. Feature flags are operational necessities that allow those features to be turned on or off in either back-end config files or administration UI pages.

Why are they so important? If you have ever had to put out a fire at 4 A.M., then you understand the need for contingency plans. You must be able to disable ratings, comments, and other auxiliary features of an application, just so the whole thing does not fall over. What's more, as new features are rolled out, sometimes the kinks do not show up until a horde of Internet users hit the site. Feature flags allow you to disable a few features, without taking the whole site offline.

6. Single Copy of the Database. You should always have at least one read replica or MySQL slave online. This allows for faster recovery in the event that the master fails, even if you are not using the slave for browsing—but you should do that, too, since you are going to build a browse-only mode, right?

Having multiple copies of a database suggests horizontal scale. Once you have two, you will see how three or four could benefit your infrastructure.

7. Using your Database for Queuing. A MySQL database server is great at storage tables or data, and relationships between them. Unfortunately, it is not great at serving as a queue for an application. Despite this, a lot of developers fall into the habit of using a table for this purpose. For example, does your app have some sort of jobs table, or perhaps a status column, with values such as "in-process," "in-queue," and "finished"? If so, you are inadvertently using tables as queues.

Such solutions run into scalability hang-ups because of locking challenges and the scan and poll process to find more work. They will typically slow down a database. Fortunately, some good open source solutions are available such as RabbitMQ (http://www.rabbitmq.com/) or Amazon's SQS (Simple Queue Service; http://aws.amazon.com/sqs/).

8. Using a Database for Full-Text Searching. Page searching is another area where applications get caught. Although MySQL has had full-text indexes for some time, they have worked only with MyISAM tables, the legacy table type that is not crash proof, not transactional, and just an all-around headache for developers.

One solution is to go with a dedicated search server such as Solr (http://lucene.apache.org/solr/). These servers have good libraries for whatever language you are using and high-speed access to search. These nodes also scale well and will not bog down your database.

Alternatively, Sphinx SE, a storage engine for MySQL, integrates the Sphinx server right into the database. If you are looking on the horizon, Fulltext is coming to InnoDB, MySQL's default storage engine, in version 5.6 of MySQL.

9. Object Relational Models. The ORM, the bane of every Web company that has ever used it, is like cooking with MSG. Once you start using it, it is hard to wean yourself off.

The plus side is that ORMs help with rapid prototyping and allow developers who are not SQL masters to read and write to the database as objects or memory structures. They are faster, cleaner, and offer quicker delivery of functionality—until you roll out on servers and want to scale.

Then your database administrator

PHOTOGRAPHS FROM SHUTTERSTOCK.COM

(DBA) will come to the team with a very slow-running, very ugly query and say, "Where is this in the application? We need to fix it. It needs to be rewritten." Your dev team then says, "We do not know!" And an incredulous look is returned from the ops team.

The ability to track down bad SQL and root it out is essential. It will happen, and your DBA team will need to index properly. If queries are coming from ORMs, they do not lend themselves to all of this. Then you are faced with a huge technical debt and the challenge of ripping and replacing.

10. Missing Instrumentation. Instrumentation provides speedometers and fuel guages for Web applications. You would not drive a car without them, would you? They expose information about an application's internal workings. They record timings and provide feedback about where an application spends most of its time.

One very popular Web services solution is New Relic (http://newrelic.com/), which provides visual dashboards that appeal to everyone—project managers, developers, the operations team, and even business units all can peer at the graphs and see what is happening.

Some open source instrumentation projects are also available.

10 Obstacles to Scaling Beyond Optimization Speed

Speed is not the only thing that can gum up scalability. The following 10 problems affect the ability to maintain and build scalability for a Web application. Best practices can avoid these issues.

1. Lack of a Code Repository and Version Control. Though it is rare these days, some Internet companies do still try to build software without version control. Those who use it, however, know the everyday advantage and organizational control it provides for a team.

If you are not using it, you are going to spiral into technical debt as your application becomes more complex. It will not be possible to add more developers and work on different parts of your architecture and scaffolding.

Once you start using version control, be sure to get all components in there, including configuration files and other essentials. Missing pieces that have to be located and tracked down at deployment time become an additional risk.

2. Single Points of Failure. If your data is on a single master database, that is a single point of failure. If your server is sitting on a single disk, that is a single point of failure. This is just technical vernacular for an Achilles heel.

These single points of failure must be rooted out at all costs. The trouble is recognizing them. Even relying on a single cloud provider can be a single point of failure. Amazon's data center or zone failures are a case in point. If it had multiple providers or used Amazon differently, AirBNB would not have failed when part of Amazon Web Services went down in October 2012 (http://www.iheavy.com/2012/10/23/airbnb-didnt-have-to-fail/).

3. Lack of Browse-only mode. If you have ever tried to post a comment on Yelp, Facebook, or Tumblr late at night, you have probably gotten a message to the effect, "This feature is not available. Try again later." "Later" might be five minutes or 60 minutes. Or maybe you are trying to book airline tickets and you have to retry a few times. To nontechnical users, the site still appears to be working normally, but it just has this strange blip.

What's happening here is that the application is allowing you to browse the site, but not make any changes. It means the master database or some storage component is offline.

Browse-only mode is implemented by keeping multiple read-only copies of the master database, using something such as MySQL replication or Amazon read replicas. Since the application will run almost fully in browse mode, it can hit those databases without the need for the master database. This is a big, big win.

4. Weak communication. Communication may seem a strange place to take a discussion on scalability, but the technical layers of Web applications cannot be separated from the social and cultural ones that the team navigates.

Strong lines of communication are necessary, and team members must know whom to go to when they are in trouble. Good communication demands confident and knowledgeable leadership, with the openness to listen and improve.

5. Lack of Documentation. Documentation happens at a lot of layers in a Web application. Developers need to document procedures, functions, and pages to provide hints and insight to future generations looking at that code. Operations teams need to add comments to config files to provide change history and insight when things break. Business processes and relationships can and should be documented in company wikis to help people find their own solutions to problems.

Documentation helps at all levels and is a habit everyone should embrace.

6. Lack of Fire drills. Fire drills always get pushed to the backburner. Teams may say, "We have our backups; we're covered." True, until they try to restore those backups and find they are incomplete, missing some

PHOTOGRAPH FROM SHUTTERSTOCK.COM

config file or crucial piece of data. If that happens when you are fighting a real fire, then something you do not want will be hitting that office fan.

Fire drills allow a team to run through the motions, before they really need to. Your company should task part of its ops team with restoring its entire application a few times a year. With AWS and cloud servers, this is easier than it once was. It is a good idea to spin up servers just to prove that all your components are being backed up. In the process you will learn how long it takes, where the difficult steps lie, and what to look out for.

7. Insufficient Monitoring and Metrics. Monitoring falls into the same category of best practices as version control: it should be so basic you cannot imagine working without it; yet there are Web shops that go without, or with insufficient monitoring—some server or key component is left out.

Collecting this data over time for system and server-level data, as well as application and business-level availability, are equally important. If you do not want to roll your own, consider a Web services solution to provide your business with real uptime.

8. Cowboy Operations. You roll into town on a fast horse, walk into the saloon with guns blazing, and you think you are going to make friends? Nope, you are only going to scare everyone into complying but with no real loyalty. That is because you will probably break things as often as you fix them. Confidence is great, but it is best to work with teams. The intelligence of the team is greater than any of the individuals.

Teams need to communicate what they are changing, do so in a managed way, plan for any outage, and so forth. Caution and risk aversion win the day. Always have a Plan B. You should be able to undo the change you just made, and be aware which commands are destructive and which ones cannot be undone.

Caution and risk aversion win the day. Always have a Plan B.

9. Growing Technical Debt. As an app evolves over the years, the team may spend more and more time maintaining and supporting old code, squashing bugs, or ironing out kinks. Therefore, they have less time to devote to new features. This balance of time devoted to debt servicing versus real new features must be managed closely. If you find your technical debt increasing, it may be time to bite the bullet and rewrite. It will take time away from the immediate business benefit of new functionality and customer features, but it is best for the long term.

Technical debt is not always easy to recognize or focus on. As you are building features or squashing bugs, you are more attuned to details at the five-foot level. It is easy to miss the forest for the trees. That is why generalists are better at scaling the Web (http://www.iheavy.com/2011/10/25/why-generalists-better-scaling-web/).

10. Insufficient Logging. Logging is closely related to metrics and monitoring. You may enable a lot more of it when you are troubleshooting and debugging, but on an ongoing basis you will need it for key essential services. Server syslogs, Apache and MySQL logs, caching logs, among others should all be working. You can always dial down logging if you are getting too much of it, or trim and rotate log files, discarding the old ones.

Conclusion

These 20 obstacles can affect scalability and result in performance degradation of a Web application. By avoiding these obstacles and following the practices outlined here, Web-application developers will be able to minimize latency and guarantee that their applications can scale as needed. C

Related articles on queue.acm.org

Building Scalable Web Services
Tom Killalea
http://queue.acm.org/detail.cfm?id=1466447

Improving Performance on the Internet
Tom Leighton
http://queue.acm.org/detail.cfm?id=1466449

Scalable SQL
Michael Rys
http://queue.acm.org/detail.cfm?id=1971597

Sean Hull (hullsean@gmail.com; http://www.iheavy.com/blog/) is the founder and senior consultant at Heavyweight Internet Group in New York. He has more than two decades of experience as a technology consultant and adviser working with clients such as AppNexus, The Hollywood Reporter, Billboard, NBC Universal, and Zagats, among other hyper-growth companies that handle up to 100 million unique visitors per month.

DOI:10.1145/2500468.2500477

Article development led by acmqueue
queue.acm.org

NUMA becomes more common because memory controllers get close to execution units on microprocessors.

BY CHRISTOPH LAMETER

An Overview of Non-Uniform Memory Access

NON-UNIFORM MEMORY ACCESS (NUMA) is the phenomenon that memory at various points in the address space of a processor have different performance characteristics. At current processor speeds, the signal path length from the processor to memory plays a significant role. Increased signal path length not only increases latency to memory but also quickly becomes a throughput bottleneck if the signal path is shared by multiple processors. The performance differences to memory were noticeable first on large-scale systems where data paths were spanning across motherboards or chassis. These systems required modified operating-system kernels with NUMA support that explicitly understood the topological properties of the system's memory (such as the chassis in which a region of memory was located) in order to avoid excessively long signal path lengths. (Altix and UV, SGI's large address space systems, are examples. These products had to modify the Linux kernel to support NUMA; in these machines, processors in multiple chassis are linked via a proprietary interconnect called NUMALINK).

Today, processors are so fast they usually require memory to be directly attached to the socket they are on. A memory access from one socket to memory from another has additional latency overhead to accessing local memory—it requires the traversal of the memory interconnect first. On the other hand, accesses from a single processor to local memory not only have lower latency compared to remote memory accesses but also do not cause contention on the interconnect and the remote memory controllers. It is good to avoid remote memory accesses. Proper placement of data will increase the overall bandwidth and the latency to memory.

As the trend toward improving system performance by bringing memory even nearer to processor cores continues, NUMA will play an increasingly important role in system performance. Modern processors have multiple memory ports, and the latency of access to memory varies even only depending on the position of the core on the die relative to the controller. Future generations of processors will have increasing differences in performance as more cores on chip necessitate more sophisticated caching. As the access properties of these different kinds of memory continue to diverge, new functionality may be needed in operating systems to allow for good performance.

NUMA systems today are mostly encountered on multisocket systems. A typical high-end business-class server today comes with two sockets and will therefore have two NUMA nodes. Latency for a memory access (random access) is about 100ns. Access to memory on a remote node adds another 50% to that number.

Performance-sensitive applications can require complex logic to handle memory with diverging performance characteristics. If a developer requires explicit control of the placement of memory for performance reasons, some operating systems provide APIs for this (for example, Linux, Solaris, and Microsoft Windows provide system calls for NUMA). However, various heuristics have been developed in the operating systems that manage memory access to allow applications to transparently utilize the NUMA characteristics of the underlying hardware.

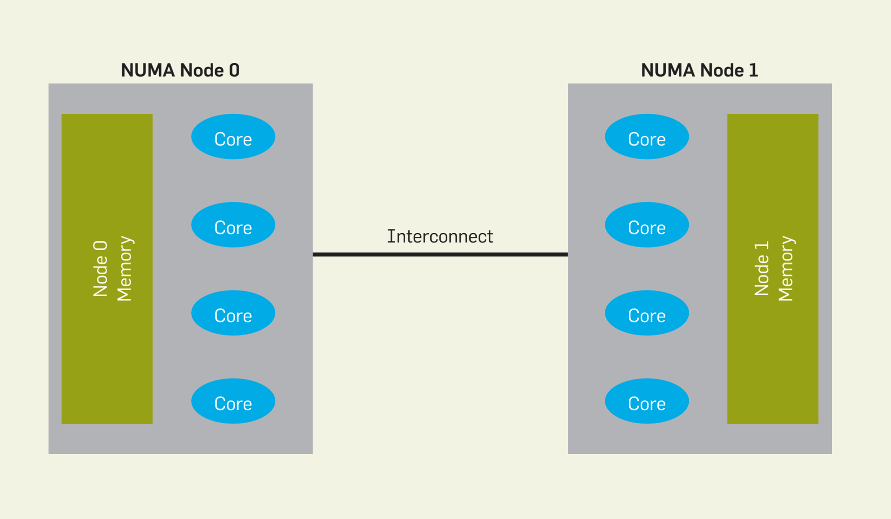
A NUMA system classifies memory into *NUMA nodes* (what Solaris calls *locality groups*). All memory available in one node has the same access characteristics for a particular processor. Nodes have an affinity to processors and to devices. These are the devices that can use memory on a NUMA node with the best performance since they are locally attached. Memory is called *node local* if it was allocated from the NUMA node that is best for the processor. For the example, the NUMA system exhibited in Figure 1 has one node belonging to one socket with four cores each.

The process of assigning memory from the NUMA nodes available in the system is called *NUMA placement*. As placement influences only performance and not the correctness of the code, heuristics approaches can yield acceptable performance. In the special case of noncache-coherent NUMA systems, this may not be true since writes may not arrive in the proper sequence in memory. However, noncache-coherent NUMA systems have multiple challenges when attempting to code for them. We restrict ourselves here to the common cache-coherent NUMA systems.

The focus in these discussions will be mostly on Linux since it is an operating system with refined NUMA facilities and is widely used in performance-critical environments today. The author was involved with the creation of the NUMA facilities in Linux and is most familiar with those.

Solaris also has somewhat comparable features,[a] but the number of systems deployed is orders of magnitude less. Work is under way to add support to other Unix-like operating systems, but that support so far has been mostly confined to operating-system tuning parameters for placing memory accesses. Microsoft Windows also has a developed NUMA subsystem that allows placing memory structures effectively, but the software is used mostly for enterprise applications rather than high-performance computing. Requirements on memory-access speeds for enterprise-class applications are frequently more relaxed than in high-performance computing, meaning that less effort is spent on NUMA memory handling in Windows compared with Linux.

Figure 1. A system with two NUMA nodes and eight processors.



How Operating Systems Handle NUMA Memory

There are several broad categories in which modern production operating systems allow for the management of NUMA: accepting the performance mismatch, hardware memory striping, heuristic memory placement, a static NUMA configurations, and application-controlled NUMA placement.

Ignore the difference. Since NUMA placement is a best-effort approach, one option is simply to ignore the possible performance benefit and just treat all memory as if no performance differences exist. This means the operating system is not aware of memory nodes. The system is functional, but performance varies depending on how memory happens to be allocated. The smaller the differences between local and remote accesses, the more viable this option becomes.

This approach allows software and the operating system to run unmodified. Frequently, this is the initial ap-

[a] For details, see http://docs.oracle.com/cd/E19963-01/html/820-1691/gevog.html; http://docs.oracle.com/cd/E19082-01/819-2239/6n4hsf6rf/index.html; http://docs.oracle.com/cd/E19082-01/819-2239/madv.so.1-1/index.html/.

proach for system software when systems with NUMA characteristics are first used. The performance will not be optimal and will likely be different each time the machine and/or application runs, because the allocation of memory to performance-critical segments varies depending on the system configuration and timing effects on boot-up.

Memory striping in hardware. Some machines can set up the mapping from memory addresses to the cache lines in the nodes in such a way that consecutive cache lines in an address space are taken from different memory controllers (interleaving at the cache-line level). As a result, the NUMA effects are averaged out (since structures larger than a cache line will then use cache lines on multiple NUMA nodes). Overall system performance is more deterministic compared with the approach of just ignoring the difference, and the operating system still does not need to know about the difference in memory performance, meaning no NUMA support is needed in the operating system. The danger of overloading a node is reduced since the accesses are spread out among all available NUMA nodes.

The drawback is the interconnect is in constant use. Performance will never be optimal since the striping means that cache lines are frequently accessed from remote NUMA nodes.

Heuristic memory placement for applications. If the operating system is NUMA-aware (under Linux, NUMA must be enabled at compile time and the BIOS or firmware must provide NUMA memory information for the NUMA capabilities to become active; NUMA can be disabled and controlled at runtime with a kernel parameter), then it is useful to have measures that allow applications to allocate memory that minimizes signal path length so that performance is increased. The operating system has to adopt a policy that maximizes performance for as many applications as possible. Most applications run with improved performance using the heuristic approach, especially compared with the approaches discussed earlier.

A NUMA-aware operating system determines memory characteristics from the firmware and can therefore tune its own internal operations to the memory configuration. Such tuning requires coding effort, however, so only performance-critical portions of the operating system tend to get optimized for NUMA affinities, whereas less-performance-critical components tend to continue to operate on the assumption that all memory is equal.

A NUMA-aware operating system determines memory characteristics from the firmware and can therefore tune its own internal operations to the memory configuration.

The most common assumptions made by the operating system are that the application will run on the local node and that memory from the local node is to be preferred. If possible, all memory requested by a process will be allocated from the local node, thereby avoiding the use of the cross-connect. The approach does not work, though, if the number of required processors is higher than the number of hardware contexts available on a socket (then processors on both NUMA nodes must be used); if the application uses more memory than available on a node; or if the application programmer or the scheduler decides to move application threads to processors on a different socket after memory allocation has occurred.

In general, small Unix tools and small applications work very well with this approach. Large applications that make use of a significant percentage of total system memory and of a majority of the processors on the system will often benefit from explicit tuning or software modifications that take advantage of NUMA.

Most Unix-style operating systems support this mode of operation. Notably, FreeBSD and Solaris have optimizations to place memory structures to avoid bottlenecks. FreeBSD can place memory round-robin on multiple nodes so the latencies average out. This allows FreeBSD to work better on systems that cannot do cache-line interleaving on the BIOS or hardware level (Additional NUMA support is planned for FreeBSD 10). Solaris also replicates important kernel data structures per locality group.

Special NUMA configuration for applications. The operating system provides configuration options that allow the operator to tell the operating system that an application should not be run with the default assumptions regarding memory placement. It is possible to establish memory-allocation policies for an application without modifying code.

Command-line tools exist under Linux that can set up policies to determine memory affinities (`taskset`, `numactl`). Solaris has tunable parameters for how the operating system allocates memory from locality groups as well. These are roughly comparable to Linux's process memory-allocation policies.

Application control of NUMA allocations. The application may want fine-grained control of how the operating system handles allocation for each of its memory segments. For that purpose, system calls exist that allow the application to specify which memory region should use which policies for memory allocations.

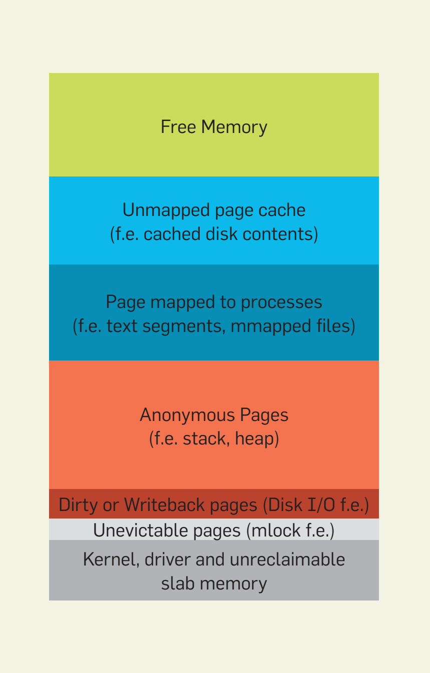
The main performance issues typically involve large structures that are accessed frequently by the threads of the application from all memory nodes and that often contain information that needs to be shared among all threads. These are best placed using interleaving so the objects are distributed over all available nodes.

How Does Linux Handle NUMA?

Linux manages memory in zones. In a non-NUMA Linux system, zones are used to describe memory ranges required to support devices that are not able to perform DMA (direct memory access) to all memory locations. Zones are also used to mark memory for other special needs such as movable memory or memory that requires explicit mappings for access by the kernel (HIGHMEM), but that is not relevant to the discussion here. When NUMA is enabled, then more memory zones are created and they are also associated with NUMA nodes. A NUMA node can have multiple zones since it may be able to serve multiple DMA areas. How Linux has arranged memory can be determined by looking at `/proc/zoneinfo`. The NUMA node association of the zones allows the kernel to make decisions involving the memory latency relative to cores.

On boot-up Linux will detect the organization of memory via the ACPI (Advanced Configuration and Power Interface) tables provided by the firmware and then create zones that map to the NUMA nodes and DMA areas as needed. Memory allocation then occurs from the zones. Should memory in one zone become exhausted, then memory reclaim occurs where Linux will scan through the least recently used pages trying to free a certain number of pages. Counters that show the current status of memory in various nodes/zones can also be seen in `/proc/zoneinfo`. Figure 2 shows types of memory in a zone/node.

Figure 2. Types of memory in a zone/node.


Free Memory
Unmapped page cache
(f.e. cached disk contents)
Page mapped to processes
(f.e. text segments, mmapped files)
Anonymous Pages
(f.e. stack, heap)
Dirty or Writeback pages (Disk I/O f.e.)
Unevictable pages (mlock f.e.)
Kernel, driver and unreclaimable
slab memory

Memory policies. How memory is allocated under NUMA is determined by a memory policy. Policies can be specified for memory ranges in a process's address space, or for a process or the system as a whole. Policies for a process override the system policy, and policies for a specific memory range override a process's policy.

The most important memory policies are:

NODE LOCAL. The allocation occurs from the memory node local to where the code of the process is currently executing.

INTERLEAVE. Allocation occurs round-robin. First a page will be allocated from node 0, then from node 1, then again from node 0, and so on. Interleaving is used to distribute memory accesses for structures that may be accessed from multiple processors in the system in order to have an even load on the interconnect and the memory of each node.

There are other memory policies that are used in special situations that are not mentioned here for brevity's sake. The two policies just mentioned are generally the most useful and are used by default by the operating system. NODE LOCAL is the default allocation policy if the system is up and running.

The Linux kernel will use the INTERLEAVE policy by default on bootup. Kernel structures created during bootstrap are distributed over all the nodes available in order to avoid putting excessive load on a single memory node later when processes require access to the operating-system structures. The system default policy is changed to NODE LOCAL when the first userspace process (`init` daemon) is started.

The active memory allocation policies for all memory segments of a process (and information that shows how much memory was actually allocated from which node) can be seen by determining the process `id` and then looking at the contents of `/proc/<pid>/numa_maps`.

Basic operations on process startup. Processes inherit the memory policy from their parent. Most of the time the policy is left at the default, which means NODE LOCAL. When a process is started on a processor, then memory is allocated for that process from the local NUMA node. All other allocations of the process (through growing the heap, page faults, mmap, and so on) will also be satisfied from the local NUMA node.

The Linux scheduler will attempt to keep the process cache hot during load balancing. This means the preference of the scheduler is to leave the process on processors that share the L1-processor cache, then on processors that share L2, and then on processors that share L3 with the processor the process ran on last. If there is an imbalance beyond that, then the scheduler will move the process to any other processor on the same NUMA node.

As a last resort the scheduler will move the process to another NUMA node. At that point the code will be executing on the processor of one node, while the memory allocated before the move will have been allocated on the old node. Most memory accesses from the process will then be remote, which will cause the performance of the process to degrade.

There has been some recent work in making the scheduler NUMA-aware to ensure the pages of a process can be moved back to the local node, but that work is available only in Linux 3.8 and later, and is not considered mature yet. Further information on the state of affairs may be found on the Linux kernel mailing lists and in articles on http://lwn.net.

Reclaim. Linux typically allocates all available memory in order to cache data that may be used again later. When memory begins to be low, reclaim will be used to find pages that are either not in use or unlikely to be used soon. The effort required to evict a page from memory and to get the page back if the need arises varies by type of page. Linux prefers to evict pages from disk that are not mapped into any process space because it is easy to drop all references to the page. The page can be reread from disk if required later. Pages that are mapped into a process's address space require the page first be removed from that address space before the page can be reused. A page that is not a copy of a page from disk (anonymous pages) can be evicted only if the page is first written out to swap space (an expensive operation). There are also pages that cannot be evicted at all, such as `mlocked()` memory or pages in use for kernel data.

The impact of reclaim on the system can therefore vary. In a NUMA system there will be multiple types of memory allocated on each node. The amount of currently free space on each node will vary. So if there is a request for memory and the local node would require reclaim but another node has enough memory to satisfy the request without reclaim, then the kernel has two choices:

- Run a reclaim pass on the local node (causing kernel processing overhead) and then allocate node-local memory to the process.
- Just allocate from another node that does not need a reclaim pass. Memory will not be node local, but we avoid frequent reclaim passes. Reclaim will be performed when all zones are low on free memory. This approach reduces the frequency of reclaim and allows more of the reclaim work to be done in a single pass.

For small NUMA systems (such as the typical two-node servers) the kernel defaults to the second approach. For larger NUMA systems (four nodes and higher) the kernel will perform a reclaim in order to get node-local memory whenever possible because the latencies have higher impacts on process performance.

There is a knob in the kernel that determines how the situation is to be treated in `/proc/sys/vm/zone_reclaim`. A value of 0 means that no local reclaim should take place. A value of 1 tells the kernel that a reclaim pass should be run in order to avoid allocations from the other node. On boot-up a mode is chosen based on the largest NUMA distance in the system.

If zone reclaim is switched on, then the kernel still attempts to keep the reclaim pass as lightweight as possible. By default, reclaim will be restricted to unmapped page-cache pages. The frequency of reclaim passes can be further reduced by setting `/proc/sys/vm/min_unmapped_ratio` to the percentage of memory that must contain unmapped pages in order to run a reclaim pass. The default is 1%.

Zone reclaim can be made more aggressive by enabling write-back of dirty pages or the swapping of anonymous pages, but in practice doing so has often resulted in significant performance issues with reclaim.

Basic NUMA command-line tools. The main tool used to set up the NUMA execution environment for a process is `numactl`, which also allows the display of the system NUMA configuration, as well as the control of shared memory segments. It is possible to restrict processes to a set of processors, as well as to a set of memory nodes. `Numactl` can be used, for example, to avoid task migration between nodes or restrict the memory allocation to a certain node. Note that additional reclaim passes may be required by the kernel if the allocation is restricted. Those cases are not influenced by zone-reclaim mode because the allocation is restricted by a memory policy to a specific set of nodes, and therefore the kernel does not have a choice simply to pick memory from another NUMA node.

Another tool that is frequently used for NUMA is `taskset`. It basically allows only binding of a task to processors and therefore has only a subset of `numactl`'s capability. `Taskset` is heavily used in non-NUMA environments, and therefore the familiarity results in developers preferring to use `taskset` instead of `numactl` on NUMA systems.

NUMA information. There are numerous ways to view information about the NUMA characteristics of the system and of various processes currently running. The hardware NUMA configuration of a system can be viewed through the use of `numactl --hardware`. This includes a dump of the SLIT (system locality information table) that shows the cost of accesses to different nodes in a NUMA system. The example in Figure 3 shows a NUMA system with two nodes. The distance for a local access is 10. A remote access costs twice as much on this system (20). This is the

Figure 3. Displaying NUMA characteristics of a system.

```
$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
node 0 size: 131026MB
node 0 free: 588MB
node 1 cpus: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
node 1 size: 131072MB
node 1 free: 169MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

convention, but the practice of some vendors (especially for two-node systems) is simply to provide 10 and 20 without regard to the actual latency differences to memory.

`Numastat` is another tool that is used to show how many allocations were satisfied from the local node. Of particular interest is the `numa_miss` counter, which indicates the system assigned memory from a different node in order to avoid reclaim. These allocations also contribute to other node. The remainder of the count are intentional off-node allocations. The amount of off-node memory can be used as a guide to figure out how effectively memory was assigned to processes running on the system (see Figure 4).

How memory is allocated to a process can be seen via a status file in `/pro/<pid>/numa_maps` (illustrated in Figure 5).

The output shows the virtual address of the policy and then some information about what the NUMA characteristics are of the memory range. `Anon` means the pages do not have an associated file on disk. `Nx` shows the number of pages on the respective node.

The information about how memory is used in the system as a whole is available in `/proc/meminfo`. The same information is also available for each NUMA node in `/sys/devices/system/node/node<X>/meminfo`. Numerous other bits of information are available from the directory where `meminfo` is located. It is possible to compact memory, get distance tables, and manage huge pages and mlocked pages by inspecting and writing values to key files in that directory.

First-touch policy. Specifying memory policies for a process or an address range does *not* cause any allocation of memory, which is often confusing to newcomers. Memory policies specify what should happen *when* the system needs to allocate memory for a virtual address. Pages in a process's memory space that have not been touched or that are zero do not have memory assigned to them. The processor will generate a hardware fault when a process touches or writes to an address (*page fault*) that is not populated yet. During page-fault handling by the kernel, the page is allocated. The instruction that caused the fault is then restarted and will be able to access the memory as needed.

What matters, therefore, is the memory policy in effect *when the allocation occurs*. This is called the *first touch*. The first-touch policy refers to the fact that a page is allocated based on the effective policy when some process first uses a page in some fashion.

Figure 4. Displaying NUMA statistics of the system.

```
$ numastat
                node0        node1
numa_hit        13273229839  4595119371
numa_miss       2104327350   6833844068
numa_foreign    6833844068   2104327350
interleave_hit  52991        52864
local_node      13273229554  4595091108
other_node      2104327635   6833872331
```

Figure 5. Displaying NUMA settings and statistics of the system.

```
# cat /proc/1/numa_maps
7f830c175000 default anon=1 dirty=1 active=0 N1=1
7f830c177000 default file=/lib/x86_64-linux-gnu/ld-2.15.so anon=1 dirty=1
active=0 N1=1
7f830c178000 default file=/lib/x86_64-linux-gnu/ld-2.15.so anon=2 dirty=2
active=0 N1=2
7f830c17a000 default file=/sbin/init mapped=18 N1=18
7f830c39f000 default file=/sbin/init anon=2 dirty=2 active=0 N1=2
7f830c3a1000 default file=/sbin/init anon=1 dirty=1 active=0 N1=1
7f830dc56000 default heap anon=223 dirty=223 active=0 N0=52 N1=171
7fffb6395000 default stack anon=5 dirty=5 active=1 N1=5
```

The effective memory policy on a page depends on memory policies assigned to a memory range or on a memory policy associated with a task. If a page is only in use by a single thread, then there is no ambiguity as to which policy will be followed. However, pages are often used by multiple threads. Any one of them may cause the page to be allocated. If the threads have different memory policies, then the page may as a result seem to be allocated in surprising ways for a process that also sees the same page later.

For example, it is fairly common that text segments are shared by all processes that use the same executable. The kernel will use the page from the text segment if it is already in memory regardless of the memory policy set on a range. The first user of a page in a text segment will therefore determine its location. Libraries are frequently shared among binaries, and especially the C library will be used by almost all processes on the system. Many of the most-used pages are therefore allocated during boot-up when the first binaries run that use the C library. The pages will at that point become established on a particular NUMA node and will stay there for the time the system is running.

First-touch phenomena limit the placement control that a process has over its data. If the distance to a text segment has a significant impact on process performance, then dislocated pages will have to be moved in memory. Memory could look like it was allocated on NUMA nodes not permitted by the memory policy of the current task because an earlier task already brought the data into memory.

Moving memory. Linux has the capability to move memory. This means the virtual address of the memory in the process space stays the same. Only the physical location of the data is moved to a different node. The effect can be observed by looking at `/proc/<pid>/numa_maps` before and after a move.

Migrating all of a process's memory to a node can optimize performance of an application by avoiding cross-connect accesses should the system have placed pages on other NUMA nodes. However, a regular user can move only pages of a process that are referenced by only that process alone (otherwise, the user could interfere with performance optimization of processes owned by other users). Only `Root` has the capability to move all pages of a process.

It can be difficult to ensure all pages are local to a process since some text segments are heavily shared and there can be only one page backing an address of a text segment. This is particularly an issue with the C library or other heavily shared libraries.

Linux has a `migratepages` command-line tool to manually move pages around by specifying a `pid`, as well as the source and destination nodes. The memory of the process will be scanned for pages currently allocated on the source node. Those will be moved to the destination node.

NUMA scheduling. The Linux scheduler had no notion of the page placement of memory in a process until Linux 3.8. Decisions about migrating processes were made on an estimate of the cache hotness of a process's memory. If the Linux scheduler moved the execution of a process to a different NUMA node, then the performance of that process could be significantly impacted because its memory now would require access via the cross-connect. Once that move was complete the scheduler would estimate the process memory is cache hot on the remote node and leave the process there as long as possible. As a result, administrators who wanted the best performance felt it best not to let the Linux scheduler interfere with memory placement. Processes were often pinned to a specific set of processors using `taskset`, or the system was partitioned using the `cpusets` feature to isolate applications to stay within the NUMA node boundaries.

In Linux 3.8 the first steps were made to address this situation by merging a framework that will enable the scheduler at some point to consider the page placement and perhaps automatically migrate pages from remote nodes to the local node. However, there is a significant development effort still needed, and the existing approaches do not always enhance the performance of a given computing load. This was the state of affairs earlier this year; for more recent information on the Linux kernel mailing list, see http://vger.kernel.org or articles from *Linux Weekly News* (http://lwn.net; for example, http://lwn.net/Articles/486858/).

Conclusion

NUMA support has been around for a while in various operating systems. NUMA support in Linux has been available since early 2000 and is being continually refined. Frequently kernel NUMA support will optimize process execution without the need for user intervention, and in most use cases an operating system can simply be run on a NUMA system, providing decent performance for typical applications.

Special NUMA configuration through tools and kernel configuration comes into play when the heuristics provided by the operating system do not provide satisfactory application performance to the end user. This is typically the case in high-performance computing, high-frequency trading, and for real-time applications, but recently these issues have become more significant for regular enterprise-class applications. Traditionally, NUMA support required special knowledge about the application and hardware for proper tuning using the knobs provided by the operating systems. Recent developments point (especially around the Linux NUMA scheduler) to developments that will result in the ability of the operating systems to automatically balance a NUMA application load properly over time.

The use of NUMA needs to be guided by the increase in performance that is possible. The larger the difference between local and remote memory access, the greater the benefits that arise from NUMA placement. NUMA latency differences are due to memory accesses. If the application does not rely on frequent memory accesses (because, for example, the processor caches absorb most of the memory operations), then NUMA optimizations will have no effect. Also for I/O-bound applications the bottleneck is typically the device and not memory access. An understanding of the characteristics of the hardware and software is required in order to optimize applications using NUMA. C

Related articles on queue.acm.org

Photoshop Scalability: Keeping It Simple
Clem Cole and Russell Williams
http://queue.acm.org/detail.cfm?id=1858330

The Cost of Virtualization
Ulrich Drepper
http://queue.acm.org/detail.cfm?id=1348591

Performance Anti-Patterns
Bart Smaalders
http://queue.acm.org/detail.cfm?id=1117403

Additional Reading

Braithwaite, R., McCormick, P., Feng, W.
Empirical memory-access cost models in multicore NUMA architectures. Virginia Tech Department of Computer Science, 2011.

Hacker, G.
Using NUMA on RHEL 6; http://www.redhat.com/summit/2012/pdf/2012-DevDay-Lab-NUMA-Hacker.pdf.

Kleen, A.
***A NUMA API for Linux*. Novell, 2005; http://developer.amd.com/wordpress/media/2012/10/LibNUMA-WP-fv1.pdf.**

Lameter, C.
Effective synchronization on Linux/NUMA systems. Gelato Conference, 2005. Effective synchronization on Linux/NUMA systems. Gelato Conference, 2005.

Lameter, C.
2006. Remote and local memory: Memory in a Linux/NUMA system. Gelato Conference, 2006.

Li, Y., Pandis, I., Mueller, R., Raman, V., Lohman, G.
NUMA-aware algorithms: The case of data shuffling. University of Wisconsin-Madison / IBM Almaden Research Center, 2013.

Love, R.
2004. Linux Kernel Development. Indianapolis: Sams Publishing.

Oracle.
Memory and Thread Placement Optimization Developer's Guide, 2010; http://docs.oracle.com/cd/E19963-01/html/820-1691/.

Schimmel, K.
Unix Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers. Addison-Wesley, 1994.

Christoph Lameter specializes in high-performance computing and high-frequency trading technologies. As an operating-system designer and developer, he has been developing memory management technologies for Linux to enhance performance and reduce latencies. He is fond of new technologies and new ways of thinking that disrupt existing industries and cause new development communities to emerge.

DOI:10.1145/2500889

How to create and resolve discomfort for a thrilling and memorable experience.

BY STEVE BENFORD, CHRIS GREENHALGH, GABRIELLA GIANNACHI, BRENDAN WALKER, JOE MARSHALL, AND TOM RODDEN

Uncomfortable User Experience

THE INCREASING USE of computers in games, rides, performances, installations, and other cultural experiences is shifting the focus of user-experience design from the traditional usability goals of learnability, performance, and minimizing errors to new ones, like fostering emotional and aesthetic engagement.[17] This switch inspires unconventional approaches that turn traditional interaction design on its head, as in, say, celebrating the role of ambiguity rather than clarity[11] and transforming system limitations into opportunities.[4] Here, we integrate perspectives from human-computer interaction (HCI) and performance studies to explore the deliberate engineering of discomfort as a way to create intense, memorable interactions and engage challenging themes.

Uncomfortable interaction—managed carefully and ethically—may become an important tool for designers, promoting entertainment, enlightenment, and sociality. We draw on our experience creating and studying interactive performances and amusement park rides to explore how discomfort can address the following questions:

- What are the potential benefits of uncomfortable interaction?;
- What forms can such interaction take?;
- How can discomfort be created?;
- How can it be embedded in an experience?; and
- What ethical challenges must be addressed?

Benefits

Uncomfortable interaction causes a degree of suffering to the user, mentally through suspense, fear, and anxiety or even physically through movement, exertion, and pain. While suffering is not the goal of a cultural experience, discomfort is often employed in a transitory way to realize three key benefits:

Entertainment. Discomfort can arouse and excite and so entertain us. Amusement park rides employ extreme acceleration, sudden drops, and inversions to create the visceral sensation of thrill, while games and films (rides, too) employ an uncomfortable feeling of suspense through anticipation of dangers to come. Discomfort may increase the subjective intensity and memorability of such an experience, heightening a participant's sense of flow, or the psychological state of deep focus associated with immersive activities like computer games.[5]

» key insights

- **The deliberate use of discomfort in interaction design can help produce a more entertaining, enlightening, socially bonding cultural experience.**
- **Designers can employ combinations of visceral and cultural discomfort by distorting control and social relationships.**
- **Embedding discomfort into an overall user experience must be done with care and reflect ethical considerations.**

PHOTO ILLUSTRATION BY BARRY DOWNARD

Enlightenment. Discomfort can frame our engagement with challenging themes, provoking us to reflect on our feelings and responses. Artistic works that confront challenging themes may employ discomfort to establish an appropriate tone, demand personal commitment, avoid trivialization, and promote empathy and respect. Religious and spiritual practices may involve abstinence, fasting, and asceticism.

Sociality. Confronting discomfort can prompt social bonding through shared rites of passage, as in, say, a child's "first" roller coaster ride[8] or adolescent boys watching horror films together.[12] The same principle is in effect in team-development activities involving physically demanding tasks.

Designers may prefer materials that are rough, tight, prickly, sweaty, or otherwise physically unpleasant.

Examples

The arts, especially the performing arts, involve a longstanding tradition of discomfort. In the 1930s, German poet and playwright Bertold Brecht proclaimed theater should contain some level of *verfremdung* (alienation), causing unease or discomfort by encouraging the audience to look at something or someone from another's point of view.[3] The latter part of the 20th century saw numerous performances that pushed the boundaries of discomfort, including Marina Abramovi ́c's "Rhythm O" (1974) where the audience was encouraged to apply a gun, bullet, pocket knife, axe, and matches to a performer's body and Vito Acconci's "Project for Pier 17" (1971) where the audience was invited to a late-night meeting on a derelict pier to hear Acconci confess a secret. Since the 1990s, the Cypriot-Australian performance artist Stelarc has created a series of works in which audience members are invited to observe his suspended body being moved and controlled by machinery and, in one memorable case, remotely controlled through electric stimuli.[21]

While artists may intend to push the boundaries, discomfort is also found in mainstream entertainment, from the visceral thrills and scares of an amusement park ride to the suspense of computer games, with the latter including even commodity electric-shock game controllers.[9] HCI researchers and designers of tangible interfaces have also experimented with discomfort; for example, users of "The Meatbook" (2007) interacted with the system by manipulating raw meat;[14] exertion games involving intense or stressful physical interaction (such as punching, kicking, and hanging from ceiling bars);[18] "I Seek the Nerves Under Your Skin" requiring participants to increase their running pace to hear a frantic punk poem;[15] and "Mediated Body" transgressing conventional social norms by requiring participants to stroke a performer's body in public view to explore an interactive soundscape.[13]

Even this brief dip into the arts and entertainment reflects how routinely discomfort is employed in all manner of cultural experiences. In order to ground a more in-depth exploration of the phenomenon consider the following examples:

"Breathless." This prototype amusement park ride, created as part of the Horizon Centre for Digital Economy Research Day in the Park project, focuses on entertainment in the mainstream setting of an amusement park as part of a long-term exploration of future ride technologies. An early prototype in which a conventional bucking-bronco ride was controlled through a chest-strap breathing monitor highlighted the potential of using breathing to control rides, especially by requiring riders to simultaneously battle the ride and their own physical response, focusing them inward on their own feelings.[16] "Breathless" extended this approach in 2011 by upping the discomfort level through respiration sensors embedded into a Wi-Fi-enabled gas mask to control a powered swing while requiring the rider to breathe in harmony with the swing's resonant frequency to make it go higher.

This control mechanism was embedded in an overall ride experience through a design inspired by Fragonard's painting "The Swing" (1767), an erotic scene involving three people: a woman on a swing, a voyeur in the bushes watching the woman's exposed legs, and a bishop controlling the swing through a pull rope (see Figure 1). This scene was mapped onto a ride structure in which a participant would move among three distinct roles: voyeur, rider, and controller. Upon arrival, participants would join a queue, to be fitted with a gas mask when reaching the front. They were then taken to a specific location where they would be the voyeur watching a floodlit rider swinging before them. Next, they mounted the swing as the new rider and subsequently took the role of controller. Each ride began with the controller's breathing driving the swing but transferring swing control over to the rider halfway through.

"Ulrike and Eamon Compliant." In contrast, in an example from the arts, the British artists Blast Theory are renowned for their mobile and interactive performances, several of which have been studied within the HCI context.[2] "Ulrike and Eamon Compliant," created for the 2009 Venice Biennale, addresses the theme of terrorism, inviting participants to enter the minds of one of two notorious international terrorists, Ulrike Meinhof and Eamon Collins. The work takes the form of a solo city walk where participants receive a series of automated phone calls guiding their direction while narrating episodes from the lives of either Ulrike or Eamon, detailing the events leading to their terrorist acts, their subsequent arrests, and ultimately their deaths. The instructions are designed to establish a sense of constant surveillance and increasing compliance through such tactics as requiring participants to perform physical gestures (such as stopping in the middle of a bridge and touching their heads) (see Figure 2) or taking off their sunglasses and sitting on a bench. They are twice asked to confirm they wish to proceed.

Participants are eventually guided to a deserted alleyway leading to a canal (or similar landmark in other cities) where they are asked to make one final commitment to continuing the journey. If yes (nearly all do), they are guided to a waiting performer who leads them to an interview room with two chairs and a mirror to be interviewed by a second performer who asks their personal views on terrorism, leading to the question, "Could you imagine a situation in which your community is being attacked, with people killing your neighbors and friends at random, and where you

Figure 1. Rider on the Swing, with human controller in background, in "Breathless."

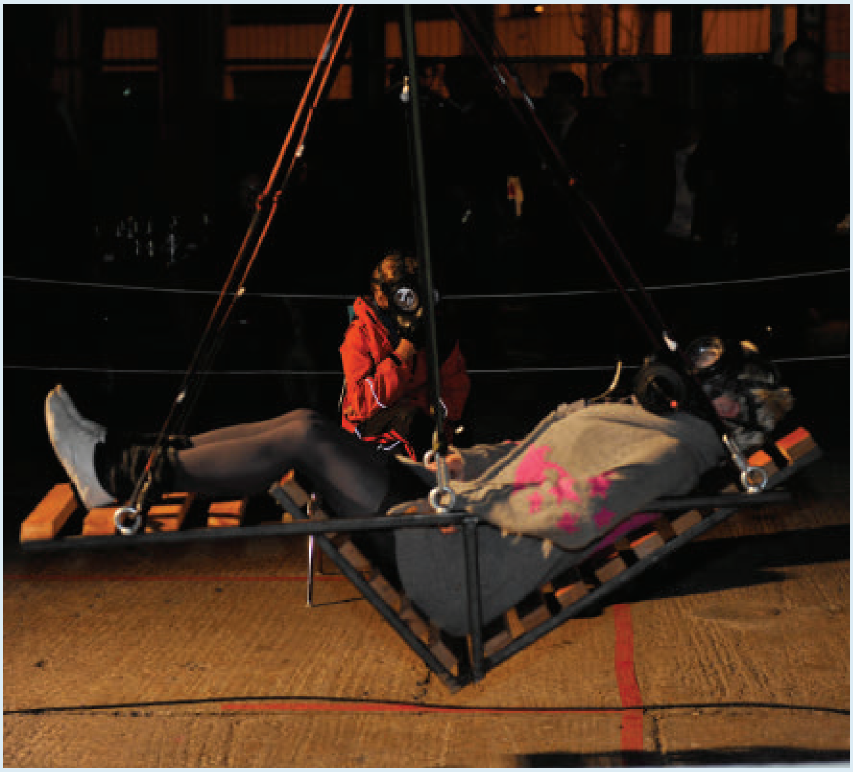


Figure 2. Complying with an instruction in "Ulrike and Eamon Compliant."

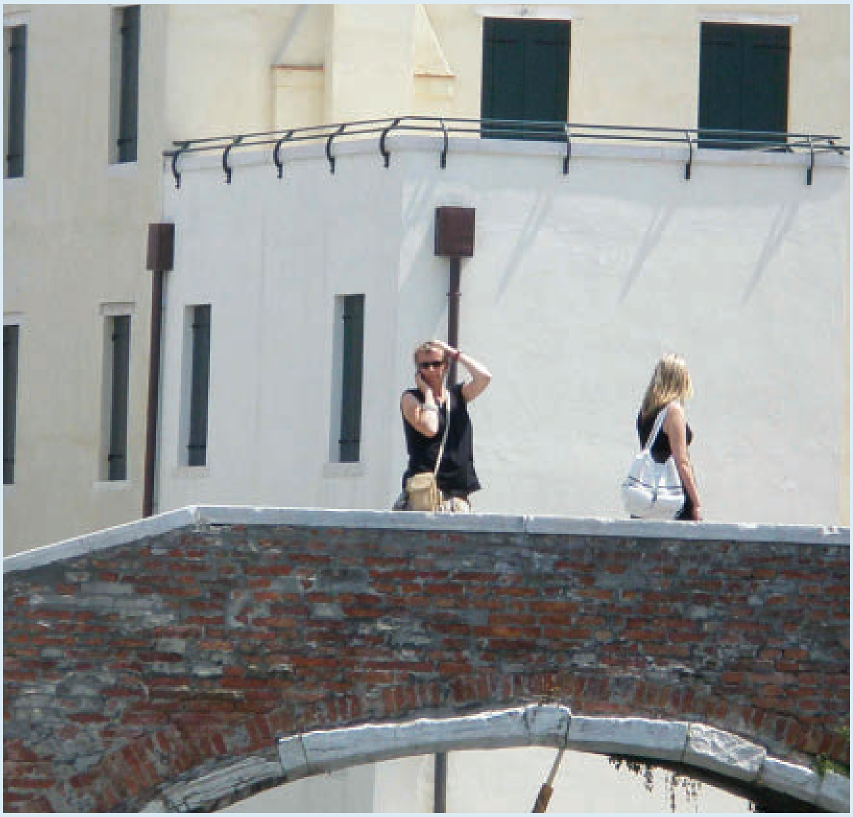


Figure 3. Final interview in "Ulrike and Eamon Compliant."

might have to fight?" As they are led from the interview room, they are invited to pause behind a (one-way) mirror to briefly watch the next participant being interviewed (see Figure 3).

Uncomfortable Interaction

These scenarios reflect how uncomfortable interaction can be in practice, identifying four primary forms of discomfort, each leading to a set of design tactics:

Visceral. In light of the growing interest in physically embodied interaction,[7] we first consider visceral discomfort, referring to the aspects of our personal experience relating most directly to physical sensation, from the unpleasant sensation of materials to demanding stressful or strenuous movement to causing pain. They reflect three tactics for creating visceral discomfort:

Design unpleasant wearables and tangibles. Devices can be uncomfortable to touch, hold, and wear. The gas mask from "Breathless" has a striking physicality—hot, sweaty, and claustrophobic, with an overpowering rubbery smell—while the tactile sensations of "The Meatbook" evoke disgust. Designers may prefer materials that are rough, tight, prickly, sweaty, or otherwise physically unpleasant;

Encourage strenuous physicality. The second tactic is to drive interaction through unusually strenuous physical activity. Roller coasters and other thrill rides place physical stress on the body through high g-forces, inversions, rolls, and drops, while "I Seek the Nerves" and other exertion experiences generate intense feelings through exertion or stressful positions (such as hanging from ceiling bars);[18] and

Cause pain. The most extreme tactic is to cause pain, as through, say, electric-shock game controllers. An effective tactic here is to deliver "acute" pain (in the sense of transitory rather than especially strong) as opposed to "chronic" pain while not causing physical damage.

Cultural. A contrasting form of discomfort invokes dark cultural associations:

Confront challenging themes and difficult decisions. The cultural acceptability of material considered adult, difficult, or vulgar provides a significant (and shifting) boundary for discomfort. Interactive works increase discomfort by requiring users to take difficult moral decisions directly, rather than being left to observe; for example, "Ulrike and Eamon Compliant" invited participants to defend or reject the actions of terrorists; and

Design culturally resonant devices. Cultural associations extend to the form of the interface itself. In addition to visceral discomfort, gas masks may invoke chilling associations with, or even memories of, warfare and civil unrest. Such resonance may be culturally and contextually specific, as in, say, the contrasting associations of a gas mask in a war museum compared to a fetish-themed nightclub.

Control. HCI guidelines have long maintained that the locus of control should remain with the user;[20] that is, it is generally good when people control the interface rather than the interface controls them. Experience designers can therefore generate discomfort by distorting this relationship:

Surrender control to the machine. Part of the thrill of a ride involves giving up control to a machine, being strapped in and unable to dismount. Interactive experience opens up the possibility of partial or unreliable control; for example, the "Broncomatic" invoking the powerful feeling of simultaneously battling to control a ride and one's own body while ultimately losing control of both;

Surrender control to others. Theatrical performances typically involve surrendering control to performers, possibly engendering uncomfortable feelings of disempowerment. This surrender is a familiar tactic in many everyday conventional performances, as in, say, a comedian singling out an audience member; for example, "Ulrike and Eamon Compliant" demands deep compliance with detailed instructions, while "Breathless" involves surrendering control to another participant; and

Require participants take greater control. Discomfort can be found in assuming greater control of others, as it may invoke feelings of power, responsibility, capriciousness, and mischief. Thus, "Breathless" requires participants to control others, as well as being controlled by others, while Blast Theory's performance "Uncle Roy All Around You" invites online participants to control pedestrians on the streets of a remote city.[1]

Intimacy. Computers are increasingly employed to maintain social relationships, giving rise to various social tactics for creating discomfort:

Isolate people. Isolating a participant from friends and family is a common tactic, leaving them alone in an unfamiliar environment. Isolation is not only disturbing but naturally focuses people inward on their own feelings. Both "Ulrike and Eamon Compliant" and "Breathless" exploited this tactic, with the former requiring solo exploration of Venice and the latter using gas masks to anonymize participants, reduce their ability to communicate, and focus them on their own breathing;

Establish intimacy with strangers. In contrast, intimate encounters with strangers can be especially uncomfortable. The one-to-one interview in "Ulrike and Eamon Compliant" is a challenge, while the "Mediated Body" required participants to physically touch a stranger's body; and

Employ surveillance and voyeurism. This final tactic emphasizes the sense of vulnerability inherent in surveillance by unseen observers, as implied by the instructions in "Ulrike and Eamon Complaint." There is also discomfort in watching others, as in, say, the helplessness a viewer would feel watching loved ones on a dramatic roller coaster ride. The reverse is the illicit thrill of voyeurism exploited by "Ulrike and Eamon Compliant" when participants are invited to look through a one-way mirror.

A variety of risks must be considered, from physical danger and injury to emotional trauma to social embarrassment.

Embedding Discomfort in the Experience

Having identified tactics for creating uncomfortable interactions, recall that our intention is to employ them in the longer-term pursuit of entertainment, enlightenment, and sociality. Discomfort is not our overall goal but rather a transitory point on a journey. Again, an experience designer can turn to the field of performance studies for assistance. The European Renaissance of the 14th–17th centuries saw development of the classic five-act performance structure consisting of exposition, rising action, climax, falling action, and dénouement, as visualized in Gustav Freytag's pyramid (see Figure 4) based on Aristotle's earlier three-act structure.[10] The pyramid gives an experience designer an elegant way to embed uncomfortable interaction into an experience:

Exposition. The first act addresses the initial framing of the experience to set an appropriate expectation. In "Ulrike and Eamon Compliant" the exposition takes the form of an initial briefing that explains the work, while the branding and ratings of rides support judgment of what is appropriate;

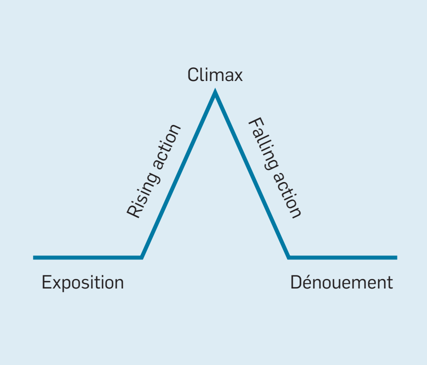
Rising action. Anticipation of discomfort increases as the experience proper begins and suspense builds; for example, a roller coaster gradually rises up a ramp toward the first drop;

Climax. Anticipation is now transformed into experience. Two important principles guide the design of this moment: First, it must be transitory, or relatively brief compared to the exposition and rising action, with effects that pass quickly. Thus, electric-shock game controllers deliver brief shocks after long periods of suspense, while the initial drop on a roller coaster takes seconds compared to perhaps an hour of queuing and waiting. Lingering feelings of nausea are a different matter, and it is unlikely that anyone would deliberately design a ride to deliver such discomfort;

Falling action. Discomfort is followed by a moment of release or catharsis that may be associated with feelings of intense pleasure, even euphoria. The designer might seek to extend such feelings for a while by, say, adding gentle curves to the end of a ride; and

Dénouement. The final act addresses the importance of reflection, letting participants assimilate the discomfort, share it with others through storytelling, deliver new insight, or simply enjoy the bragging rights of having survived, supported by a photo and,

Figure 4. Freytag's pyramid.



perhaps, other souvenir objects.

This structure can be extended in various ways; for example, multiple climatic moments can be embedded into a complex experience, as in, say, designing climbs, drops, and loops into a roller coaster or twists and false endings into a narrative. An experience may deliver different feelings when repeated (such as when suspense gives way to the pleasure of physical movement) or adapted to participants so as to provide them with a fresh experience each time round; some roller coasters even involve control of individual seats. A final option is to reveal how discomfort was engineered during the dénouement (such as participants looking back through the one-way mirror in "Ulrike and Eamon Compliant"), reflecting the way stage magicians sometimes reveal their trickery as part of the set up for a further trick.

An experience designer might ask: Would the participants be happy in hindsight with what has occurred?

Ethics of Uncomfortable Interaction

Finally, deliberately introducing discomfort into an experience requires ethical consideration; the following comments therefore address key ethical challenges:

The first overarching question is to consider on what basis an experience designer might justify the deliberate use of discomfort at all. While deontological ethical systems are based on an axiomatic definition of the rightness or wrongness of actions, other schools of ethical thought since the 19th century British philosopher Jeremy Bentham have argued a consequentialist position that assesses the goodness of an action solely in terms of that goodness or otherwise of its consequences.[6] Adopting this point of view, might an experience designer justify a degree of short-term discomfort through the longer-term benefits to participants of entertainment, enlightenment, and sociality? An experience designer might ask: Would the participants be happy in hindsight with what has occurred? And given what they know afterward would they still have chosen to take part?

A second potential route to justifying an uncomfortable interaction focuses on an individual's right to choose. Contemporary Western ethics and human rights follow the 18th century German philosopher Immanuel Kant and others in assigning a primary value to the individual, and in particular to free choice and self-determination, possibly including the right to deliberately choose discomfort, subject to limitations of its effect on others and assuming the individual is competent to make such a decision. This idea is commonly invoked in relation to an artistic or entertainment experience where both artist and audience could claim a right to freely express themselves through acts of creation or participation. However, the same principles of individual value and autonomy also disallow the arbitrary imposition of discomfort on another, at least against that person's will.

Such arguments do not, of course, provide blanket justification for uncomfortable interaction. Rather, designers must carefully weigh each experience, focusing on specific ethical concerns, and balance any temporary discomfort against the longer-term value of entertainment, enlightenment, and social bonding:

Informed consent. The idea of informed consent is challenging for cultural experiences, especially those involving surprise, where, by definition, participants do not necessarily know in advance what they signed up for. This surprise is further complicated when playing up the anticipation of discomfort beyond the actual experience, though this would seem preferable to experiences where actual discomfort exceeds anticipated discomfort.

Requiring written formal consent to take part in such an experience is rare; rather, consent is often achieved through the careful framing of the experience in advertising, ticketing, branding, and trust in the hosting venue, all representing an unwritten contract with participants. Peer pressure is another factor designers must consider. In recognizing the importance of social bonding, they must be aware of the possibility of personal social pressure on participants to participate. Some members of groups may be more hesitant than others, and designers may wish to be wary of an experience in which the leader determines the level of discomfort for an entire group;

The right to withdraw. It may be impossible for participants to withdraw from an experience once a key point is passed, as in, say, dismounting a moving roller coaster, though such a ride is typically short and carefully regulated to minimize risk to participants. Consequently, it may be justifiable for a designer to limit opportunities for withdrawal than would be the case in other contexts. While one might argue an experience should be clear about any point of no return, explicit warnings about the right to withdraw are employed to further increase suspense in some rides, even in "Ulrike and Eamon Compliant";

Privacy and anonymity. An individual's right to privacy is another ethical principle. However, tactics that generate discomfort by distorting intimacy, especially through voyeurism, clearly impinge personal privacy. A designer must therefore consider whether private actions would become visible to those outside the "performance frame." Breeches of privacy and anonymity should be restricted largely to those within the frame, especially in situations involving a degree of symmetry (such as where observers are themselves observed); and

Managing risk. Finally, experience designers have a clear responsibility to consider and manage risk. Given the breadth of the tactics we have covered here, a variety of risks must be considered, from physical danger and injury to emotional trauma to social embarrassment. Dealing with them is a practical matter requiring assessment and management within a variety of professional codes and regulations, standard practice for design professionals working in the cultural sector in galleries, theaters, and theme parks. Second are the contingencies incorporated into the experience or its related alternative "paths." Finally, there is "orchestration," or the set of procedures and supporting technologies that enable human controllers to monitor and intervene in an experience from behind the scenes.[2]

Conclusion

We have argued here for the deliberate design of uncomfortable interaction so as to deliver an entertaining, enlightening, socially bonding cultural experience. While this idea is familiar in the worlds of art and entertainment, it is unconventional in HCI. We therefore aimed to unpack the various ways deliberate discomfort could potentially be achieved, identifying four primary forms of discomfort and associated set of design tactics for each. Most important, we have urged the embedding of such tactics within an experience, along with careful consideration of ethical challenges.

Our intent is to stimulate discussion around the challenges of cultural applications of computers across games, rides, performances, and installations. An open question is whether it has implications for other types of computing.

Acknowledgments

This work is funded by the Research Councils UK (http://www.rcuk.ac.uk/) through Horizon Digital Economy Research grant (EP/G065802/1) and by the Engineering and Physical Sciences Research Council (http://www.epsrc.ac.uk/) through Dream Fellowship award EP/J005215/1. We also thank the artists of Blast Theory (http://www.blasttheory.co.uk/) for their collaboration.

References

1. Benford, S., Crabtree, A., Reeves, S., Sheridan, J., Dix, A., Flintham, M., and Drozd, A. The frame of the game: Blurring the boundary between fiction and reality in mobile experiences. In *Proceedings of the Conference on Human Factors in Computing Systems* (Montréal, Canada, Apr. 24–27). ACM Press, New York, 2006, 427–436.
2. Benford, S. and Giannachi, G. *Performing Mixed Reality.* MIT Press, Cambridge, MA, 2011.
3. Brecht, B. The modern theatre is the epic theatre. In *Brecht on Theatre*, J. Willett, Ed. Methuen, London, 1993, 33–42.
4. Chalmers, M. and Galani, A. Seamful interweaving: Heterogeneity in the theory and design of interactive systems. In *Proceedings of the Fifth Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (Cambridge, MA, Aug. 1–4). ACM Press, New York, 2004, 243–252.
5. Csíkszentmihályi, M. *Finding Flow: The Psychology of Engagement with Everyday Life.* Basic Books, A Member of the Perseus Books Group, 1997.
6. Deigh, J. *An Introduction to Ethics.* Cambridge University Press, Cambridge, U.K., 2010.
7. Dourish, P. *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, Cambridge, MA, 2006.
8. Durrant, A., Rowland, D., Kirk, D., Benford, S., Fischer, J., and McAuley, D. *Automics: Souvenir-generating photoware for theme parks.* In *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, May 7–12). ACM Press, New York, 2011, 1767–1776.
9. Electric Shock Games & Gadgets; http://www.electricshock.org/electric-shock-games.html
10. Freytag, G. *Die Technik des Dramas.* Verlag von S. Hirzel, Leipzig, Germany, 1863.
11. Gaver, W., Beaver, J., and Benford, S. Ambiguity as a resource for design. In *Proceedings of the Conference on Human Factors in Computing Systems* (Fort Lauderdale, FL, Apr. 5–10). ACM Press, New York, 2003, 233–240.
12. Goldstein, J. The attractions of violent entertainment. *Media Psychology 1*, 3 (1999), 272–282.
13. Hobye1, M. and Löwgren, J. Touching a stranger: Designing for engaging experience in embodied interaction. *International Journal of Design 5*, 3 (2011), 31–48.
14. Levisohn, A., Cochrane, J., Gromala, D., and Seo, J. The Meatbook: Tangible and visceral interaction. In *Proceedings of the First International Conference on Tangible and Embedded Interaction* (Baton Rouge, LA, Feb. 15–17). ACM Press, New York, 2007, 91–92.
15. Marshall, J. and Benford, S. Using fast interaction to create intense experiences. In *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, May 7–12). ACM Press, New York, 2011, 1255–1265.
16. Marshall, J., Rowland, D., Egglestone, S., Benford, S., Walker, B., and McAuley, D. Breath control of amusement rides. In *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, May 7–12). ACM Press, New York, 2011, 73–82.
17. McCarthy J. and Wright, P. *Technology As Experience.* MIT Press, Cambridge, MA, 2007.
18. Mueller, F., Edge, D., Vetere, F., Gibbs, M., Agamanolis, S., Bongers, B., and Sheridan, J., Designing sports: A framework for exertion games. In *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, May 7–12). ACM Press, New York, 2011, 2651–2660.
19. Schnädelbach, H., Rennick Egglestone, S., Reeves, S., Benford, S., Walker, B., and Wright, M. Performing thrill: Designing telemetry systems and spectator interfaces for amusement rides. In *Proceedings of the Conference on Human Factors in Computing Systems* (Florence, Italy, Apr. 5–10). ACM Press, New York, 2008, 1167–1176.
20. Shneiderman, B. *Designing the User Interface, Second Edition.* Addison-Wesley Publishing Co., Reading, MA, 1992.
21. Smith, M. *Stelarc: The Monograph.* MIT Press, Cambridge, MA, 2007.

Steve Benford (sdb@cs.nott.ac.uk) is an EPSRC-funded Dream Fellow and a professor of collaborative computing in the Mixed Reality Laboratory and Horizon at the University of Nottingham, Nottingham, U.K.

Chris Greenhalgh (chris.greenhalgh@nottingham.ac.uk) is a professor of computer science in the Mixed Reality Laboratory and Horizon at the University of Nottingham, Nottingham, U.K.

Gabriella Giannachi (g.giannachi@exeter.ac.uk) is a professor of performance and new media and Director of the Centre for Intermedia in the Department of English at the University of Exeter, Exeter, U.K.

Brendan Walker (studio@aerial.fm) is a senior research fellow in Horizon at the University of Nottingham, professor of creative industries at Middlesex University, and director of design practice at Aerial, U.K.

Joe Marshall (jqm@cs.nott.ac.uk) is a Leverhulme-funded research fellow in the Mixed Reality Laboratory at the University of Nottingham, Nottingham, U.K.

Tom Rodden (tar@cs.nott.ac.uk) is a professor of interactive systems in the Mixed Reality Laboratory and Horizon at the University of Nottingham, Nottingham, U.K.

DOI:10.1145/2500890

How to test the usefulness of computation for understanding and predicting continuous phenomena.

BY MARK BRAVERMAN

Computing with Real Numbers, from Archimedes to Turing and Beyond

REAL NUMBERS ARE at the center of our mathematical reasoning about the world around us. Computational problems, from computing the number π to predicting an asteroid's trajectory, all deal with real numbers. Despite the abundance of inherently continuous problems, computers are discrete, finite-precision devices. The need to reason about computing with real numbers gives rise to the kind of fascinating challenges explored here.

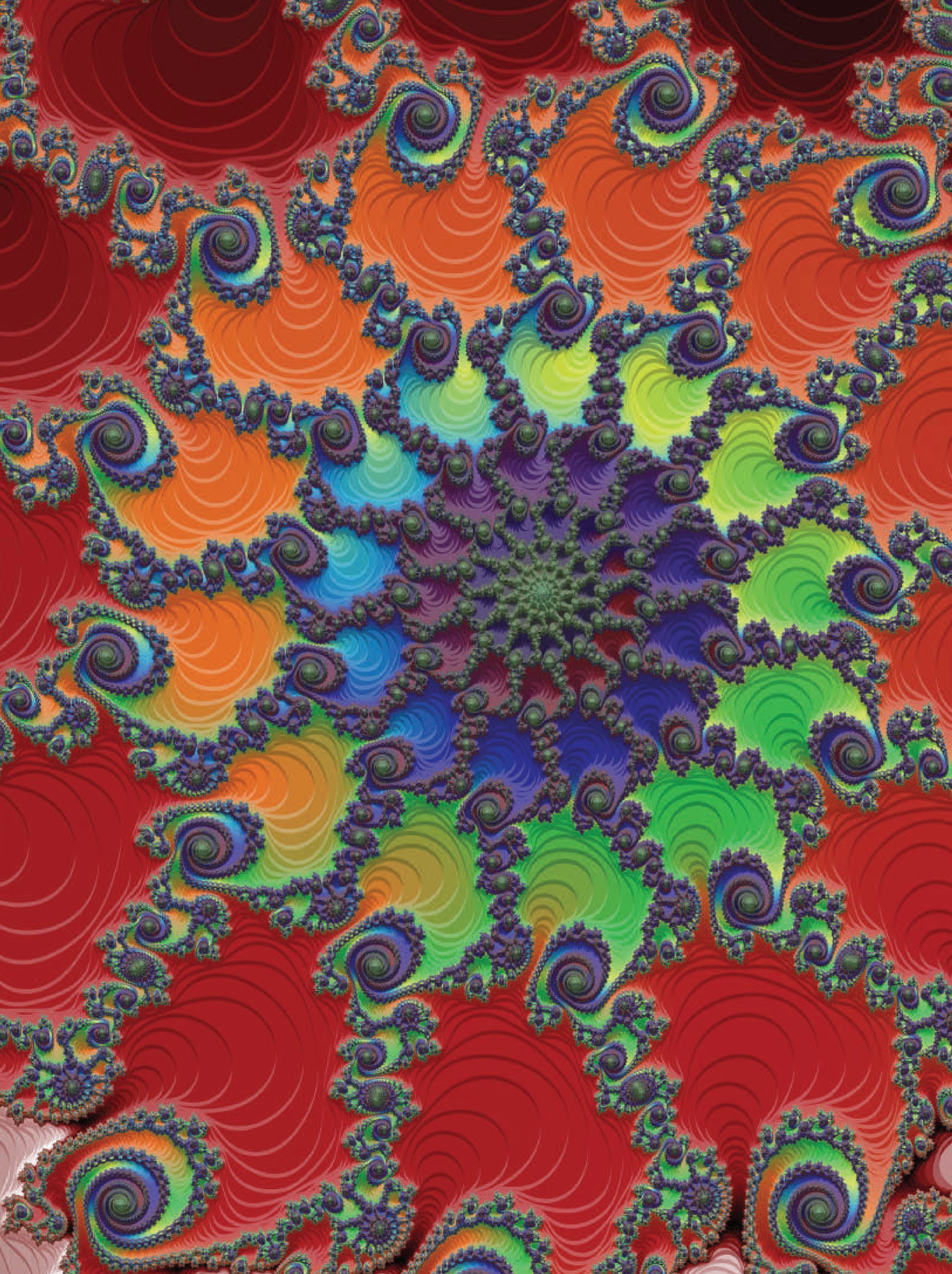
We are so immersed in numbers in our daily lives it is difficult to imagine humans once got by without them. When numbers were finally introduced in ancient times, they were used to represent specific quantities (such as commodities, land, and time); for example "four apples" is just a convenient way to rephrase "an apple and an apple and an apple and an apple"; that is, numbers had algorithmic meaning millennia before computers and algorithmic thinking became as pervasive as it is today. The natural numbers 1, 2, 3, . . . are the easiest to define and "algorithmize." Given enough time (and apples), one can easily produce a pile with any natural number of apples.

Fractions are not as easy to produce as whole natural numbers, yet the algorithm for them is fairly straightforward. To produce 2/3 of an apple, one can slice an apple into three equal parts, then take two of them. If one considers positive rational numbers, there is little divergence between the symbolic representation of the number and the algorithm one needs to "construct" this number out of apples; the number practically shouts a way to construct it. These numbers were the ones that populated the world of the ancient Greeks (in Archimedes's time) who often viewed numbers and fractions through the lens of geometry, identifying them with geometric quantities. In the geometric language, natural numbers are just integer multiples of the unit inter-

key insights

- **The study of algorithms dealing with real numbers and functions over the reals requires extending the reach of traditional computability theory but is a notable challenge for mathematicians and computer scientists.**
- **The theory of computation over the reals can be applied to the study of computational hardness of dynamical systems involving a range of natural and artificial phenomena.**
- **Predicting a system's long-term properties is easy in some cases; in others it can be as hard as trying to solve the undecidable Halting Problem.**

IMAGE BY PATRICK CATUDAL



val, and positive rational numbers are integer fractions of these intervals.

It was tempting at the time to believe that all numbers, or all possible interval lengths, are rational and can be constructed in this manner. However, it turns out not to be the case. The simplest example of an irrational number is $\sqrt{2}$. The number $\sqrt{2}$ is easily constructed geometrically (such as by using a ruler and compass) and is the length of the diagonal of a 1×1 square. On the other hand, a simple elegant proof, first given by the Pythagorean philosopher Hippasus, shows one cannot write $\sqrt{2}$ as m/n for integers m and n. Hippasus's result was controversial at the time since it violated the belief that all mathematical quantities are rational. Legend has it Hippasus was drowned for his theorem. History offers many examples of scientists and philosophers suffering for their discoveries, but we are not aware of another example of a mathematician being punished for *proving* a theorem. In modern terms, the conflict can be framed through a question: What family of algorithms suffices if one wants to compute all real numbers representing plottable lengths?; Hippasus's opponents supposed integer division would suffice.

Even more intriguing is the number π, which represents the circumference of a circle of diameter 1. Perhaps the most prominent mathematical constant, π can also be shown to be irrational, although the proof is not as simple as for $\sqrt{2}$ and was not known in ancient times. We cannot represent either $\sqrt{2}$ or π as rational fractions. We can "construct" them from mesh wire using their geometrical interpretations, but can we also figure out their numerical values? Unlike the names "4" and "2/3" the names "$\sqrt{2}$" and "π" are not helpful for actually evaluating the numbers. We can calculate approximations of these numbers; for example, for π, we can write

$$3.1415926 < \pi < 3.1415927$$

or perhaps we can follow Archimedes, who carried out the earliest theoretical calculations of π, and write

$$\frac{223}{71} < \pi < \frac{22}{7}\,.$$

One reason numbers and mathematics was developed in the first place was to understand and control natural systems.

Both representations are correct, giving us a good handle on the value of π, but both have limited precision, thus losing some information about the true value of π. All real numbers can be written using their infinite binary (or decimal) expansion, which can be used to name the number, specifying it unambiguously.

The infinite representation $\pi = 3.1415926\,.\,.\,.$ unambiguously specifies the number π. Alas, however, we cannot use it to write π in a finite amount of space. An ultimate representation would take a finite amount of space but also allow us to compute π with any desired precision. In modern language, such a representation should be algorithmic. As there are many formulas for π, there are likewise many ways to represent π this way; for example in approximately the year 1400, Madhava of Sangamagrama gave this formula

$$\pi = 4\,\Sigma_{k=0}^{\infty}\frac{(-1)^k}{2k+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \ldots \quad (1)$$

It allows us to compute π with any precision, although the convergence is painfully slow; to compute the first n digits of π one needs to take approximately 10^n terms of this sum. Many elegant formulas for computing π have been devised since, some allowing us to compute π in time polynomial in the number of digits. One such formula, known as the Bailey-Borwein-Plouffe formula, is given by

$$\pi = 4\,\Sigma_{k=0}^{\infty}$$

$$\left[\frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)\right]\ldots \quad (2)$$

The fact that the terms in formula (2) decrease exponentially fast in k causes the sum to converge rapidly. Mathematically speaking, formulas (1) and (2) are both valid "names" for π, although the latter is better because it corresponds to a much more efficient algorithm.

Can all numbers be given names in a way that allows us to compute them? No, as it turns out. Surprisingly, it took until Alan Turing's seminal paper[4] in 1936 to properly pose and answer the question. Turing had to overcome a profound philosophical difficulty. When showing a real number is computable, we would need only to describe an algorithm able to compute it with any prescribed precision, as we did with the number π. In showing that a number $x \in \mathbb{R}$ is not computable, we need to rule out all potential ways of computing x. The first major step in any such proof is formalizing what "computing" means by devising a model of computation. This is exactly what Turing did, defining his famous Turing Machine as an abstract device capable of performing all mechanical computations. Turing's paper started the modern field of computability theory. Remarkably, it happened about a dozen years before the first computers (in the modern sense of the word) were built. Turing used his new theory to define the notion of computable numbers. Not surprisingly, a modern reinterpretation of Turing's definition says a number x is computable if we can write a C++ or Java program that (given sufficient time and memory) can produce arbitrarily precise approximations of x. One of Turing's key insights was the Halting Problem $\mathcal{H}$ (which takes an integer n and outputs $\mathcal{H}(n) = 1$ if and only if $n = [P]$ is an encoding of a valid program P and P terminates) is "undecidable"; no algorithm exists that, given a program P, is capable of deciding whether or not P terminates.

The Halting Problem allows us to give a specific example of a noncomputable number. Write down the values of the function $\mathcal{H}(\bullet)$; the number

$$X_{\mathcal{H}} = 0.\mathcal{H}(1)\mathcal{H}(2)\mathcal{H}(3)\ldots = \sum_{n=1}^{\infty} 10^{-n}\mathcal{H}(n).$$

is not computable, since computing it is equivalent to solving the Halting Problem. Fortunately, "interesting" mathematical constants (such as π and e) are usually computable.

One reason numbers and mathematics was developed in the first place was to understand and control natural systems. Using the computational lens, we can rephrase this goal as reverse-engineering nature's algorithms. Which natural processes can be computationally predicted? Much of this article is motivated by this question. Note, unlike digital computers, many natural systems are best modeled using continuous quantities; that is, to discuss the computability of natural systems we have to extend the discrete model of computation to functions and sets over the real numbers.

Real Functions and Computation

We have established that a number $x \in \mathbb{R}$ is computable if there is an algorithm that can compute x with any prescribed precision. To be more concrete, we say an algorithm $\mathcal{A}_x$ computes x if on an integer input $n \in \mathbb{N}$, $\mathcal{A}_x(n)$ outputs a rational number x_n such that $|x_n - x| < 2^{-n}$. The algorithm $\mathcal{A}_x$ can be viewed as a "name" for x, in that it specifies the number x unambiguously. The infinite-digit representation of x is also its "name," albeit not compactly presented.

What does it mean for a function $f:\mathbb{R}\to\mathbb{R}$ to be computable? This question was first posed by Banach, Mazur, and colleagues in the Polish school of mathematics shortly after Turing published his original paper, starting the branch of computability theory known today as "computable analysis." Now step back to consider discrete Boolean functions. A Boolean function $F:\{0,1\}^*\to\{0,1\}^*$ is computable if there is a program $\mathcal{A}_F$ that given a binary string $s\in\{0,1\}^*$ outputs $\mathcal{A}_F(s)=F(s)$. By analogy, an algorithm computing a real-valued function f would take a real number x as an input and produce $f(x)$ as an output. Unlike the Boolean case, "input" and "output" must be qualified in this context. What we would like to say is given a name for the value $x \in \mathbb{R}$ we should be able to produce a name for the output $f(x)$; that is, we want $\mathcal{A}_f$ to be a program that, given access to arbitrarily good approximations of x, produces arbitrarily good approximations of $f(x)$.

A function $f:(a,b) \to \mathbb{R}$ is computable if there is a discrete algorithm $\mathcal{A}_f$ that, given a precision parameter n and access to arbitrarily good rational approximations of an arbitrary input $x \in (a,b)$, outputs a rational y_n such that

$$|y_n - f(x)| < 2^{-n}.$$

This definition easily extends to functions that take more than one input (such as the arithmetic operations +: $\mathbb{R}\times\mathbb{R}\to\mathbb{R}$ and ×: $\mathbb{R}\times\mathbb{R}\to\mathbb{R}$). As with numbers, all "nice" functions, including those usually found on a scientific calculator, are generally computable. Consider the simple example of the function $f(x) = x^2$ on the interval (0,1). Our algorithm for squaring numbers should be able to produce a 2^{-n} approximation of x^2 from approximations of x. And consider this simple algorithm

```
SimpleSquare(x,n)
  1. Request $q = x_{n+1}$ , a rational $2^{-(n+1)}$-
     approximation of the input $x$.
  2. Output $q^2$.
```

Note the algorithm SimpleSquare operates only with rational numbers. To see that the algorithm works, we need to show for all $x \in (0,1)$ the output q^2 satisfies $|x^2 - q^2| < 2^{-n}$. Since x is in the interval (0,1), without loss of generality we may assume q is also in (0,1). Therefore, $|x + q| < |x| + |q| < 2$, and we have

$$|x^2 - q^2| = |x+q|\cdot|x-q| \le 2\,|x-q| < 2\cdot 2^{-(n+1)} = 2^{-n}.$$

This shows the SimpleSquare algorithm indeed produces a 2^{-n} approximation of x^2. Although the function $f(x) = x^2$ is computable on the entire real line $\mathbb{R} = (-\infty,\infty)$, in this case, the algorithm would have to be modified slightly to work.

A more interesting example is the function $g(x) = e^x$, which is defined on the entire real line. Indeed, for any x, we can compute e^x with any precision by requesting a sufficiently good rational approximation q of x and then using finitely many terms from the series

$$e^q = \sum_{n=0}^{\infty}\frac{q^n}{n!} = 1 + \frac{q}{1} + \frac{q^2}{2} + \frac{q^3}{6} + \frac{q^4}{24} + \ldots$$

Throughout the discussion of the computability of these functions, we did not have to assume the input x to a computable function is itself computable. As long as the Request command gives us good approximations of x we do not care whether these approximations were obtained algorithmically. Now, if the number x is itself computable, then the Request commands may be replaced with a subroutine that computes x with the desired precision. Thus if f is computable on (a,b) and $x \in (a,b)$ is a computable number, then $f(x)$ is also a computable number. In particular, since e^x is a computable function and π is a computable number, e^{π} and $e^{e^{\pi}}$ are computable numbers as well.

One technical limitation of the Request-based definition is we can never be sure about the exact value of the input x; for example, we are unable to decide whether the real-valued input x is equal to, say, 42 or not. Thus the function

$$f(x) = \begin{cases} 1 & \textit{if } x = 42.0 \\ 0 & \textit{otherwise} \end{cases}$$

is not computable. The reason for this inability is while we can Request x with any desired precision, no finite-precision approximation of x will ever allow us to be sure x is exactly 42.0. If we take the requested precision high enough, we may learn $x = 42.0 \pm 10^{-1{,}000}$. This still does not mean $f(x) = 1$, as it is possible the first disagreement between x and 42.0 occurs after the 1,000th decimal place (such as if $x = 42 + 2^{-2000} \neq 42.0$). The Request function can be viewed as a physical experiment measuring x. By measuring x we can narrow down its value to a very small interval but can never be sure of its exact value. We refer to this difficulty as the "impossibility of exact computation." More generally, similar reasoning shows only continuous functions may be computable in this model.

On the other hand, and not too surprisingly, all functions that can be computed on a calculator are computable under this definition of function computability. But, as with Turing's original work, the main goal of having a model of computation dealing with real functions is to tell us what *cannot* be done, or proving fundamental bounds on our ability to computationally tackle continuous systems. First we need to explore the theory of computation over the reals a little further.

Computability of subsets in $\mathbb{R}^d$. In addition to numbers and functions we are also interested in computing sets of real numbers; see Figure 1 for example subsets of $\mathbb{R}^2$. Sets we might be interested in include simple geometric shapes (such as a circle), graphs of functions, and the more complicated ones, like the Koch snowflake and the Mandelbrot set. When is a set $\mathcal{S}$ in, say, the plane $\mathbb{R}^2$, computable? It is tempting to mimic the discrete case

Figure 1. Examples of subsets of $\mathbb{R}^2$: the graph of $y = c^x$, the Koch snowflake, and the Mandelbrot set.

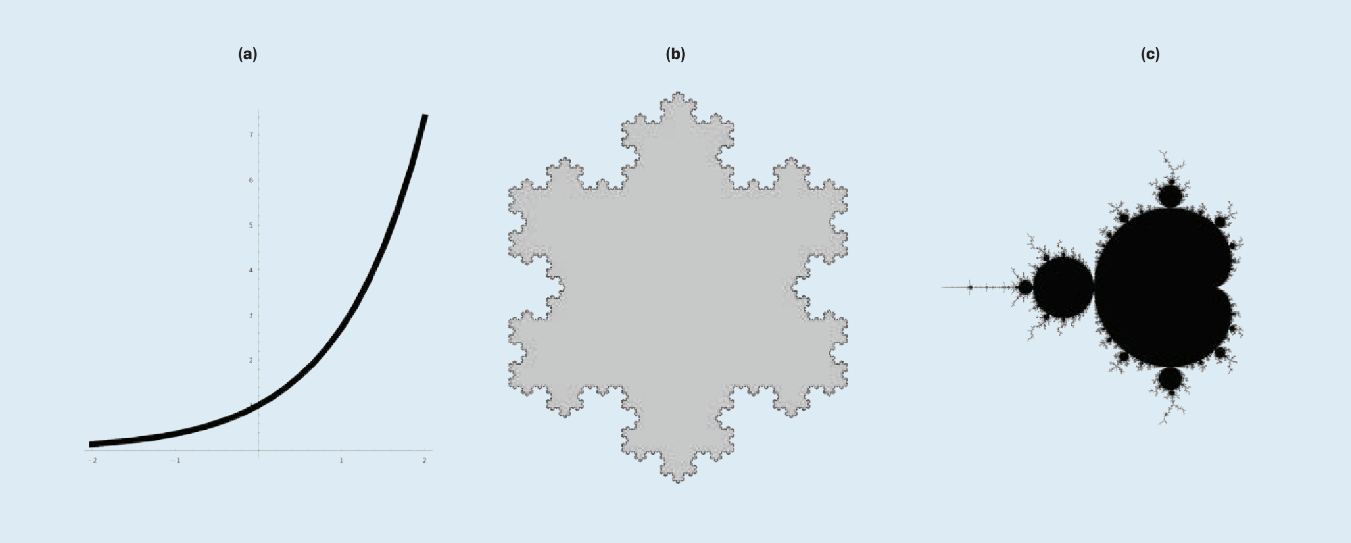
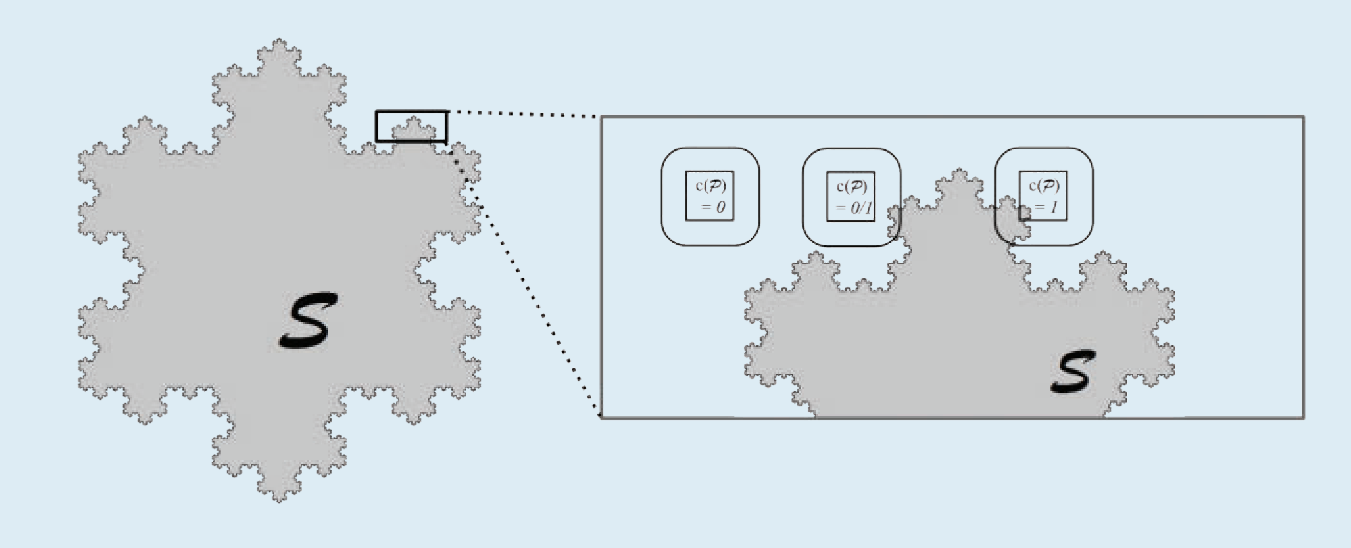


Figure 2. The process of deciding the color $c(\mathcal{P})$ for an individual pixel $\mathcal{P}$.



and say $\mathcal{S}$ is computable whenever the membership function

$$\chi_{\mathcal{S}}(x) = \begin{cases} 1 & \textit{if } x \in \mathcal{S} \\ 0 & \textit{otherwise} \end{cases}$$

is decidable. However, this definition involves a serious technical problem, the same impossibility-of-exact-computation problem present in the $x \overset{?}{=} 42.0$ example. If x happens to lie on the boundary of $\mathcal{S}$, we will never be able to decide whether $x \in \mathcal{S}$ through a finite number of Request queries. To address this problem we proceed by analogy with the computability of numbers. Rather than try to compute $\mathcal{S}$, we should try to approximate it with any prescribed precision 2^{-n}.

What does it mean to "approximate" a set? There are many ways to address this question, and many "reasonable" definitions are equivalent. First take a "graphical" approach. Consider the process of producing a picture of the set $\mathcal{S}$. The process would consist of many individual decisions concerning whether to color a pixel black or white; for example, we need to make 600 × 600 such decisions per square inch if $\mathcal{S}$ is being printed on a 600dpi printer. Thus a discussion about drawing $\mathcal{S}$ with precision 2^{-n} can be reduced to a discussion about deciding on the color of individual pixels, bringing us back to the more familiar realm of 0/1-output algorithms.

To be concrete, let $\mathcal{S}$ be a subset of the plane $\mathbb{R}^2$ and let

$$P = [x - 2^{-n-1}, x + 2^{-n-1}] \times [y - 2^{-n-1}, y + 2^{-n-1}]$$

be a square pixel of dimensions $2^{-n} \times 2^{-n}$. The coloring of pixels should satisfy the following conditions:

1. If $\mathcal{P}$ intersects with $\mathcal{S}$, we must color it `black`;
2. If $\mathcal{P}$ is 2^{-n}-far from $\mathcal{S}$ we must color it `white`. It is natural to ask why not simply require $\mathcal{P}$ to be colored white if it does not intersect $\mathcal{S}$. The reason is, if we did, we would again run into the impossibility of exact computation. Thus we allow for a gray area in the image; and
3. If $\mathcal{P}$ does not intersect $\mathcal{S}$ but is 2^{-n} close to it, we do not care whether it is colored black or white.

This gray area allows us to avoid the impossibility of exact computation problem while still producing a faithful image of $\mathcal{S}$ (see Figure 2).

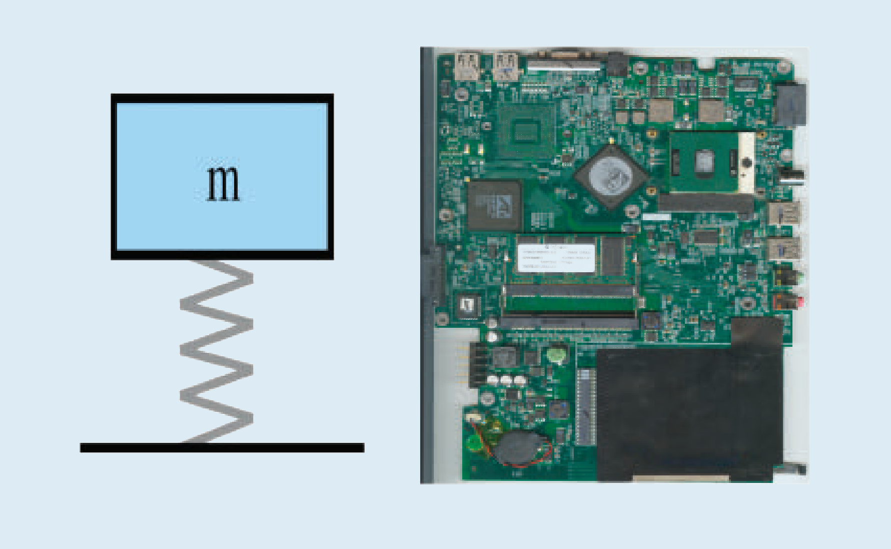


Figure 3. One of the "easiest" (left) and one of the "hardest" (right) dynamical systems.

The definition of set computability presented here may seem ad hoc, appearing to be tied in to the way we choose to render the set $\mathcal{S}$. Somewhat surprisingly, the definition is robust—equivalent to the "mathematical" definition of $\mathcal{S}$ being "approximable" in the Hausdorff metric, a natural metric one can define on subsets of $\mathbb{R}^d$. The definition is also equivalent to the distance function $d_{\mathcal{S}}(x)$ that measures how far the point x is from the set $\mathcal{S}$ being a computable function.

Just as "nice" calculator functions are computable, "nice" sets are likewise computable; for example, a circle $C(o,r)$ with center $o = (x, y)$ and radius r is computable if and only if the numbers x, y, and r are computable. Graphs of computable functions are computable sets. Thus the graph of the function $x \mapsto e^x$ is computable. For a more interesting example, consider the Koch snowflake K (see Figure 1b). This fractal set has dimension $\log_3 4$ and lacks a nice analytic description. However, it is computable and is, in fact, the limit set of a sequence of finite snowflakes. Each finite snowflake K_n is just a polygon and thus easily computable. To approximate K we need to draw only the appropriate finite snowflake K_n, exactly how the Koch snowflake is drawn in practice, as in Figure 1b.

Now that we have the notion of computable real functions and real sets we can turn to formulating the computational hardness of natural and artificial systems, as studied in the area of dynamical systems.

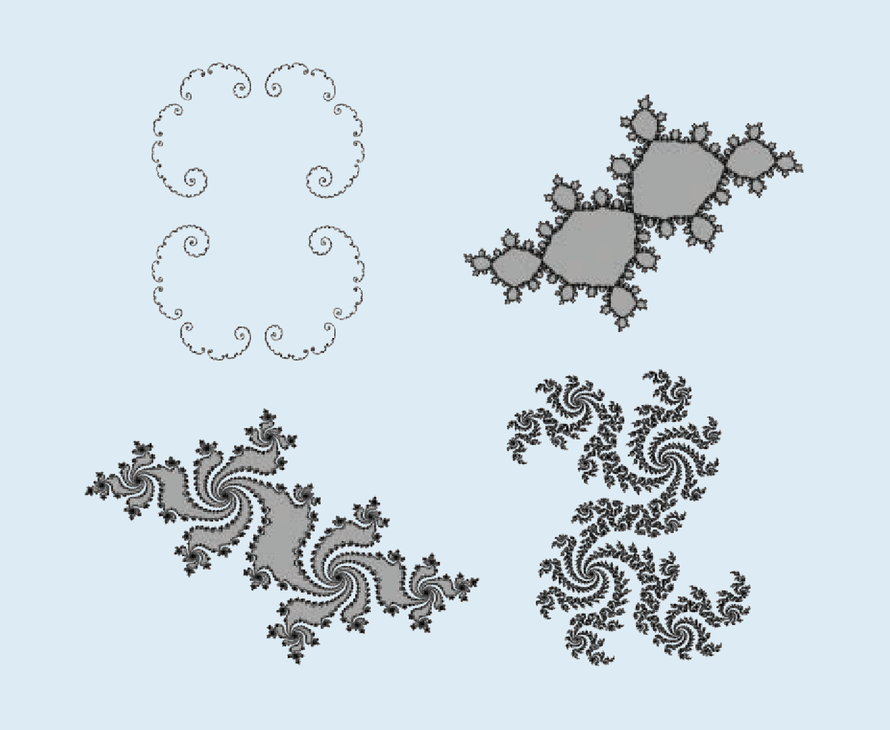
Computing Nature: Dynamical Systems

At a high level, the area of dynamical systems studies all systems that evolve over time. Systems ranging from an electron in a hydrogen atom to the movement of galaxies to brain activity can thus be framed in the context of dynamical systems. A dynamical system consists of a set of states $\mathcal{X}$ and a set of evolution rules $\mathcal{R}$. Evolution of the system occurs over time. The state of the system at time t is denoted by $X_t \in \mathcal{X}$. The time may move either discretely or continuously. If time is discrete, the evolution of the system is given by the sequence $X_1, X_2, X_3, \ldots$, and the rule $\mathcal{R}$ specifies the dependence of X_{t+1} on X_t. If time is continuous, the evolution $\mathcal{R}$ of the system $X_t = X(t)$ is usually given by a set of differential equations specifying the rates of change in $X(t)$ depending on its current state.

As an example, consider a simple harmonic oscillator. A mass in Figure 3 is attached to a spring and is moving in a periodical fashion. Assuming no friction, the system X_t^{osc} evolves in continuous time, and its state at any time is described fully by two numbers: the location of the mass on the line and its velocity. Thus the state X_t^{osc} can be represented by the vector $X_t^{osc} = (\ell(t), \upsilon(t))$, where $\ell(t)$ represents the location of the mass, and $\upsilon(t)$ represents its velocity. The evolution rule of the system is given in this case by high-school physics

$$\begin{cases} \ell'(t) = \upsilon(t) \\ \nu'(t) = -\alpha \cdot \ell(t) \end{cases} \tag{3}$$

Figure 4. Example Julia sets.



where α is a parameter that depends on the spring and the mass of the weight. X^{osc} is a very simple system, and we can answer pretty much any question about it; for example, we can solve this system of equations to obtain the full description of the system's behavior

$$X_t^{osc} = (A\sin(\sqrt{\alpha}\ \cdot t + \phi),\\ A\sqrt{\alpha}\ \cos(\sqrt{\alpha}\ \cdot t + \phi)) \qquad (4)$$

where the parameters A and ϕ depend on the initial condition $X_0^{osc} = (\ell(0), \upsilon(0))$ of the system at time 0; that is, if we know the exact state of the system at time 0, we can compute the state of the system at any time in the future. It is not just the prediction problem that is easy for this system. Using the analytic solution (4) we can answer almost any question imaginable about it; for example, we can describe the set of all possible states the system being released from state X_0^{osc} will reach.

At the other extreme, predicting some dynamical systems in the long run is incredibly difficult. One important set of examples of "hard" dynamical systems comes from computer science itself. Consider the Turing Machine or its modern counterpart, a RAM computer with unlimited RAM, as a dynamical system. The state-space X^{comp} is the (infinite) state space of the computer. The system X^{comp} evolves in discrete time, with X_t^{comp} representing the state of the computer at step t of the execution. The evolution rule $\mathcal{R}$ is the rule according to which the computation proceeds; that is, $\mathcal{R}(X)$ is the state of the computer in the next time step if its current state is X. Call this system the "computer dynamical system."

The computer dynamical system is easy to simulate computationally; all we must do is simulate the execution of the computation. On the other hand, unlike the oscillator example, answering long-term questions about the system is difficult; for example, given an initial condition X_0^{comp}, there is no computational procedure that can tell us whether the system will ever reach a given state Y; determining whether the system will reach a terminating state is equivalent to solving the Halting Problem for programs, that, as discussed, is computationally undecidable. It can likewise be shown that almost any nontrivial question about the long-term behavior of X^{comp} is noncomputable in the worst case.

These examples exist at the two extremes of computational hardness: X^{osc} is a linear dynamical system and fully computable. X^{comp} is (trivially) computationally universal, capable of simulating a Turing Machine, and reasoning about its long-term properties is as computationally hard as solving the Halting Problem. What kinds of systems are prevalent in nature? For example, can an N-body system evolving according to the laws of Newtonian gravity simulate a computer, in which case predicting it would be as difficult as solving he Halting Problem? Is predicting it computationally easy? Or something in-between?

We do not know the answers for most natural systems. Here, we consider an interesting in-between example that is one of the best-studied dynamical systems. We consider dynamics on the set of complex numbers $\mathbb{C}$, or a number of the form $a + bi$, evolving by a quadratic polynomial. For the rest of this discussion, the set of complex numbers is identified with the 2D complex plane, with the number $a + bi$ corresponding to the point (a,b), allowing us to visualize subsets of $\mathbb{C}$ nicely. Let $c \in \mathbb{C}$ be any complex number. Denote

$$p_c(z) := z^2 + c.$$

Define the discrete-time dynamical system X_t^c by

$$X_{t+1}^c = p_c(X_t^c).$$

The polynomial $p_c(z)$ is arguably the simplest nonlinear transformation one can apply to complex numbers, yet this system is already complicated enough to exhibit a variety of interesting and complicated behaviors. In particular, it is impossible to give a closed-form expression for X_t^c in terms of X_0^c as we did with the oscillator example. Dynamical systems of the form X_t^c are studied by a branch of the theory of dynamical systems known as complex, or holomorphic, dynamics. Within mathematics, one of the main reasons for studying these systems is the rich variety of behaviors they exhibit allows us to learn about the behavior of more general (and much more difficult to study) dynamical systems.

Outside mathematics, complex dynamics is best known for the fascinating fractal images it generates. These images, known as Julia sets (see Figure 4), depict a global picture relevant to the long-term behavior of the system X^c. More specifically, J_c is the subset of initial conditions in the complex plane on which the long-term behavior of X^c is unstable. To understand what this means, we need to take a slightly closer look at the system X^c. Consider an initial point $X_0^c = x_0$, as mapped by $p_c(z) = z^2 + c$

$$x_0 \mapsto x_0^2 + c \mapsto (x_0^2 + c)^2 + c \mapsto \ldots$$

If we start with an x_0 with a very high absolute value, say, $|x_0| > |c| + 2$, then the absolute value of $p_c(x_0) = x_0^2 + c$ will be larger than $|x_0|$, and $|p_c(p_c(x_0))|$ will be larger still and the state of the system will diverge to ∞. The set of starting points for which the system does not diverge to ∞ is called the filled Julia set of the system X^c and is denoted by K_c. The Julia set[a] J_c is the boundary ∂K_c of the filled Julia set.

The Julia set J_c is the set of points around which the system's long-term behavior is highly unstable. Around each point z in J_c are points (just outside K_c) with trajectories that ultimately escape to ∞. There are also points (just inside K_c) with trajectories that always stay within the bounded region K_c. The Julia set itself is invariant under the mapping $z \mapsto z^2 + c$. This means trajectories of points that start in J_c stay in J_c.

The Julia set J_c provides a description of the long-term properties of the system X_c. Julia sets are therefore valuable for studying and understanding these systems. In addition, as in Figure 4, Julia sets give rise to an amazing variety of beautiful fractal images. Popularized by Benoit Mandelbrot and others, Julia sets are today some of the most drawn objects in mathematics, and hundreds of programs for generating them can be found online.

Formally speaking, the problem of computing the Julia set J_c is one of evaluating the function $\mathcal{J}: c \mapsto J_c$. $\mathcal{J}$ is a set-valued function whose computability combines features of function and set computability discussed earlier. The complex input $c \in \mathbb{C}$ is provided to the program through the request command, and the program computing $\mathcal{J}(c) = J_c$ is required to output an image of the Julia set J_c within a prescribed precision 2^{-n}. $\mathcal{J}$ is a fascinating function worth considering in its own right; for example, the famous Mandelbrot $\mathcal{M}$ (see Figure 1c) can be defined as the set of parameters c for which $\mathcal{J}(c)$ is a connected set. It turns out the function $\mathcal{J}(c)$ is discontinuous, at least for some values of c (such as for $c = \frac{1}{4} + 0 \cdot i$). This means for every ε there is a parameter $c' \in C$ that is ε-close to $\frac{1}{4}$ but for which the Julia set $J_{c'}$ is very far from $J_{\frac{1}{4}}$. Due to the impossibility of exact computation, this discontinuity implies there is no hope of producing a single program $P_{\mathcal{J}}$ that computes $\mathcal{J}(c)$ for all parameters c; on inputs close to $c = 1/4$, such a program would need to use `request` commands to determine whether c is in fact equal to 1/4 or merely very close to it, which is impossible to do.

As with numbers, all "nice" functions, including those usually found on a scientific calculator, are generally computable.

If there is no hope of computing $\mathcal{J}$ by one program, we can at least hope that for each c we can construct a special program P_{J_c} that evaluates J_c. Such a program would still need access to the parameter c through the `request` command, since only a finite amount of information about the continuous parameter $c \in \mathbb{C}$ can be "hardwired" into the program P_{J_c}. Nonetheless, by requiring P_{J_c} to work correctly on only one input c we manage to sidestep the impossibility of exact computation problem.

Most of the hundreds of online programs that draw Julia sets usually draw the complement $(\overline{K}_c) = \mathbb{C} \setminus K_c$ of the filled Julia set, or the set of points whose trajectories escape to ∞. It turns out that from the computability viewpoint, the computability of $\overline{K}_c$ is equivalent to the computability of J_c, allowing us to discuss these problems interchangeably.[b] The vast majority of the programs follows the same basic logic; to check whether a point z_0 belongs to $\overline{K}_c$, we need to verify whether its trajectory $z_0, p_c(z_0), p_c(p_c(z_0)), \ldots$ escapes to ∞. We pick a (large) number M of iterations. If z_0 does not escape within M steps, we assume it does not escape. To put this approach in a form consistent with the definition of computability of sets in $\mathbb{R}^2$, let $\mathcal{P}$ be a pixel of size 2^{-n}. The naïve decision procedure for determining whether the pix-

[a] "Julia" is not a first name in this context but rather the last name of the French mathematician Gaston Julia (1893–1978).

[b] As mentioned here, the computability of $(\overline{K}_c)$ and J_c is not equivalent to the computability of the filled Julia set K_c.

el $\mathcal{P}$ overlaps with $\overline{K}_c$ or is 2^{-n}-far from it thus looks roughly like this:

```
A naïve heuristic for drawing $\overline{K}_c$:
  1. Let $z_0$ be the center of the pixel $\mathcal{P}$;
  2. Let $M = M(n)$ be the number of iterations;
  3. for $i = 0$ to $M$
      3.1.  Set $z_{i+1} \leftarrow z_i^2 + c$;
      3.2.  if $|z_{i+1}| > |c|+2$, return "$\mathcal{P}$ intersects $\overline{K}_c$";
  4. if $|z_{M+1}| \leq |c|+2$ return "$\mathcal{P}$ is $2^{-n}$ far from $\overline{K}_c$"
```

If the pixel $\mathcal{P}$ is evaluated to intersect with $\overline{K}_c$, it is colored `black`; otherwise it is left `white`. However, there are multiple problems with these heuristics that make the rendered pictures imprecise and sometimes just wrong. The first is we take one point z_0 to be "representative" of the entire pixel $\mathcal{P}$. This approach means even if $\mathcal{P}$ intersects $\overline{K}_c$ or some point $w \in \mathcal{P}$ has its trajectory escape to ∞, we might miss it if the trajectory of z_0 does not escape to ∞. This problem highlights one of the difficulties encountered when developing algorithms for continuous objects. We need to find the answer not just for one point z_0 but for the uncountable set of points located within the pixel $\mathcal{P}$. However, there are computational ways to remedy this problem. Instead of tracing just one point z_0 we can trace the entire geometric shape $\mathcal{P}, p_c(\mathcal{P}), p_c(p_c(\mathcal{P})), \ldots$ and see whether any part of the M^{th} iteration of $\mathcal{P}$ escapes to ∞. This may increase the running time of the algorithm considerably. Nonetheless, we can exploit the peculiarities of complex analytic functions to make the approach work. John Milnor's "Distance Estimator" algorithm does exactly that, at least for a large set of "good" parameters c.

The heuristic also involves a much deeper problem—choosing the parameter $M(n)$. In the thousands of Java applets available online, selection of the number of iterations M is usually left to the user. Suppose we wanted to automate this task or make the program evaluate a "large enough" M such that M iterations are sufficient (see Figure 5 for the effect of selecting an M that is too small); that is, we want to find a parameter M such that if the M^{th} iteration z_M of z_0 did not escape to ∞, then we can be sure no further iterations will escape to ∞, and it is safe to assert z_0 lies within the filled Julia set K_c. Computing such an M is equivalent to establishing termination of the loop

```
Loop($z_0$): $i \leftarrow 0$
    while $|z_i| \leq |c|+2$
        $z_{i+1} \leftarrow z_i^2 + c$
        $i \leftarrow i + 1$
```

Systems ranging from an electron in a hydrogen atom to the movement of galaxies to brain activity can be framed in the context of dynamical systems.

In general, the termination of loops, as with the Halting Problem $\mathcal{H}$ is a computationally undecidable problem. If the loop terminates, we are sure z_0 has escaped. But if the loop keeps running there is no general way of knowing it will not terminate later. There is thus no simple solution to figuring out the appropriate M; the only way to know the loop does not terminate is to understand the system X^c and the set $\overline{K}_c$ well enough. Turning the naïve heuristic into an algorithm necessarily involves a deep understanding of the underlying dynamical system. As with the systems X^{osc} and X^{comp} discussed earlier, it all boils down to understanding the underlying system. Fortunately, complex dynamicists have developed a rich theory around this system since its introduction around 1917 by the French mathematicians Gaston Julia and Pierre Fatou. This knowledge is enough to give precise answers to most questions concerning the computability of K_c, $\overline{K}_c$, and J_c. One can formalize the naïve heuristic discussed here and show that (with slight modifications) it works for the vast majority of values of c.

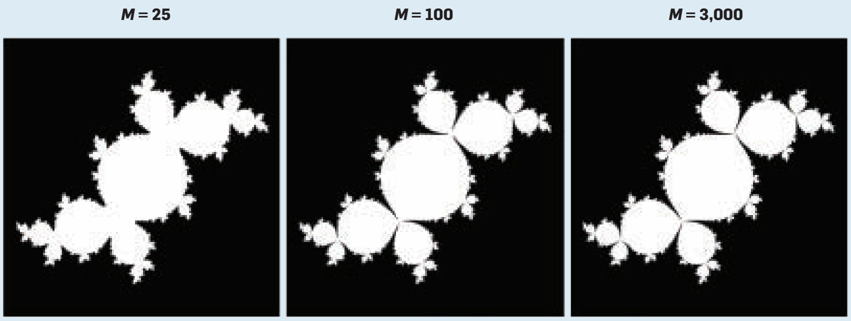
However, it also turns out there are also noncomputable Julia sets:

THEOREM 1.[2] *There exist parameters c such that the Julia set J_c is not computable. Moreover, there are such parameters $c \in \mathbb{C}$ that can be produced algorithmically.*

That is, one can produce a parameter c such that drawing the picture of J_c is as computationally hard as solving the Halting Problem. What does such a Julia set look like? All sets in Figure 4 are computable, as they were produced algorithmically for inclusion here. Unfortunately, Theorem 1 means we will most likely never know what these noncomputable Julia sets look like.

The negative result is delicate; in a surprising twist, it does not extend to the filled Julia set K_c:

Figure 5. Example outcomes of the heuristic algorithm with $c \approx -0.126 + 0.67i$ and $M = 25$, $M = 100$, and $M = 3,000$ iteration. Note with the lower values of M the fjords do not reach all the way to the center since the points close to the center do not have time to "escape" in M iterations. The difficulty selecting a "large enough" M is a crucial obstacle in computing the exterior of the filled Julia set K_c.



THEOREM 2.[2] *For all parameters c,* the filled Julia set K_c:

THEOREM 2.[2] *For all parameters c, the filled Julia set K_c is computable.*

Is the Universe a Computer?

We have explored three examples of dynamical systems: The first, harmonic oscillator X^{osc}, is simple; its behavior can be calculated in closed form, and we can answer pretty much any question about the long-term behavior of the harmonic oscillator computationally. The second, computer system X^{comp}, is at the opposite extreme; predicting it is computationally hard, and it is (relatively) easy to show it is computationally hard through a reduction to the Halting Problem. The third, complex dynamics, requires an involved approach. For some (in fact, most) parameters c, the long-term behavior of the system X^c is easy in almost any sense imaginable. Showing there are parameters c for which no amount of computational power suffices to compute the Julia set J_c required a full understanding of the underlying dynamical system developed over nearly a century.

Our experience with the computability of Julia sets, as well as the relative success of the field of automated verification at solving undecidable problems in practice,[5] indicates there is likely to be a gap between computability in dynamics in the worst case and in the typical case. This gap means it is possible that while questions surrounding many natural systems (such as the *N*-body problem and protein assembly) are provably noncomputable in the worst case, a typical case in practice is tractable.

A related interesting possibility is that noncomputable structures in many systems are too delicate to survive the random noise present in all natural systems. Noise is generally viewed as a "prediction destroying" force, in that making predictions in the presence of noise is computationally more difficult. On the other hand, if we are interested in predicting the statistical distribution of possible future states, then noise may actually make the task easier. It is likely there are natural systems that (if implemented with no noise) would be computationally impossible to predict but where the presence of noise makes statistical predictions about the system computationally tractable.

Another lesson from the study of the computational properties of Julia sets is that mapping out which of the Julia sets J_c are and which are not computable requires a nuanced understanding of the underlying dynamical system. It is likely this is the case with other natural dynamical systems; the prerequisite to understanding its computational properties would be understanding its other properties. Indeed, understanding the role (non)computability and computational universality play in natural dynamical systems probably requires significant advances in both real computation and dynamical systems. The role of computational universality—the ability of natural systems to simulate generic computation—in nature is therefore likely to remain one of the most tantalizing open problems in natural philosophy for some time to come.

Bibliographic Notes

The following references include extensive bibliographies for readers interested in computation over the reals; computability of real numbers was first discussed in Turing's seminal paper,[4] which also started the field of computability. There are two main modern visions on computability over the real numbers: computable analysis and the Blum-Shub-Smale (BSS) framework. My presentation here is fully based on the framework of computable analysis, as presented in-depth by Weihrauch.[6] The BSS framework is more closely related to algebraic geometry and presented by Blum et al.[1] I focused on computable analysis, as it appears more appropriate for the study of the computational hardness of natural problems over the reals. The results on the computability and complexity of Julia sets was presented by Braverman and Yampolsky.[2] Computational universality of dynamical systems is discussed in several sources, including Moore[3] and Wolfram,[7] but many basic questions remain open.

Acknowledgments

Work on this article has been supported in part by an Alfred P. Sloan Fellowship, National Science Foundation awards CCF-0832797 and CCF-1149888, and a Turing Centenary Fellowship from the John Templeton Foundation.

References

1. Blum, L., Cucker, F., Shub, M., and Smale, S. *Complexity and Real Computation.* Springer-Verlag, New York, 1998.
2. Braverman, M. and Yampolsky, M. *Computability of Julia Sets.* Springer Verlag, Berlin Heidelberg, 2009.
3. Moore, C. Unpredictability and undecidability in dynamical systems. *Physical Review Letters 64*, 20 (May 1990), 2354–2357.
4. Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society 42*, 2 (Nov. 12, 1936), 230–265.
5. Vardi, M. Solving the unsolvable. *Commun. ACM 54*, 7 (July 2011), 5.
6. Weihrauch, K. *Computable Analysis.* Springer-Verlag, Berlin, 2000.
7. Wolfram, S. *A New Kind of Science.* Wolfram Media, Champaign, IL, 2002.

Mark Braverman (mbraverm@cs.princeton.edu) is an assistant professor in the Department of Computer Science at Princeton University, Princeton, NJ.

DOI:10.1145/2494558

Exploring autonomous systems and the agents that control them.

BY MICHAEL FISHER, LOUISE DENNIS, AND MATT WEBSTER

Verifying Autonomous Systems

IN THIS ARTICLE we consider the question: How should autonomous systems be analyzed? In particular, we describe how the confluence of developments in two areas—autonomous systems architectures and formal verification for rational agents—can provide the basis for the formal verification of autonomous systems behaviors.

We discuss an approach to this question that involves:

1. Modeling the behavior and describing the interface (input/output) to an agent in charge of making decisions within the system;

2. Model checking the agent within an unrestricted environment representing the "real world" and those parts of the systems external to the agent, in order to establish some property, φ;

3. Utilizing theorems or analysis of the environment, in the form of logical statements (where necessary), to derive properties of the larger system; and

4. If the agent is refined, modify (1), but if environmental properties are clarified, modify (3).

Autonomous systems are now being deployed in safety, mission, or business critical scenarios, which means a thorough analysis of the choices the core software might make becomes crucial. But, should the analysis and verification of autonomous software be treated any differently than traditional software used in critical situations? Or is there something new going on here?

Autonomous systems are systems that decide for themselves what to do and when to do it. Such systems might seem futuristic, but they are closer than we might think. Modern household, business, and industrial systems increasingly incorporate autonomy. There are many examples, all varying in the degree of autonomy used, from almost pure human control to fully autonomous activities with minimal human interaction. Application areas are broad, ranging from healthcare monitoring to autonomous vehicles.

But what are the reasons for this increase in autonomy? Typically, autonomy is used in systems that:

1. must be deployed in *remote* environments where direct human control is infeasible;

2. must be deployed in *hostile* environments where it is dangerous for humans to be nearby, and so difficult for humans to assess the possibilities;

3. involve activity that is too lengthy

» key insights

- **Autonomous systems are systems that decide for themselves what to do. They are currently making large impacts in a variety of applications, including driverless cars, unmanned aircraft, robotics, and remote monitoring.**
- **A key issue for autonomous systems is determining their safety and trustworthiness: How can we be sure the autonomous systems will be safe and reliable? Methodologies to enable certification of such systems are urgently needed.**
- **The choices made by agent-based autonomous systems can be formally verified to provide evidence for certification. Sample applications include search and rescue robots, satellite systems, and unmanned aircraft.**

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

5.06
MAX
145.25
TUNE
145.25
CALL
4.79
2.08
8.88
3.20
2.20
5.09
0 1 2 3 4 5 6 7 8
x1000 r/min
1/1
H
R
C
SHIFT
0 20 40 60 80 100 120 140 160 180 200 220 240
km/h
1°C
6:39
ODO
117507 km

and/or repetitive to be conducted successfully by humans; or

4. need to react *much* more quickly than humans can.

However, it may actually be cheaper to use an autonomous system. After all, humans need training, monitoring, safe environments, medical support, legal oversight, and so on.

Examples. There are many autonomous systems that have either been deployed or are in development. We clearly cannot survey them all so only provide a broad selection noted here.

Robotics and robot swarms. As we move from the restricted manufacturing robots seen in factories toward robots in the home and robot helpers for the elderly, so the level of autonomy required increases.

Human-robot teamwork. Once we move beyond just directing robots to undertake tasks, they become robotic companions. In the not too distant future, we can foresee teams of humans and robots working together but making their decisions individually and autonomously.

Pervasive systems, intelligent monitoring, among others. As sensors and communications are deployed throughout our physical environment and in many buildings, so the opportunity to bring together a multiplicity of sensor inputs has led to autonomous decision-making components that can, for instance, raise alarms and even take decisive action.

Autonomous road vehicles. Also known as "driverless cars," autonomous road vehicles have progressed beyond initial technology assessments (for example, DARPA Grand Challenges) to the first government-licensed autonomous cars.[35]

As we can see from these examples, autonomous systems are increasingly being used in safety/mission/business critical areas. Consequently, they need rigorous analysis. One traditional way to achieve this, at least in non-autonomous systems, is to use formal verification. While applying formal verification techniques to autonomous systems can be difficult, developments in autonomous system architectures are opening up new possibilities.

Autonomous Systems Architectures. Many autonomous systems, ranging over unmanned aircraft, robotics, satellites and even purely software applications, have a similar internal structure, namely layered architectures[23] as summarized in Figure 1. Although purely connectionist/sub-symbolic architectures remain prevalent in some areas, such as robotics,[10] there is a broad realization that separating out the important/difficult choices into an identifiable entity can be very useful for development, debugging, and analysis. While such layered architectures have been investigated for many years[3,23] they appear increasingly common in autonomous systems.

Notice how the system in Figure 1 is split into real-world interactions, continuous control systems, and discontinuous control. For example, a typical unmanned aircraft system might incorporate an aircraft, a set of control systems encapsulated within an autopilot, and a high-level decision-maker that makes the key choices. Once a destination has been decided, the continuous dynamic control, in the form of the autopilot, will be able to fly there. The intelligence only becomes involved if either an alternative destination is chosen, or if some fault or unexpected situation occurs.

But what is this intelligent decision-making component? In the past this has often been conflated with the dynamic control elements, the whole being described using a large, possibly hierarchical, control system, genetic algorithm, or neural network. However, architectures are increasingly being deployed in which the autonomous, intelligent decision-making component is captured as an "agent."

Agents as Autonomous Decision Makers. The development and analysis of autonomous systems, particularly autonomous software, is different to traditional software in one crucial aspect. In designing, analyzing, or monitoring "normal" software we typically care about

- *what* the software does, and
- *when* the software does it.

Since autonomous software has to make its own decisions, it is often vital to know not only what the software does and when it does it, but also

- *why* the software chooses to do it.

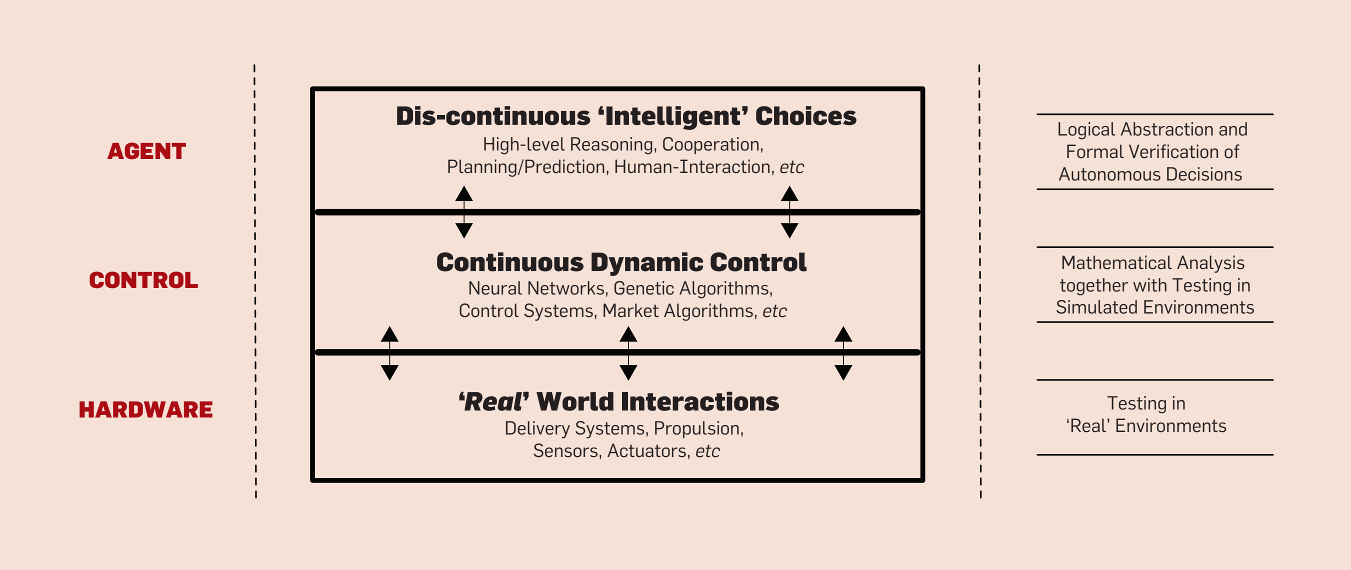
This requirement—describing why a system chooses one course of action over another—provides new entities to be analyzed. But how shall we describe these new entities? A very useful abstraction for capturing such autonomous behavior within complex, dynamic systems turns out to be the concept of an agent.[22] Since the agent concept came into prominence in the 1980s, there has been vast development within both academia and industry.[4,14,20,34] It has become clear this agent metaphor is very useful for capturing many practical situations involving complex systems comprising flexible, autonomous, and distributed components. In essence, agents must fundamentally be capable of flexible autonomous action.[38]

However, it turns out the "agent" concept on its own is still not enough! Systems controlled by neural networks, genetic algorithms, and complex control systems, among others, can all act autonomously and thus be called agents, yet the reasons for their actions are often quite opaque. Because of this, such systems are very difficult to develop, control, and analyze.

So, the concept of a rational agent has become more popular. Again, there are many variations[9,33,39] but we consider this to be an agent that *has explicit reasons for making the choices it does, and should be able to explain these if necessary*.

Therefore, a rational agent can be examined to discover why it chose a certain course of action. Such agents are often programmed and analyzed by describing their motivations (for example, "goals"), information (for example, "knowledge"), and how these change over time (as we will discuss later). Rational agents can adapt their autonomous behavior to cater for the dynamic aspects of their environment, their requirements and their knowledge. Typically, they can also modify their decision-making following interactions with their environment. The predominant form of rational agent architecture is that provided through the Beliefs, Desires, and Intentions (BDI) approach.[32,33] Here, the beliefs represent the agent's (probably incomplete, possibly incorrect) information about itself, other agents, and its environment, desires represent the agent's long-term goals, and intentions represent the goals the agent is actively pursuing.

Figure 1. Typical hybrid autonomous system architecture—with suitable analysis techniques noted.


AGENT
CONTROL
HARDWARE
Dis-continuous 'Intelligent' Choices
High-level Reasoning, Cooperation, Planning/Prediction, Human-Interaction, *etc*
Continuous Dynamic Control
Neural Networks, Genetic Algorithms, Control Systems, Market Algorithms, *etc*
'Real' World Interactions
Delivery Systems, Propulsion, Sensors, Actuators, *etc*
Logical Abstraction and Formal Verification of Autonomous Decisions
Mathematical Analysis together with Testing in Simulated Environments
Testing in 'Real' Environments

Before we consider how we might verify autonomous systems and, in particular the rational agent that makes the core decisions, we first recap formal verification. In particular we will motivate and outline the tools and techniques for the agent verification we have developed.

Formal Verification

So, we are clear now that autonomous systems are important, that their key decision-making components can usefully be represented through the rational agent concept, and their increasing use in critical areas means a deep and comprehensive form of analysis will be desirable. These concerns have led us to use formal logics for describing the required properties of our rational agents and then formal verification techniques to analyze how well the actual agents match these requirements. Formal verification encompasses a range of techniques that use mathematical and logical methods to assess the behavior of systems. The most common approach is to exhaustively assess all the behaviors of a system against a logical specification.[13] But, how do we logically specify what an agent should do? In particular, how do we specify what decisions an agent can make and what motivations it has for making those decisions?

Logical Agent Specification. Logics provide a well-understood and unambiguous formalism for describing the behaviors, requirements, and properties of systems. They have clear syntax and semantics, well-researched structural properties, and comparative expressive power. Importantly, from our viewpoint, there are very many formal logics. This allows us to choose a logic appropriate to the types of properties and level of abstraction we require; for example:

- dynamic communicating systems → *temporal logics*
- information → *modal logics of knowledge*
- autonomous systems → *modal logics of motivation*
- situated systems → *modal logics of belief and context*
- timed systems → *real-time temporal logics*
- uncertain systems → *probabilistic logics*
- cooperative systems → *cooperation/coalition logics*

So, we can usually choose logics that have the properties we require. Crucially, we can even construct new logics as the combinations of simpler logics. This turns out to be very useful for developing logical theories for rational agents as these typically consist of several dimensions:

Dynamism—temporal or dynamic logic;

Information—modal/probabilistic logics of belief or knowledge; and

Motivation—modal logics of goals, intentions, desires.

For example, the BDI approach combines:[31] a (branching) temporal/dynamic logic; a (KD45) modal logic of belief; a (KD) modal logic of desire; and a (KD) modal logic of intention. (For detail on different modal varieties, see Blackburn.[2])

Formal Agent Verification. Once we have such a logical requirement, together with an autonomous system architecture wherein rational agent(s) encapsulate high-level decision-making, we have many options for carrying out formal verification, ranging across model-checking,[13] runtime verification,[24] and formal proof.[21]

While there are also several approaches to agent verification,[7,29] the particular approach we adopt involves checking a BDI logical requirement against all practical executions of a program. This is termed the model checking of programs[36] and depends on being able to extract all these possible program executions, for example through symbolic execution. This contrasts to many model checking approaches in which an abstract model of the program must first be constructed before it can be checked against a property. In the case of Java, model checking of programs is feasible as a modified virtual machine can be used to manipulate the program executions.[26] It is this last approach to agent verification we adopt. In order to do so, we must also give a very brief overview of agent programming languages.

Programming Rational Agents. We have seen how the rational agent approach provides the key model for describing autonomous decision-making.

But, how can rational agents be programmed? Typically, programming languages for rational agents provide:

- a set of *beliefs*, representing information the agent has;
- a set of *goals*, representing motivations the agent has;
- a set of *rules/plans*, representing the agent's mechanisms for achieving goals;
- a set of *actions*, corresponding to the agent's external acts (delegated to other parts of the system); and
- deliberation mechanisms for deciding between alternative goals/plans/actions.

A typical agent rule/plan in such a language is:

```
Goal(eat) : Belief(has_money),
            Belief(not has_food)
          <-  Goal(go_to_shop),
              Action(buy_food),
              Goal(go_home),
              Action(eat),
              +Belief(eaten).
```

The meaning of this rule is that, if the agent's goal is to "eat" and if the agent believes it has money but does not have food, then it will set up a new goal to go to the shop. Once that goal has been achieved, it will buy some food (delegated to a subsystem) and then set up a new goal to get home. Once at home it will eat and then update its beliefs to record that it believes it has eaten.

Such languages are essentially rule-based, goal-reduction languages, with the extra aspect that *deliberation*, the ability to change between goals and change between plan selection strategies at any time, is a core component. Almost all of these languages are implemented on top of Java, and the large number of agent platforms now available[5,6] has meant the industrial uptake of this technology is continually increasing. The key ancestor of most of today's agent programming languages is AgentSpeak,[30] which introduced the programming of BDI agents using a modification of Logic Programming. Of the many descendants of AgentSpeak, we use Gwendolen,[19] which is based upon *Jason*,[8] for programming our rational agents. Consequently, it is such programs that we directly verify.

A full operational semantics for Gwendolen is presented in Dennis and Farwer.[15] Key components of a Gwendolen agent are a set, Σ, of beliefs that are ground first order formulae and a set, I, of intentions that are stacks of *deeds* associated with some event. Deeds include (among other things) the addition or removal of beliefs, the establishment of new goals, and the execution of primitive actions. Gwendolen is event driven and events include the acquisition of new beliefs (typically via perception), messages, and goals.

The key ancestor of most of today's agent programming languages is AgentSpeak, which introduced the programming of BDI agents using a modification of Logic Programming.

A programmer supplies plans that describe how to react to events by extending the deed stack of the intention associated with the event. The main task of a programmer working in Gwendolen is defining the system's initial beliefs and plans; these then describe the dynamic behavior of the agent. A Gwendolen agent executes within a reasoning cycle that includes the addition of beliefs from perception, the processing of messages, the selection of intentions and plans, and the execution of deeds.

Model Checking Agent Programs. We begin with program model checking, specifically the *Java PathFinder 2* system (JPF2), an open source explicit-state model checker for Java programs.[26,36] Since the vast majority of agent languages are built on top of Java, we have extended JPF2 to the Agent JPF (AJPF) system[19] incorporating the checking of agent properties. However, in order to achieve this the semantics of the agent constructs used must be precisely defined. Such semantics can be given using the Agent Infrastructure Layer (AIL),[16] a toolkit for providing formal semantics for agent languages (in particular BDI languages) built on Java. Thus, AJPF is essentially JPF2 with the theory of AIL built in; see Dennis et al.[19]

The whole verification and programming system is called MCAPL and is freely available on Sourceforge.[a] As the model checker is based on JPF2, the modified virtual machine is used to exhaustively explore all executions of the system. As each one is explored it is checked against the required property. If any violation is found, that execution is returned as a counterexample.

[a] http://cgi.csc.liv.ac.uk/MCAPL/

The Gwendolen language, mentioned earlier, is itself programmed using the AIL and so Gwendolen programs can be model checked directly via AJPF.

Verifying Autonomous Systems

We now return to our original question: How do we go about verifying autonomous systems? Recall the architecture in Figure 1. For the traditional parts there are well known, recognized, and trusted approaches, such as testing for real-world interaction and analytic techniques for continuous dynamics. But what about the agent that makes high-level, intelligent choices about what to do? As we will explain, it is our approach to use formal verification of the potential choices the agent can take. This is feasible since, while the space of possibilities covered by the continuous dynamics is huge (and potentially infinite), the high-level decision-making within the agent typically involves navigation within a discrete state space. The agent rarely, if ever, bases its choices directly on the exact values of sensors, for example. It might base its decision on values reaching a certain threshold, but relies on its continuous dynamics to alert it of this, and such alerts are typically binary valued (either the threshold has been reached or it has not). Thus, we propose the mixture of techniques in Figure 1 to provide the basis for the formal verification of autonomous systems.

Verifying Autonomous Choices. How shall we verify autonomous decision-making? Our main proposal is to use program verification to demonstrate that the core rational agent always endeavors to act in line with our requirements and never deliberately chooses options it believes will lead to bad situations (for example, ones where the agent believes something is unsafe). Thus, we do not try to verify all the real-world outcomes of the agent's choices, but instead verify the choices themselves. In particular, we verify the agent always tries to achieve its goals/targets to the best of its knowledge/beliefs/ability. Thus, the agent targets situations it believes to be good and avoids situations it believes to be bad. Consequently, any guarantees here are about the autonomous system's decisions, not about its overall effects.

This lets us distinguish between a rational agent knowingly choosing a dangerous/insecure option and a rational agent unknowingly doing so based on an imperfect representation of the actual environment. Indeed, we argue the most crucial aspect of autonomous system verification, for example concerning safety, is to identify the agent never deliberately makes a choice it believes to be unsafe. We wish to ensure that if an unsafe situation arises it is because of unforeseen consequences of an agent's actions (that is, its model of the environment was too weak), not because the agent chose an option known to lead to a bad outcome.

Aside: Accidental or deliberate? Are all dangerous situations equally bad? What if a robot deliberately took an action that it knew would cause danger? Is this more serious than a robot accidentally causing this danger? This distinction can be important, not least to the public, and if a robot is being "vindictive," then few safeguards can protect us. Importantly, our approach allows us to distinguish between these cases. We can verify whether the agent beliefs were simply not accurate enough (in which case, the agent is "innocent") or whether the agent knew about the danger and decided to proceed anyway.

One reason for our approach of verifying what the agent chooses, based on its beliefs, involves the purely practical issue of trying to model the real world. We can never have a precise model of the real world and so can never say, for certain, what the effect of any action the system could choose might be. We might construct increasingly precise models approximating the real world, but they can clearly never be perfect.

A second reason is to treat the agent, to some extent, as we might treat a human. In assessing human behavior, we are happy if someone is competent and tries their best to achieve something. In particular, we consider someone as exhibiting "safe" behavior, if they have taken all the information they have access to into account and have competently made the safest decision they consider possible. Just as with humans, an agent's beliefs capture its partial knowledge about the real world. The agent's beliefs might be wrong, or incorrect, but we only verify the agent never chooses a course of action that it believes will lead to a bad situation. The agent's beliefs could be wrong and, of course, these beliefs might be refined/improved providing a better (more accurate) abstraction of the real situation.

We can contrast this with the traditional approach to formal (temporal) verification where we verify that bad things never happen and good things eventually happen. Instead, we only need to verify the agent *believes* these to be the case. This also has an impact upon the agent's selection of intentions/goals. As the agent is required to believe that no bad thing should occur, then it should never select an intention that it believes will lead to something bad.[b]

$$\mathtt{B}\ (\varphi \Rightarrow \Diamond\ \mathtt{bad}) \Rightarrow \Box\neg\mathtt{I}\varphi$$

So, if the agent believes that achieving φ eventually leads to something bad, it will never intend to undertake φ.

In the context of the verifications discussed in this article we use the property specification language that is provided with AJPF.[19] This language is propositional linear temporal logic (PLTL), extended with specific modalities for checking the contents of the agent's belief base (**B**), goal set (**G**), actions taken (**A**) and intentions—goals that are associated with a deed stack—(**I**).

This approach is clearly simpler as we can carry out verification without comprehensive modeling of the real world. Thus, we verify the *choices* the agent has, rather than all the real-world effects of those choices. Clearly, some parts of an agent's reasoning are still triggered by the arrival of information from the real world and we must deal with this appropriately. So, we first analyze the agent's program to assess what these incoming *perceptions* can be and then explore, via the AJPF model checker, all possible combinations of these. This allows us to be agnostic about how the real world might actually behave and simply verify how the agent behaves no matter what information it receives.

[b] Here, '**B**' means "the agent believes," '◇' means "at some future moment in time," '□' means "at all future moments in time," and '**I**' means "the agent intends."

Furthermore, this allows us to use hypotheses that explicitly describe how patterns of perception may occur in reality. Taking such an approach clearly gives rise to a large state space because we explore all possible combinations of inputs to a particular agent. However it also allows us to investigate a multi-agent system in a compositional way. Using standard assume-guarantee (or rely-guarantee) approaches,[25,28] we need only check the internal operation of a single agent at a time and can then combine the results from the model checking using deductive methods to prove theorems about the system as a whole.

Example Scenarios

To exemplify this approach, we review several different scenarios that have been implemented using Gwendolen and verified formally using AJPF.[17,37] In all these examples, the distinction in Figure 1 is central. The agent makes a decision, passes it on to the continuous control to implement the fine detail, and then monitors the activity. The agent only becomes involved again if a new situation is reached, if a new decision is required, or if the agent notices some irregularity in the way the continuous control is working.

RoboCup Rescue Scenario. Imagine an "urban search and rescue" scenario, of the form proposed in the RoboCup Rescue challenge,[27] where autonomous robots are searching for survivors after some natural disaster (for example, an earthquake). A robot builds up beliefs about some area using sensor inputs. Based on these beliefs, the robot makes decisions about whether to search further. So, we might verify:[17]

$$\Box(\mathbf{B}\,\texttt{can\_leave} \rightarrow (\mathbf{B}\,\texttt{found} \vee \mathbf{B}\,\texttt{area\_empty}))$$

meaning if the searching robot believes it can leave the area, then it either believes a human is found or it believes the area is empty. We can verify this, but need to provide some abstraction of the sensor inputs. We model the environment by supplying, randomly, all relevant incoming perceptions to the robot. In this case it either detects a survivor or does not detect a survivor, at any location. It is important to note the robot could be wrong. Its sensors might not detect a survivor (for example, buried under rubble). However, this does not make the autonomous system incorrect; it has made the best decisions it can given the information it had.

When AJPF encounters a random choice in Java it treats it as a branch in the possible execution of the model and explores both branches—that is, it checks the property holds both in the situation where the perception was received by the agent and the situation where the perception was not received. We can extend this to proving properties given simple assumptions about the behavior of the real world. These assumptions might be verified using other forms of analysis. Given the verification here, we might assume the robot's sensors accurately detect the human, and that its motor control operates correctly. This allows us to prove a stronger property that the agent will either find the human or the area is actually empty. These deductive aspects can be carried out by hand, or by using a suitable prover.

In more sophisticated scenarios we may want to check properties of groups of systems/agents working together. Imagine we now have another robot, capable of lifting rubble. The two robots work as a team: the "searching" robot will find the human; the "lifting" robot will then come and remove the rubble. We will refer to the beliefs of the lifting robot as $\mathbf{B}_l$. Ideally, if these two work together as expected then we would like to show that eventually the lifter believes the human is free: $\Diamond\mathbf{B}_l\,\texttt{free(human)}$. However, this depends on several things, for example that any communication between the robots is reliable. We can check the behaviors of each agent separately, then combine these component properties with statements about communication, in order to verify whether the robots can cooperate.

We have been verifying the beliefs agents form about their environment in lieu of verifying actual facts. However, some choices we may legitimately wish to verify depend upon the outcomes of previous choices being as expected. Suppose our lifting agent does not deduce that the human is free (because it has moved some rubble), but continues to lift rubble out of the way until its sensors tell it the area is clear. We cannot verify the robot will eventually believe the human is free since we cannot be sure it will ever believe the human is clear of rubble. However, we can establish (and have verified) that assuming that, whenever the lifter forms an intention to free the human it will eventually believe the rubble is clear, then receipt of a message from the searching robot that a trapped human is located will eventually result in the lifter believing the human is free.

$$\Box(\mathbf{I}_l\,\texttt{free(human)} \Rightarrow \Diamond\mathbf{B}_l\,\texttt{clear}) \Rightarrow (\mathbf{B}_l\,\texttt{receive(searcher,found)} \Rightarrow \Diamond\mathbf{B}_l\,\texttt{free(human)})$$

While much simplification has occurred here, it should be clear how we can carry out compositional verification, mixing agent model checking and temporal/modal proof. The input from sensors can be modeled in various ways to provide increasingly refined abstractions of the real world. Crucially, we can assess the choices the agent makes based on its beliefs about its environment and not necessarily what actually happens in its environment.[c]

Autonomous Satellite Scenario. Consider a satellite orbiting the Earth and attempting to keep on a particular path.[18] We want to establish $\mathbf{B}\,\Box\texttt{on\_path}$, that is, the satellite believes it is always on the path. Yet, we cannot establish this since the satellite's agent cannot be sure it will never leave the path (since this would be an impossibly strong assumption about the environment).

However, we can show that

1. if it does leave its path, then the satellite will eventually recognize this; and

2. once this situation is recognized, the satellite will have a goal (that is, "intends") to move back onto the path as soon as possible.

In other words, if anything goes wrong, the satellite will recognize

[c] Agent code written in Gwendolen for this scenario together with sample verified properties is available from the MCAPL repository on Sourceforge.

this and will try to fix it. It might fail, but all we can show is that it always tries to succeed. Note that (1) is a property that needs to be established concerning the satellite's sensors, but (2) is indeed something we can verify of the agent.

Engineers and mathematicians have developed strong techniques for analyzing control systems and scenarios and proving that a certain property holds. For example, we might separately prove that a continuous path planning algorithm works and so capture that as a behavior in a simplified model of the environment (here, '**A**' means "the agent executes the external action of"):

$$\mathbf{A}\,\mathtt{go\_to\_path} \Rightarrow \Diamond\mathtt{on\_path}$$

Thus, if the agent executes some action based on continuous path planning to reach some destination it will eventually reach that destination. Again, notice how the verification of this will be carried out using other methods; we will just use this assumption during verification of the agent choices. As examples, we can verify several different properties:[17]

1. Using a simple model of the environment where the satellite simply receives information about its position, we can verify that if, whenever an agent uses continuous planning to move to a path, it eventually believes it reaches the path and if, whenever it activates path maintenance procedures it always believes it remains on the path, then eventually the satellite always believes it is on the path:

$$\begin{aligned}&\Box(\mathbf{A}\,\mathtt{go\_to\_path} \Rightarrow \Diamond\mathbf{B}\,\mathtt{on\_path}) \wedge\\ &\quad\Box(\mathbf{A}\,\mathtt{maintain\_path} \Rightarrow (\mathbf{B}\,\mathtt{on\_path} \Rightarrow\\ &\quad\Box\mathbf{B}\,\mathtt{on\_path}))\\ &\qquad\qquad\Rightarrow\Diamond\Box\mathbf{B}\,\mathtt{on\_path}\end{aligned}$$

2. It is possible for venting from a broken fuel line to knock a satellite off path. In this situation the satellite first needs to correct the problem with the thruster (for example, by switching valves between fuel lines) and then calculate a new path to its destination. So we can verify if the satellite notices it is no longer on the path then it will form an intention to return to the path:

$$\Box(\mathbf{B}\neg\,\mathtt{on\_path} \Rightarrow \Diamond\mathbf{I}\,\mathtt{on\_path})$$

Any autonomous system in control of an unmanned aircraft must be "human equivalent" or better. Human equivalence is, clearly, difficult to specify.

Note: If the satellite receives a message requesting it to move to a different position during this process, then subtle interactions between the agent's goals and plans can result in the satellite attempting to move to two locations at once. Attempting (and failing) to verify that, under suitable conditions, the agent would always eventually get on to the path led to the detection of a number of bugs such as this.

3. If we relax our hypotheses, for instance to allow the possibility of unfixable errors in the thrusters, then we can still verify some properties. For instance, eventually either the agent always believes it is on the path or it has informed ground control of a problem.

$$(\Diamond(\Box\mathbf{B}\,\mathtt{on\_path} \vee \mathtt{B}\ \mathtt{informed}(\mathit{ground}, \mathit{problem}))$$

Autonomous Unmanned Aircraft Scenario. Unmanned aircraft are set to undertake a wide variety of roles within civil airspace. For safety, and to obtain regulatory approval, unmanned aircraft must be shown to be equivalent to manned aircraft and transparent to other airspace users.[12] In essence, any autonomous systems in control of an unmanned aircraft must be "human equivalent" or better. Human equivalence is, clearly, difficult to specify. But perhaps a good place to start extracting desirable human behaviors is the statutory and regulatory documents designed to specify and exemplify ideal human behaviors, for example, the "Rules of the Air."[11] In order to begin to verify the human equivalence of unmanned aircraft autonomy, we identified a very small (but salient) subset of the Rules of the Air,[37] including the following.

1. Detect and Avoid: *"...when two aircraft are approaching head-on ... and there is danger of collision, each shall alter its course to the right."* (Section 2.4.10)

2. Navigation in Aerodrome Airspace: *"[An aircraft in the vicinity of an aerodrome must] make all turns to the left unless [told otherwise]."* (Section 2.4.12(1)(b))

3. Air Traffic Control Taxi Clearance: *"An aircraft shall not taxi on the apron or the maneuvering area of an aerodrome without [permission]."* (Section 2.7.40)

A decision-making agent for an unmanned aircraft was written. A simulated environment was also developed using GWENDOLEN, consisting of: a sensor unit to generate alerts related to intruder aircraft and other air traffic; a navigation manager to generate alerts about the current flight path; and an aerodrome air traffic controller unit to simulate aerodrome air traffic control. In order to formally verify the agent controlling the unmanned aircraft will follow the three rules here they were translated into the logical formulae and verified using the AJPF model checker:[37]

1. *"It is always the case that if the agent believes that an object is approaching head-on, then the agent believes that the direction of the aircraft is to the right."*

$$\square(\mathbf{B}\,\texttt{objectIsApproaching} \Rightarrow \mathbf{B}\,\texttt{direction}(\mathit{right}))$$

2. *"It is always the case that if the agent believes that it is changing heading (that is, turning as part of navigation) and it believes it is near an aerodrome and it believes it has not been told to do otherwise, then the agent will not believe that its direction is to the right."*

$$\square\left(\begin{bmatrix}\mathbf{B}\,\texttt{changeHeading}\wedge\\ \mathbf{B}\,\texttt{nearAerodrome}\wedge\\ \neg\mathbf{B}\,\texttt{toldOtherwise}\end{bmatrix}\right)\Rightarrow\neg\mathbf{B}\,\texttt{direction}(\mathit{right})$$

3. *"It always the case that if the agent believes it is taxiing, then it believes that taxi clearance has been given."*

$$\square(\mathbf{B}\,\texttt{taxiing} \Rightarrow \mathbf{B}\,\texttt{taxiClearanceGiven})$$

Verifying such requirements not only shows the autonomous system makes choices consistent with these Rules of the Air, but can also highlight inconsistencies within the rules themselves.[37]

Summary and Future Work

Once autonomous systems have a distinguished decision-making agent, then we can formally verify this agent's behavior. In particular, we have developed model checking techniques for rational agents, allowing us to explore all possible choices the agent might make. Notably, the architecture and the logical framework together allow us to verify not only what the agent chooses, but why it chooses it.

A central theme of our analysis of autonomous systems, and of the agents that control them, is to verify what the agent tries to do. Without a complete model of the real environment, then we cannot say the system will always achieve something, but we can say it will always try (to the best of its knowledge/ability) to achieve it. This is not only as much as we can reasonably say, it is entirely justifiable as we wish to distinguish accidental and deliberate danger. So, when considering safety, we cannot guarantee our system will never reach an unsafe situation, but we can guarantee the agent will never "knowingly" choose to move toward such a situation. Thus, all the choices of the agent/system are verified to ensure it never chooses goals/actions it believes will lead to bad situations. Crucially, this analysis concerns just the agent's internal decisions and so verification can be carried out without having to examine details of the real world. Thus, we verify the choices the agent has, rather than the (continuous/ uncertain) real-world effect of those choices.

A central theme of our analysis of autonomous systems, and of the agents that control them, is to verify what the agent tries to do.

Overall, we can see this as a shift from considering whether a system is correct to considering two aspects of systems:

1. analysis of whether the (autonomous) system makes only correct choices, given what it believes about its environment, together with
2. analysis of how accurate and reliable the system's beliefs are about its environment.

We have considered (1) in this article. However, (2) may be discrete, if abstractions are used, or continuous and uncertain, requiring more complex analytical techniques.

This work is only just at the beginning, and the theme of verifying what autonomous systems try to do, rather than the effects they have, has much potential. However, there are many avenues of future work, the foremost currently being incorporation of uncertainty and probability. So, rather than verifying the agent never chooses a course of action it believes will lead to a bad situation, we would like to verify the agent *never chooses a course of action that it believes is more likely to reach a "bad" situation than its other options.*

In addition, there are clearly various different forms of "bad" situation, with different probabilities and measures concerning their seriousness. Again, these measures and probabilities should be incorporated into the properties verified.

Similarly, there are important aspects of truly autonomous behavior, such as the ability to plan and learn that we have not considered in any detail. We are interested in exploring how an agent might reason about new plans, for instance, to ensure their execution did not violate any important properties and so provide guarantees about the agents overall behavior even in the face of changing internal processes.

It is also important to assess if, and how, other approaches to the formalization of autonomous behaviors, for example, Arkin,[1] can be involved in our verification.

Toward Certification. Certification can be seen as the process of negotiating with a certain legal authority in order to convince them that relevant safety requirements have been explored and mitigated in an appropriate way. As part of this process, various items of evidence are provided to advance the applicant's safety argument. This approach is widely used for the certification of real systems, from aircraft to safety critical software.

Clearly, we are mainly concerned with the certification of autonomous systems. As noted, systems might generally be analyzed with respect to the question, "Is it safe?" If there is a human involved at some point, for example, a pilot or controller, then some view must be taken on whether the human acts to preserve safety or not. For example, within aircraft certification arguments, it is usually assumed that a pilot, given appropriate information and capabilities, will act to preserve the aircraft's safety. Yet in a safety analysis, we rarely go any further. Essentially, the human is assumed to be benevolent.

Our approach provides a mechanism for analyzing the agent choices in the case of autonomous systems. Thus, while a standard safety argument might skip over human choices, assuming the pilot/driver/operator will endeavor to remain safe, we can formally verify the agent indeed tries its best to remain safe. In this way, our approach allows wider analysis—while the intentions and choices of a pilot/driver/operator must be assumed to be good, we can actually examine the intentions and choices of an autonomous system in detail.

Acknowledgments. This work was partially supported by EPSRC, while the Virtual Engineering Centre is a University of Liverpool project partially supported by both NWDA and ERDF. We are grateful to our many collaborators, but particularly Rafael Bordini, Neil Cameron, Mike Jump, Alexei Lisitsa, Nick Lincoln, Bertie Müller, and Sandor Veres.

References

1. Arkin, R. Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture. Technical Report GIT-GVU-07-11. Georgia Tech, 2007.
2. Blackburn, P., van Benthem, J. and Wolter, F. eds. *Handbook of Modal Logic.* Elsevier, 2006.
3. Bonasso, P., Firby, J., Gat, E., Kortenkamp, D., Miller, D. and Slack, M. Experiences with an architecture for intelligent, reactive agents. *J. Exp. Theor. Artif. Intel. 9*, 23 (1997), 237–256.
4. Bond, A. and Gasser, L. eds. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1988.
5. Bordini, R., Dastani, M., Dix, J. and El Fallah-Seghrouchni, A. eds. *Multi-Agent Programming: Languages, Platforms and Applications.* Springer, 2005.
6. Bordini, R., Dastani, M., Dix, J. and El Fallah-Seghrouchni, A. eds. *Multi-Agent Programming: Languages, Tools and Applications.* Springer, 2009.
7. Bordini, R., Fisher, M., Visser, W. and Wooldridge, M. Verifying multi-agent programs by model checking. *J. Autonomous Agents and Multi-Agent Systems 12*, 2 (2006), 239–256.
8. Bordini, R., Hübner, J. and Wooldridge, M. *Programming Multi-agent Systems in* `AgentSpeak` *using Jason.* Wiley, 2007.
9. Bratman, M. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.
10. Brooks, R. A Robust layered control system for a mobile robot. *IEEE J. Robotics and Automation 2*, 10 (1986).
11. Civil Aviation Authority. CAP 393 Air Navigation: The Order and the Regulations; http://www.caa.co.uk/docs/33/CAP393.pdf, April 2010.
12. Civil Aviation Authority. CAP 722 Unmanned Aircraft System Operations in UK Airspace—Guidance; http://www.caa.co.uk/docs/33/CAP722.pdf, April 2010.
13. Clarke, E., Grumberg, O. and Peled, D. Model Checking. MIT Press, 1999.
14. Cohen, P. and Levesque, H. Intention is choice with commitment. *Artificial Intelligence 42* (1990), 213–261.
15. Dennis, L. and Farwer, B. Gwendolen: A BDI Language for verifiable agents. In *Workshop on Logic and the Simulation of Interaction and Reasoning.* AISB, 2008.
16. Dennis, L., Farwer, B., Bordini, R., Fisher, M. and Wooldridge, M. A common semantic basis for BDI languages. In *Proc. 7th Int. Workshop on Programming Multiagent Systems*, LNAI 4908 (2008). Springer, 124–139.
17. Dennis, L., Fisher, M., Lincoln, N., Lisitsa, A. and Veres, S. Verifying Practical Autonomous Systems. (Under review.)
18. Dennis, L., Fisher, M., Lisitsa, A., Lincoln, N. and Veres, S. satellite control using rational agent programming. *IEEE Intelligent Systems 25*, 3 (May/June 2010), 92–97.
19. Dennis, L., Fisher, M., Webster, M. and Bordini, R. Model checking agent programming languages. *Automated Software Engineering 19*, 1 (2012), 5–63.
20. Durfee, E., Lesser, V. and Corkill, D. Trends in cooperative distributed problem solving. *IEEE Trans. Knowledge and Data Engineering 1*, 1 (1989).
21. Fisher, M. *An Introduction to Practical Formal Methods Using Temporal Logic.* Wiley, 2011.
22. Franklin, S. and Graesser, A. Is it an agent, or just a program? A taxonomy for autonomous agents. *Intelligent Agents III*, LNCS 1193 (1996), 21–35.
23. Gat, E., Bonnasso, R., Murphy, R. and Press, A. On three-layer architectures. *Artificial Intelligence and Mobile Robots.* AAAI Press, 1997, 195–210.
24. Havelund, K. and Rosu, G. Monitoring programs using rewriting. In *Proc. 16th IEEE Int. Conf. Automated Software Engineering* (2001). IEEE Computer Society, 135–143.
25. Jones, C. *Systematic Software Development Using VDM.* Prentice Hall International, 1986.
26. Java PathFinder. javapathfinder.sourceforge.net.
27. Kitano, H. and Tadokoro, S. RoboCup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine 22*, 1 (2001), 39–52.
28. Manna, Z. and Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1992.
29. Raimondi, F. and Lomuscio, A. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic 5*, 2 (2007), 235–251.
30. Rao, A. `AgentSpeak`(L): BDI agents speak out in a logical computable language. In *Proc. 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, LNCS 1038* (1996). Springer, 42–55.
31. Rao, A. Decision procedures for propositional linear-time belief-desire-intention logics. *Journal of Logic and Computation 8*, 3 (1998), 293–342.
32. Rao, A.S. and Georgeff, M.P. BDI agents: From theory to practice. In *Proc. 1st Int. Conf. Multi-Agent Systems* (San Francisco, CA, 1995), 312–319.
33. Rao, A.S. and Georgeff, M.P. An abstract architecture for rational agents. In *Proc. 1st Int. Conf. Knowledge Representation and Reasoning* (1992), 439–449.
34. Shoham, Y. Agent-oriented programming. *Artificial Intelligence 60*, 1 (1993), 51–92.
35. United States of America State of Nevada Legislature. Nevada Revised Statutes Chapter 482A—Autonomous Vehicles, Mar. 2012.
36. Visser, W., Havelund, K., Brat, G.P., Park, A. and Lerda, F. Model checking programs. *Automated Software Engineering 10*, 2 (2003), 203–232.
37. Webster, M., Fisher, M., Cameron, N. and Jump, M. Formal methods and the certification of autonomous unmanned aircraft systems. In *Proc. 30th International Conference on Computer Safety, Reliability and Security. LNCS 6894* (2011). Springer, 228–242.
38. Wooldridge, M. *An Introduction to Multiagent Systems.* Wiley, 2002.
39. Wooldridge, M. and Rao, A., Eds. *Foundations of Rational Agency.* Kluwer Academic Publishers, 1999.

Michael Fisher (mfisher@liverpool.ac.uk) is a professor in the Department of Computer Science at the University of Liverpool, U.K.

Louise Dennis (l.a.dennis@liverpool.ac.uk) is a research associate in the Department of Computer Science at the University of Liverpool, U.K.

Matt Webster (matt@liverpool.ac.uk) is a research associate in the Virtual Engineering Centre at Daresbury Laboratory Warrington, U.K.

LEARNING @ SCALE

ATLANTA, GEORGIA MARCH 4–5 2014

Call for Papers

LEARNING @ SCALE

ACM will host the first **ACM Conference on Learning at Scale** to be held March 4-5, 2014 in Atlanta, GA.

Inspired by the emergence of Massive Open Online Courses (MOOCs) and the shift in thinking about education, ACM created this conference as a new scholarly venue to explore how learning and teaching can improve when done "at scale."

ABOUT THE CONFERENCE

Learning at Scale refers to new approaches for students to learn and teachers to teach, when engaging hundreds or even thousands of students; be it face-to-face or remotely, synchronous or asychronous.

Topics will include (but are not limited to) Usability Studies, Tools for Automated Feedback and Grading, Learning Analytics, Investigation of Student Behavior and Correlation with Learning Outcomes, New Learning Tools and Techniques at Scale.

CALL FOR PAPERS

Authors are encouraged to visit the Learning @ Scale website
http://learningatscale.acm.org/

- for details about the conference and guidelines for paper submissions, including an array of subject examples to explore.
- Authors are encouraged to consider submitting a full paper (10-pages max); a tutorial proposal for a 1-4 hour discussion on a relevant tool, technology, or methodology related to learning at scale; or a work-in-progress poster or demo.
- Papers must tackle topics "at scale."

IMPORTANT DATES

NOVEMBER 8, 2013
Paper submissions due.

NOVEMBER 8, 2013
Tutorial proposals due.

DECEMBER 23, 2013
Notification to authors of accepted full papers.

JANUARY 2, 2014
Work-in-progress submissions due.

JANUARY 14, 2014
Notification to authors of accepted work-in-progress.

JANUARY 17, 2014
ALL revised and camera-ready material due.

research highlights

DOI:10.1145/2500468.2500480

Technical Perspective
Progress in Visual Categorization

By Pietro Perona

OUR VISUAL SYSTEM helps us carry out our daily business: walking, driving, reading, playing sports, or socializing. It is difficult to think of an activity that does not depend on vision. Our eyes and brain help us by measuring shapes, trajectories, and distances in world around us, and by recognizing materials, objects, and scenes. How is this done? Can we reproduce these abilities in a machine?

The following paper by Felzenszwalb et al. describes what is currently the best system for detecting object categories (a pedestrian, a bottle, a cat) in images. Like much work in computer vision, their system is built upon insight coming from a diverse set of areas of science and engineering: biological vision, geometry, signal processing, machine learning, and computer algorithms.

Three ingredients make their system successful. First, objects are described as collections of visually distinctive parts (for example, eyes, nose, and mouth in a face) that appear in a consistent, although not rigid, mutual position, or shape. This idea may be traced back to Fischler and Elschlager,[6] although much work was necessary to make it work in practice; for example, making representations invariant to scale, representing the fact that parts are sometimes occluded and thus invisible, and giving shape and occlusion probabilistic interpretation.[2]

The second ingredient is representing parts (eyes, among others) using patterns of local orientations in the image. This simple idea makes a big difference. It turns out that orientation is less sensitive to changes in lighting conditions and viewpoint than pixel values. This observation comes from studying biological vision systems[4] and is the foundation of the most successful descriptors for image patches: shape contexts, SIFT, and HOG.[1,3,7] The authors here add one twist to the idea: rather than building detectors based on what the part looks like, it is better to build detectors as discriminative classifiers; that is, optimizing their ability to tell the difference between a given part (for example, the head of a pedestrian) and the environment that typically surrounds it (bookshelves, the shoulders, and arms of the pedestrian).

> **The following paper describes what is currently the best system for detecting object categories (a pedestrian, a bottle, a cat) in images.**

The third ingredient is an efficient search algorithm, originating with Felzenszwalb's thesis,[5] which detects an object in a handful of seconds, focusing computation only on the most promising areas of the image.

Is detecting visual categories a solved problem? The reader will be amused by how poorly our best algorithms work. A quick perusal of Table 1 in Felzenszwalb et al. will reveal that, on a good day, less than half of the people are detected in the PASCAL VOC dataset. Boats and birds are even more difficult to find. This is precisely what makes computer vision an exciting field of research today: there is much progress to be made; we are still a few big ideas away from the ultimate design. Twenty years ago we only had nebulous ideas about how to approach visual categorization, and 10 years ago the performance numbers would have probably been in the few percent.

What is missing? Quite a few things; I will mention a couple. First of all, our models are purely phenomenological, based on statistics of how objects look in 2D images. We do not take into account 3D geometry, nor the properties and materials of surfaces. Second, today's goal is to recognize widely different categories: bottle vs. cat vs. person. There is a whole world of fine distinctions, for example, Anopheles vs. Culex mosquito, Siamese vs. Burmese cat. We do not yet know how to handle such fine-grained classifications. Third, people can learn to recognize new categories with just a few training examples; how many femurs does a medical student need to see to learn the category? Our algorithms must see thousands of training examples to become halfway decent. The mother of all challenges is scaling: there are millions of meaningful visual categories to recognize (10^5 vertebrate species, 10^7 insect species, not to speak of shoes, wristwatches, and handbags). We need to develop systems able to train themselves by using information available on the Web, and that are able to tap into the expertise of knowledgeable humans by asking them intelligent questions.

A growing number of talented researchers are hard at work tackling these questions. It is an exciting moment for computer vision. Stay tuned.

References

1. Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 4 (2002), 509–522.
2. Burl, M., Weber, M., and Perona, P. A probabilistic approach to object recognition using local photometry and global geometry. *European Conference on Computer Vision, II* (1998), 628–641.
3. Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society (2005), 886–893.
4. Edelman, S., Intrator, N., and Poggio, T. Complex cells and object recognition. Unpublished; http://cogprints.org/561/2/199710003.ps.
5. Felzenszwalb, P.F. and Huttenlocher, D.P. Pictorial structures for object recognition. *International Journal of Computer Vision 61*, 1 (2005), 55–79.
6. Fischler, M. and Elschlager, R. The representation and matching of pictorial structures. *IEEE Transactions on Computers 22* (1973), 67–92.
7. Lowe, D.G. Distinctive image features from scale-invariant keypoints. *Int. J. Comput Vision 60*, 2 (2004), 91–110.

Pietro Perona is the Allen E. Puckett Professor of Electrical Engineering and Computational and Neural Systems at the California Institute of Technology, Pasadena, CA, and director of the Ph.D. program in Computation and Neural Systems at Caltech.

DOI:10.1145/2494532

Visual Object Detection with Deformable Part Models

By Pedro Felzenszwalb, Ross Girshick, David McAllester, and Deva Ramanan

Abstract

We describe a state-of-the-art system for finding objects in cluttered images. Our system is based on deformable models that represent objects using local part templates and geometric constraints on the locations of parts. We reduce object detection to classification with latent variables. The latent variables introduce invariances that make it possible to detect objects with highly variable appearance. We use a generalization of support vector machines to incorporate latent information during training. This has led to a general framework for discriminative training of classifiers with latent variables. Discriminative training benefits from large training datasets. In practice we use an iterative algorithm that alternates between estimating latent values for positive examples and solving a large convex optimization problem. Practical optimization of this large convex problem can be done using active set techniques for adaptive subsampling of the training data.

1. INTRODUCTION

Object recognition is a fundamental challenge in computer vision. Consider the problem of detecting objects from a category, such as people or cars, in static images. This is a difficult problem because objects in each category can vary greatly in appearance. Variations arise from changes in illumination, viewpoint, and intra-class variability of shape and other visual properties among object instances. For example, people wear different clothes and take a variety of poses while cars come in various shapes and colors.

Early approaches to object recognition focused on three-dimensional geometric models and invariant features.[22, 24, 25] More contemporary methods tend to be based on appearance-based representations that directly model local image features.[21, 27] Machine learning techniques have been very successful in training appearance-based models in restricted settings such as face detection[29, 30] and handwritten digit recognition.[23] Our system uses new machine learning techniques to train models that combine local appearance models with geometric constraints.

To apply machine learning techniques to object detection we can reduce the problem to binary classification. Consider a classifier that takes as input an image and a position and scale within the image. The classifier determines whether or not there is an instance of the target category at the given position and scale. Detection is performed by evaluating the classifier at a dense set of positions and scales within an image. This approach is commonly called "sliding window" detection. Let x specify an image and a position and scale within the image. In the case of a linear classifier we threshold a score $\beta \cdot \Phi(x)$, where β is a vector of model parameters (often seen as a template) and $\Phi(x)$ is a feature vector summarizing the appearance of an image region defined by x. A difficulty with this approach is that a linear classifier is likely to be insufficient to model objects that can have significant appearance variation.

One representation, designed to handle greater variability in object appearance is that of a pictorial structure,[14, 18] where objects are described by a collection of parts arranged in a deformable configuration. In a pictorial structure model each part encodes local appearance properties of an object, and the deformable configuration is characterized by spring-like connections between certain pairs of parts.

Deformable models such as pictorial structures can capture significant variations in appearance but a single deformable model still cannot represent many interesting object categories. Consider modeling the appearance of bicycles. Bicycles come in different types (e.g., mountain bikes, tandems, penny-farthings with one big wheel and one small wheel) and we can view them from different directions (e.g., frontal versus side views). We use mixtures of deformable models to deal with these more significant variations.

Our classifiers treat mixture component choice and part locations as latent variables. Let x denote an image and a position and scale within the image. Our classifiers compute a score of the form

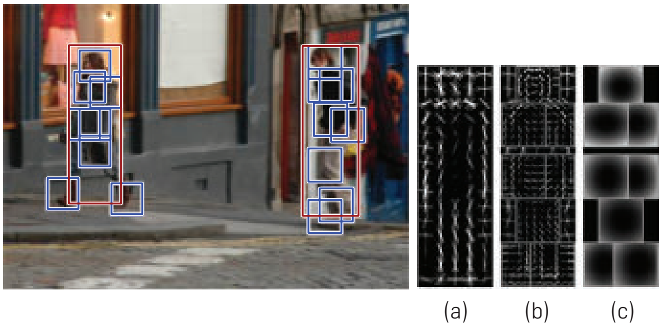
$$f_\beta(x) = \max_z \beta \cdot \Phi(x, z). \tag{1}$$

Here β is a vector of model parameters, z are latent values, and $\Phi(x, z)$ is a feature vector. If the score is above a threshold, our model will produce a detection at x. Associated with every detection are the inferred latent values, $z^* = \text{argmax}_z\ \beta \cdot \Phi(x, z)$, which specify a mixture component choice and the locations of the parts associated with that component. Figure 1 shows two detections and the inferred part locations in each case. We note that (1) can handle very general forms of latent information. For example, z could specify a derivation under a rich visual grammar.[15]

One challenge in training deformable part models is that it is often difficult to obtain training data with part annotations. Annotating parts can be time consuming and genuinely ambiguous. For example, what are the right parts for a sofa model? We train our models from weakly labeled data in the form of bounding boxes around target objects. Part structure and latent part locations are automatically

This work first appeared in the *IEEE CVPR 2008 Conference* and in the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 32, No. 9, September 2010.

Figure 1. Detections obtained with a single component person model. The model is defined by a coarse root filter (a), several higher resolution part filters (b), and a spatial model for the location of each part relative to the root (c). The filters specify weights for histogram of oriented gradients features. Their visualization shows the positive weights at different orientations. The visualization of the spatial models reflects the "cost" of placing the center of a part at different locations relative to the root.



inferred during learning. To achieve this, we developed a general framework for discriminative training of latent-variable classifiers of the form in (1). This leads to a formalism that we call latent support vector machine (LSVM).

Sliding window detection leads to imbalanced classification problems. There are vastly more negative examples than positive ones. To obtain high performance using discriminative training it is often important to make exhaustive use of large training sets. This motivates a data subsampling process that searches through all of the negative instances to find the hard negative examples and then trains a model relative to those instances. A heuristic methodology of data mining for hard negatives was adopted by Dalal and Triggs[7] and goes back at least to the training methods used by Schneiderman and Kanade[28] and Viola and Jones.[30] We developed simple data mining algorithms for subsampling the training data for SVMs and LSVMs that are guaranteed to converge to the optimal model defined in terms of the entire training set.

We formally define our models in Section 2. We describe a general framework for learning classifiers with latent variables in Section 3. Section 4 describes how we use this framework to train object detection models. We present experimental results in Section 5 and conclude by discussing related work in Section 6.

2. MODELS

A core component of our models is templates, or filters, that capture the appearance of object parts based on local image features. Filters define scores for placing parts at different image positions and scales. These scores are combined using a deformation model that scores an arrangements of parts based on geometric relationships. Detection involves searching over arrangements of parts using efficient algorithms. This is done separately for each component in a mixture of deformable part models.

2.1. Filters

Our models are built from linear filters that are applied to dense feature maps. A feature map is an array whose entries are d-dimensional feature vectors computed on a dense grid of image locations (e.g., every 8×8 pixels). Each feature vector describes a small image patch while introducing some invariants. The framework described here is independent of the specific choice of features. In practice we use a low-dimensional variation of the histogram of oriented gradient (HOG) features from Dalal and Triggs.[7] HOG features introduce invariances to photometric transformations and small image deformations.

A linear filter is defined by a $w \times h$ array of d-dimensional weight vector. Intuitively, a filter is a template that is tuned to respond to an iconic arrangement of image features. Filters are typically much smaller than feature maps and can be applied at different locations within a feature map. The score, or response, of a filter F at a particular feature map location is obtained by taking the dot product of F's array of weight vectors, concatenated into a single long vector, with the concatenation of the feature vectors extracted from a $w \times h$ window of the feature map. Because objects appear at a wide range of scales, we apply the same filter to multiple feature maps, each computed from a rescaled version of the original image. Figure 2 shows some examples of filters, feature maps, and filter responses. To fix notation, let I be an image and $p = (x, y, s)$ specify a position and scale in the image. We write $F \cdot \phi(I, p)$ for the score obtained by applying filter F at the position and scale specified by p.

2.2. Deformable part models

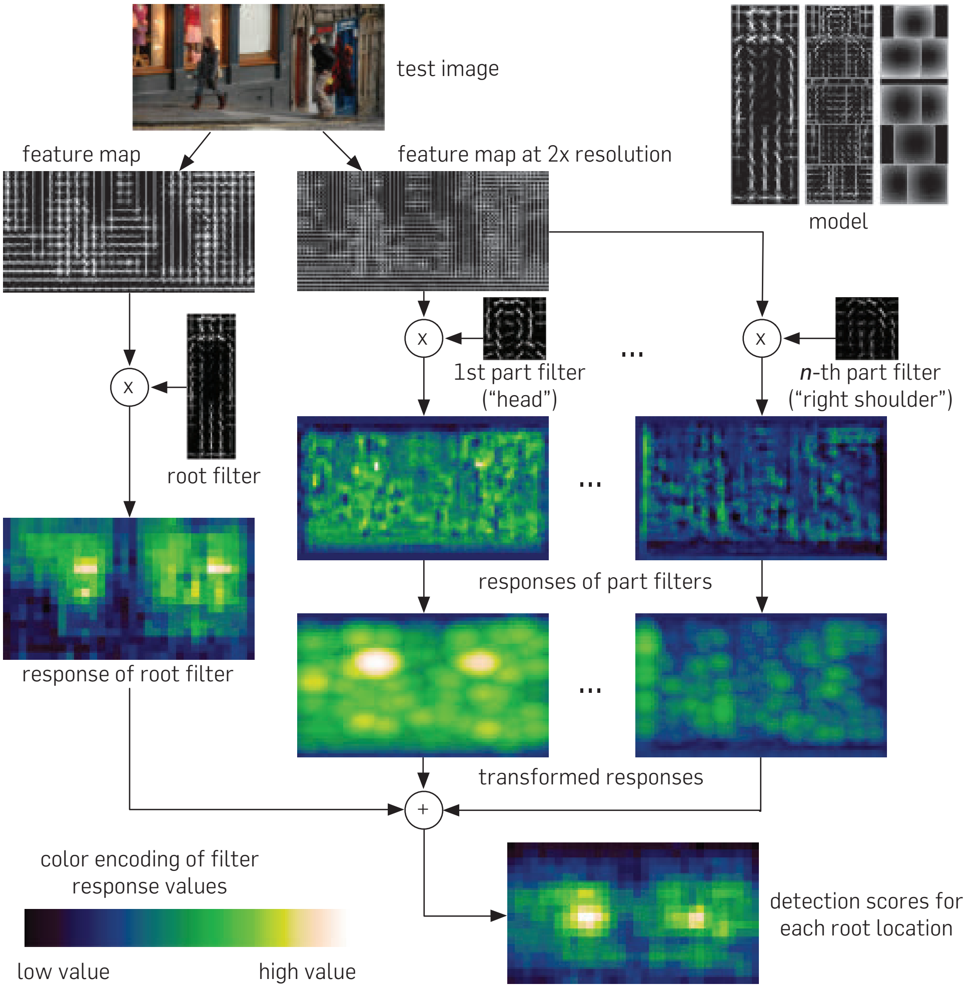
To combine a set of filters into a deformable model we define spring-like connections between some pairs of filters. Thinking of filters as vertices and their pairwise connections as edges, a model is defined by a graph. Here we consider models represented by star graphs, where one filter acts as the hub, or root, to which all other filters are connected.

In our star models, a low resolution root filter, that approximately covers an entire object, serves as the star's hub. Higher resolution part filters, that cover smaller regions of the object, are connected to the root. Figure 1 illustrates a star model for detecting pedestrians and its two highest scoring detections in a test image.

We have found that using higher resolution features for defining part filters is essential for obtaining high recognition performance. With this approach the part filters capture finer resolution features that are localized to greater accuracy when compared to the features captured by the root filter. Consider building a model for a face. The root filter might capture a coarse appearance model for the face as a whole while the part filters might capture the detailed appearance of face parts such as eyes, nose, and mouth.

The model for an object with n parts is defined by a set of parameters $(F_0, (F_1, d_1), \ldots, (F_n, d_n), b)$ where F_0 is a root filter, F_i is a part filter, d_i is a vector of deformation parameters, and b is a scalar bias term. The vector d_i specifies the coefficients of a quadratic function that scores a position for filter i relative to the root filter's position. We use a quadratic deformation model because it is relatively flexible while still amenable to efficient computations. A quadratic score over relative positions can be thought of as a spring that connects a part filter to the root filter. The rest position and rigidity of the spring are determined by d_i.

Figure 2. ***Detection at one scale*. Responses from the root and part filters are computed on different resolution feature maps. Distance transforms are used to solve equation (7) efficiently for all possible part placements. The transformed responses are combined to yield a final score for each root location. We show the responses and transformed responses for the "head" and "right shoulder" parts. Note how the "head" filter is more discriminative. The combined scores clearly show two good hypotheses for the object at this scale.**



An object hypothesis is given by a configuration vector $z = (p_0, \ldots, p_n)$, where $p_i = (x_i, y_i, s_i)$ specifies the position and scale of the i-th filter. The score of a hypothesis is given by the scores of each filter at their respective locations (the data term) minus a deformation cost that depends on the relative position of each part with respect to the root (the spatial prior), plus the bias,

$$\text{score}(z) = \sum_{i=0}^{n} F_i \cdot \phi(I, p_i) - \sum_{i=1}^{n} d_i \cdot \psi(p_i, p_0) + b, \qquad (2)$$

$$\text{where} \quad \psi(p_i, p_0) = (dx_i, dy_i, dx_i^2, dy_i^2),$$

$$\text{with} \quad dx_i = x_i - x_0 \quad \text{and} \quad dy_i = y_i - y_0.$$

Each term in the second summation in (2) can be interpreted as a spring deformation model that anchors part i to some ideal location relative to the root.

The score of a hypothesis z can be expressed in terms of a dot product, $\beta \cdot \Phi(I, z)$, between a vector of model parameters β and a feature vector $\Phi(I, z)$,

$$\beta = (F_0, \ldots, F_n, d_1, \ldots, d_n, b). \qquad (3)$$

$$\Phi(I, z) = (\phi(I, p_0), \ldots \phi(I, p_n), -\psi(dx_1, dy_1, \ldots, -\psi(dx_n, dy_n), 1). \qquad (4)$$

This makes a connection between deformable part models and linear classifiers. We use this representation for learning the model parameters with the latent SVM framework.

2.3. Detection

To detect objects in an image we compute an accumulated score for each root filter location p_0 according to the best possible placement of the parts relative to p_0

$$\text{score}(p_0) = \max_{p_1, \ldots, p_n} \text{score}(p_0, \ldots, p_n) \qquad (5)$$

$$= F_0 \cdot \phi(I, p_0) + \sum_{i=1}^{n} m_i(p_0) + b \qquad (6)$$

$$\text{where} \quad m_i(p_0) = \max_{p_i} F_i \cdot \phi(I, p_i) - d_i \cdot \psi(p_i, p_0). \qquad (7)$$

Let k be the number of possible locations for each filter. A naive computation of the accumulated score for p_0 would take $O(nk)$ time. Since there are k choices for p_0 this would lead to an $O(nk^2)$ time algorithm for computing all accumulated scores. A much faster approach can be obtained using the generalized distance transform algorithms from Felzenszwalb and Huttenlocher.[13] This leads to a method that computes all of the accumulated scores in $O(nk)$ time total. The approach is illustrated in Figure 2.

We obtain a set of detections by finding the local maxima of score(p_0) that exceed a user-specified confidence threshold. This non-maximal suppression step removes redundant detections that differ slightly in position and scale and thus are largely supported by the same image evidence.

2.4. Mixture models

As described in the introduction many interesting object categories exhibit more intra-class variation than can be accounted for by a single deformable model. A natural extension involves using a mixture of deformable models.

Formally, a mixture model with m components is defined by a m-tuple, $M = (M_1, \ldots, M_m)$, where M_c is the model for the c-th component. An object hypothesis, $z = (c, p_0, \ldots, p_{n_c})$ for a mixture model specifies a mixture component, $1 \le c \le m$, and a location p_i for each filter of M_c. The score of this hypothesis is the score of the hypothesis $z' = (p_0, \ldots, p_{n_c})$ for the c-th model component. As in the case of a single deformable model the score of a hypothesis for a mixture model can be expressed by a dot product between a vector of model parameters (the concatenation of the parameters for each mixture component) and an appropriately constructed feature vector that depends on the image I and the hypothesis z.

To detect objects using a mixture model, we first compute the accumulated root scores independently for each component, and then for each root location we select the highest scoring component hypothesis at that location.

3. LATENT SVM

Our models involve binary classifiers with latent variables. To train these classifiers we use a latent support vector machine (LSVM).[a] To formulate the LSVM training objective consider scoring functions of the following form

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z). \qquad (8)$$

Here x is an input, such as a detection window; β is a vector of model parameters; and z is an assignment of values to latent variables such as part placements. The set $Z(x)$ defines the possible latent values for an example x. A binary label for x can be obtained by thresholding this score.

In analogy to classical SVMs we can train β from labeled examples $D = (\langle x_1, y_1 \rangle, \ldots, \langle x_n, y_n \rangle)$, where $y_i \in \{-1, 1\}$, by minimizing the following objective function,

$$L_D(\beta) = \frac{1}{2} ||\beta||^2 + C \sum_{i=1}^{n} \max\left(0, 1 - y_i f_\beta(x_i)\right), \qquad (9)$$

where $\max(0, 1-y_i f_\beta(x_i))$ is the standard hinge loss and the constant C controls the relative weight of the regularization term. Note that if there is a single possible latent value for each example ($|Z(x_i)| = 1$) then f_β is linear in β, and we obtain linear SVMs as a special case of LSVMs.

3.1. Semi-convexity

Because the scoring function (8) is nonlinear in β, the LSVM objective function (9) is non-convex in β. However, the training problem becomes convex once latent information is specified for the positive training examples.

To see this, note that $f_\beta(x)$ as defined in (8) is a maximum of functions each of which is linear in β. Hence $f_\beta(x)$ is a max of convex functions and is hence convex. This implies that the hinge loss, $\max(0, 1 - y_i f_\beta(x_i))$, is convex in β when $y_i = -1$. That is, the loss function is convex in β for negative examples. Now if we only allow a single setting of the latent variables for each positive example, i.e., if we fix the latent values for the positives, then the hinge loss becomes linear in β, and hence convex, on the positive examples also. So fixing the latent information on the positive examples makes the overall training objective convex. This observation motivates the following training algorithm:

1. Holding β fixed, select the best latent value for each positive example,

$$z_i = \operatorname*{argmax}_{z \in Z(x_i)} \beta \cdot \Phi(x, z).$$

2. Fixing the latent variables for the positive examples to $Z(x_i) = \{z_i\}$, solve the (now convex) optimization problem defined by (9).

This procedure can be seen as a block coordinate descent optimization of an auxiliary training objective $L(\beta, Z_p)$ that depends on both β and a choice of latent values for the positive examples Z_p. Moreover, if the pair (β, Z_p) minimizes the auxiliary objective $L(\beta, Z_p)$ then β minimizes the original LSVM objective $L(\beta)$. This justifies training via minimization of $L(\beta, Z_p)$. The semi-convexity property plays an important role in this approach because it leads to a convex optimization problem in Step 2, even though the latent values for the negative examples are not fixed.

3.2. Optimization with data subsampling

When the latent values for the positive examples are fixed the LSVM objective function is convex and can be optimized using a variety of methods. However, a classical difficulty that arises when training a sliding window classifier is that a single training image yields an overwhelming number of negative examples. This difficulty has been previously addressed using heuristics that start with a small subset of the negative examples and alternate between training a model and growing the negative training set with false positives generated by the previous model.[7, 30]

We have developed a version of this heuristic process that is tailored for discriminative training with a hinge loss. It involves repeatedly training models using relatively small

[a] A latent SVM is equivalent to a multiple instance SVM.[2]

subsets of the training data and is guaranteed to find an optimal model under the original large dataset. The approach is applicable for both standard SVM and latent SVM.

Our method maintains a subset C of the training data, trains the model parameters β on the subset C, and then updates C. To describe the procedure more formally, we first define the "hard examples" for a model β as follows, where i ranges over the full training set

$$H(\beta) = \{(x_i, y_i) : y_i f_\beta(x_i) \leq 1\}.$$

Our algorithm initializes C to an arbitrary set of examples (such as all positives and a small random subset of negatives). It then repeats the following steps where $\beta^*(C)$ is the model minimizing the training objective on the training set C.

1. Set $\beta := \beta^*(C)$.
2. Shrink C by removing elements not in $H(\beta)$.
3. Grow C by adding new examples from $H(\beta)$.

Recall that we are holding the latent values of positive examples fixed, so the objective function is convex. If C contains all of $H(\beta)$ after Step 1 then the subgradients of the training objective with respect to C equal the subgradients of the training objective with respect to the entire dataset and we can terminate the process. Furthermore, we can prove that the process will always terminate. The basic insight is that the value of the training objective on the set C is non-decreasing. Note that the training objective on C does not change when we shrink C in Step 2, because the hinge loss of the examples being removed is zero. The objective also does not decrease when new elements are added to C in Step 3. In fact the training objective on C grows over time and since the number of possible subsets C is finite, the process must terminate.

4. TRAINING MODELS

Suppose we have training images with bounding boxes around objects in a particular category. We define a positive example from each bounding box. Bounding boxes do not specify mixture component labels or filter locations, so we treat these as latent variables during training. We use the bounding box information to constraint the placement of the root filter in each positive example. We define a very large set of negative examples from images that do not contain objects from the target category. Each position and scale in such an image yields a different negative example.

Together, the positive and negative examples lead to a latent SVM training problem where we want to select a model that gives high score for positive examples and low score for negatives. We use the block coordinate descent algorithm from Section 3.1 to optimize the LSVM training objective. Since this algorithm is susceptible to local minima it must be initialized carefully.

Initialization: We begin by learning root filters for each component of a mixture model. We partition the positive examples into m disjoint groups based on the aspect ratio of their bounding boxes. For each group, we warp the image data in the bounding boxes to a canonical size and train a root filter with a standard SVM. To initialize the part filters, we greedily place a fixed number of parts (eight, in all of our experiments) to cover high-energy regions of the root filter. The part filter coefficients are initialized by interpolating the root filter to twice the spatial resolution, and the part deformation parameters are initialized to a value that penalizes large displacements. Figure 3 shows the initial model obtained for a two-component car model.

Coordinate descent: Given an initial model β, Step 1 of the coordinate descent algorithm estimates latent values for each positive example. This includes a mixture component label and filter locations. We constrain the locations of the root filters to placements that overlap with the bounding box of a positive example by a significant amount. During Step 2 of coordinate descent, we learn a new model β by solving a large scale convex program with stochastic subgradient descent and data subsampling over the negative examples (Section 3.2). Note that we repeatedly update the latent values for each positive example, including a mixture component label. Therefore our algorithm naturally performs a "discriminative clustering" of the positive examples.

5. EMPIRICAL RESULTS

The system described here has been evaluated on the PASCAL VOC datasets. We refer to Everingham et al.[9] for details, but emphasize that the PASCAL VOC challenges are widely acknowledged as difficult testbeds for object detection.

Each dataset contains thousands of real-world images, and specifies ground-truth bounding boxes for 20 object classes. At test time, the goal is to predict the bounding boxes of all objects of a given class in an image (if any). In practice a system will output a set of bounding boxes with confidence scores, and these scores are thresholded at different points to obtain a precision-recall curve across all images in the test set. For a particular threshold the precision is the fraction of the reported bounding boxes that are correct detections, while recall is the fraction of the objects found.

A reported bounding box is considered correct if it overlaps more than 50% with a ground-truth bounding box. When a system reports several bounding boxes that overlap with a single ground-truth bounding box, only one detection is considered correct. One scores a system by the average precision (AP) of its precision-recall curve, computed for each object class independently.

Figure 3. Initialization. (a) The initial root filters for a car model. (b) and (c) The part filters and deformation models initialized from (a).

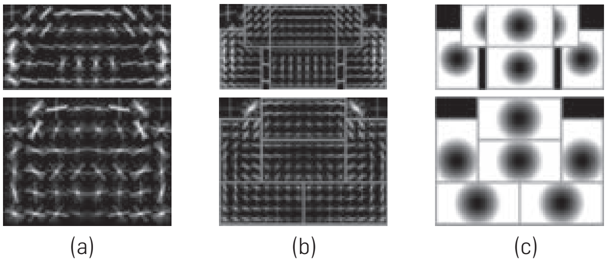


Figure 4. Visualizations of some of the models learned on the PASCAL 2010 dataset.

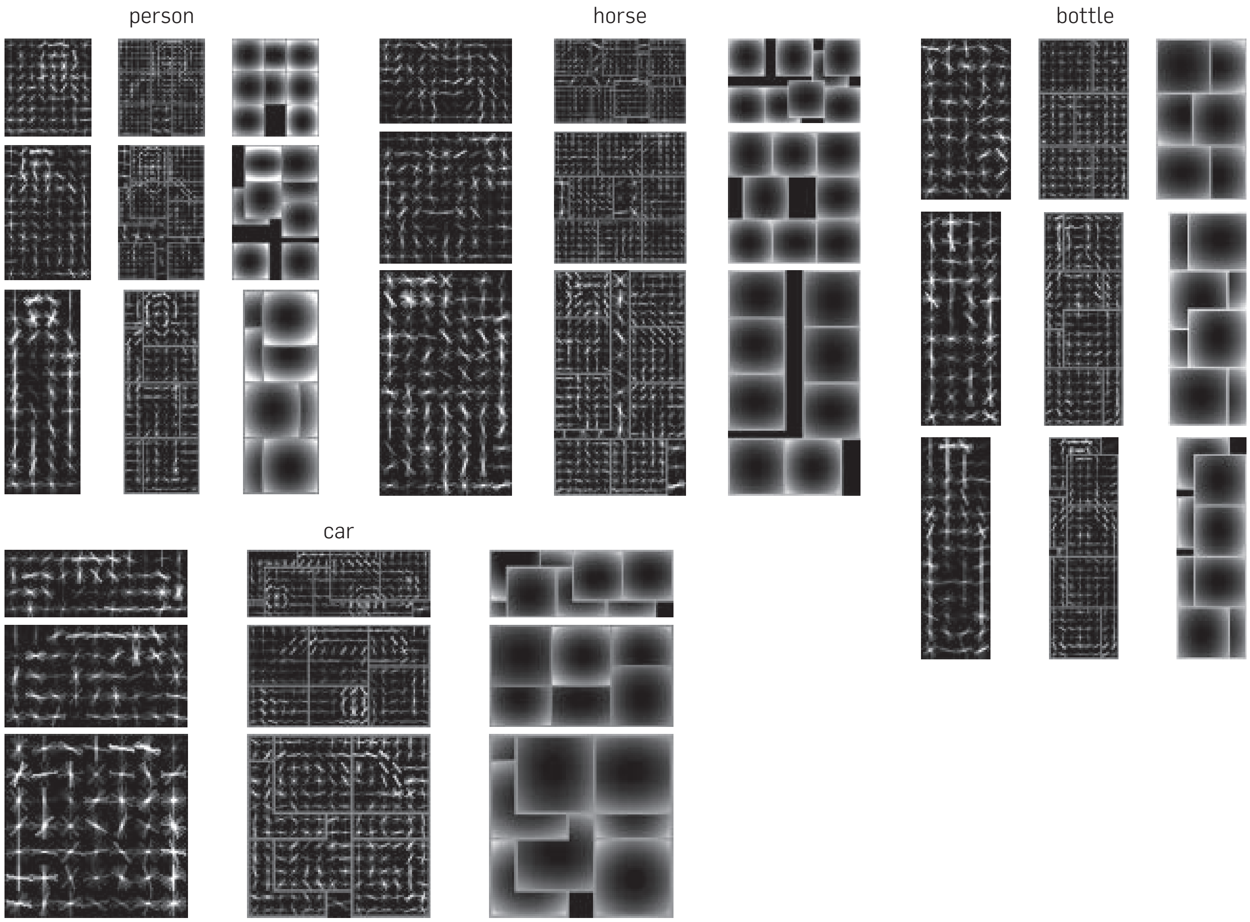


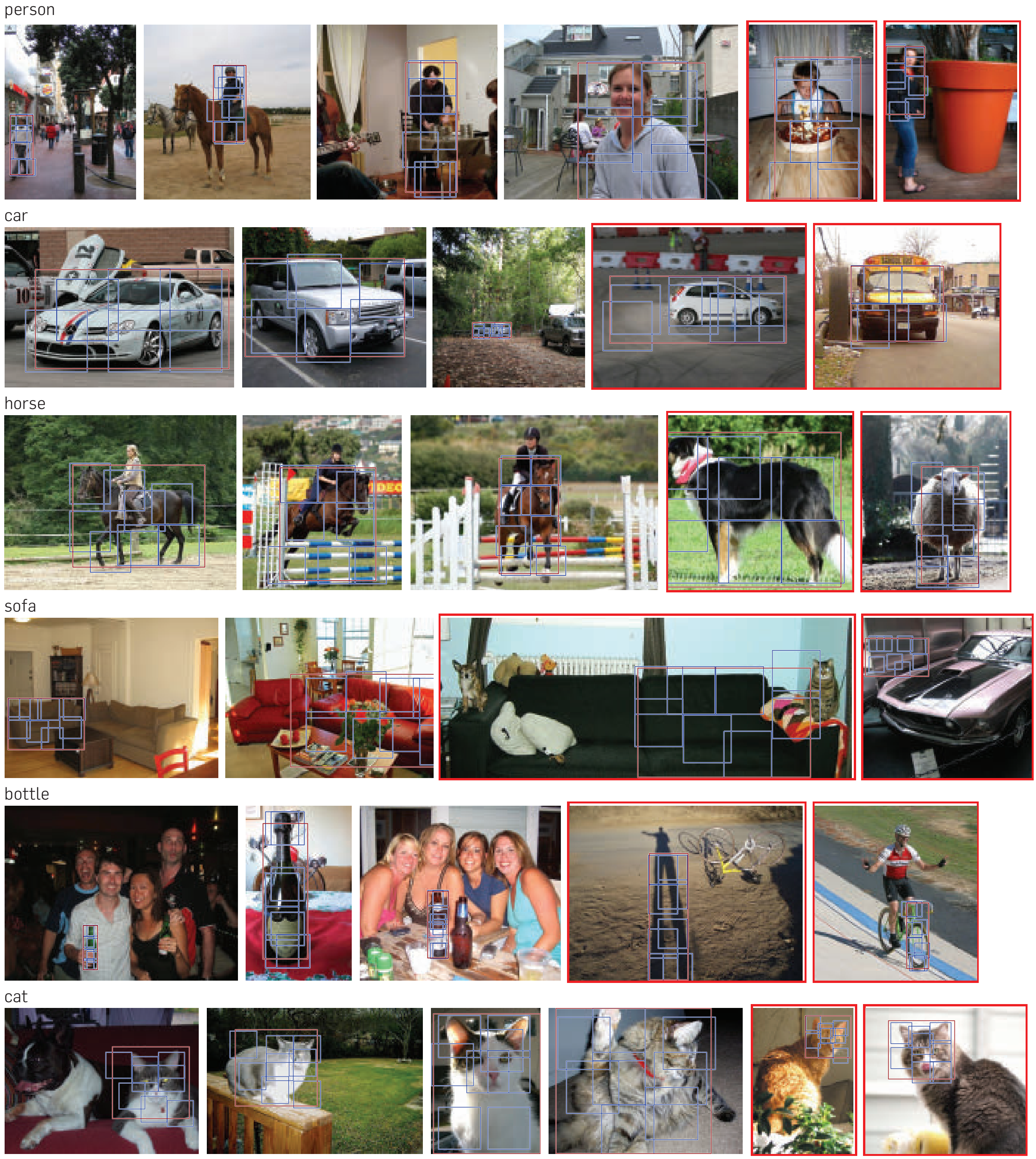
Figure 4 shows some models learned from the PASCAL VOC 2010 dataset. Figure 5 shows some example detections using those models. We show both high-scoring correct detections and high-scoring false positives. These examples illustrate how our models can handle significant variations in appearance such as in the case of cars and horses.

In some categories our false detections are often due to similarities among objects in different categories, such as between horse and cow or between car and bus. In other categories false detections are often due to the relatively strict bounding box overlap criteria. The two false positives shown for the person category are due to insufficient overlap with the ground-truth bounding box. The same is true for the cat category, where we often detect the face of a cat and report a bounding box that has relatively little overlap with the correct bounding box that encloses the whole object. In fact, the top 20 highest scoring false positive detections for the cat category correspond to a cat face. This is an extreme case but it gives an explanation for our low AP score in this category. Many positive training examples of cats contain only the face, and our cat mixture model has a component dedicated to detect cat faces, while another component captures an entire cat. Sometimes the wrong mixture component has the highest score, suggesting that our scores across different components could be better calibrated.

In the 2007, 2008, and 2009 PASCAL VOC competitions our system obtained the highest AP score in 6, 7, and 7 out of 20 categories, respectively.[9] Our entry was declared the winner of the competition in 2008 and 2009. In the 2010 competition, our system won in 3 of 20 categories, and the 3 systems that achieved a higher mean AP score (averaged over all classes) were all extensions of our system using additional features, richer context, and more parts.[9] Table 1 summarizes the AP scores of our system on the 2010 dataset, together with the best scores achieved across all systems that entered the official competition. We also show the effect of two post-processing methods that improve the quality of our detections.

The first method, bounding-box prediction, demonstrates the added benefit that comes with inferring latent structure at test time. We use a linear regression model to predict the true bounding box of a hypothesis from the inferred part configuration. The second method, context rescoring, computes a new confidence score for each detection with a polynomial kernel SVM whose features are the base detection score and the highest score for each of the 20 object-class detectors in the same image. This method can learn co-occurrence constraints between object classes; because cars and sofas tend not to co-occur, car detections should be downweighted if an image contains a high-scoring sofa. This context rescoring method

Figure 5. Examples of high-scoring detections on the PASCAL 2007 dataset. The red-framed images (last two in each row) illustrate false positives for each category. Many false positives (such as for person and cat) are due to the stringent bounding box overlap criteria.



currently outperforms more complex approaches, such as that proposed by Desai et al.[8]

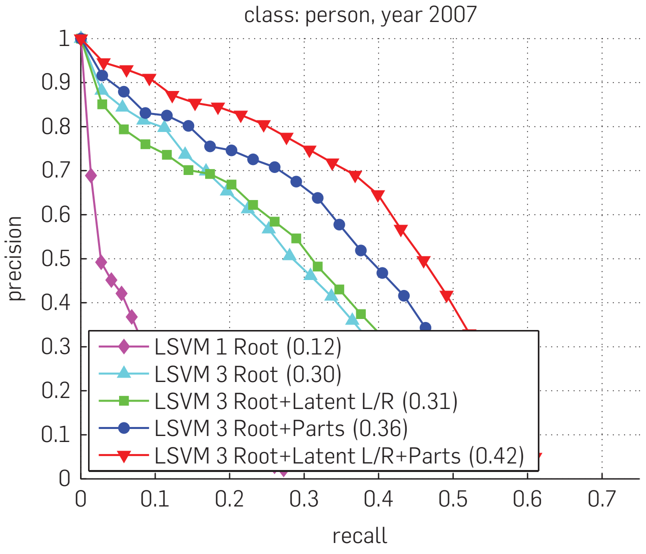
We evaluated different aspects of our system on the longer-established PASCAL VOC 2007 dataset. Figure 6 summarizes results of different models for the person category. We trained models with 1 and 3 components, with and without parts, and forcing mirror symmetry in each component or allowing for asymmetric models. We see that the use of parts can significantly improve the detection accuracy. Mixture models are also very important in the

Table 1. ***PASCAL VOC 2010 results.***

	Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	Mbike	Person	Plant	Sheep	Sofa	Train	TV
Base[a]	47.2	50.8	8.6	12.2	32.2	48.9	44.4	28.1	13.6	22.7	11.3	17.4	40.4	47.7	44.4	7.6	30.0	17.3	38.5	34.3
BB[b]	48.7	52.0	8.9	12.9	32.9	51.4	47.1	29.0	13.8	23.0	11.1	17.6	42.1	49.3	45.2	7.4	30.8	17.1	40.6	35.1
Context[c]	52.4	54.3	13.0	15.9	35.1	54.2	49.1	31.8	15.5	26.2	13.5	21.5	45.4	51.6	47.5	9.1	35.1	19.4	46.6	38.0
Best[d]	58.4	55.3	19.2	21.0	**35.1**	55.5	**49.1**	47.7	20.0	31.5	27.7	37.2	51.9	56.3	**47.5**	13.0	37.8	33.0	50.3	41.9

[a] Score of our base system.
[b] The system with bounding box prediction.
[c] The final system with context rescoring.
[d] The highest score over all systems entered into the 2010 competition (bolded numbers indicate that our system obtained the highest score).

Figure 6. ***Precision/Recall curves for models trained on the person category of the PASCAL 2007 dataset.*** **We show results for 1- and 3-component models with and without parts. For the 3-component models, we show results where the models are forced to be symmetric and where the models are allowed to be asymmetric and left vs. right orientation is treated as a latent variable during both training and testing ("Latent L/R"). In parentheses we show the average precision score for each model.**



person category where there are many examples of people truncated at various heights (e.g., by desks). Allowing for asymmetric models, where the object's facing direction is treated as a latent variable, produces a very small change when working with root-filter only models. However, after adding parts, latent direction yields a significant improvement.

6. DISCUSSION

Object detection is difficult because instances can vary greatly in appearance and because objects tend to appear in cluttered backgrounds. Latent-variable models provide a natural formalism for dealing with appearance variation. This represents a departure from other approaches that rely primarily on invariant features.[26] Rather, we find that a combination of both approaches, namely a latent variable model built on top of an invariant local image descriptor,[7] works quite well. Our models are robust to cluttered backgrounds by way of their discriminative training. This requires using a very large number of negative training examples to emulate the distribution of positive and negatives encountered at test-time.

There is a rich body of work in the use of deformable models of various types for object detection, including several kinds of deformable template models (e.g., Cootes et al.,[4] Coughlan et al.,[5] Grenander et al.,[20] and Yuille et al.[33]) and a variety of part-based models (e.g., Amit and Trouve,[1] Burl et al.,[3] Crandall et al.,[6] Felzenszwalb and Huttenlocher,[14] Fergus et al.,[17] Fischler and Elschlager,[18] and Weber et al.[31]). Our models are based on the pictorial structures formulation from Felzenszwalb and Huttenlocher[14] and Fischler and Elschlager[18], which evaluates a dense set of possible part positions and scales in an image. We are able to do so in an efficient manner using the fast matching algorithms of Felzenszwalb and Huttenlocher.[14] Our approach differs from past work on deformable models with its use of highly engineered local features[7] and weakly supervised discriminative learning algorithms.

The work described here was originally published in Felzenszwalb et al.[12] and Felzenszwalb et al.[16] with associated code releases available online.[11] We have extended this work in a variety of ways. In Felzenszwalb et al.[10] we explored cascade classifiers that evaluate the filters in a deformable part model sequentially and prune the computation using intermediate thresholds. This approach results in an order-of-magnitude speedup and real-time performance with little loss in accuracy. In Felzenszwalb and McAllester[15] and Girshick et al.,[19] we pursued grammar-based models that generalized deformable part models to allow for objects with variable structure, mixture models at the part level and reusability of parts across components and object classes. Finally, our method is still limited somewhat by its sensitivity to initialization. One approach to reducing this sensitivity is to use partially or fully annotated data with part and mixture labels. Our recent work has shown that one can use such a framework to achieve competitive results for facial analysis[34] and articulated pose estimation.[32]

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS 0746569, IIS 0811340 and IIS 0812428.

References

1. Amit, Y., Trouve, A. POP: Patchwork of parts models for object recognition. *Int. J. Comput. Vis. 75*, 2 (2007), 267–282.
2. Andrews, S., Tsochantaridis, I., Hofmann, T. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems* (2003), volume 15.
3. Burl, M., Weber, M., Perona, P. A probabilistic approach to object recognition using local photometry and global geometry. In *European Conference on Computer Vision* (1998).
4. Cootes, T., Edwards, G., Taylor, C. Active appearance models. *IEEE Trans. Pattern Anal. Mach. Intell. 23*, 6 (2001), 681–685.
5. Coughlan, J., Yuille, A., English, C., Snow, D. Efficient deformable template detection and localization without user initialization. *Comput. Vis. Image Understand. 78*, 3 (2000), 303–319.
6. Crandall, D., Felzenszwalb, P., Huttenlocher, D. Spatial priors for part-based recognition using statistical models. In *IEEE Conference on Computer Vision and Pattern Recognition* (2005).
7. Dalal, N., Triggs, B. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (2005).
8. Desai, C., Ramanan, D., Fowlkes, C. Discriminative models of multi-class object layout. *Int. J. Comput. Vis. 95*, 1 (2011), 1–12.
9. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A. The PASCAL Visual Object Classes Challenges. http://www.pascal-network.org/challenges/VOC/index.html.
10. Felzenszwalb, P., Girshick, R., McAllester, D. Cascade object detection with deformable part models. In *IEEE Computer Vision and Pattern Recognition* (2010).
11. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D. Discriminatively trained deformable part models. http://people.cs.uchicago.edu/~pff/latent/.
12. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D. Object detection with discriminatively trained part based models. *IEEE Trans. Pattern Anal. Mach. Intell. 32*, 9 (2010), 1627–1645.
13. Felzenszwalb, P., Huttenlocher, D. Distance transforms of sampled functions. Technical Report 2004–1963, CIS Dept., Cornell University, 2004.
14. Felzenszwalb, P., Huttenlocher, D. Pictorial structures for object recognition. *Int. J. Comput. Vis. 61*, 1 (2005), 55–79.
15. Felzenszwalb, P., McAllester, D. Object detection grammars. Technical Report TR-2010–02, CS Dept., University of Chicago, 2010.
16. Felzenszwalb, P., McAllester, D., Ramanan, D. A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition* (2008).
17. Fergus, R., Perona, P., Zisserman, A. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition* (2003).
18. Fischler, M., Elschlager, R. The representation and matching of pictorial structures. *IEEE Trans. Comput. C-22*, 1 (1973), 67–92.
19. Girshick, R., Felzenszwalb, P., McAllester, D. Object detection with grammar models. In *Advances in Neural Information Processing Systems* (2011), volume 24.
20. Grenander, U., Chow, Y., Keenan, D. *HANDS: A Pattern-Theoretic Study of Biological Shapes*, Springer-Verlag, 1991.
21. Huttenlocher, D., Klanderman, G., Rucklidge, W. Comparing images using the hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell. 15*, 9 (1993), 850–863.
22. Lamdan, Y. Wolfson, H. Geometric hashing: A general and efficient model-based recognition scheme. In *IEEE International Conference on Computer Vision* (1988).
23. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE 86*, 11 (1998), 2278–2324.
24. Lowe, D. Three-dimensional object recognition from single two-dimensional images. *Artif. intell. 31*, 3 (1987), 355–395.
25. Marr, D., Nishihara, H. Representation and recognition of the spatial organization of three-dimensional shapes. *Proc. Roy. Soc. Lond. B Biol. Sci. 200*, 1140 (1978), 269–294.
26. Mundy, J., Zisserman, A., et al. *Geometric Invariance in Computer Vision*, volume 92, MIT press, Cambridge, MA, 1992.
27. Murase, H., Nayar, S. Visual learning and recognition of 3-d objects from appearance. *Int. J. Comput. Vis. 14*, 1 (1995), 5–24.
28. Schneiderman, H., Kanade, T. A statistical method for 3D object detection applied to faces and cars. In *IEEE Conference on Computer Vision and Pattern Recognition* (2000).
29. Sung, K.K., Poggio, T. Example-based learning for view-based human face detection. *IEEE Trans. Pattern Anal. Mach. Intell. 20*, 1 (1998), 39–51.
30. Viola, P., Jones, M. Robust real-time face detection. *Int. J. Comput. Vis. 57*, 2 (2004), 137–154.
31. Weber, M., Welling, M., Perona, P. Towards automatic discovery of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition* (2000).
32. Yang, Y., Ramanan, D. Articulated pose estimation using flexible mixtures of parts. In *IEEE Conference on Computer Vision and Pattern Recognition* (2011).
33. Yuille, A., Hallinan, P., Cohen, D. Feature extraction from faces using deformable templates. *Int. J. Comput. Vis. 8*, 2 (1992), 99–111.
34. Zhu, X., Ramanan, D. Face detection, pose estimation, and landmark localization in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition* (2012).

Pedro Felzenszwalb School of Engineering and Department of Computer Science, Brown University.

Ross Girshick EECS, UC Berkeley.

David McAllester Toyota Technological Institute at Chicago.

Deva Ramanan Department of Computer Science, UC Irvine.

California State University, East Bay (Hayward, CA)

Department of Mathematics and Computer Science
Faculty Position in Computer Science
Assistant Professor

POSITION (**OAA Position 14-15 MCS-COMPUTATIONAL-TT) The Department invites applications for a tenure-track appointment as Assistant Professor in Computer Science (preference to theory) starting Fall 2014.**

Teaching includes day and evening courses at B.S. and M.S. levels, with a typical teaching responsibility of three classes/Quarter. Required: potential for excellent teaching, research and curriculum development; ability to teach, advise and mentor students from diverse backgrounds; potential to serve Department and University. Applicants must have a Ph.D. by September, 2014.

Please submit an application letter and a current and complete vita through the following URL:
https://my.csueastbay.edu/psp/pspdb1/
EMPLOYEE/HRMS/c/HRS_HRAM.HRS_
CE.GBL

Additionally, please email graduate transcripts, 3 letters of recommendation, 3 references, a statement of teaching philosophy, and evidence of teaching and research abilities to the Computer Science Search Committee at the email address below.

A detailed position announcement is available at:
http://www20.csueastbay.edu/about/career-opportunities/

For questions, email:
CSSearch@mcs.CSUEastBay.edu

APPLICATION DEADLINE **Review of applications will begin November 1, 2013.**

CSUEB, situated in the hills overlooking San Francisco Bay, is an EOE, committed to "educational excellence for a diverse society".

Princeton University

Computer Science
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. Candidates should expect to receive their PhD before Fall, 2014. 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 be underway by December 2013. Applications in the area of Machine Learning are particularly encouraged and are strongly urged to apply by November 1, 2013.

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/.

Stanford University

Graduate School of Business
Faculty Positions in Operations, Information and Technology

The Operations, Information and Technology (OIT) area at the Graduate School of Business,

TU WIEN
FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Announcement of an open position at the Faculty of Informatics, Vienna University of Technology, Austria

FULL PROFESSOR (TENURED) of SECURITY

The Vienna University of Technology invites applications for a Full Professor position (tenured) at the Faculty of Informatics.

The successful candidate will have an outstanding research record in the field of computer security. He/She should have a comprehensive research agenda, and his/her work ideally spans multiple subareas of the field. We expect the successful candidate to establish links to several of the main research areas of the Faculty of Informatics (Computational Intelligence, Computer Engineering, Distributed and Parallel Systems, Media Informatics and Visual Computing, as well as Business Informatics).

We offer excellent working conditions in an attractive research environment in a city with an exceptional quality of life.

A more detailed announcement and information on how to apply can be found at:
http://www.informatik.tuwien.ac.at/security

Application Deadline: **October 20, 2013**

Technical University of Denmark DTU

ASSOCIATE/ASSISTANT PROFESSOR

Language Based Technology

DTU Compute. Applications are invited for one or more positions as Associate or Assistant Professor. The successful candidate will be part of an international team focusing on a broad portfolio of research projects using formal methods for modeling and analysis of systems (MT-LAB, IDEA4CPS, TREsPASS, FutureID, SESAMO, PaPP). The candidate is expected to strengthen the research profile of the section within these areas.

Application deadline: **1 October 2013**

DTU is a technical university providing internationally leading research, education, innovation and public service. Our staff of 5,000 advance science and technology to create innovative solutions that meet the demands of society; and our 7,600 students are educated to address the technological challenges of the future. DTU is an independent academic university collaborating globally with business, industry, government, and public agencies.

Further details: career.dtu.dk

SCIENCE AND TECHNOLOGY AT A GLOBAL SCALE
- SET THE STANDARDS FOR THE FUTURE
See our PhD-programmes at dtu.dk/phd

Stanford University, is seeking qualified applicants for full-time, tenure-track positions, starting in the 2013-2014 academic year. All ranks and relevant disciplines will be considered. Applicants are considered in all areas of Operations, Information and Technology (OIT) that are broadly defined to include the analytical and empirical study of technological systems, in which technology, people, and markets interact. It thus includes operations, information systems/technology, and management of technology. Applicants are expected to have rigorous training in management science, engineering, computer science, economics, and/or statistical modeling methodologies. The appointed will be expected to do innovative research in the OIT field, to participate in the school's PhD program, and to teach both required and elective courses in the MBA program. Junior applicants should have or expect to complete a PhD by September 1, 2014.

Applicants are ***strongly encouraged*** to submit their applications electronically by visiting the web site http://www.gsb.stanford.edu/recruiting and uploading their curriculum vitae, research papers and publications, and teaching evaluations, if applicable, on that site. Alternatively, all materials may be sent by e-mail to faculty_recruiter@gsb.stanford.edu, or by postal mail (non-returnable) to Office of Faculty Recruiting, Graduate School of Business, Stanford University, 655 Knight Way Way, Stanford, CA 94305-7278. However, submissions via e-mail and postal mail can take 4-6 weeks for processing. For an application to be considered complete, each applicant must have three letters of recommendation emailed to the preceding email address, or sent via postal mail. **The application deadline is November 15, 2013.**

Stanford University is an equal opportunity employer and is committed to increasing the diversity of its faculty. It welcomes nominations of and applications from women and members of minority groups, as well as others who would bring additional dimensions of diversity to the University's research, teaching and clinical missions.

ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to *acmmediasales@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 by typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

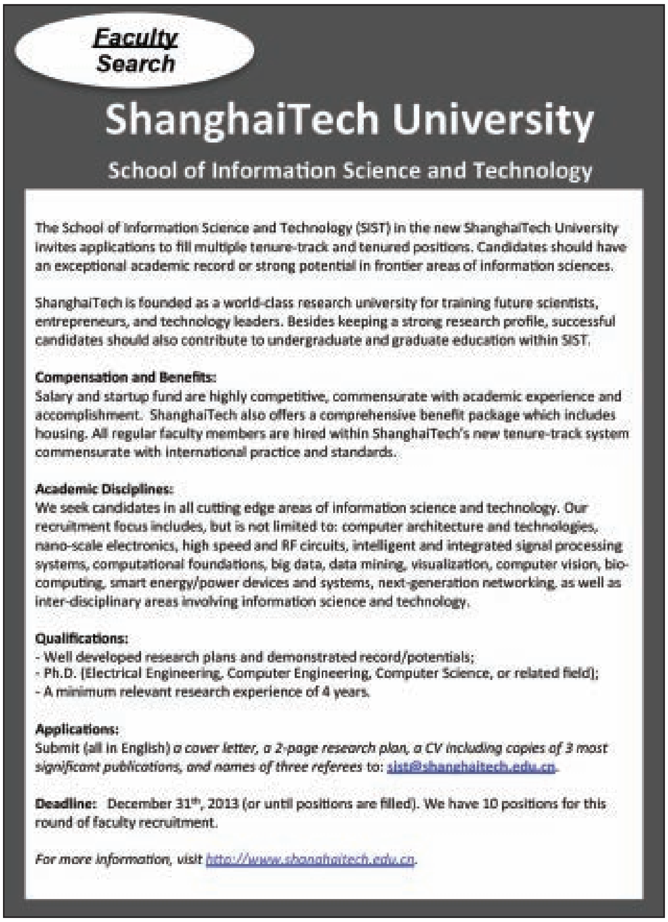
Rates: $325.00 for six lines of text, 40 characters per line. $32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact:
acmmediasales@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
acmmediasales@acm.org



Faculty Search

ShanghaiTech University

School of Information Science and Technology

The School of Information Science and Technology (SIST) in the new ShanghaiTech University invites applications to fill multiple tenure-track and tenured positions. Candidates should have an exceptional academic record or strong potential in frontier areas of information sciences.

ShanghaiTech is founded as a world-class research university for training future scientists, entrepreneurs, and technology leaders. Besides keeping a strong research profile, successful candidates should also contribute to undergraduate and graduate education within SIST.

Compensation and Benefits:
Salary and startup fund are highly competitive, commensurate with academic experience and accomplishment. ShanghaiTech also offers a comprehensive benefit package which includes housing. All regular faculty members are hired within ShanghaiTech's new tenure-track system commensurate with international practice and standards.

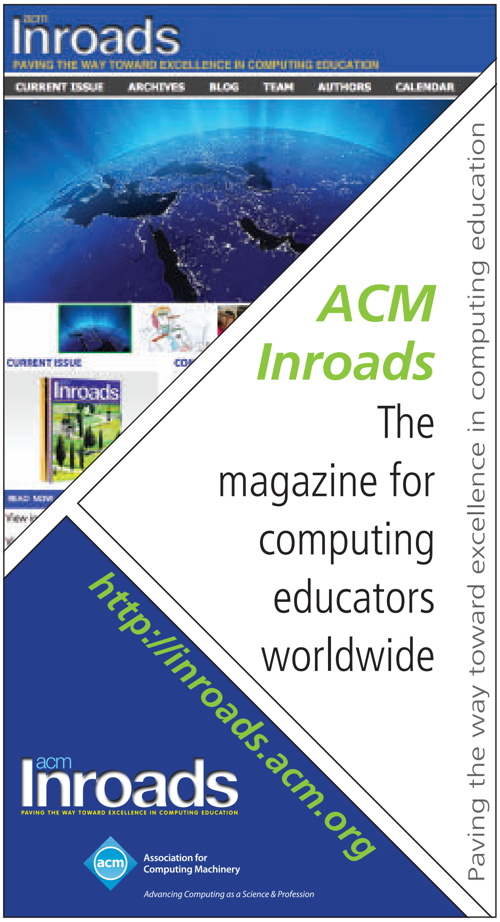
Academic Disciplines:
We seek candidates in all cutting edge areas of information science and technology. Our recruitment focus includes, but is not limited to: computer architecture and technologies, nano-scale electronics, high speed and RF circuits, intelligent and integrated signal processing systems, computational foundations, big data, data mining, visualization, computer vision, bio-computing, smart energy/power devices and systems, next-generation networking, as well as inter-disciplinary areas involving information science and technology.

Qualifications:
- Well developed research plans and demonstrated record/potentials;
- Ph.D. (Electrical Engineering, Computer Engineering, Computer Science, or related field);
- A minimum relevant research experience of 4 years.

Applications:
Submit (all in English) *a cover letter, a 2-page research plan, a CV including copies of 3 most significant publications, and names of three referees* to: sist@shanghaitech.edu.cn.

Deadline: December 31^{th}, 2013 (or until positions are filled). We have 10 positions for this round of faculty recruitment.

For more information, visit http://www.shanghaitech.edu.cn.


acm Inroads
PAVING THE WAY TOWARD EXCELLENCE IN COMPUTING EDUCATION
CURRENT ISSUE
ARCHIVES
BLOG
TEAM
AUTHORS
CALENDAR
ACM Inroads
The magazine for computing educators worldwide
http://inroads.acm.org
Paving the way toward excellence in computing education
acm Inroads
PAVING THE WAY TOWARD EXCELLENCE IN COMPUTING EDUCATION
acm
Association for Computing Machinery
Advancing Computing as a Science & Profession

DOI:10.1145/2500468.2500479

Peter Winkler

Puzzled
Solutions and Sources

Last month (Aug. 2013) you needed to win several chess games in a row, alternately playing white and black, and had to decide with which color you should start.

The first puzzle was adapted from Martin Gardner's *Colossal Book of Short Puzzles & Problems* (W. W. Norton & Co., New York, 2006), Problem 2.10. In his solution, Gardner observed since you need to win two in a row of three games, you must win the middle game; moreover, you must win a game with the black pieces, so having two chances to do it seemed like a good idea. There are two arguments for playing black first, even though it means you get to play white only once. In fact, neither argument is conclusive by itself, nor do they constitute a proof even together. Gardner offered an algebraic proof that starting with black is best, but extending it to the second and third puzzles would be too horrible to contemplate. Is there a watertight argument that circumvents algebra? Yes. Probabilists sometimes use a technique called "coupling" in which random events are tied to the same experiment, and it can be used to good effect here.

1. White or black?

Imagine you play *four* games against Ioana, alternating white-black-white-black, or WBWB. You must still win two in a row and decide ahead of time whether to discount the first game or the last. This question is clearly equivalent to the earlier one, but now you are in a position to limit the discussion to outcomes in which the decision makes a difference: namely, WWLX and XLWW. In words, what you decided matters only if you win the first two games and lose the third (in which case you should have discounted the fourth) or if you win the last two games and lose the second (in which case you should have discounted the first). The first two and last two games are both WB, so it comes down to comparing the probability of losing the third game with the probability of losing the second game. Since you are black in the second game, the second scenario is more likely, so discount the first game; that is, play BWB.

2. Still need two in a row.

Here you play 17 games, still needing two wins in a row. Note if you start with black you are black, not white, in the middle (ninth) game. Does that change the answer? No. Assume you actually play 18 games, WBWB…WB, and must decide in advance whether to discount the first or last game. Your decision matters only when you win the first two games, lose the third, and win no other two in a row (in which case, you should have discounted the last game) or you win the last two, lose the 15th, and win no other two in a row (in which case you should have discounted the first game). It is easy (but not as easy as before) to see the second scenario as more likely, so again start with black.

3. 10 in a row.

Here you must win 10 in a row out of 49 games, but the argument hardly changes. Begin with black. In general, if the total number of games played is even, time-symmetry implies it does not matter whether you began with white or black. If the total games played *and* the number of consecutive wins needed are both odd, then, if you made it this far, you should have no trouble showing one should always start with white.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science, at Dartmouth College, Hanover, NH.

[CONTINUED FROM P. 112] scalability was a real challenge. Our indexing and serving systems had to rework things very quickly in order to both be able to update the index files and deal with the queries coming in. The indexing system starts with a bunch of pages collected from the Internet. Eventually, you want to end up with an inverted index where words map to the documents that contain them, along with a bunch of other information about those documents: things like page rank of the document, what language the document is in, and then you want to eliminate duplicates … so there was this whole collection of operations you have to perform, starting with raw document contents that we had crawled on disk, and ultimately ending up with an inverted index and other data structures needed for handling search requests.

SANJAY: Each of these operations had to process a lot of data, so we had to divide up the work across many machines so it would finish in a reasonable amount of time. This division of work was very boring and repetitive. We had to implement the same boilerplate for every new data processing task. That's where MapReduce came in. We abstracted the repetitive parts into a library and allowed the author of the new data processing task to just plug in the specific operations they wanted to apply to their data; the MapReduce library took care of the rest.

JEFF: It was originally done in the context of these eight or 10 phases of the indexing system, but as we looked around, we realized it was much more general-purpose.

One thing that makes it so versatile is that the programmer doesn't have to worry about how to map his or her computations across the processors.

JEFF: MapReduce makes it easy for people without a lot of systems experience to get the answers they want, without being experts in how you automatically parallelize computations or handle failures.

Speaking of failures, you've called MapReduce a software answer to a hardware problem. Can you elaborate?

SANJAY: If you have 1,000 servers, then three of them are dying every day. So if you're running a MapReduce computation across thousands of machines, given the law of large numbers, chances are that something will go wrong. We could try to make the underlying hardware more reliable, but I think the costs are against that.

> **"MapReduce makes it easy for people without a lot of systems experience to get the answers they want."**

So because MapReduce spreads computations across so many machines, it doesn't matter if one or two fail.

JEFF: Doing recovery in software means that if a particular machine dies after it has done 100 pieces of work, you can map each of those 100 pieces of work to one of 100 other machines. The recovery is very fast because it can happen with a high degree of parallelism.

SANJAY: And there are other reasons to make error-handling a central part of these systems, because you can leverage it for other things. So consider a job scheduling system that is handling jobs submitted by many people. When a new job comes in, the job scheduler might kick out parts of other jobs to make room for the new job, so the old job has to be prepared to handle failures introduced by this preemption. If the old job is a MapReduce, the MapReduce library will deal with these preemptions automatically.

After your 2004 paper, MapReduce inspired the open source system that became Hadoop, along with countless other projects. What about internally?

JEFF: It's used in thousands of ways in hundreds of products and underlying systems. Lots of batch-oriented computations, processing things like logs data, web pages, images, satellite imagery, source code, etc., in order to compute summary or derived information of various kinds. It's often used in multistage pipelines with different MapReduce operations forming different stages of the pipeline.

Tell me about BigTable, a database that spreads rows of data across multiple machines.

SANJAY: So the initial motivation for BigTable was the cycle of periodic crawling and indexing. In particular, we wanted to make things a lot more real time. When a page goes up on the Web, you want it to be searchable quickly. Every month we would crawl and index from scratch and that would take a while. We wanted to reduce that delay.

JEFF: If the row is the URL, there are a bunch of different columns with information about that URL. Then you basically have all of the information about a page, and you can, with very low latency, crawl a new version of the page, and re-index the page with all the information we know about it in an incremental fashion, rather than waiting for the next large batch update.

In 2007, Jeff served as the inspiration for a popular April Fool's gag—a collection of statements modeled after Chuck Norris Facts, like "Jeff Dean once failed a Turing Test when he correctly identified the 203rd Fibonacci number in less than a second." Sanjay, do you have a favorite Jeff Dean fact?

SANJAY: Hmm, let me think. Jeff is a fast typist, but he's also a very hard typist, so—

JEFF: I wear out keyboards fairly quickly.

SANJAY: So we were working together and all of a sudden, from the next office, we heard: "Is it raining?" "No, that's just Jeff typing."

Leah Hoffmann is a technology writer based in Piermont, NY.

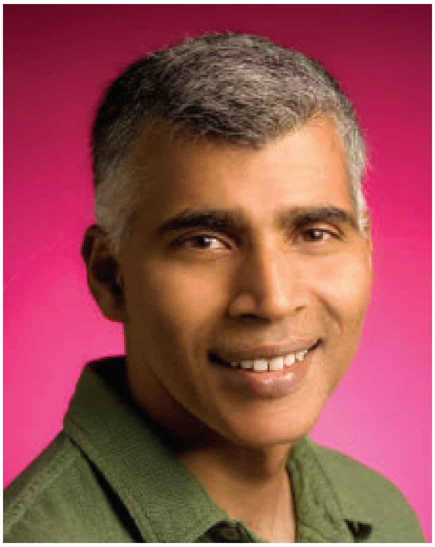
DOI:10.1145/2500468.2500478

Leah Hoffmann

Q&A
Big Challenge

Jeff Dean and Sanjay Ghemawat talk about scalability.

AFTER MEETING IN the 1990s at Digital Equipment Corporation (DEC) and forging a productive friendship at a gelato stand between their two labs, ACM-Infosys Foundation Award in the Computing Sciences recipients Jeff Dean and Sanjay Ghemawat moved to Google where, for nearly 15 years—and often coding at the same computer—they have transformed Internet-scale computing. Spurred by the challenge of handling an ever-growing volume of web pages and search requests, the two built scalable computing platforms that distributed computations across thousands of servers, and have since been leveraged by thousands of projects both inside and beyond Google.



Sanjay Ghemawat

Jeff Dean

I understand you met at Digital Equipment Corporation.

JEFF: We both started at DEC within about a year of each other. Sanjay was at the Systems Research Center, and I was at the Western Research Lab. The two labs were conveniently separated by a gelato stand. We initially started collaborating on a project to build a low-overhead profiling system.

SANJAY: Then we moved on to an optimizing compiler for Java.

JEFF: I got interested in a side project working on information retrieval. A colleague had built a system that kept the entire graph of all the connectivity structure from an AltaVista crawl in memory, and built a simple API so you could see what pages pointed to which other pages—and more importantly, which pages pointed to a given page. So I started looking into the link structure of the Web, and I decided to leave the research lab. Two months later, I told Sanjay he should come to Google, too.

At Google, you share an office, and you even code together.

JEFF: We've shared an office for most of the time that we've been here, but we were actually coding together at DEC. We've been doing that for a number of years. We work really well in that mode because we each understand where the other one is going with an idea, both on a very small scale—like how we should implement some data structure that we need—and at the large scale of a big system. It's a very fluid style.

“We work really well in that mode because we each understand where the other is going with an idea.”

SANJAY: It's all over the place. Jeff has a lot of energy and excitement. We usually sit, and one of us is typing and the other is looking on, and we're chatting all the time about ideas, going back and forth.

Let's talk about scalability. One of your best-known projects at Google is MapReduce, which enables programmers to spread computations across multiple machines.

JEFF: For the first few years we were at Google,

[CONTINUED ON P. 111]

IUI 2014

Haifa, Israel | February 24-27, 2014 | www.iuiconf.org

2014 INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES

Conference Committee

General Chairs
Tsvi Kuflik, The University of Haifa
Oliviero Stock, FBK-irst

Program Chairs
Joyce Chai, Michigan State University
Antonio Kruger, Saarland University and DFKI GmbH

Treasurer
Melinda Gervasio, SRI International

Workshop Chairs
Ido Guy, IBM Research - Haifa
Tracy Hammond, Texas A&M University

Demo Chairs
Doron Friedman, Interdisciplinary Center (IDC) Herzliya
Massimo Zancanaro, FBK-irst

Industry Track Chairs
Claudia Goldman, General Motors - Advanced Technical Center - Israel
Doron Friedman, Interdisciplinary Center (IDC) Herzliya
Massimo Zancanaro, FBK-irst

Students Consortium Chairs
Shlomo Berkovsky, NICTA
Helmut Prendinger, National Institute of Informatics, Tokyo

Social Media Chair
Angel Puerta, RedWhale Software

Local Arrangement Chairs
Joel Lanir, The University of Haifa
Eran Toch, Tel Aviv University

Registration Chair
Amit Tiroshi, The University of Haifa

Publicity Chair
Nava Tintarev, University of Aberdeen

Sponsorship Chair
Doug Riecken, Columbia University

Sponsors

IUI 2014 is the 19th annual meeting of the intelligent interfaces community and serves as the principal international forum for reporting outstanding research and development of intelligent user interfaces.

IUI is where the community of people interested in Human-Computer Interaction (HCI) meets the Artificial Intelligence (AI) community. We are very interested in contributions that bridge these two fields and also related fields, such as psychology, cognitive science, computer graphics, the arts, etc. IUI researchers are interested in improving the symbiosis between humans and computers, so that interface design and interactive experiences yield higher performance outcomes. This may involve designing interfaces that incorporate intelligent automated capabilities, if the net impact is a human-computer interaction that improves performance or usability in critical ways. It may also involve designing an interface that effectively leverages human skills and capabilities, so that human performance with an application excels. In other cases, such as educational interface design, it may involve exercising judgment in when not to automate a function so that humans are encouraged to exert themselves as they acquire new skills or domain knowledge.

We call for original submissions that describe novel technologies and applications to intelligent user interfaces.

Submission Venues

Full and Short Papers

We invite original paper submissions that describe novel user interfaces, applications, interactive and intelligent technologies, empirical studies, or design techniques. Accepted papers will be published in the ACM Digital Library. IUI 2014 especially encourages submissions on innovative and visionary new concepts or directions for interface design. We do not require evaluations with users, but we do expect papers to include an appropriate evaluation for their stated contribution.

Full papers should make substantial, novel and relevant contributions to the field. Short papers can either contain smaller contributions, novel ground-breaking ideas, or work in progress. Accepted full papers will be invited for oral presentation and short papers will be presented orally or as posters.

Demonstrations

The demonstrations track complements the overall program of the conference. Demonstrations show implementations of novel and interesting intelligent user interface concepts or systems. We invite submissions relevant to intelligent user interfaces and which address, but are not limited to, the topics of the conference. All submissions are intended to convey scientific results or work in progress and should not be advertisements for commercial software packages.

Industrial Track

The industrial track represents an opportunity for the companies to present their late works (as presentations, posters or interactive demos) and to receive valuable feedback from the research community. It also provides an opportunity for recruitment and networking. Submissions in the form of 1-page expression of interested proponents are welcome before November 15, 2013. The industry track will not be peer-reviewed but the relevance of proposals for the field will be judged by a jury. Submissions should be sent to the industry chairs address and contain a short description of the company highlighting the relevance in the fields of AI and UI together with a short description of the talk or the demo.

Workshops

Workshops will be held on the first day of the conference. We invite submissions of full-day (6 hours) and half-day (3 hours) workshop proposals on any of the conference topics.

Student Consortium

The IUI 2014 Student Consortium provides an opportunity for students to present and receive feedback about their research in an interdisciplinary workshop, under the guidance of a panel of mentors, selected from senior people in the field. We invite students to apply for this unique opportunity to share their work with students in a similar situation as well as senior researchers in the field.

Important Dates

Papers: (abstracts)	October 4, 2013
(final papers due)	October 9, 2013
Demonstrations:	December 12, 2013
Workshops:	September 16, 2013
Student Consortium:	November 1, 2013
Industrial Track:	Contact track chairs

Topics

User input
Processing of multi-modal input
Natural language and speech processing
Gestures, eye gaze, face recognition

Generation of system output
Intelligent visualization tools
Intelligent generation of user-consumable content

Ubiquitous computing
Intelligent interfaces for ubiquitous computing
Smart environments and tangible computing

Help and Persuasive Technologies
Intelligent assistants for complex tasks
Support for collaboration in multiuser environments
Intelligent information and knowledge management
Persuasive technologies in IUI

Personalization
User-adaptivity in interactive systems
Recommender systems
Modeling and prediction of user behavior

AI Techniques in IUI
Planning and plan recognition
Reasoning in interfaces
Knowledge-based systems

Social Computing
Affective, social and aesthetic interfaces
Social networks and collaboration

IUI Design
Knowledge-based approaches to user interface design and generation
Proactive and agent-based paradigms for user interaction
Example- and demonstration-based interfaces

User studies
User studies concerning intelligent interfaces
Evaluations of implemented intelligent user interfaces

Semantic Web
Query interfaces and novel interfaces for Linked Data
Consuming Linked Data
Interfaces for creating and using large ontologies