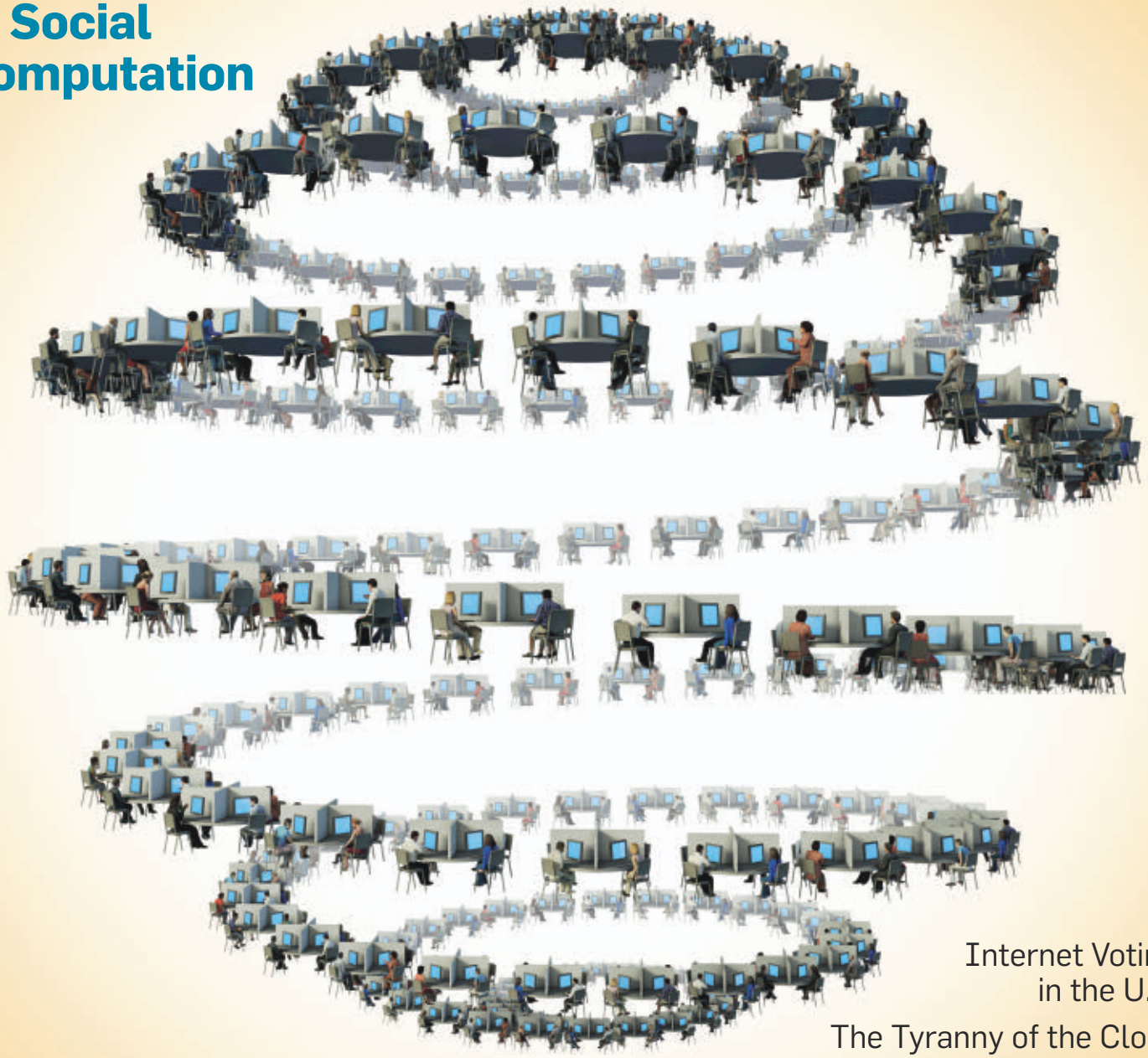
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

10/2012 VOL.55 NO.10

Experiments in Social Computation



Internet Voting in the U.S.

The Tyranny of the Clock

Redesigning the Data Center

Things to Know About Machine Learning

Where is the Science in Computer Science?

Association for Computing Machinery

The 20th Anniversary ACM Multimedia 2012

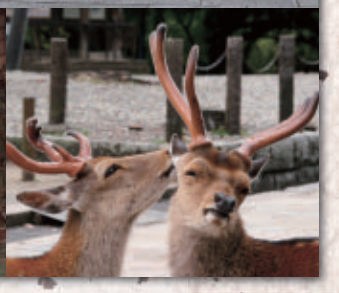
NARA

October 29 – November 2, 2012

ACM Multimedia is the premier conference and worldwide event bringing together multimedia experts and practitioners across academia and industry. The central feature of the conference, which continues this year as in every year since its inception, is the outstanding Technical Program. This year's conference features both oral and poster presentations covering all aspects of the multimedia field chosen through a highly selective review process. Notably, this year's conference includes special Technical Program activities recognizing the 20th anniversary of ACM Multimedia.

In addition to the Technical Program, this year's conference features a diverse range of activities including Panels, Demonstrations and Tutorials. Additionally, a wide array of Workshops brings focus on new topics for investigation. The conference features also special sessions on Brave New Ideas, a Grand Challenge contest and an Open Source Software Competition and includes a Doctoral Symposium for mentoring graduate students. Finally, the conference provides a rich Multimedia Art Exhibition to stimulate artists and researchers alike to meet and discover the frontiers of multimedia artistic communication!

http://www.acmmm12.org/



20th Anniversary Keynote Talk

Future Direction of Digital Content

Masahiro Fujita (Sony Corporation, JP)

20th Anniversary Panel

Coulda, Woulda, Shoulda: 20 Years of Multimedia Opportunities

Organizers

Klara Nahrstedt (Univ. of Illinois at Urbana-Champaign, US)
Malcolm Slaney (Microsoft, US)

Panelists

Dick Bulterman (CWI / Vrije Univ., NL)
Ramesh Jain (Univ. of California, Irvine, US)
Larry Rowe (Univ. of California, Berkeley / FXPAL, US)
Ralf Steinmetz (Darmstadt Univ. of Technology, DE)

Multimedia Art Exhibition 2012 *Eternal / Moment*

Multimedia Art Exhibition 2012 entitled as ***Eternal / Moment*** presents integrated perspectives spanning art, entertainment and culture. It shows this year's fruitful juried/curated works in multiple media forms; hybrid art, zero-gravity-space art, installations and interactive media, digital communities, mobile media, and aesthetic/scientific exploration.

Art Venue
Todaiji Culture Center
Oct.20–Nov.4

empathetic heartbeat

©Hideyuki Ando, Junji Watanabe, Masahiko Sato

2kg Deer

©Satoru Tamura

The Silent Power

©Shih-Ting Tsa

acm Association for Computing Machinery

SIGMM

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, 2012** 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 2013.

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/html/dda.cfm

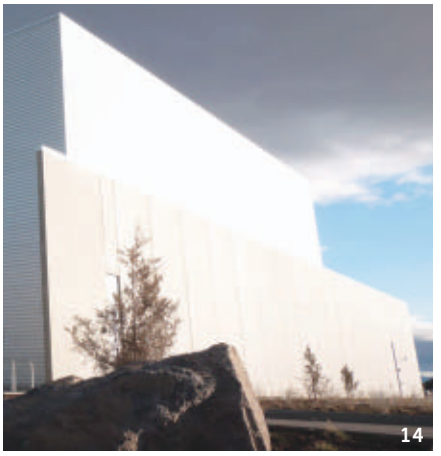
Association for Computing Machinery

COMMUNICATIONS OF THE ACM

Departments

Last Byte

News



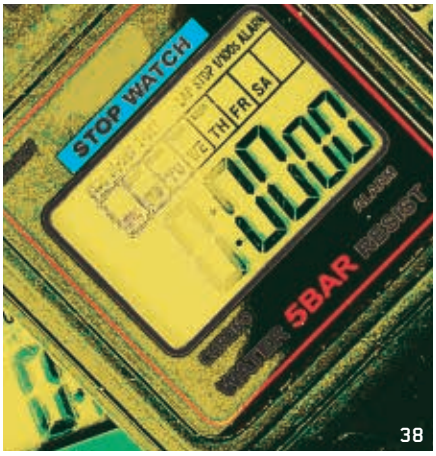
Viewpoints

Association for Computing Machinery
Advancing Computing as a Science & Profession

PHOTOGRAPH BY CHUCK GOOLSBEE

10/2012
VOL. 55 NO. 10

Practice

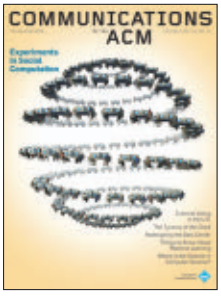


Articles' development led by acmqueue
queue.acm.org

Contributed Articles

Review Articles

Research Highlights



About the Cover:
This month's cover story (p. 56) shares results from an array of experiments that illustrate the power of social computation; showing ways groups of human subjects are able to solve challenging tasks from only local, networked interactions. Cover illustration by Randy Lyhus.

ILLUSTRATION (ON RIGHT) BY RANDY LYHUS

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.

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.

DOI:10.1145/2347736.2347737

Vinton G. Cerf

Where is the Science in Computer Science?

We are all members of the Association for Computing Machinery. Sounds sort of electromechanical doesn't it? Given today's computing technology, it is probably a good

thing we are mostly known as ACM! There was a time when the physical artifact—the computer—really was the focus of attention. These behemoths occupied rooms full of equipment. Now, in fairness, if you have ever visited a cloud computing data center, the dominant impression is still a (vast) room full of machinery. But we carry huge quantities of computing power in our pockets and purses too. Computing is a remarkable artifact and its origins centered on the ability to make a piece of equipment calculate under programmable control. Alan Turing, whose 100th birthday we celebrated this year, drew dramatic attention to the artificiality of these systems with what we now call the Universal Turing Machine. This conceptual artifact emphasizes the artificial nature of computation.

In the physical world, science is largely about models, measurement, predictions, and validation. Our ability to predict likely outcomes based on models is fundamental to the most central notions of the scientific method. The term "computer science" raises expectations, at least to my mind, of an ability to define models and to make predictions about the behavior of computers and computing systems. I think we have a fairly good capability to measure and predict the physical performance of our computing devices. We can measure clock speeds, latencies, memory sizes, and computational capacity against standard computing tasks. In my view, however, we are much less able to make models and predictions about the behavior and performance of the artifact we label "software." An almost flippant analogy is the difference between measuring, modeling, and predicting neural brain functions and trying to do the same for "thought."

That software is an artifact seems obvious. Moreover, it is a strikingly complex artifact filled with layer upon layer of components that exhibit dependencies and complex and often unpredicted (not to say unpredictable) behaviors. Even though we *design* software systems and ought to have some clues about how these systems behave and perform, we generally do not have a reliable ability to anticipate the states these systems can get into, their vulnerabilities, their performance, and ability to adapt to changing conditions.

When we write a piece of software, do we have the ability to predict how many mistakes we have made (that is, bugs)? Do we know how long it will take to find and fix them? Do we know how many new bugs our fixes will create? Can we say anything concrete about vulnerability? What about the probability of exploitation? Murphy's Law suggests that if there is a bug that can be exploited for nefarious purposes, it will be. ACM Turing Award recipient Fred Brooks' wonderful book, *The Mythical Man-Month*[1] captures some of the weakness of our understanding of the nature of software. A complementary look at this topic is found in ACM Turing recipient Herbert A. Simon's *The Sciences of the Artificial*.[2] Chapter 8 deals with hierarchy and complexity, touching on the way in which we try to bound complexity through modular and hierarchical structures but are still challenged by the emergent behaviors masked, in some ways, by the beguiling apparent simplicity of the hierarchy.

The richness of our field has only grown in the 65 years of our existence as an organization. Computers, computing, software, and systems are seemingly omnipresent. We are growing increasingly dependent upon what must be billions of lines of code. Some unknown wag once quipped that the only reason all the computers in the world have not failed at once is that they are not yet all on the Internet. But that may be coming (not the collapse, I hope, but the interconnection of a vast number of programmable devices through the Internet or its successor(s)).

As a group of professionals devoted to the evolution, understanding, and application of software and hardware to the myriad problems, opportunities, and activities of modern society, we have a responsibility to pursue the science in computer science. We must develop better tools and much deeper understanding of the systems we invent and a far greater ability to make predictions about the behavior of these complex, connected, and interacting systems. I consider membership in the ACM a mark of recognition of that responsibility. I hope you share that view and will encourage others in our profession to join ACM in the quest for the science in our discipline. C

References

1. Brooks, F.P. *The Mythical Man-Month*, Anniversary edition, 1995. Addison-Wesley, Reading, PA, ISBN 0-201-83595-9.
2. Simon, H.A. *The Sciences of the Artificial*, 3rd edition, 1996. MIT Press, Cambridge, MA, ISBN 0-262-19374-4.

Vinton G. Cerf is Vice President and Chief Internet Evangelist at Google Inc. and the president of ACM.

DOI:10.1145/2347736.2347738

When Harm to Conference Reputation Is Self-Inflicted

Citing conferences sponsored by the World Scientific and Engineering Academy and Society, Moshe Y. Vardi's Editor's Letter "Predatory Scholarly Publishing" (July 2012) reminded me of my own participation in the WSEAS flagship summer conference (the International Conference on Circuits, Systems, Communications and Computers) several years ago, contributing papers, tutorials, and even a plenary lecture, as an ad hoc replacement for a missing speaker. I recall WSEAS adopting a new policy saying papers would not be accepted for publication unless they included at least two references pointing to previous WSEAS proceedings or transactions. At first, I thought it odd that a scientific association would mandate self-citation to deliberately and artificially increase its citation impact but imagined it was simply common practice among conference organizers.

Visiting the Scholarly Open Access Web site (http://scholarlyoa.com) Vardi recommended, I realized that such a policy should indeed be viewed as harmful to an academic publisher's credibility and reputation. I would therefore like to thank Vardi for pointing out such publisher behavior contrary to the interests of all scholarly publishing. It is particularly important for those of us whose conference travel is not sponsored by governments and other institutions.

Miroslav Skoric, Novi Sad, Serbia

Don't Blame Modular Programming

In "Large-Scale Complex IT Systems" (July 2012) Ian Sommerville et al. reached unwarranted conclusions, blaming project failures on modular programming: "Current software engineering is simply not good enough." Moreover, they did so largely because they missed something about large-scale systems. Their term, "coalition," implies alliance and joint action that does not exist among real-world competitors. They said large-scale systems "coalitions" have different owners with possibly divergent interests (such as in the 2010 Flash Crash mentioned in the article) and then expect the software "coalition" used by the owners to work cooperatively and well, which makes no sense to me. Even if the owners, along with their best minds and sophisticated software, did cooperate to some extent, they would in fact be attempting to deal with some of the most difficult problems on earth (such as earning zillions of dollars in competitive global markets). Expecting software to solve these problems in economics makes no sense when even the most expert humans lack solutions.

Alex Simonelis, Montréal

Reading Ian Sommerville et al. (July 2012), I could not help but wonder whether new initiatives and institutions are really needed to study and create ultra/large-scale complex artificial systems. We should instead ponder how the behavior and consequences of such systems might be beyond our control and so should not exist in the first place. I am not referring to grand-challenge projects in science and engineering like space exploration and genomics with clear goals and benefits but the ill-conceived, arbitrary, self-interest-driven monstrosities that risk unpredictable behavior and harmful consequences. Wishful thinking, hubris, irresponsible tinkering, greed, and the quest for power drive them, so they should be seen not as a grand challenge but as a grand warning.

Why invent new, ultimately wasteful/destructive "interesting" problems when we could instead focus on the chronic "boring" deadly ones? From war, polluting transportation, and preventable disease to lack of clean water and air. These are real, not contrived, with unglamorous solutions that are infinitely more beneficial for all.

Todd Plessel, Raleigh, NC

Konrad Zuse and Floating-Point Numbers

In his news story "Lost and Found" (July 2012), Paul Hyman characterized Konrad Zuse's Z9 as "the world's first program-controlled binary relay calculator using floating-point arithmetic." This description is not correct but should indeed be the other way round; the Z9/M9 was the only one of Zuse's computers to use binary-coded-decimal fixed-point arithmetic.

Zuse used binary floating point from the time of his earliest computer designs, because his own thorough analysis showed binary representation reduced the complexity of the arithmetic unit and that floating point is adequate for engineering calculations, which, as a civil engineer, is what he primarily had in mind.

Among the pioneers of early computing, from Babbage to Aiken to Wilkes, Zuse alone used floating-point arithmetic; his general-purpose computers Z1 (1938), Z3 (1941), Z4 (1945), Z5 (1953), and Z11 (1955) all used binary floating-point arithmetic. Beginning with the Z22 (1958), the computers developed by the Zuse Company used binary fixed-point arithmetic, implementing floating-point arithmetic through microprograms.

Zuse invented a format for binary floating-point numbers similar to that of IEEE 754, using it in his very first machine, the Z1; Donald Knuth attributes the invention of normalized floating-point numbers to Zuse. The Z3 included representations for 0 (zero) and ∞ (infinity). Operations involving these "exceptional" numbers were performed as usual, as in $0 - 0 = 0$ and $\infty + 5 = \infty$. Operations with an undefined result (such as 0/0, $\infty - \infty$, and ∞/∞) were detected automatically,

signaled by a special light before the machine stopped.

Zuse discussed the question of binary vs. decimal arithmetic with Howard Aiken, writing, "We also had some differences of opinion on technical matters. Aiken advocated the decimal system and had developed very beautiful codes for decimal numbers using several bits. I was more a proponent of pure binary representation—in any case, at least where large scientific computers were concerned. However, I had also used encoded decimal numbers in the mechanical solution for the punch card machine."[1] The "punch card machine" was the Z9/M9.

Jürgen F.H. Winkler,
Feldkirchen-Westerham, Germany

Reference
1. Zuse, K. The Computer–My Life. Springer, New York, 1993.

Attack on Piracy vs. New Creation

Notwithstanding the anti-copyright logo illustrating Joel Waldfogel's Viewpoint "Digitization and Copyright: Some Recent Evidence from Music" (May 2012), legal attacks on unauthorized copying did not begin with Napster in 1999 but have been around since at least the first read/write media allowed copying, including audio and video cassettes in the 1980s and recordable CDs in the 1990s.

The earliest corporate legal resistance to illegal copying of music in the Napster era involved the MP3 format, not cassettes, reflecting the recording and distribution industry's concern over the new technology-enabled ease of copying and sharing with no loss of quality. This was before any particular company, including Napster, had the ability to complicate the business situation by providing the peer-to-peer option to millions of everyday users. The target was individual sharers, not the media that made sharing possible. However, all such campaigns ultimately failed to prevent unauthorized copying of copyrighted work, even as some facilitating organizations were sued out of existence.

Napster also heralded more permissively licensed music. Creative Commons licensing followed, making it easier for artists, as well as rights holders, to adopt revenue models different from the traditional numbers-of-bound-copies as set by distributors. Though difficult to say which came first, there was a strong cross-influence (so at least some correlation) between zero-cost copying and permissively licensed creation of free-to-copy works. Waldfogel said that, given there is more illegal copying today than ever before, there should likewise be a decline in production, as a new musical work should earn less money when it can just be copied. However, the volume of new work has not declined.

So Waldfogel's hypothesis (or my understanding of it) means creation is not inhibited, and some newer creation, distribution, and payment models are based on metrics other than traditional per-copy margin. It is not that Waldfogel's metrics distort the legal foundation of copyright-protected music publishing and distribution but that the legal foundation has produced a market in which what the distributor is able to measure—reviews—is distorted by that foundation. Music critics are more likely to comment on a work produced and marketed by, say, Sony than on the equivalent work recorded and distributed independently by an individual musical artist directly.

Even though some well-known bands (notably Nine Inch Nails and Radiohead) have produced permissively licensed (and widely reviewed) albums, they follow Creative Commons licensing, not borrowing and redistributing permissively licensed music but creating new works covered by a permissive license to begin with. Most composers never attract much attention because they are not part of the established distribution and promotion ecosystem, even if their work reaches a potentially worldwide audience online, free of licensing barriers. Waldfogel's metrics reflect the commercial and cultural effects of digital creation, rather than raw sales numbers.

Gunnar Wolf, Mexico City

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.

BLOG@CACM

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

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

DOI:10.1145/2347736.2347739 http://cacm.acm.org/blogs/blog-cacm

Online Privacy; Replicating Research Results

Daniel Reed offers three ideas about the future of personal online information management. Ed H. Chi writes about replication of experiments and how experiments are often the beginning, rather than the end, of a scientific inquiry.

Daniel Reed
"Information Privacy: Changing Norms and Expectations"
http://cacm.acm.org/blogs/blog-cacm/108232
May 11, 2011

Our notions of privacy and security are deeply tied to our social and historical notions of person and place. The aphorism "A man's home is his castle" captures that notion and its roots in English common law. This Castle Doctrine followed settlers to the colonies and was later codified in the Fourth Amendment to the U.S. Constitution.

Your family heirlooms may be secure in your personal castle, but what of the information about you that lives on the Internet? The legacy of physical place and norms around it are far less relevant.

Equally importantly, we often convolve privacy and security without considering their differences. One needs security to protect private information, but one can have security without privacy, as many world events have shown. Security is a topic for another day; let's talk about the evolving notions of electronic information privacy.

Riding the Light

That picture of you at a family reunion, squinting into the sun, can rarely be delimited by a physical location. It might be on disk two, machine nine, rack 23 in a North Carolina data center, but it probably will not be for long.

Instead, information flows freely in radio waves among our wireless devices and on photon beams along the fiber-optic cables that connect the burgeoning network of worldwide cloud data centers. It's cached, distributed, forwarded, copied, mirrored, and indexed.

All of which suggests that we need to rethink our notions of information privacy, moving beyond concepts rooted primarily in person and place, and considering logical privacy. These issues are complex and emotionally charged for they challenge many of our social, cultural, legal, and economic assumptions. I would not presume to offer a definitive answer here. Instead, let me offer three ideas to stimulate our debate about the future of information and electronic personal management in this brave new world.

Let's return to that family reunion photograph, captured on a smartphone and posted to a social network site. What might I, as a person in the picture, wish to specify and who else might be involved?

First, I might well like to specify a *bounded lifetime* for the photograph, after which it would be inaccessible to anyone. Of course, the bound might be infinity, allowing it to remain in the electronic ether forever. That is the current default, as more than one person has learned to their chagrin.

Second, I might choose to define the *transitivity of access*. I could share the photograph with my extended family, but not allow any of them to share it with their friends. Or I might limit access to an overlapping circle of personal or professional friends, preventing viral propagation. This is challenging because our overlapping spheres of social, professional, and familial influence rarely have hard boundaries, as anyone who has configured their social network privacy settings knows all too well.

The usability of specification interfaces for privacy and security deserves far more attention than it has received. All too often, the only options presented are a broad and vague end-

user license agreement that one must accept to use a service or a Byzantine set of confusing service configuration options whose effects are less than obvious. Privacy specifications must be made far simpler and more intuitive.

Third, I might wish to define a *claims-based access* policy. This is not a binary access specification, but rather a statement that this person or this entity can access this photograph for this and only this purpose. Thus, I might grant my cousin the right to look at this photograph, but not to sell, alter, or combine it with other media.

Ownership, privacy, reputation, and decision-making are intertwined in subtle ways. What if I posed for a reunion photograph but one of my crazy cousins was dancing on the table behind me? Who controls that family reunion photograph—me, the drunken dancer in the background, the photographer with the smartphone, all of us? The shifting nature of social relationships further exacerbates these challenges.

Let me end with another aphorism: "Possession is nine-tenths of the law." In a digital world where images, video, and text can proliferate globally in seconds, we need to rethink what "possession" means.

I don't have all the answers, but I do have lots of questions.

Ed H. Chi
"On the Importance of Replication in HCI and Social Computing Research"
http://cacm.acm.org/blogs/blog-cacm/109916
June 22, 2011

I was asked to serve as a panelist at the CHI2011 conference to discuss the issue of replication of research results. As part of this RepliCHI panel, I wrote an essay arguing that replication isn't just replication of experiments or rebuilding of systems, but instead is used as an important step in building up greater understanding of a domain. Many panelists, including myself, were surprised when many people showed up at the panel (more than 100?), ready to discuss this seemingly dry academic issue. Here is my essay, slightly edited: One mainstream perspective on HCI is that it is a discipline built upon applied psychological science. "Psychological science" here refers to the understanding of mind and behavior, while "applied" here means that it is the application of approaches of methods, findings, models, and theories from the psychology domain. One has to only look at the CHI annual proceedings to see that it is full of *borrowed* methods from experimental psychology, a particular approach to understanding mind and behavior based on scientific experimental methods. This approach worked well for HCI, since computers can be seen as a kind of stimuli that is not only interesting, but could augment cognition and intelligence.[1]

Experimental psychology is based on the idea that if you design the experiment and control the laboratory setting well enough, you will end up with evidence to believe that the results of the experiment will generalize. These ideas about controlled experiments form the basis of the scientific method. As part of the scientific discovery process, we ask researchers to document the methodology and results, so they can be archived and replicated by others.

But my position is that replication is not the only goal. More importantly, if there are limitations to the study, later experiments might expand on the original experiment to examine new contexts and other variables. In these ways, the idea behind the replication and reproducibility of experiments is not just to ensure validity of the results, but it is also an essential part of the scientific dialog. After all, the reason we value research publications so much is not just because they document and archive the results of the research, but also so that others might stand on the shoulders of giants to reproduce and build on top of the results.

Take, for example, the great CHI 97 Browse Off in Atlanta that aimed to put together a number of hierarchical browsers to see which is the "best." At the event, the Hyperbolic Browser[2] was the clear winner. While the event was not meant to be a controlled experiment, it was widely publicized. Several years later, the experiment was replicated in a lab setting at PARC[3] with the top two performing systems during the event—Hyperbolic Browser and Windows Explorer. Not just once, but twice, under different task conditions!

In the first experiment, the results were at odds with the 97 Browse Off. Not only was there *no difference between the browsers* in terms of performance, it appears that *subject variation* had more effect on the results than any other variable.

Further analyses showed there was an interesting interaction effect between the amount of information scent available via the interface conditions and performance, with better information scent resulting in lower retrieval task times with Hyperbolic Browser.

In the second experiment, when restricted to retrieval tasks rather than also including comparison tasks, Hyperbolic Browser was faster, and users appeared to learn more of the tree structure than with Explorer.

What's interesting is the interpretation of the results suggests that squeezing more information on the screen does not improve subjects' perceptual and search performance. Instead, the experiment shows there is a very complex interaction between visual attention/search with density of information of the display. Under high scent conditions, information seems to "pop out" in the Hyperbolic Browser, helping to achieve higher performance.

The above example shows there are a number of fundamental problems with viewing experimental results as the end result of a line of research inquiry. Instead, they are often the beginning. Further experiments often shed light on the complex interaction between the mind/behaviors of the user and the system. Replication/duplication of results and further research efforts examining other contexts and variables are not just desirable, but are an important part of the whole scientific exercise.

References

1. Engelbart, D.C., Augmenting human intellect: a conceptual framework, 1962.
2. Lamping, J., Rao, R., and Pirolli, P., A focus + context technique based on hyperbolic geometry for visualizing large hierarchies, ACM Conference on Human Factors in Computing Systems, 1995.
3. Pirolli, P., Card, S.K., and Van Der Wege, M.M. The effect of information scent on searching information: visualizations of large tree structures, *Proceedings of the Working Conference on Advanced Visual Interfaces*, N.Y., N.Y., 2000.

Daniel Reed is vice president, Technology Strategy & Policy and eXtreme Computing Group, Microsoft.
Ed H. Chi is a research scientist at Google.

ACM Member News

IVAN SUTHERLAND RECEIVES KYOTO PRIZE

Ivan Sutherland, a visiting scientist at Portland State University, was recently awarded the 2012 Kyoto Prize in Advanced Technology for "pioneering achievements in the development of computer graphics and interactive interfaces."

Sutherland is best known for Sketchpad, a graphical interface program that revolutionized computer graphics in the early 1960s and laid the groundwork for today's computer-aided design systems. Today, the ACM A.M. Turing Award recipient is working on something a little different: time itself.

His work at Portland State involves asynchronous computing, or self-timed computing. Sutherland, who founded the Asynchronous Research Center at Portland State with his wife and collaborator Marly Roncken in 2009, is partly motivated by his observation of how time is now different from Alan Turing's era. In Turing's day, logic was slow and expensive but, relative to logic, communication methods (or wires) were fast and inexpensive. Today, the opposite is true.

This "clocked design paradigm," as Sutherland described it in his talk at the Turing Centenary Celebration, slows down communication, by far the more costly of the two factors. [A Viewpoint by Sutherland, "The Tyranny of the Clock," appears on p. 35.—*Ed.*]

"I think in terms of the tyranny of the clock," says Sutherland. "You've got to do the next step exactly on time. It's quite unnecessary."

Sutherland believes the clocked design paradigm has been kept largely by mistake. "It's the kind of mistake that's very hard to change," he says. "What we're working on is trying to change the paradigm from a synchronous time paradigm to a self-timed paradigm. I think there are major benefits to be had, but I also think there's a lot of education to be done."

—Logan Kugler

PHOTO BY DICK LYON

Science | DOI:10.1145/2347736.2347741 Neil Savage

Digging for Drug Facts

With the right approach, data mining can discover unexpected side effects and drug interactions.

When people with high blood pressure start taking thiazide diuretics as treatment, they are warned about possible heart-related side effects, including palpitations, fainting, and even sudden death. Patients taking a certain class of antidepressants face similar risks. But what if they are taking both drugs together? Would the bad effects be more likely? No one knew.

Not, that is, until researchers at Stanford University used data-mining techniques to pore through a database of side effects. They discovered that people taking both a thiazide diuretic and a selective serotonin reuptake inhibitor, such as Prozac, were about one-and-a-half times as likely as people taking either drug separately to show a prolonged QT interval, a measurement in cardiology that increases the risk of those heart problems. In mining the database, Russ Altman, professor of medicine and biomedical informatics research, and Nicholas Tatonetti, who has since earned his Ph.D. in biomedical informatics, found an additional 46 drug pairs that interacted to cause side effects that had not been known before beyond those of either drug alone.

PHOTOGRAPH BY ERIN DEMAY

Clinical trials do not test for drug interaction; it would be costly and impractical to test every drug against every other drug. And although the trials can identify some side effects, other side effects do not show up until a medication is given to a larger population of patients over a longer period of time.

To keep tabs on unexpected complications from already approved drugs, the U.S. Food and Drug Administration (FDA) created the Adverse Events Reporting System (AERS), a database of more than four million negative events reported since 1969. Every three months, the FDA releases a new batch of event reports, which statisticians can use to look for previously unrecognized drug-related problems. Pharmaceutical companies are required to report bad effects associated with their medications, and health professionals and patients can also submit reports.

"This database from the FDA is very large and on first blush looks like an amazing resource," says Altman. But teasing out a relationship between a particular drug and an adverse event can be challenging. "People are on multiple medications, they have multiple diseases, and they have multiple side effects, and establishing one-to-

Informatics, Drugs, and Chemical Properties

Predicting the side effects of a medication is not only useful after the drugs are on the market. Catching potential problems before the compounds even reach clinical trials can save pharmaceutical companies time and money by screening out failing candidates early on.

Researchers at the Novartis Institutes for Biomedical Research and the University of California, San Francisco, applied a method called the similarity ensemble approach (SEA) to the problem. Drugs work by binding to and interacting with target proteins in the body, but they also often bind to off-target proteins and cause side effects. SEA examined the chemical properties of 656 approved drugs to see how similar they were to other molecules known to bind to 73 different proteins that are associated with side effects. This was the first time such an informatics approach was applied systematically to the search for side effects, according to the researchers.

The computer compared the chemical structure of those drugs to a database of more than 285,000 molecules known to interact with 1,500 human proteins. It predicted about 1,200 potential interactions, nearly 900 of which had never been explored. The researchers searched other databases, ran their own chemical tests, and confirmed almost half of the computer's predictions. Eugen Lounkine, a researcher at Novartis and first author of the study, which was published in *Nature,* says the computer acts as a screening tool to make the development of medication more efficient. "It's early in drug discovery, where you have more compounds than you have resources to test with biochemical assays," he explains.

Lounkine says finding off-target proteins can also help explain some previously unexplained side effects. For instance, chlorotrianisene, a synthetic estrogen sometimes used to treat prostate cancer, can cause upper abdominal pain, but no one knew why. The Novartis study discovered the drug interfered with an enzyme known as COX-1 in much the same way a blood thinner, which is known to cause such pain, would.

one correspondence between those is really daunting."

The reports are "spontaneous," meaning they are not standardized and are based on individual judgments about symptoms someone noticed and deemed significant. The data is statistically noisy, full of biases and confounding factors that may not be easily identifiable. One well-known bias, for instance, is what Tatonetti calls "the Vioxx effect"; when a link was discovered between the painkiller Vioxx and heart attacks, the resulting publicity prompted people using Vioxx to report more heart-related symptoms, which made the background rate for those symptoms seem greater than normal, thereby masking the drug's real effects. There are also symptoms that might be associated with a drug but are not caused by it. Someone taking a medication for diabetes, for instance, could have symptoms caused by the underlying disease, though an algorithm would only notice the association between the symptoms and the drug, and could incorrectly conclude the drug was causing the problem. Someone using an arthritis medicine might report complications that are the result of being elderly, and not from the medication. Modern signal-detection algorithms try to account for biases, but have not addressed all the possible sources, the researchers say. "There are a lot of scientific computational challenges to this database," says Robert O'Neill, director of the Office of Biostatistics in the FDA's Center for Drug Evaluation and Research.

One way to separate the effects of a drug from the effects of related factors is to remove such covariates from the sample. But the database often does not list potential confounding factors such as age, sex, or underlying disease. Existing methods that control for confounding factors by creating subsets according to the covariates cannot work when the covariates are unknown.

But Tatonetti realized that in many cases he could figure out what those covariates were based on the combination of drugs that patients were taking and the set of symptoms they described. If a patient is on a cholesterol-lowering drug, for instance, he is likely to have a high-fat diet. A patient taking antidepressants is somewhat more likely to be female. If the patient is using birth control pills, she is definitely female, whereas someone taking a prostate medication is clearly male. Tatonetti also grouped people by which drugs they were taking in addition to the drug being investigated, and discarded side effects known to be caused by those other drugs.

For his data source, Tatonetti gathered more than 1.8 million reports in the AERS database from 2004 to early 2009, along with another 300,000 reports from a similar Canadian database. He also used a database of known side effects and drug indications mined from medications' FDA labels. And he included information about the biological targets the drugs were aimed at.

In the end, for each drug the researchers wanted to study, they wound up with two groups: one taking that drug and one that matched the first group in as many other ways as possible, except for that one drug. For each drug they studied, their search found an average of 329 bad reactions that were associated with the drug but that were not listed as known side effects. They then applied the same method to find drug interactions, comparing groups that were on only Drug A, only Drug B, or both. To validate their prediction that the combination was causing a side effect, they looked at lab test results from the Stanford hospital system's electronic health records for people on those drugs, and found 47 combinations that seemed to cause problems.

The researchers' hypotheses are a valuable step in discovering unknown health problems. With the hypotheses in hand, drug companies can reexamine their clinical trial data to see if they can verify the problem or even run additional trials. Regulatory agencies can warn doctors to watch for new side effects, or even pull drugs from the market. O'Neill says that by identifying side effects common to a class of drugs, such surveillance could help drug developers screen out failing drug candidates sooner, before they devote too much effort to their development. (See the "Informatics, Drugs, and Chemical Properties" sidebar on this page.)

Niklas Noren, chief science officer at the Uppsala Monitoring Centre in Uppsala, Sweden, which monitors drug safety for the World Health Organization, calls the Stanford approach "interesting and innovative." He has used a different statistical method to correct for the Vioxx effect and for

false associations between drugs and symptoms; his approach does not include information about drug indications, but he says it could. "To me their attempt to directly control for this important bias is a key contribution to the field," he says. Noren has also used reports that contain covariate information such as age, sex, time period, and country to sort patients into matched groups that can be compared, but it detects only associations in those subsets and cannot reveal syndromes—groups of drug interactions that tend to be reported together. One of the big questions in the field, he says, is how mining these spontaneous reporting databases, like AERS and similar European systems, fits with the growing use of electronic medical records (EMRs). The value of EMRs is that they have much more detailed information about patients, but they may lack the power to detect rare events. "Spontaneous reporting clearly has a role," he says. "What we need to figure out now is how to use each type of data in the best way."

Multiple Sources

Rave Harpaz, a research scientist at Stanford's Center for Biomedical Informatics Research but not involved with Altman's work, is looking for the best ways to combine information from multiple sources, using not only AERS and EMRs, but also mining medical literature for clues, picking up patient reports of symptoms from social networks, and adding basic information from biology and chemistry. "There's many ways to combine these data sources, depending on what you want to achieve," Harpaz says.

The advantage of these large and growing datasets is that researchers can use different approaches to amplify signals from rare or hidden events that might otherwise go unnoticed. With enough data, Altman explains, researchers can discard some of it as they remove some of the noise, but still have plenty to work with. "You can throw away lots of it, and as long as you maintain statistical significance, you can still get useful answers."

Harpaz adds that it is possible for a weak signal to "borrow" statistical significance from other sources. For instance, with 14,000 different codes for events in the FDA's coding dictionary, the same event might wind up being described by several different terms, with no single term showing up often enough to appear statistically significant. "If you group all these terms together into a sort of hyperterm or hyperevent, you might be able to find that adverse drug event," Harpaz says. Adds Altman, "Multiple pieces of weak data, when combined, can equal one very strong piece of evidence."

The FDA, too, wants to move beyond what O'Neill calls the passive surveillance of the AERS database to active surveillance, in which it combs through electronic records held by healthcare providers such as the U.S. Department of Veterans' Affairs and Kaiser Permanente. The agency is in the early stages of setting up what it calls the Sentinel Initiative to accomplish just that.

Still, O'Neill cautions the data-mining algorithms alone cannot prove that particular drugs cause particular side effects. They can provide clues that need to be checked in other ways. "Out of every 100 things you find, there may be 10 that are worth pursuing," O'Neill says. "How to separate the wheat from the chaff, that's the trick." C

Further Reading

Altman, R.
Russ Altman @ PMWC 2012: Data Mining EMR's, http://www.youtube.com/watch?v=XPBrCYaV050, Feb. 24, 2012.

Harpaz, R., DuMouchel, W., Shah, N.H., Ryan, P., and Friedman, C.
Novel data mining methodologies for adverse drug even discovery and analysis, *Clinical Pharmacology and Therapeutics*, May 2, 2012.

Hopstadius, J., and Noren, G.N.
Robust discovery of local patterns: subsets and stratification in adverse drug reaction surveillance, 2012 ACM SIGHIT International Health Informatics Symposium, Miami, FL, Jan. 28–30, 2012.

Lounkine, E., et al.
Large-scale prediction and testing of drug activity on side-effect targets, *Nature 486*, 7403, June 10, 2012.

Tatonetti, N.P., Ye, P.P., Daneshjou, R., and Altman, R.
Data-driven prediction of drug events and interactions, Science Translational Medicine 4, 125, March 14, 2012.

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

Education

Latinos and STEM

Of all the certificates and degrees conferred in science, technology, engineering, and math (STEM) in the U.S. in 2009–2010, 8% were earned by Latinos, reports a new study, *Finding Your Workforce*, by Excelencia In Education, a nonprofit organization advocating Latino educational success.

The 25 higher-education institutions that granted the most STEM certificates and degrees to Latinos were concentrated in just six states—Texas, Florida, California, Arizona, Illinois, and New Mexico—and Puerto Rico. About 40% of the students came from those schools.

At the undergrad level, the majority of the schools were categorized as Hispanic-Serving Institutions by the federal government, meaning that at least 25% of the student population is Hispanic.

Regarding the STEM certificates and degrees:

- **Of the total 30,801 conferred to Latinos, the majority were bachelor's degrees (60%), while 16% were associate's degrees, 12% certificates, 9% master's degrees, and 2% doctoral degrees.**
- **Of those certificates/degrees, 35% were in science, 32% in engineering, 28% in technology, and 5% in math.**
- **The top 25 institutions awarding STEM degrees at the doctoral level in 2009–2010 awarded fewer than 430 to Latinos.**
- **At the doctoral level, the top institutions conferring degrees in science to Latinos in 2009–2010 awarded 28 degrees during that period. The top institution in computer science awarded two doctorates to Latinos, the top institution in engineering awarded 12 doctorates to Latinos, and the top institution in math awarded five doctorates to Latinos.**
- **Only 15 institutions awarded any doctorate degrees to Latinos in computer/information sciences in 2009–2010. The schools awarding the highest number are MIT (2) and the University of Puerto Rico-Mayaguez (2). The other 13 institutions awarded one doctorate each.**

The report can be found at: http://edexcelencia.org/sites/default/files/exc2012fyw_stem_final_web_2.pdf.

—Paul Hyman

Technology | DOI:10.1145/2347736.2347742 Gregory Mone

Redesigning the Data Center

Faced with rising electricity costs, leading companies have begun revolutionizing the way data centers work, from the hardware to the buildings themselves.

Late last year, Stanford University researcher Jonathan Koomey released a report detailing a few surprising trends about the energy squanderers known as data centers. Previous estimates suggested that electricity consumption in massive server farms would double between 2005 and 2010. Instead, the number rose by 56% worldwide, and merely 36% in the U.S. The slower-than-expected growth stemmed from a number of changes, including a stagnant economy and the rise of virtualization software.

Yet experts say a more fundamental change is also starting to take effect—one that could lead to much greater improvements in efficiency. Over the past seven or so years, leading companies have begun revising the way they design, maintain, and monitor data centers, from the physical building all the way down to the hardware doing the computation. Recently, Google, Facebook, and other major companies have begun releasing details on the efficiency of their facilities, and revealing a few of the technological tricks they have devised to achieve those gains.

Still, these leaders are the exception rather than the rule. There are no solid estimates of the total number of data centers in the U.S., and the Silicon Valley giants are secretive about exactly how many they operate, but they hardly dominate from an energy standpoint. In all, U.S. facilities consume between 65 and 88 billion kilowatt hours per year, and Google, for instance, accounts for less than 1% of that figure.

The fact remains that the average data center is still largely inefficient. The standard measure of a data center's efficiency is its PUE, or power usage effectiveness. PUE is the total energy used to operate a data center divided by the amount devoted to actual computing. That total includes lighting, fans, air conditioners, and even electrical losses as power is transferred from the grid to physical hardware. Ideally, a data center would run at a PUE of 1.0, and all of the electricity would go toward computing. Yahoo!, Facebook, and Google have all touted facilities scoring below 1.1. Across industries, though, these numbers are hardly common. "What happens in the typical data center is that it's more like 2.0," explains Koomey.

Over the past seven or so years, leading companies have begun revising the way they design, maintain, and monitor data centers, from the physical building all the way down to the hardware doing the computation.

Until recently, most companies did not even bother measuring PUE. They had little sense of how and where energy was used or lost within the facility. "The primary reason all of this happens is because there's not great accounting of the energy in data centers," says Raju Pandey, the chief technical officer of Synapsense, a Folsom, CA-based company that performs data center optimizations. "There's an incredible amount of wastage."

Heating Up

If you had walked into the average data center 10 years ago, you would have needed a sweater. The American Society of Heating, Refrigerating and Air-Conditioning Engineers recommended these facilities be maintained at temperatures between 60 and 65 degrees Fahrenheit to prevent the equipment inside from overheating. And the machines that cool the space are often inefficient. "Traditional data centers basically have the same air-conditioning unit you'd put in your house," says Bill Weihl, Facebook's manager of energy efficiency and sustainability.

The rationale was that warmer temperatures could lead to hardware failures, but several experts doubted this was actually the case. When Google began planning a new breed of data center in 2004, the company started testing the temperature limits of its hardware. "We started running our servers warmer and monitoring the failure rates," says Joe Kava, Google's director of data center operations. In the end, Google simply did not see any major problems. "The servers ran just fine," he adds, "and if you know your servers can run at 80 degrees, you can redesign your cooling system entirely."

Google found that it could avoid relying on giant air-conditioning units, as did other companies. The most efficient data centers now hover at temperatures closer to 80 degrees Fahrenheit, and instead of sweaters, the technicians walk around in shorts. Facebook's data centers in Lulea, Sweden and Prineville, OR, have no me-

Ambient air flows through Facebook's data center in Prineville, OR, and cools the servers inside the 334,000-square-foot facilty.

chanical chillers at all. "We don't need them," says Weihl, who was previously Google's energy efficiency czar.

At Facebook's Prineville facility, ambient air flows into the building, passing first through a series of filters to remove bugs, dust, and other contaminants, then into a long corridor. On hot days, when the outside temperature rises above 80 degrees Fahrenheit, the air moves through a fine mist of micron-sized droplets of water suspended in the corridor. Some of the mist evaporates on contact with the warmer outside air. This reduces the temperature, and the mildly chilled air then passes through another filter, which captures the droplets of water that did not evaporate. The end result is a rush of cool air flowing into the building.

When it is too cold outside, the control system automatically mixes in some of the 85 to 90 degree Fahrenheit air coming out of the back of the servers to bring it up to the right temperature. "We don't want 20 degree Fahrenheit air going into our servers," Weihl says. Drastic changes in temperature could cause components to expand and contract, creating mechanical stresses that might lead to permanent damage.

Smart Monitoring

The source of the cool air in traditional data centers is only part of the problem. Companies have also begun demonstrating the importance of managing circulation within the space. When Synapsense audits a facility, its technicians install wireless sensors throughout the building to measure temperature, pressure, humidity, and more. Pandey says Synapsense often identifies intense hot spots—warmer areas that force the fans and mechanical chillers to work harder to manipulate the temperature, thus increasing energy usage. "You might have enough cool air, but it's not going to the right places," he explains. "There might be mixing of the air or there might be areas where it's leaking."

The most efficient data centers now hover at temperatures closer to 80 degrees Fahrenheit, and instead of sweaters, the technicians walk around in shorts.

In one case, Synapsense installed 3,674 sensors throughout a 100,000-square-foot data center. The sensors fed Synapsense's control system a stream of data on temperature, pressure, and humidity, and the company's software built a live-updated map of these metrics throughout the facility. With this data, Synapsense was able to figure out how to optimize energy use by turning up certain fans or shutting down specific air conditioning units. It ended up saving the company 8,244 megawatt hours per year—or $766,000 in annual electrical bills.

In other instances, the changes can be simpler to identify. "In many data centers," Pandey says, "the hot side of a server might be blowing air into a cool side." When this happens, the chilled air rising from the floor is partially wasted, so Synapsense and others advocate arranging data centers into hot and cold aisles. In one aisle, you will be faced with the backs of the racks on both sides expelling warm air, whereas the two adjacent aisles will be comparatively cool, with only the front ends of the hardware facing out. Weihl says Facebook installs plastic panels around its hot aisles, creating a corridor that ferries hot air straight to the ceiling. Then it is either exhausted to the outside or mixed with incoming, colder ambient air to lower it to the ideal temperature.

Focusing on the air flow and conditions within the hardware itself has proven critical as well. Both Google and Facebook advocate simpler, stripped-down server hardware without the standard plastic or metal plates that often bear a manufacturer's logo. "The more obstructions you put in the way of the air flow, the harder the fans have to work and the more energy you use," Weihl says.

Those vanity plates are only one problem with standard hardware. "If you take a standard off-the-shelf server there are probably quite a few things that need to be improved to have it work more efficiently," says Bianca Schroeder, a computer scientist at the University of Toronto and the coauthor of a recent paper on data center efficiency (see the Further Reading list). For example, Schroeder notes

PHOTOGRAPH BY CHUCK GOOLSBEE

that the standard machine won't have internal temperature sensors to monitor whether one of its hard disk drives might be overheating. On the other hand, Facebook's Open Vault, a freely accessible server hardware design, has 10 thermal gauges spaced throughout. These sensors link to a self-monitoring system that can adjust the speed of six fans that help to ensure the server stays cool. Furthermore, the fans themselves consume less energy than the industry standard.

Efficiency for All

The Open Vault design is part of a larger Facebook effort, OpenCompute, that makes the company's data center-related efficiency tricks publicly available. Weihl says Facebook released this information in part because it does not see much of a competitive advantage in locking up its energy-saving secrets. "We've done a lot of cool things," he says, "and the conclusion here was that we should figure out how to work with the industry to be as efficient as possible."

Facebook's goal is to have a larger impact on data center energy consumption on the whole. And Weihl says the company was thrilled to see that server manufacturers like Dell and Hewlett-Packard have incorporated some of its recommendations, like the removal of vanity plates. Such changes could translate into warmer data centers, and more savings on the cost of cooling the huge buildings.

At Facebook's Prineville, OR, facility, ambient air flows into the building, passing first through a series of filters to remove bugs, dust, and other contaminants.

Despite the evidence, and examples from the efficiency leaders, many companies are still afraid to turn up the thermostat, says Schroeder. Her own research suggests that this fear is unjustified. "We can safely say that increasing the temperature by a few degrees will not significantly increase failure rates," she explains, "and increasing temperature even a few degrees will save significant amounts in cooling."

Koomey argues a number of roadblocks remain. For instance, traditional data centers can last for 15 to 20 years, preventing the wholesale adoption of the more efficient new designs, and many of these older facilities are filled with "comatose" servers that suck up power but no longer handle any computation. "There's still a long way to go," he says.

Further Reading

Barroso, L.A. and Urs Hölzle, U.
***The Data Center as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan & Claypool Publishers, San Francisco, CA, 2009.**

El-Sayed, N., Stefanovici, I., Amvrosiadis, G., Hwang, A., and Schroeder, B.
Temperature management in data centers: Why some (might) like it hot, *Proceedings of the 12th ACM SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, June 11–15, 2012, London, England.

Google, Inc.
Google's Hamina data center, http://www.youtube.com/watch?v=VChOEvKicQQ&feature=player_embedded, May 23, 2011.

Hamilton, J.
Perspectives, a blog by Amazon Web Services infrastructure efficiency expert James Hamilton, http://perspectives.mvdirona.com/.

Koomey, J.
***Growth in Data Center Electricity Use 2005 to 2010*, Analytics Press, Oakland, CA, 2011.**

Gregory Mone is a Boston, MA-based science and technology writer and the author of the novel *Dangerous Waters*.

Privacy

Vehicle License Plates and Surveillance

Systems for reading, matching, and storing vehicle license plate data are proliferating, with automatic license-plate reader (ALPR) cameras being deployed by law enforcement agencies at all levels of government, often without the knowledge of the populations under surveillance.

ALPR cameras, on stationary objects or atop police cars, can photograph hundreds of license plates per minute. The license plate numbers are converted into machine-readable text and matched against crime databases, alerting a patrol officer when a "hit" appears. The data, which includes GPS location, time, and date, is generally kept for every reading, and often for an indefinite period.

"The technology is spreading rapidly, thanks in part to grants from the Departments of Homeland Security, Justice, and Transportation to police departments nationwide," says Kade Crockford, privacy rights coordinator for the American Civil Liberties Union (ACLU). She adds that the ACLU is not opposed to using the technology to identify stolen vehicles or those owned by someone for whom there is an outstanding arrest warrant. "What isn't okay, however, is for the police to use license-plate readers to collect vast amounts of data on ordinary motorists," Crockford adds. "We already know that this is happening in various parts of the country, usually without so much as a public conversation on the privacy risks involved."

The online computer technology journal *Ars Technica* recently reported on an ALPR system in California that produced numerous false positives, and which failed to read the license plate on a reporter's car. The false positives stemmed from the fact that the California Department of Motor Vehicles takes a long time to update its vehicle-crime database, and because the system could not distinguish out-of-state plates.

The Electronic Privacy Information Center (EPIC) recently received the results of a Freedom of Information Act request about ALPRs from U.S. Customs and Border Protection. "These documents offer clear proof that location data obtained through license-plate readers is being shared extensively, not just among law enforcement agencies, but also with third parties," such as the nonprofit National Insurance Crime Bureau, says Ginger McCall, an EPIC director.

—Gary Anthes

Society | DOI:10.1145/2347736.2347743 Leah Hoffmann

Computer Science and the Three Rs

A growing sense of crisis prevails as computer science searches for its place in the K–12 curriculum.

Not long ago, the educational crisis most talked about by computer scientists was the steep decline in the number of undergraduates who picked the field as their major. Today, with those declines in reverse at many U.S. universities, there is a growing recognition—both in the U.S. and internationally—that the real crisis lies not at the college level, but in primary and secondary school.

"The college level is, in a certain sense, too late," says Stephen Cooper, an associate professor of computer science at Stanford University and chairman of the board of the Computer Science Teachers Association (CSTA), a membership organization that serves U.S. K–12 educators. "Though young women decide what they're majoring in much later than young men, they also decide what they're not majoring in much earlier than men do. If they decide they're not interested in computer science at the end of 8th grade, there's nothing we can do at the college level to change that. We've missed the opportunity. But if we change K–12, we have a chance to solve this problem."

Of course, it is not just about recruiting majors or correcting the field's gender imbalance and lack of minorities. Computer science has become an integral part of nearly all science, technology, engineering, and mathematics (STEM) jobs, and it is of increasing conceptual relevance to a variety of other professions, as well. "There's really no science that doesn't involve computation in a pretty fundamental way, and if you look beyond science, what people need to know is what computation can do for them. What are the parts of your job that could be solved computationally? Would you know what kinds of tools to ask for and how to put them together to solve your particular problem?" explains Jan Cuny, a program officer at the National Science Foundation who has headed a number of initiatives to support computer science education.

Elementary school students learn how computers enable science during a field trip sponsored by Lawrence Berkeley National Laboratory in May 2010.

PHOTOGRAPH BY ROY KALTSCHMIDT, COURTESY OF LAWRENCE BERKELEY NATIONAL LABORATORY

Yet over the past 20 years, Cuny notes, the percentage of students who take computer science in U.S. high schools has dropped from 25% to 19%. In many schools, the subject is simply not taught; in others, what is covered are soft skills like touch typing and the use of Microsoft Office. Because of the decentralized nature of the U.S. school system, precise student numbers are difficult to discern. However, according to *Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age*, a 2010 report released by CSTA and ACM, only 14 states have adopted recognized education standards for computer science instruction, and only nine states allow computer science courses to count toward the hours required to graduate for either science or mathematics.

Fitting computer science into the curriculum is a particular problem in the U.S., where each state defines its own academic standards. That is why many educators have focused their attention on the computer science course offered by the Advanced Placement (AP) program, a set of courses that are administered by the College Board, a membership organization that administers standardized tests, and follow a curriculum designed to be equivalent to college-level courses. The current AP computer science course focuses on Java programming and is taught in roughly 2,000 high schools out of 29,000 across the country. Approximately 20,000 students take the end-of-year AP computer science test, as compared to the several hundred thousand who take the AP calculus exam. To increase participation and attract new students to the field, Owen Astrachan of Duke University, Amy Briggs of Middlebury College, CSTA, and others are developing a new AP course that focuses on principles rather than programming. Seizing on a National Research Council directive to rethink the way several AP courses in STEM subjects are taught—urging instructors to emphasize process over knowledge and understanding over facts—the group is developing a pilot program to be released in 2016. The idea, proponents say, is to attract a more diverse student body by concentrating on computational thinking, and to get the course into more schools using the cachet of the AP brand.

"When kids have had no experience with computing, it's hard to convince them that they want to spend a year learning how to program," says Cuny,

If young women "decide they're not interested in computer science at the end of 8th grade," says Stephen Cooper, "there's nothing we can do at the college level to change that."

who has helped coordinate NSF's support for the development of the new course. "What's more, AP is taught all over the country, and schools like AP because it makes them look academic."

Managing the way computer science is taught can be a challenge even for countries that do have a standard national curriculum. Israel, for example, has several well-regarded computer science programs at the secondary level, but the subject is considered an elective. According to Tami Lapidot, executive manager of Machshava, the Israeli National Center for Computer Science Teachers, some officials and administrators still fail to understand the importance of the subject. "[They] think that CS and Facebook are the same," she explains. "In the Ministry of Education, CS is not considered a scientific subject but rather a technology subject."

England, by contrast, made Information Communications Technology (ICT) compulsory 12 years ago, but left the syllabus quite broad. "It did allow more interesting stuff if the teachers were so minded," explains Steve Furber, ICL Professor Of Computer Engineering at the University of Manchester and chair of a Royal Society advisory group whose report, *Shut Down or Restart? The Way Forward for Computing in UK Schools*, was released in January. "But an awful lot of schools identified the lowest common denominator and were teaching ICT through fairly routine Office applications." *Shut Down or Restart* points to a nexus of interrelated factors that reinforce the status quo, where ICT is delivered in the form of digital literacy lessons, dismissed by students as uninteresting, and further devalued by schools. Indeed, several recent reports—and the 2011 MacTaggart Lecture by Google executive chairman Eric Schmidt, in which he warned that the U.K.'s IT curriculum "focuses on teaching how to use software, but gives no insight into how it's made"—have underscored the importance of improving the country's computer science instruction. Would-be reformers were therefore quite encouraged when Secretary of State for Education Michael Gove announced the "disapplication" of the current ICT curriculum, along with plans to replace it with a more academically rigorous alternative.

Singapore has also seen efforts to rethink its pretertiary computer science curriculum. The republic's Infocomm Development Authority (IDA) has worked with industry partners and schools to develop courses that are anchored in computational thinking through the CS Reloaded program, which also helps offset course fees. Since March 2011, more than 700 students have been trained under the program in courses that are anchored in computational thinking.

A Lack of Qualified Teachers

Yet even as curricula improve, a larger challenge looms for international computer science education: attracting qualified teachers and keeping them abreast of new developments in a fast-changing field. Many teachers have had no formal training in computer science, coming instead from domains like math, business, and physics. "That isn't to say we don't have a lot of great teachers—we do," explains Jan Cuny. "But students don't go into the field thinking they're going to be teachers, and we don't do a good job of telling them that could be an important thing to do." In England, only 30% of high school computer science teachers have qualifications that are recognized by the nation's Department for Education (as opposed to 75%–80% in established subjects). In the U.S., only half of the states even have certification programs in the first place.

Encouragingly, a number of initiatives have sprung up to address the

problem. In the U.S., the NSF has funded a project called CS10K, which set the goal of training 10,000 computer science teachers across the country. Launched last year, CS10K aims to build a consortium across academia, industry, government agencies, and private foundations, and is part of a larger NSF-sponsored project called Computing Education for the 21st Century, or CE21, which will also fund projects in computing education research and broadening student participation. CSTA, meanwhile, runs a variety of professional development initiatives, including an annual conference and workshops. It also maintains an online repository of research and resources.

"You have maybe one computer science teacher per school," says CSTA's Cooper. "At best they have to go to other schools in their district or to other districts to find other teachers who look like them. Offering them the ability to network and build a community is incredibly important." Israel's Machshava takes a similar approach, with an annual conference, courses, and summer seminars aimed at fostering professional leadership. In England, a grass-roots group called Computing at School (CAS) has formed a partnership with the British Computer Society to sponsor workshops and coordinate advocacy efforts.

"Everybody's picked up this sense that change is in the air, and that teachers who want to stay on top of this have got to be proactive," says Furber. "The good news is CAS membership went up from 300 to 600 or 700" after the publication of *Shut Down or Restart*. "The bad news is that's still only one teacher for every 10 secondary schools in the country."

A variety of university-led programs have begun to appear, as well. Purdue University's CS4EDU is a joint effort between computer science faculty and the school's College of Education to create a teaching licensure program that prepares education majors to teach computer science in secondary schools. Carnegie Mellon's CS4HS reaches out to current high school teachers through workshops aimed to help them teach computer science principles; it has since received funding from Google's Education Group. Finally, there are regional initiatives like Jane Margolis's Exploring Computer Science—an NSF-supported program that has helped bring computer science courses to more than 25 schools in the Los Angeles Unified School District—which will also inspire work in other cities.

Managing the way CS is taught can be a challenge even for countries that have a standard national curriculum.

CSTA's Cooper is optimistic. "We're making progress. Ten years ago, you couldn't talk about major school systems in CS, and now there's work being done in Chicago, Los Angeles, and San Jose. It starts to slowly add up." C

Further Reading

Brandes, O., Vilner, T., and Zur, E.
Software design course for leading CS in-service teachers, *Proceedings of the 4th International Conference on Informatics in Secondary Schools*, Zurich, Switzerland, Jan. 13–16, 2010.

CSTA Teacher Certification Task Force
Ensuring Exemplary Teaching in an Essential Discipline: Addressing the Crisis in Computer Science Teacher Certification, Computer Science Teachers Association, New York, NY, Sept. 2008.

The Royal Society
Shut Down or Restart? The Way Forward for Computing in UK Schools, The Royal Society, London, U.K., Jan. 2012.

Schmidt, E.
"The James MacTaggart Lecture," MediaGuardian Edinburgh International TV Festival 2011, Edinburgh, Scotland, http://www.edinburghguide.com/video/deregulateandgoglobalgooglechieftellstvfest, August 26, 2011.

Wilson, C., Sudol, L.A., Stephenson, C., and Stehlik, M.
Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age, Association for Computing Machinery, New York, NY, 2010.

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

Milestones

CS Awards

The Royal Society and National Center for Women & Information Technology (NCWIT) recently honored five leading computer scientists.

ROYAL SOCIETY MILNER AWARD

The newly created Royal Society Milner Award is given for outstanding achievement in computer science by a European researcher. The inaugural recipient is Gordon D. Plotkin, professor of theoretical computer science, University of Edinburgh, who was selected for "his fundamental research into programming semantics with lasting impact on both the principles and design of programming languages."

NCWIT UNDERGRADUATE RESEARCH MENTORING AWARD

NCWIT recently announced the inaugural recipients of its Undergraduate Research Mentoring Award. Four computer scientists were selected "for their outstanding mentorship, creation of high-quality research opportunities, recruitment of women and minority students, and efforts to encourage and advance undergraduate students in computing-related fields." The 2012 recipients are:

- **Diana Franklin, tenure-equivalent teaching faculty, University of California, Santa Barbara. Franklin has developed a retention pipeline by recruiting students during their freshman year and has mentored 18 students, half of whom are female or from other underrepresented groups.**
- **Juan Gilbert, IDEaS professor and chair of the Human-Centered Computing Division, Clemson University. Gilbert's lab is home to nearly 8% of the nation's African-American computer science Ph.D. students.**
- **Scott McCrickard, associate professor, Virginia Tech. McCrickard conducted specific, targeted outreach to undergraduate women by partnering with local women's colleges.**
- **Mingrui Zhang, professor, Winona State University. Though his computer science department's student body is only 4% female, Zhang actively recruits women to his research programs and seeks funding to support them.**

—Jack Rosenberger

DOI:10.1145/2347736.2347744

Michael A. Cusumano

Technology Strategy and Management

Reflecting on the Facebook IPO

Exploring some factors that reflect a company's worth.

What should a company be worth? This simple question is difficult to answer and became a heated topic of debate with the public debut of Facebook in this past May. The social networking giant, now with close to one billion users, opened with an initial public offering (IPO) stock price of $38 and a market value of $104 billion. These numbers quickly fell 25% within a week.[5,6] Then Facebook disappointed investors with its first earnings announcement in late July, showing "only" 32% growth over the prior year. The company had higher costs and took $1.3 billion in charges related to pre-IPO stock compensation. Analysts also worried that too many users were accessing Facebook on mobile devices, resulting in lower growth rates for ad revenues as well as reduced operating profits. Not surprisingly, the stock price dropped another 20% or so overnight.[8] Where the company's market value will end up, no one knows. But how low or high should the IPO stock price have been?

Most new software and Internet services companies pay little or no dividends, do not place much emphasis on profits, and invest mainly in intellectual capital rather than physical assets. Therefore, traditional metrics used to value a company, such as dividend yields, price-to-earnings ratios, or return on assets, do not work well. Nonetheless, investment bankers will suggest a value, investors will consider whether or not to invest, and analysts will comment after the fact. One columnist for *The Wall Street Journal*, for example, argued that Facebook's initial price should have been no more than $13.80—if investors were to achieve average stock market returns (11% annually) and a Google-like market valuation-to-sales ratio five years from now (currently 6 or 7 to 1 compared to 15 times revenues at its 2003 IPO). The lower stock price implies Facebook should have been worth at IPO no more than $38 billion.[2]

Facebook and its entire generation of social media stocks may all have been overvalued.

Zynga and Groupon have also fallen steeply from their IPO prices, giving many investors reason to pause: Facebook and its entire generation of social media stocks may all have been overvalued. My purpose in this column is not to speculate on prices but to make two more-general observations: First, I can see no way to determine the *short-term* market value of a company, before or after IPO, with any type of "scientific" precision. And second, investors can still use some "objective" metrics to consider whether a company might become a good *long-term* investment. To make this judgment, we need to understand the underlying economics of the business—how a company makes and spends money, as well as how fast it is growing relative to its peers.

ILLUSTRATION BY ANTHONY FREDA

For example, Table 1 compares Facebook with several companies frequently in the news—Microsoft, Google, Oracle, Apple, Infosys, SAP, IBM, Salesforce, and LinkedIn. The data is from the prior fiscal year, with market values as of January 2012 except for Facebook, for which I use its IPO value from May 2012. The order is determined by the second column, which is *operating profit percentage* (profits from ongoing operations, before taxes, divided by sales—excluding extraordinary income like sales of real estate or a business division). Every year I compile such a list and Microsoft nearly always comes out on top—at least partially the result of selling not hardware but packaged software products, which comprise 90% of revenues (the last column in Table 1). On this measure of operating performance, Facebook does extraordinary well—beating Microsoft, 47% to 39%. (Of course, we now know these numbers excluded the $1.3 billion in stock-related charges that Facebook took in the first quarter after its IPO!)

At the bottom is the professional social networking firm LinkedIn, a prominent 2011 IPO and the smallest firm on the list in sales. It has only 4% operating profits. But a young firm should be spending heavily on salespeople, ads and marketing campaigns, and new products and services. Operating profits represent money left over after these expenses and the direct cost of sales (as well as some other expenses). It would not be surprising for LinkedIn to rank low in operating profits if it has been investing in growth.

The next column gets a lot of attention in business schools and the business press: *gross margin*. This represents sales minus the direct cost of making those sales. In conventional manufacturing, like automobiles or semiconductors, this is an important number because it is costly to bend metal, mold plastic, or transform silicon wafers into microprocessors. It is also revealing in professional service businesses like Infosys or the consulting divisions of SAP, Oracle, and IBM because it represents the direct costs (mostly personnel) of delivering those services. In software products or Internet services, this is not the case. The marginal cost of replicating any digital good is close to zero, though Internet services can require big investments in server farms. But, in general, companies that do not rely on physical products or labor-intensive services should do well on this metric, and this is what we see.

LinkedIn leads in gross margins (84%), followed by Salesforce.com (80%), the pioneer in Software as a Service (SaaS), and then Microsoft (78%) and Facebook (77%). The lag-

Table 1. Comparing Facebook with other companies.

2010/ 11 Jan. 2012	Prior Year Sales ($B)	Op. Profit %	Gross Margin %	Market Value x Sales	Sales Growth %	Sales/Person	Sales, Mktg, GA%	R&D%	Product Sales%
Facebook	$3.7	47%	77%	28x	88%	$1.2 M	20%	10%	15%
Microsoft	$70	39%	78%	3x	12%	$778 K	26%	13%	90%
Google	$29	35%	65%	7x	24%	$1.2 M	16%	13%	4%
Oracle	$36	34%	76%	4x	33%	$333 K	21%	13%	38%
Apple	$108	31%	41%	4x	66%	$1.7 M	7%	2%	94%
Infosys	$6	29%	42%	5x	26%	$46 K	12%	2%	5%
SAP	$16.5	21%	69%	4x	17%	$308 K	26%	14%	26%
IBM	$100	19%	46%	2x	4%	$234 K	19%	6%	41%
Salesforce	$1.7	6%	80%	9x	27%	$321 K	63%	11%	94%
Linkedin	$0.52	4%	84%	13x	215%	$260 K	46%	25%	20%

Source: Compiled from company 10-K, 20-F, and S-1 reports submitted to the U.S. Securities and Exchange Commission, available on the company Web sites under Investor Relations.

Table 2. Examining the Salesforce.com profit rate.

Salesforce.com	2011 ($million)	Percentage
Revenues	$1,657	**100%**
Subscription and Support	1,551	94
Professional Services, etc.	106	6
Cost of Revenues	324	**20**
Subscription and Support	208	13
Professional Services, etc.	116	7
Gross profit (and gross margin)	**1,333**	**80**
Operating expenses		
Research and development	188	11
Marketing and sales	792	48
General and administrative	256	15
Total operating costs	1,236	**74**
Operating profit	97	**6**

Source: Calculated from Salesforce.com, Inc., "Form 10-K," Washington, D.C.: United States Securities and Exchange Commission Fiscal Year Ended January 31, 2011, p. 31.

gard is Apple (41%), which also trailed Microsoft, Google, and Oracle (in addition to Facebook) in operating profit rate. Apple's innovative iPhone, iPad, iPod, and iMac products are at the core of its strategy, and it can still charge a premium. But the low gross margin reflects that Apple's direct costs of making products are much higher than at software or Internet service companies. The more revenue Apple generates from automated digital services, software sales, or transaction fees through iTunes, App Store, iCloud, iBooks, and iAds, then the more its gross margins are likely to rise.

What about *market value times prior year sales*? LinkedIn's IPO price in May 2011 was $45 per share, with an initial market value of $4.25 billion.[7] This was 17 times 2010 revenues of $243 million. A year later, LinkedIn still leads with a valuation of 13 times 2011 revenues ($522 million). Salesforce.com is next (9x), while at the bottom we see IBM (2x) and Microsoft (3x). Even Apple is only 4x—remarkably low for a company that grew sales 66%. Why is that?

LinkedIn's valuation makes more sense when we look at the next column—*sales growth rate* (versus the prior year). Here again, LinkedIn leads at 215%, though 2010 revenues were small. Facebook again impresses at 88%. By contrast, IBM lags at 4%. Low growth and low operating profits (19%) help explain why IBM's market value is so low.

Another metric is *productivity*—here defined as total sales divided by year-end employees. Our star is Apple ($1.7 million per employee), followed by Google and Facebook (each with $1.2 million per employee). But we need to look at other metrics to interpret this number. For example, comparing sales productivity with gross margins indicates if a firm has high sales costs and production expenses—which Apple does, making its high nominal sales productivity less impressive.

An IT services company like Infosys should struggle in sales productivity and it does ($46,000 per employee). It can compensate by paying low wages and using lots of people to generate revenues. In fact, Infosys does this and had impressive gross margins (42%, better than Apple) and a notably high operating profit rate (29%, merely 2% behind Apple). Investors apparently see this and high growth (26%) and value Infosys (5x sales) more than Apple, Oracle, and SAP (4x) or Microsoft (3x) and IBM (2x). But Infosys' dependence on labor-intensive services (95% of sales) suggests that future growth will be difficult. Historical data suggests Infosys grows revenues at approximately a 1:1 ratio to increases in employees. In 2011, Infosys had 131,000 employees and sales of $6

billion. It also reported employee attrition of 17%.[3] To grow sales again by 26% in 2012, assuming the same sales productivity, Infosys must hire 34,000 employees plus another 22,000 to replace departures!

Other numbers help explain why gross margins can be high and operating profits low: general expenses that come after the direct cost of sales (but still before taxes). For software products and SaaS companies, and for Internet service companies like Google and Facebook, these expenses can be huge. Salesforce.com spent a whopping 63% of 2011 revenues simply on sales, marketing, and general and administrative expenses. Add to this 11% of sales spent on research and development (R&D) and we get 74%. Combined with a 20% cost of sales (the inverse of the 80% gross margin), we can see why Salesforce.com managed a profit rate of only 6% (see Table 2). The company was indeed investing heavily to achieve a 27% growth rate.

Similarly, LinkedIn spent 46% of revenues on sales, marketing, and general and administrative expenses, and 25% on research and development: 71%. This total, combined with 16% direct cost of sales (100 minus 84% gross margin), and another 9% of sales in other expenses,[4] explains why LinkedIn shows just 4% of sales as operating profit. But investors seem to like LinkedIn nonetheless, possibly due to high growth and because it does not simply rely on advertising. Fifty percent of last year's revenues came from job placement fees paid by corporate clients and another 20% from premium subscriptions ("product sales").[4]

We can also understand Apple's operating profit rate of 31% more clearly, as well as its low valuation. Apple's growth has been so high (66%) that revenues have probably outpaced prior-year plans for expenses, leaving a surprisingly low percentages for sales, marketing, and administration costs (7% of sales) and research and development (2%). Only Infosys (2%) was in Apple's territory for R&D spending. But, as a service company, with merely 5% product sales, we expect Infosys to do little research and development. We expect Apple to do a lot, and its expense ratios will rise quickly if and when growth slows. Perhaps investors do not believe Apple can continue to grow so fast so cheaply.

The obvious unknown is how much and how fast Facebook will continue to grow, and how much that growth will cost.

The last column in Table 1 points to percentage of revenues coming from *new product sales*. The balance of revenues comes mainly from services or maintenance, including software license renewals. New product sales or subscriptions in software and some hardware businesses (like the iPhone) have high gross margins and high growth potential without adding costly headcount or expensive factories. It is a useful metric for Microsoft, Oracle, SAP, and Salesforce.com, and even for Apple and IBM, but has little meaning for Google and Facebook. Their "products" are free automated services like searching the Internet or connecting with friends. What they sell is primarily advertising. Facebook logged some virtual goods and technology sales (mostly to Zynga), but these amounted to only 15% of revenues.[1] SaaS companies are hybrids with a product delivered and priced like a service, with some technical support and maintenance bundled into a monthly fee. But they still can have very high gross margins and scale economies. This is why Salesforce.com and LinkedIn should have high valuations as long as they are growing fast. But their high expense ratios are worrisome for the future.

Conclusion

The Facebook IPO price seems very expensive in retrospect and assumed a continuation of extremely high growth (and profit) rates. But there really is no way to determine what the price should have been in advance. On "objective" measures like sales growth as well as operating profits, gross margin, sales productivity, and expense ratios, the company looked very strong prior to the IPO and solidly in the class of Microsoft, Google, Apple, and other elite performers. The obvious unknown is how much and how fast Facebook will continue to grow, and how much that growth will cost. User behavior with mobile ads is apparently different and less profitable than with PCs, and Facebook will have to adjust. Investors must also adjust their expectations as they speculate about the future. For example, if Facebook can average 32% annual growth for the next five years, and if the stock price and market cap recover to the IPO level, then the multiple will be a Google-like 7x in 2016. Facebook will then seem much less expensive. But these are big "if's." The bottom line? There may be no science to IPO pricing, but even a quick look at the economics of the business, with some reasonable extrapolations and comparisons to peer firms, does shed some light on what a company is likely to be worth in the future. C

References

1. Facebook, Inc. Form S-1. United States Securities and Exchange Commission, Washington, D.C., Filed February 1, 2012.
2. Hulbert, M. Facebook's stock should trade for $13.80. *The Wall Street Journal* Marketwatch.com (May 25, 2012); http://www.marketwatch.com/story/facebooks-stock-should-trade-for-1380-2012-05-25.
3. Infosys Technologies Ltd. Form 20-F. United States Securities and Exchange Commission, Washington, D.C., Fiscal Year Ending March 31, 2011.
4. LinkedIn Corporation. Form 10-K. United States Securities and Exchange Commission, Washington, D.C., Fiscal Year Ending December 31, 2011.
5. Raice, S. and Letzing, J. Parsing the Facebook fallout. *The Wall Street Journal* (May 30, 2012), B5.
6. Saitto, S. et al. Playing the Facebook blame game. *BusinessWeek* (May 28–June 3, 2012), 45–46.
7. Selyuk, A. and Baldwin, C. LinkedIn IPO prices at $45, but risks real. *Reuters* (May 18, 2011); http://www.reuters.com/article/2011/05/18/us-linkedin-ipo-risks-idUSTRE74H0TL20110518.
8. Sengupta, S. Facebook shares plummet in an earnings letdown. *The New York Times* (July 27, 2012), B1.

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

DOI:10.1145/2347736.2347745

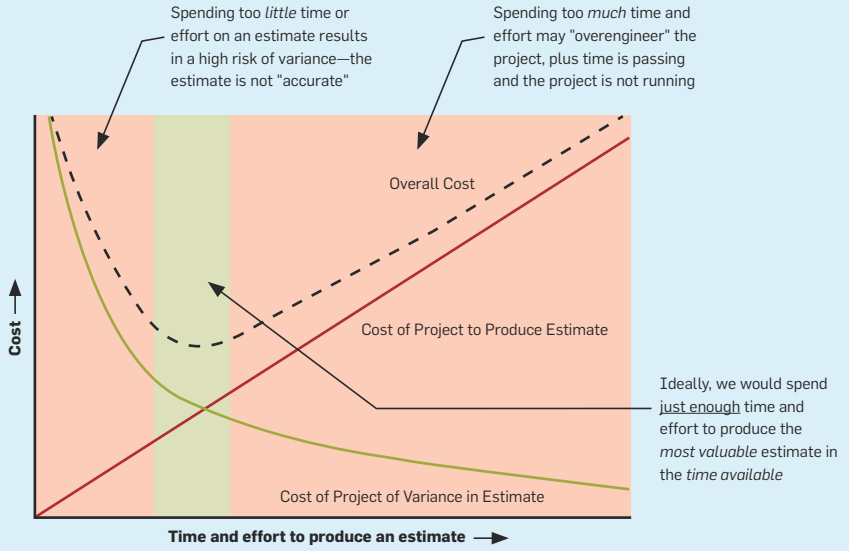
Phillip G. Armour

The Business of Software
The Goldilocks Estimate

Balancing two extremes in project estimation.

Ben was hurrying to his next meeting when his boss stopped him in the hallway. "Ben, I'm heading up to the CEO's office for the budgeting meeting," he said. "Remember the system upgrade we talked about the other day? Could you give me a quick ballpark dollar figure that I can take to the boss just as an example? You won't be held to it, of course..."

Cost versus the time and effort to produce an estimate.



The Drive-By Estimate

A drive-by estimate occurs when a senior manager corners a subordinate and demands an immediate answer to the estimation questions: "When can we get this project done?" "How much will it cost?" and "How many people do we need?" Depending on how much pressure is applied, the unhappy estimator must produce *some* numbers and do it quickly, usually without much research. Estimates derived this way are normally of low quality and making any kind of critical business decision based on them is dangerous and often costly.

The Never-Ending Estimate

At the opposite extreme, sometimes estimation can be a process that goes on and on and on. To make an estimate "safer" and more "accurate," organizations may try to remove all the uncertainty in the project and the data they use to create the estimate. One way to reduce uncertainty is simply to spend more time and effort in analyzing the situation—spending more time and effort on something almost always produces a better outcome. However, when estimating a project, the work we have to do to remove the uncertainty in the estimate is pretty much the same work we have to do to actually run the project. I have seen companies where, in order to decide if they should run a project and what resources it will need, they de facto run the project and use up the resources.

Ideally, an estimation process would aim for the Goldilocks "sweet spot" where we spend enough time and effort on the estimate to be sufficiently confident in the result, but not so much that we overengineer the estimate and actually start working the project.

Inverse Exponential Value

Most uncertainty reduction processes (such as inspections, testing, and estimation) follow an exponential decay curve. Initially there is a lot of low-hanging fruit—items that are quite clear and easy to find and fix. As we progressively clean up the more obvious issues, the remaining ones tend to be more obscure, ambiguous, and difficult to find and it takes us longer and longer to resolve the next uncertain item.

Linear Cost

If we allocate a certain number of people to produce an estimate and let them work until we are "satisfied" with the result (whatever that might mean), the cost profile will be linear and ascending with respect to time and effort. The combination of these two graphs produces a U-shaped profile, as shown in the figure here. Too far to the left and the likely cost of a poor estimate will be high. Too far to the right and the work done in producing the estimate is not balanced by

its value. Indeed, the project might actually be going ahead without its funding being reviewed and approved.

The optimal estimate time and cost occurs at the saddle point and we can calculate where that should be. It will not be the same point for the early feasibility assessment of a very large complex system and for late planning of a very small simple system so we need to adjust for different types of projects. NASA's Deep Space Network (DSN) project[1] developed a mechanism for this calculation based on just two simple parameters:

▸ **Target Cost of Project.** This is the *goal cost* of the project as first envisaged in the project concept. It is not the estimated cost of the project (which has not been calculated yet). The rationale is, when we expect and plan to spend a lot of money on a project, we should also expect to spend proportionally more for the estimate, simply because more is at risk.

▸ **The Business Practice being Supported.** Very early in a project, it is normal that the data available is sparse, ambiguous, and of low quality. At this point in time, the business practice being supported is a conceptual one: should we consider this project? If we were to invest in it, would we get a reasonable return? For such estimates, investing too much time and effort is usually not worthwhile. The estimate produced will be (or should be) only used to approve the project for continued analysis or to reject it altogether. In the latter case, developing a highly detailed and expensive estimate for a project that will not even start is not a good use of resources.

Later in a life cycle, assuming the project has been given the go-ahead, we have better data and we are at the point of committing significant resources to move the project forward. Therefore it is worth spending more time and money to produce an estimate. Here the business practice being supported is one of financial budgeting and resource allocation.

Later still, when the resources have been allocated and the project is ready to launch, we need even more detailed estimates and we often have high-quality data to support them. These estimates will provide the bounding box inside which the project plan will sit and supports the project and manpower planning activity.

The Formula. The formula devised by NASA is a simple one:

$$\text{Cost\_of\_Estimate} = \text{Practice\_Parameter} * \text{Target\_Cost}^{0.35}$$

Where the values of **Practice_Parameter** are related to the estimation phase by Table 1, and the exponent of 0.35 is fixed (for NASA).

Using the Formula. Since the formula only uses two simple parameters, it is easy to apply—a project with a Target Cost of (say) $10m should expect to spend the amounts given in Table 2. Simply by dividing the cost by a personnel rate we can arrive at a simple effort value and using a personnel loading factor we can deduce an approximate schedule to produce the estimate. Using an average personnel rate of $100/staff hour and three people allocated half time of eight-hour days to producing these estimates, our $10m project estimates would take the effort and time given in Table 3.

Table 1. Phase estimation.

Phase	Expected Precision	Value
Concept	Order of Magnitude	24
Commitment	Budgetary Estimate	60
Planning	Definitive Estimate	115

Table 2. Cost estimation.

Estimate Type	Cost of Estimate
Concept Estimate	$6,764
Commitment Estimate	$16,910
Planning Estimate	$32,411

Table 3. Effort and time estimation.

Estimate Type	Effort	Value
Concept Estimate	68 SHrs	6 Days
Commitment Estimate	169 SHrs	14 Days
Planning Estimate	324 SHrs	27 Days

Some Caveats

There are some caveats to using this approach intelligently, three of which I will address here. One is that the effort/time ratio is not linear for heavy staff loading—we cannot get the 68 Staff Hour estimate in one hour elapsed time by putting 68 people on the task for instance.

Secondly the calculation is driven by the **Target Cost** which is only, well, a target. What happens if the target is way off? A common result of producing an estimate based on the calculated time and effort determined by this formula is that we find the Target Cost is actually not achievable. This is, of course, the point of creating the estimate. In this case it is perfectly appropriate to expend whatever extra time and effort is indicated by the ***new*** (estimated) cost to further refine the estimate.

Also, the formula assumes the quality of data is only dependent on the phase and target cost, so the Planning Estimate for two $10m projects would be equivalent. If this is not the case, if one project is dealing with well-known quantities while the other is something completely new, some adjustment to the estimate effort might be necessary. The key thing is to control the estimate investment to achieve the most optimal cost-effective result.

The Never-Ending Drive-By

After Ben's boss had taken the ballpark numbers to the CEO they were, quite predictably, cast in concrete and used to fund the project—though only after they were trimmed somewhat to take out the "fat." The resulting project was a case study in under-resourcing and experienced enormous overruns. Having learned their lesson from this, much greater "accuracy" was demanded of future estimates, to the point that it was decreed that the project manager would be held personally responsible for any variation from the estimate of more than 5%—plus or minus(!). Using these guidelines, producing the "estimate" on a subsequent project took close to 35% of the expected budget before it was decided it should be canceled. Both extremes, of course, drew the public ire of senior executives.

Clearly, in the business of software, our estimation effort needs to be "just right" if we are not to suffer the anger of Papa Bear. C

Reference

1. Remer, D.S. and Buchanan, H.R. A model for the cost of doing a cost estimate. *The Telecommunications and Data Acquisition Progress Report* 42-110, NASA Jet Propulsion Laboratory, Pasadena, CA (1992), 278–283.

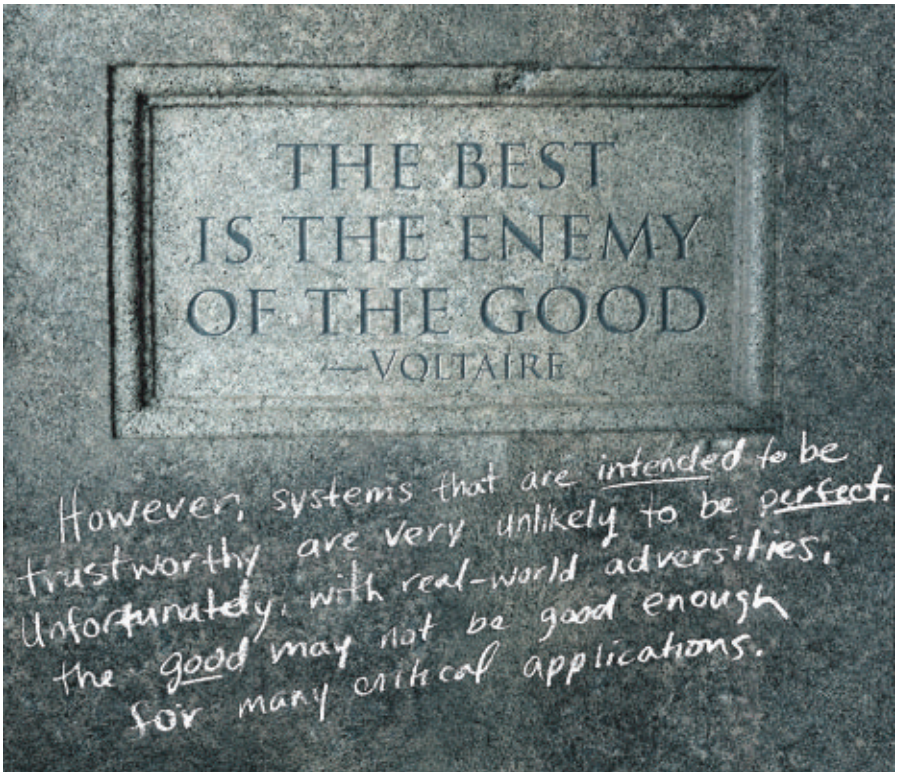
Phillip G. Armour (armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL, and a consultant at QSM Inc., McLean, VA.

DOI:10.1145/2347736.2347746

Peter G. Neumann

Inside Risks
The Foresight Saga, Redux

Short-term thinking is the enemy of the long-term future.



THE PRIMARY MESSAGE of this column is that optimizing computer-related alternatives in the short term may be seriously detrimental in the long term, particularly where trustworthy behavior is essential—with respect to satisfying requirements for security, reliability, resilience, human safety, human usability, and so on. In general, we know from experience that it can be very difficult to retrofit systems with new implementations to make them trustworthy, especially if they were developed on insecure platforms without the benefit of security-aware development practices. Thus, a well-reasoned understanding of the trade-offs is essential before potentially sacrificing possible future opportunities in an effort to satisfy short-term goals. One complicating factor is that much more knowledge of the past and the present—and appreciation of the effects of possible futures—is needed to intelligently making such trade-offs.

This column is intended for a diverse audience relating to computer-related risks, including notably researchers, system developers, curricula creators, teachers, politicians, and many others. The starting point is that almost everything that we necessarily must depend on relating to information systems, networks, and national infrastructures is today for the most part riddled with flaws that can be exploited or triggered as a result of willful or accidental misuse, hardware and software failures, environmental hazards such as power outages and earthquakes, and so on. Similarly, there is often an inherent dependence on system administrators, with the hope that they are nearly infallible.

For example, software vendors tend to release new system versions with known bugs, realizing that they can economize by letting users and application developers discover the bugs! Critics might argue that this can lead to problems that could be avoided if the designers analyzed long-term consequences—including anticipating attacks exploiting serious vulnerabilities. In addition, some software tends to outlive hardware, with flawed software sometimes lingering on. Worse yet, bad interfaces tend to persist—due to commitments on backward compatibility. Furthermore, easy programmability and a desire for lowest common denominators often trumps the advantages of functional abstractions, strong typing, controlled memory management, formal reasoning, and other somewhat more farsighted approaches.

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

As another example, hardware architectures that enabled trustworthy layered enforcement of fine-grained least-privilege access policies are no longer in the mainstream, which tends to limit the trustworthiness that can be achieved in software. Also, problematic hardware instructions tend to survive, again for backward compatibility.

The resulting shortfalls with respect to desired system, network, and enterprise behavior can be very serious. That has been noted repeatedly in the preceding 227 Inside Risks columns (including a similarly titled column[5]), and does not need much further elaboration here.

Some inspiration for writing this column came from the ACM Turing Centenary Celebration this past June, which attempted to look back at the past and to consider what might happen in the future. The talks by the Turing laureates and other invited participants ranged from near to far into both the past and the future—reminding us of some of the laureates' important past contributions, while at the same time giving diverse perspectives on the future.

Consider some of the general guidance that has emerged from our collective pasts. This may seem similar to earlier Inside Risks columns, but bears repeating because it is not widely observed in practice.

Requirements. We should anticipate the long-term needs that a system or network of systems must satisfy, and plan the development to overcome potential obstacles that might arise, even if the initial focus is on only short-term needs. This might seem to be common wisdom, but is in reality quite rare. Common requirements for security, reliability, fault tolerance, resilience, diagnostic ability, adaptability, human safety, interoperability, long-term evolvability, trustworthiness, and assurance evaluations are generally much too weak. Furthermore, highly distributed control with highly networked or cloud-dependent systems demands much greater foresight. Also, refining requirements on the fly often causes serious development problems.

System development. We can gain significantly by using effective design methodologies, basic principles, well-reasoned system/network architectures, horizontal (modular) and vertical (layered) abstraction with encapsulation and strong typing, predictable composability, use of formal methods for assurance where most effective, suitable choices of languages for requirements, specifications, programming, and so on—compatible with the sophistication of the requirements and the expertise of the developers.

There is much to be gained from farsighted thinking that also enables short-term achievements.

Research. Solving problems more generally with preplanned evolution, rather than just barely attaining short-term requirements, can be very advantageous. With some foresight and care, this can be done without losing much efficiency. Often a slightly more general solution can prove to be more effective in the long run. There is much to be gained from farsighted thinking that also enables short-term achievements. Thus, it seems most wise not to focus on one without the other. Some new clean-slate approaches are emerging in response to the needs for much greater system and enterprise trustworthiness, as are executable hardware-software co-design languages (for example, see Dave[1]). Such efforts have long-term goals, but can also have significant short-term results—especially in an ongoing formally based hybrid capability-based hardware-software architecture,[6,7] which allows legacy software to coexist securely with newly developed highly trustworthy hardware-software.

Roles of science and engineering. Computer science has evolved into a very useful collection of scientific principles and methods, with significant advances in many areas—although the use of systemwide metrics and evaluations of trustworthiness still have significant room for advances. On the other hand, the so-called field of software engineering is still sorely lacking in engineering foundations and discipline, and therefore unlike well-established engineering fields. Theoretical bases and supporting tools can be very helpful to engineering practice, in simplifying and analyzing complex systems, and especially when it comes to long-term thinking. Metatheories enhancing the predictable composition of requirements, subsystems, and measures of trustworthiness enabling evaluations of emergent properties of entire systems would be extraordinarily valuable in facilitating long-term thinking.

Experimentation. Experimental computer science and engineering tend to lurk in the background somewhat as orphan stepchildren. Exploration of alternative system architectures, easily reconfigured testbeds, parameterizable symbolic analyses and simulations, layered composable evaluations, and so on would all be very supportive of long-term thinking.

Education. We must do better at teaching principles and theoretical bases, not just how to write programs or use tools for development and analysis. Thinking to support long-term advances requires much more than rote learning, lowest-common denominator standards and academic evaluation criteria, and short-sighted elimination of historically relevant departments deemed of less commercial value. In a down economy, farsighted education remains a most essential need for the future.

People. Accepting that to err is human, and that malicious misuse is here to stay on an increasingly more pervasive scale, we need to spend more effort on anticipating the potential shortcomings of system interfaces for human-system interactions, and design systems that are much more people tolerant and people resilient. The burden on today's system administrators is enormous, but could also be greatly reduced by the presence of more long-term thinking in system design and implementations.

Cognitive vs. subconscious thinking. A recent book by Daniel Kahnemann[2] revisits some of the earlier studies of left-brain (logical, linear, methodical) versus right-brain (intuitive, subconscious, out of the box) thinking. Our educational systems tend to prod the former, while in some cases neglecting the latter. Kahnemann's fast thinking (more or less right brain) tends to be checked or modulated by slow thinking (more or less left brain). What is important in the present context is that long-term thinking inherently requires a well-integrated combination of both[4] as applied to computer system development). A holistic balance of human intelligence, experience, memory, ingenuity, creativity, and collective wisdom, with slow and fast thinking, is extremely valuable in exploring the trade-offs between short-term gains and long-term potentials within some sort of holistic big-picture foresight.

Innovation. New computing technologies tend to introduce new security vulnerabilities, as well as reintroduce earlier ones. This has occurred over many decades, with buffer-overflow attacks, man-in-the-middle attacks (for example, Needham-Schroeder), distributed denial-of-service attacks, physical attacks on crypto and mobile devices, spam, and both large-scale and targeted social engineering. When security problems continually recur, it might be time to do something different. Perhaps this tendency can be overcome with formally based architectures and developments.

High assurance. Formal methods have always had enormous promise, but have been very difficult to use. As they become more integrally and seamlessly embedded in the development process and more diverse (encompassing a variety of solvers), they may finally become more viable. The biggest remaining challenge may involve designing and analyzing systems that are composed from components that themselves have been thoroughly analyzed and whose analyses are themselves composable.

Reliance on the marketplace. Market success in hardware and software tends to produce winners that may not be adequately trustworthy. The alternative of clean-slate developments is generally unpopular and difficult to pursue in commercial enterprises, but currently reborn in several DARPA research and development programs such as CRASH (Clean-slate design of Resilient, Adaptable, Survivable Hosts) and MRC (Mission-oriented Resilient Clouds).

Interactions with other disciplines. All of the considerations noted here can be much more effective if motivated by real applications such as medical information systems, telerobotic surgery systems, real-time control systems, low-power multipurpose mobile devices, and so on. The application of long-term thinking to such applications is essential to ensure satisfaction of their critical requirements. An earlier article[3] characterizes the holistic nature of energy, agriculture, and health care—each of which requires a deeper understanding of the need for long-term thinking—and contrasts them with various computer-related risks issues.

A few illustrative requirements:

Computer system security and integrity. The relative ease of perpetrating certain attacks such as viruses, worms, and exploits such as Conficker and Stuxnet suggests that long-term concerns have been largely ignored. With particular attention to critical national infrastructure systems, we seem to have arrived at lowest-common-denominator systems and have had to live with them, in the absence of better alternatives. The standards for acceptable levels of security and best practices are typically much too simplistic and basically inadequate. Relevant efforts of various research and development communities seem to be largely ignored.

Computer-aided elections suffer from all of the aforementioned difficulties, plus more. The proprietary nature of commercial systems is a serious obstacle to meaningful oversight, as is the lack of constructive integrity throughout the entire election cycle, the lack of incisive audit trails, extensive opportunities for insider misuse of technology, and manipulation of the externalities—for example, registration, authentication, disenfranchisement, and so on.

The financial crises of the past few years present another example in which

the almost total absence of realistic long-term thinking and oversight contributed to worldwide economic problems. Optimizing for short-term gains often tends to run counter to long-term success (except for the insider investors, who having taken their profits have little interest in the more distant future). Although this may not seem like a typical Inside Risks case, it is certainly illustrative of the main theme here.

The Future

The ACM Risks Forum has helped dramatize past experiences with the effects of design faults, system security vulnerabilities, system failures and errors, and the pervasive roles of people throughout. Previous columns have highlighted the importance of understanding these experiences and applying them diligently in the future.

Social engineering (exploiting human weaknesses) is a significant factor in system penetrations, inadvertent insider misuse, and above all the spread of malicious malware and other forms of malicious misuse combined with the existence of vulnerable systems whose exploitation permits email and Web-based scams to facilitate online identity fraud and other forms of malfeasance. For example, innocently clicking on a seemingly legitimate link is a common failing. Ultimately, no matter how trustworthy individual hosts, servers, and networks might become, users will need much greater help in detecting and avoiding scams and deception. For example, human frailties such as greed, gullibility, and obliviousness to the risks will clearly persist. Thus, widespread computer literacy is an urgent goal, involving both short-term and long-term aspects. Nevertheless, designing systems, networks, and applications that wherever possible effectively mask the overall complexity and concerns for trustworthiness must also be a long-term goal.

Conclusion

Although this column has barely scratched the surface of an iceberg-like collection of problems, it addresses the need for urgently developing compelling logical and realistic justifications for embedding long-term thinking into our planning. Commodity hardware/software aims at the mass market; on the other hand, what is needed for certain critical applications such as national infrastructures, secure and resilient cloud servers, and so on is the existence of meaningfully trustworthy networked systems. Thus, there is a major disconnect that requires some long-term thinking to overcome. Unfortunately, the real-world arguments for short-term optimization are likely to continue to prevail unless significant external and internal efforts are made to address some of the long-term needs. C

New computing technologies tend to introduce new security vulnerabilities, as well as reintroduce earlier ones.

References

1. Dave, N. A Unified Model for Hardware/Software Codesign. Ph.D. thesis, MIT, Cambridge, MA, 2011.
2. Kahnemann, D. *Thinking, Fast and Slow.* Farrar, Strauss and Giroux, 2011.
3. Neumann, P.G. Holistic systems. *ACM ACM SIGSOFT Software Engineering Notes 31*, 6 (Nov. 2006), 4–5; http://www.csl.sri.com/neumann/holistic.pdf.
4. Neumann, P.G. Psychosocial implications of computer software development and use: Zen and the art of computing. In *Theory and Practice of Software Technology*, D. Ferrari, M. Bolognani, and J. Goguen, Eds., North-Holland, 1983, 221–232.
5. Neumann, P.G. The foresight saga. *Commun. ACM 50*, 9 (Sept. 2006); http://www.csl.sri.com/neumann/insiderisks06.html#195.
6. Neumann, P.G. and Watson, R.N.M. Capabilities revisited: A holistic approach to bottom-to-top assurance of trustworthy systems. Fourth Layered Assurance Workshop, U.S. Air Force Cryptographic Modernization Office and AFRL (Austin, TX, Dec. 2010); http://www.csl.sri.com/neumann/law10.pdf.
7. Watson, R.N.M. et al. CHERI: A research platform deconflating hardware virtualization and protection, runtime environments/systems, layering, and virtualized environments. RESoLVE workshop (London, U.K., Mar. 3, 2012); http://www.csl.sri.com/neumann/2012resolve-cheri.pdf.

Peter G. Neumann (neumann@csl.sri.com) chairs the ACM Committee on Computers and Public Policy and moderates the ACM Risks Forum (http://www.risks.org). He is very grateful to his committee members for their long-standing incisive feedback on Inside Risks columns—including this one!

Calendar of Events

October 15–18
31st International Conference on Conceptual Modelings,
Florence, Italy,
Contact: De Antonellis Valeria,
Email: valeria.deantonellis@ing.unibs.it

October 15–19
ACM SIGUCCS Annual Conference,
Memphis, TN,
Sponsored: SIGUCCS,
Contact: Carol Rhodes,
Phone: 812-856-2007,
Email: csrhodes@indiana.edu

October 15–19
Conference on Systems, Programming, and Applications: Software for Humanity,
Tucson, AZ,
Sponsored: SIGPLAN,
Contact: Gary T. Leavens,
Phone: 407-823-4758,
Email: leavens@eecs.ucf.edu

October 21–24
Conference on Systems, Programming, and Applications: Software for Humanity,
Tucson, AZ,
Sponsored: SIGPLAN,
Contact: Gary T. Leavens,
Phone: 407-823-4758,
Email: leavens@eecs.ucf.edu

October 22–24
The 14th International ACM SIGACCESS Conference on Computers and Accessibility,
Boulder, CO,
Sponsored: SIGACCESS,
Contact: Matt Huenerfauth,
Phone: 646-639-3815,
Email: matt@cs.qc.cuny.edu

October 22–26
International Conference on Network and Service Management,
Las Vegas, NV,
Contact: Medhi Deep,
Email: dmedhi@umkc.edu

October 22–26
5th International Conference of Security of Information and Networks,
Jaipur, India,
Contact: Manoj Singh Gaur,
Email: gaurms@gmail.com

DOI:10.1145/2347736.2347747

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious
A Nice Piece of Code

Colorful metaphors and properly reusing functions.

In my last column (A System Is Not a Product, August *Communications*) I mentioned I had recently read two pieces of code that had actually lowered, rather than raised, my blood pressure. As promised, this column covers that second piece of code.

One of the things that can make me *really* like a piece of code—other than the obvious ones such as decent documentation, proper indenting, and rational variable names—is when a function or subsystem is properly reused. Over the past month, I have been reading the IPFW (IP Firewall) code written by Luigi Rizzo at the University of Pisa, which is one of the firewalls available in FreeBSD. Like any firewall, IPFW needs to examine packets and then decide to drop, modify, or pass the packets unchanged through the system. Having reviewed several pieces of software that do similar work, I have to say that IPFW does the best job of reusing the code around it. Here are two examples.

Part of the job of a firewall is to classify packets and then decide what to do with them. There are a few ways to go about this, but what IPFW does is quite elegant. It reuses a tried and tested idea from another place in the kernel, the BPF (Berkeley Packet Filter). BPF classifies packets using a set of opcodes—sort of like a machine language for processing network packet headers—to decide whether a packet matches a filter that has been specified by the user. Using opcodes and a state machine for packet classification leads to a flexible and compact implementation of the packet classifier, compared with hand coding rules for later use. IPFW extends the set of opcodes that can be used for classifying packets, but the idea is exactly the same, and the resulting code is easy to read and understand, and therefore easier to maintain and less likely to contain bugs that might let malicious packets through. The entire state machine that executes any and all firewall rules in IPFW is only 1,200 lines of C code, including comments. Another advantage of using a set of opcodes to express the packet-processing rules is that the entire chunk of C code, which is really a bytecode interpreter, can be replaced by just-in-time compiled code, generated by an optimizing compiler. This leads to an even greater increase in packet-processing speed.

A more direct case of reuse is how IPFW directly reuses the kernel's routing-table code to store its own address lookup tables. Many of the rules in a firewall make reference to the source or destination address of a packet. While it is quite possible to write your own routines for storing and retrieving network addresses—and many people have—there is no need to rewrite this code, in particular when your firewall code will already be linked into a program that has such routines avail-

An example of good code.

```
1  int
2  ipfw_lookup_table(struct ip_fw_chain *ch, uint16_t tbl, in_addr_t addr,
       uint32_t *val)
3  {
4          struct radix_node_head *rnh;
5          struct table_entry *ent;
6          struct sockaddr_in sa;
7
8          if (tbl >= IPFW_TABLES_MAX)
9                  return (0);
10         rnh = ch->tables[tbl];
11         KEY_LEN(sa) = 8;
12         sa.sin_addr.s_addr = addr;
13         ent = (struct table_entry *)(rnh->rnh_lookup(&sa, NULL, rnh));
14         if (ent != NULL) {
15                 *val = ent->value;
16                   return (1);
17          }
18          return (0);
19 }
```

able. The radix code in the kernel can manage any type of key/value lookup, although it is optimized to handle network addresses and associated masks. The IPFW table-management code is really just a simple wrapper around the radix code, as can be seen in the accompanying figure.

All this code does is take arguments understood by IPFW, such as the chain of rules (`ch`), address table (`tbl`), address being sought (`addr`), and pack them up in a way that is usable by the radix code, which is called on line 13. The value is returned in the last argument to the function. All of the other functions in the table-management code, which add, delete, and list entries in the table, look very much the same. They are wrappers around the radix code. Treating the routing-table code as a library, as IPFW does, means writing less complex and tedious code, and results in a mere 200 lines of C code, including comments, to implement tables of network addresses. It is this sort of reuse, not the tortured kind that I more often come across, that leads me to praise this code.

Don't worry, I am sure next time I will be back to ranting about some bad bit of code, but I have to say it has been a nice surprise to have found two well-written pieces of code in two months. I think it is some sort of record.

KV

Dear KV,

One of the people on my current project keeps complaining about my use of "colorful metaphors" in code. While I understand that I should not be checking these sorts of things into our source repo, I cannot see why he complains when he sees them on my screen. I mostly use such words for debugging messages because they are shocking enough to stand out from the rest of the log messages produced by our software. I cannot really believe that KV would object to a programmer adding a bit of salt to log messages.

Kolorful Koder

Dear Kolorful,

I can understand why you might think that I might be a prolific user of colorful metaphors, given some of the things I write about in this column. You are correct, and my coworkers can tell you that, because of my occasional outbursts when dealing with particularly horrific bits of code, they have heard me say a thing or two they wish they could now forget.

One of the things that can make me *really* like a piece of code is when a function or subsystem is properly reused.

Unfortunately, for you at least, I have to come down on the side of your coworker in this dispute. While I am sure you have faithfully marked every place your code might exclaim a colorful metaphor with the well-known comment "XXX Remove This!" the fact is that if you do this enough, someday, and usually on quite the wrong day, you are going to forget. You probably think you won't, but the risk is not worth the eventual hassle. I have been through that hassle, and I am glad that, for once, the problem was not my fault.

More than a decade ago I worked for a company that produced a software IDE (integrated development environment) and some associated low-level software. One of the IDE's limitations, on a certain platform, was that every project saved by the IDE had to have an appropriate extension—those letters after the dot—that provide a clue about what type of file has just been saved. While programmers are quite used to giving their files such descriptive monikers as notes.txt, main.c, and stdlib.h, it turns out that not everyone is familiar with this sort of naming standard, and some even prefer names such as Project1 and Project2, without any type of identifying extension.

The programmer working on the IDE decided that if the user of his program declined to add an extension to the project file name, he would add one for them. He chose a four-letter word that rhymes with duck. I am not sure if he meant this to go out in the release, as a way of pointing out customers who refused to use file extensions, or if it was something he meant to change before the release, but in the end it didn't matter. Within days of our 1.0.1 maintenance release of the IDE, there was a 1.0.1b release with a single change. I do not remember if the b release had a note saying what changed, but all of the engineers working on the software knew the real reason.

Amazingly, the programmer who did this got to keep his job. I suspect there were two reasons for this, the first being that he was actually a pretty good programmer, and the second that he was the only one in the company willing to support the IDE on the platform he was working on.

While this is a pretty extreme example of a colorful metaphor gone wrong, and while I know there are programmers who will leave extremely strong language in comments, I have to say I frown on this as well. Your code is your legacy, and while your mother might never see it, you should still only check in code that would not shock her should she choose to read it.

KV

Related articles on queue.acm.org

Passing a Language through the Eye of a Needle
Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes
http://queue.acm.org/detail.cfm?id=1983083

Syntactic Heroin
Rodney Bates
http://queue.acm.org/detail.cfm?id=1071738

Debugging AJAX in Production
Eric Schrock
http://queue.acm.org/detail.cfm?id=1515745

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

DOI:10.1145/2347736.2347748

Jean-François Blanchette

Viewpoint
Computing as if Infrastructure Mattered

Understanding the technical and social fundamentals of the computing infrastructure is essential in the continuously evolving technological realm.

Since its original formulation by Jeannette Wing in 2006, "computational thinking"[9] has generated considerable interest as an approach that promises to solve a persistent gap: the dramatic expansion of the role of computing in every dimension of society and the general lack of appreciation by the public of its underlying principles. Wing argues this gap must not only be bridged, it must be reversed: the extraordinary success of computing technologies is proof that, from recursion to time-space trade-offs, the conceptual tools developed by computer scientists have broad application beyond mere computing. Indeed, Wing contends the ability to understand and apply fundamental computational principles to a wide range of human endeavors will increasingly count as a core literacy skill for the information age. Computational thinking will thus provide "a basis for lifelong learning of increasingly new and advanced computational concepts and technologies."[5]

While the spread of computational thinking will clearly do wonders to disseminate the *intellectual* achievements of the field, I want to argue that it places insufficient emphasis on the equally significant *social* and *material* contribution of the field, the computing infrastructure. It is this enormously complex system of material resources, institutional practices, economic structures, standards, and regulations that provides the existential ground for the provision of computing services. Consequently, it is understanding of the dynamics of that infrastructure that is the essential skill required for a world where, from automobile repair to the humanities, the evolution of countless professional fields has become intimately tied to the evolution of computing itself. Indeed, in pursuing their professional opportunities, students and practitioners from a wide range of disciplines will need to answer a broad range of infrastructure-related questions. For example,

What is required is a set of skills to analyze the complex forces that direct infrastructural evolution.

- What data formats and storage strategies offer the best guarantees for long-term preservation of digital assets, given a particular organization's workflow, existing computing assets, budgetary constraints, and regulatory requirements?
- How can the economies of scale afforded by cloud computing be best leveraged for a given project? What cost, efficiency, and risk trade-offs should be factored in?
- Among the dozens of metadata standards available, which one offers the best return on investment for discovery and access for a given type of digital resource?
- What new and profitable information services can be built using the Google Books API? How might these services take advantage of advances in interface and sensing technologies, including touch, speech, eye-tracking?
- What are the consequences of adopting open-source software for an institution? How does one discriminate between the different types of governance and economic structures that undergird different open-source projects?

In answering such questions, application-specific skills and fundamental principles of computation provide little guidance. Nor do they clarify the technology-related headlines of major newspapers and magazines: the pros and cons of Net neutrality, the intensity of codec warfare, or the defining infrastructural work of our time, the shift to cloud computing. To clarify such issues, current and future professionals must have access to the material and technical fundamentals of the computing infrastructure and the principles of its social organization. Indeed, for the graduate professional degrees that are increasingly becoming the educational baseline for the workforce,[7] what is required is a set of skills to analyze the

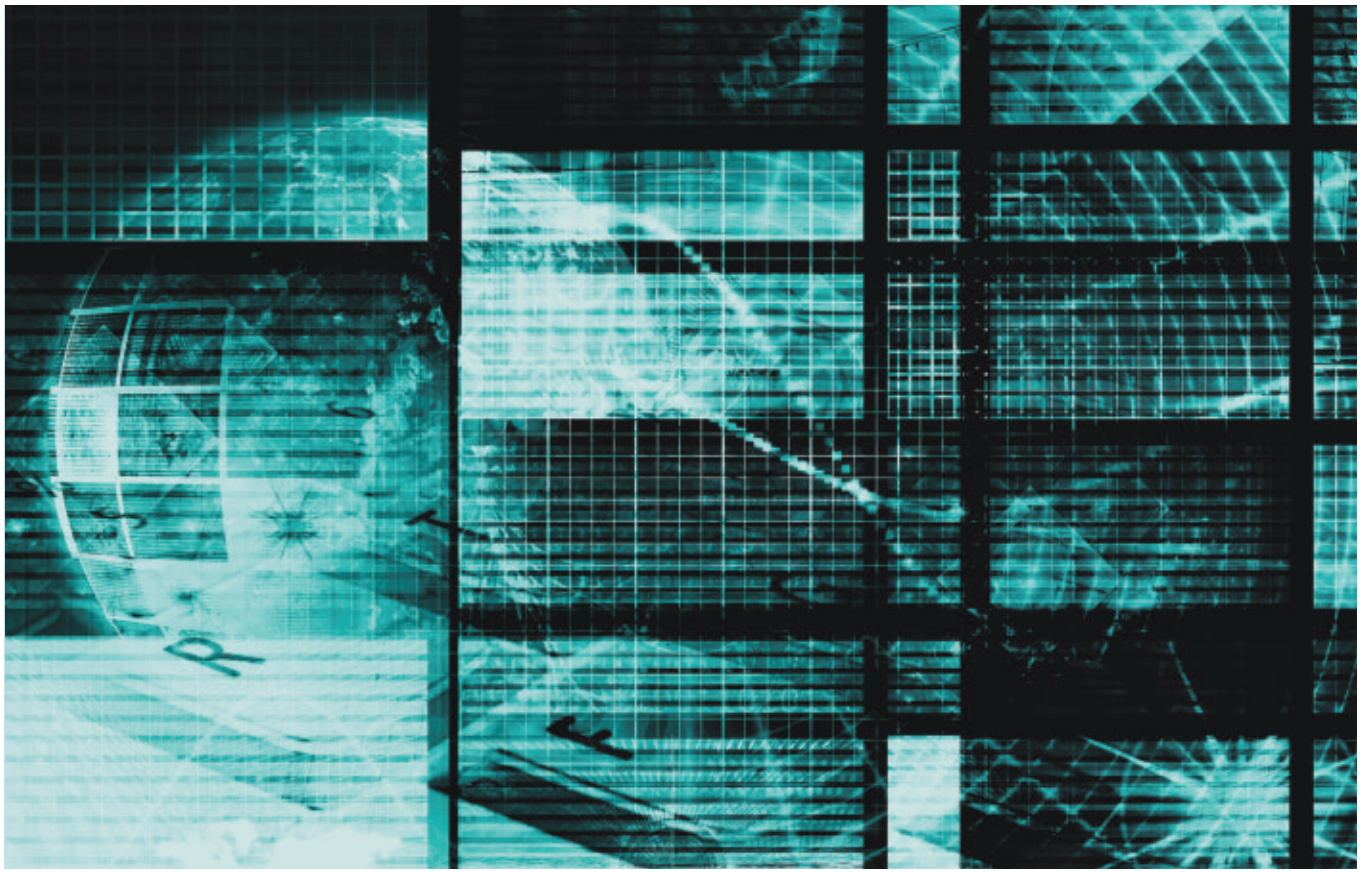


IMAGE BY KENTOH/SHUTTERSTOCK.COM

complex forces that direct infrastructural evolution. It is such skills that provide the means to anticipate the curve ahead in a continuously evolving technological world.

How Infrastructure Moves

By definition, all infrastructures strive for invisibility. Users are understandably interested in the applications that perform useful services in their lives, rather than in the layers of system abstractions or the physical components that make these services possible. Yet whether it takes place at the level of operating systems, data centers, communication protocols, file formats, standardization, or policy work, the design, planning, and operation of the computing infrastructure is a major professional activity of computer scientists. The remarkable waves of social and technical innovation unleashed by the deployment of the Internet have focused attention on some dimensions of this activity, in particular, the TCP/IP protocols and the governance mechanisms that have guided their design and implementation (for example, see Van Schewick[8]). The enshrinement of the Internet's modular structure as the technical foundation of Net neutrality has further highlighted the links between infrastructure, public policy, and economic development. These links remind us that infrastructural design is never merely dependent on technical imperatives, but also on sustaining a delicate balance between serving the general interest and maintaining economic competitiveness among infrastructural stakeholders.

This balancing act is inscribed deep within the very DNA of the computing infrastructure, as it attempts to fulfill two distinct objectives: on the one hand, mediate applications' access to the physical resources of computation—processing, storage, and connectivity—through layered abstractions; on the other hand, multiplex these limited resources so as to efficiently respond to competing demands from applications. The inherent difficulties in simultaneously meeting these two goals set in motion a series of *infrastructural dynamics*—persistence, efficiency trade-offs, scarcity, drift—that gives the computing infrastructure its distinctive evolutionary path.

In spite of the furious pace of IT innovation, the computing infrastructure evolves very slowly.

Infrastructure persists. In spite of the furious pace of IT innovation, the computing infrastructure evolves very slowly. Indeed, the system abstractions that dominate the processing, media, and transport stacks have been around for decades—more than 65 years for the von Neumann model, 45 years for the file and the packet. Abstractions persist as they become embodied in hardware (for example, routers), software (for example, protocols), and institutions (for example, programming textbooks). They further persist through the inherent rigidity of modular decomposition,

since changes to a module's interface requires renegotiating boundaries with all communicating modules. Long-term persistence provides the stability necessary for economies of scale to take hold. It is such economies that have afforded the von Neumann machine a better cost/processing power ratio than the numerous alternative architectures that for decades have failed to supplant it. At the same time, the high costs of infrastructural investments ensure computing resources are repurposed rather than merely replaced. The resource stacks must thus compose with different types of materials—serial and parallel architectures, twisted pair and fiber, tape and flash storage—and ensure their backward compatibility. Infrastructural change thus proceeds conservatively through mutation and hybridization, rather than making an outright break with the past.

Trades-offs all the way down. Modularity is a powerful design strategy: by decoupling abstraction from implementation, it breaks down complex systems in independent yet coordinated organizational units and provides for flexibility in coping with technical change. In doing so, modularity constitutes the primary social and technical order of the computing infrastructure. It is thus remarkable that the costs of modularity are rarely noted in the literature. The flexibility modularity brings to abstraction and implementation is always incurred at the price of efficiency trade-offs—for example, the von Neumann architecture and the serial model of programming it inherently favors. Because of these infrastructural biases, the pax romana of modularity is always under threat, under pressure to extract more computational work from the current organization of the stacks. This tension becomes particularly apparent as new types of computational resources require integration within the infrastructure: the shift from singlecore to multicore, magnetic to flash media, wireline to wireless will reverberate throughout the stacks, as efficiency trade-offs are renegotiated.[1] Abstraction and implementation are thus always in tension, a tension that provides a key entry point for analyzing infrastructural change.[2]

Computer science textbooks remain primarily interested in abstraction as the fundamental practice of the field.

Infrastructure manages scarce resources. Data centers already consume 3% of the world's total electrical output, a number that uncomfortably connects computing cycles to coal extraction.[4] The computing infrastructure is however not merely concerned with managing the scarcity of electrical power, but also that of processing, storage, and connectivity. Abstractions not only relieve programmers from the burdens of keeping track of the finiteness of resources, but also from how these are shared among competing applications. But sharing a resource also inevitably entails various efficiency trade-offs, favoring some types of applications over others. Reliance on networked computing for an ever broader range of essential services—from telesurgery to intelligent transportation and national security—will require providers to devise policies for prioritizing competing demands on shared computational resources, with much more significant implications than mere jittery music videos.

Slow drift. The computing infrastructure is a constantly evolving system, continuously responding to and integrating growth in size and traffic, technical evolution and decay, new applications, services and implementations, emergent behaviors, and so forth. This evolution is constrained by the dynamics outlined previously—persistence, efficiency trade-offs, and the necessity to share scarce resources. These constraints induce a limited set of possible evolutionary paths, such as co-design of layers, encapsulation, insertion of new layers (so-called middleware). Thus, infrastructural development never proceeds from some clean slate, but rather, from the push and pull of competing stakeholders working to shift its evolution in the most advantageous direction. Even the OSI model, the best example of top-down, a priori modular decomposition of a resource stack, was immediately challenged in the marketplace by the more widely implemented TCP/IP stack. Always only partially responsive to rational control, infrastructural evolution is characterized by drift, opportunity, and improvisation.[3]

An infrastructure-centered perspective thus requires students to engage with computing in a manner that fully integrates network economics, standardization, human-computer interaction, modularity, the material resources of computation, regulation, and systems design. In such a perspective, the fundamental principles at the heart of computational thinking are inseparable from the messy work required for their material realization as infrastructure. There are few pedagogical resources to help promote such a view (see Messerschmitt[6] for an exception). By and large, computer science textbooks remain primarily interested in abstraction as the fundamental practice of the field. Yet, as a breathtaking proportion of social relations becomes mediated through information technologies, it becomes necessary to think of computing as equally concerned with infrastructure building, with all the real-world complexity and engagement with the social world this entails. C

References

1. Asanovic, K. et al. A view of the parallel computing landscape. *Commun. ACM 52*, 10 (Oct. 2009), 56–67.
2. Blanchette, J-F. A material history of bits. *Journal of the American Society for Information Science and Technology 62*, 6 (June 2011), 1042–1057.
3. Ciborra, C. *From Control to Drift: The Dynamics of Corporate Information Infastructures.* Oxford University Press, 2000.
4. Cook, G. and Van Horn, J. How dirty is your data? Technical report, Greenpeace International, Amsterdam, 2011.
5. Lin, H.S., Editor. *Report of a Workshop on the Scope and Nature of Computational Thinking.* National Academies Press, Washington, D.C., 2010.
6. Messerschmitt, D.G. *Networked Applications: A Guide to the New Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.
7. Pappano, L. The master's as the new bachelor's. *The New York Times* (July 22, 2011), page ED16.
8. Van Schewick, B. *Internet Architecture and Innovation.* The MIT Press, 2010.
9. Wing, J.M. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.

Jean-François Blanchette (blanchette@ucla.edu) is an associate professor in the Department of Information Studies at the University of California, Los Angeles.

DOI:10.1145/2347736.2347749

Ivan Sutherland

Viewpoint
The Tyranny of the Clock

Promoting a clock-free paradigm that fits everything learned about programming since Turing.

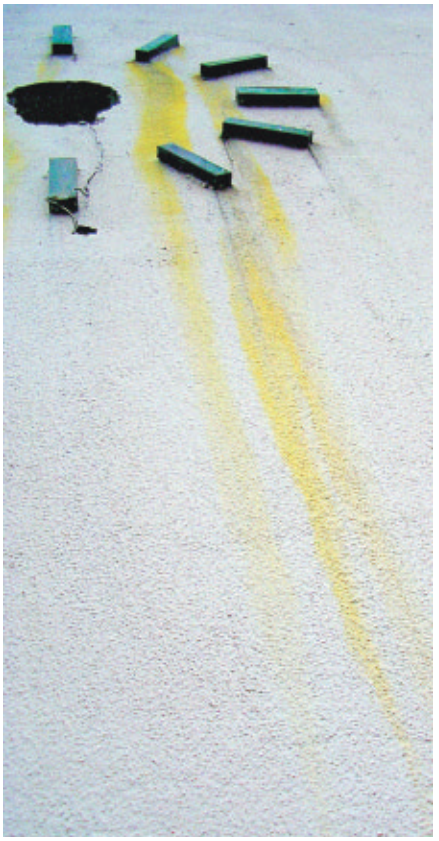
In Alan Turing's day logic was slow and costly but, relative to logic, wires were fast and almost free. Since then the costs of logic and wires have reversed: modern logic is fast and almost free but, relative to logic, wires are now slow and costly. They are costly in three ways. 1) The wires in a modern circuit chip cost most of its area; the transistors in a chip hide underneath a thick bed of tiny wires. 2) The wires in a modern circuit chip cost most of the delay. 3) Worst of all, moving electric charge onto and off of wires wastes most of the energy. The cost of logic and memory dominated Turing's thinking, but today, communication rather than logic should dominate our thinking.

Does communication dominate your thinking?

My question applies equally to hardware, software, and theory.

Today's digital design paradigm, the "clocked" design paradigm, depends on a rhythmic clock signal. The clock signal breaks time into discrete time steps. The designer knows exactly his intent for all the actions of each time step and can check that all the necessary precursors for the actions of each time step happen in earlier steps. Discrete time steps simplify the design task.

Before the telegraph, there was no easy way to synchronize time over distance. Fortunately, there was little need outside navigation to know what time it is somewhere else. "Simultaneous" did not need to apply between Chicago and New York; each city could be its own time zone. The railroad



changed that: passengers wanted to know at what local time their train would arrive, and dispatchers wanted to avoid collisions. Fortunately, the telegraph could provide a notion of "simultaneous" from New York to Chicago so that schedules could be kept. Like a railroad the clocked design paradigm makes designers *want* a concept of "simultaneous" so that clock periods can begin and end everywhere simultaneously.

As transistors and wires get smaller, the area over which one can deliver a clock signal "simultaneously" becomes smaller and thus the number of "clock zones" must increase. The clock beat of each zone differs from the beat of its neighbors in phase and often in frequency as well. A large chip may have hundreds or even thousands of separate clock zones. The clocked design paradigm helps *within* each zone, but only within the zone.

Between clock zones the clocked design paradigm retards data flow. The clocked design paradigm insists on synchronizing all incoming data to the frequency and phase of the clock in the destination zone. Synchronizing a data signal to the destination clock requires special precaution against errors. A reliable boundary crossing requires a delay of two or three clock periods. It is as if each clock zone posted customs inspectors at its borders. The clocked design paradigm exacerbates communication delay.

Ever since my 1988 Turing lecture, I have been exploring an alternative "clock-free" design paradigm. I seek change in the design paradigm to cast off the tyranny of the clock. Instead of making all logic "march to an external drum beat," let us allow each logic element to proceed at its own pace. Because each element acts only when and if necessary, such a paradigm shift will lead to designs that save energy. The clock-free paradigm will also make computers go faster because doing away with border-crossing delays speeds communication. I see a parallel to the economic efficiency Europe gains from free communication across national borders.

PHOTOGRAPH BY ERIK VELLDAL

Casting off the tyranny of the clock offers freedom to optimize the separate parts of a design. For example, Rajit Manohar and his students at Cornell report a clock-free IEEE-compliant, double precision, floating-point adder with the same throughput as an equivalent clocked design. The Cornell clock-free design uses less than half, about 40%, as much energy per addition as its clocked counterpart. The Cornell design gains simplicity and thus reduces energy by doing easy cases fast and allowing the rare hard cases to take longer. A recent paper from my group in the Asynchronous Research Center at Portland State University reports on faster division by allowing steps that merely shift to go faster than steps that must subtract.

Casting off the tyranny of the clock offers modularity as well as local optimization. Sam Fuller, then chief engineer at Digital Equipment Corporation, once told me that his process people could provide faster chips every six months. He complained that his product could not similarly improve every six months because it took 18 months to redesign an entire computer for the new clock speed. The tyranny of the clock made his design insufficiently modular to permit incremental improvement. *He chose* to march his entire machine to a single drumbeat rather than allowing each part to work at its own best speed.

Like all tyranny, the tyranny of the clock stems from the range over which *we choose* to subject *ourselves* to the tyrant's authority.

The clock-free paradigm I promote relates to the clocked design paradigm as a "free economy" relates to a "controlled economy." We can regain the efficiency of local decision making by revolting against the pervasive beat of an external clock.

Clock-free commercial products are in use today. Handshake Solutions, a computer-aided design company from the Netherlands, was proud of having 700 million of their clock-free chips in use in smart cards, passports, cellphones, and other portable devices. Fulcrum Microsystems, a Caltech spinoff recently purchased by Intel, sells a self-timed communication switch with outstanding performance.

Casting off the tyranny of the clock offers freedom to optimize the separate parts of a design.

The paradigm shift I seek faces three formidable obstacles: technical, social and courage. First, technical: Make no mistake; designing a clock-free system can face the same hard problems of parallelism that give software people nightmares. But a few pioneers have shown that clock-free design is possible and sometimes even easy. The pioneers have uncovered benefits like using less than half, 40%, of the energy per operation as reported by Cornell. Second, social: *All* of today's commercial design tools assume clocked design. All engineering schools teach clocked design. Will we ever train enough young people in the clock-free paradigm for it to self-perpetuate? Third, courage: Management knows the costs, difficulties, and results of the "tried and true" clocked design paradigm. Management chooses "to bear those ills we have rather than fly to others that we know not of."

The clock-free design paradigm must eventually prevail. It fits physics. Each increase in the relative cost of communication over logic brings us closer to the fundamental physical truth that "simultaneous" lacks meaning. The clock-free paradigm fits everything we have learned since Turing about programming. Software avoids tyrannous global time constraints. Without freedom from global time constraints, software libraries would be impossible. "Modularity" and "data hiding" are basic principles of quality software because they allow reuse and local optimization. Software is self-timed: Each subroutine runs at its own pace; its users wait for it to finish. Imagine what software would be like if subroutines could start and end only at preset time intervals. "My subroutines all start at 3.68 millisecond intervals; how often do yours start?"

Software development proceeds from correctness to performance. After software works, we tune its heavily used parts to achieve the desired performance. Performance almost always depends on only a small part of the whole. Compare this to the situation in a clocked hardware design where each and every signal must arrive "on time," even if it is rarely used. The tyranny of the clock wastes both engineering cost at design time and energy at runtime. What a needless waste!

Only a small handful of intrepid entrepreneurs and academic researchers have yet dared to explore the clock-free paradigm I promote. I predict that sometime soon, some courageous management will tire of wasting money on the tyranny of the clock and adopt the clock-free design paradigm. Such courage will reap giant rewards. I shall be disappointed but not at all surprised if that courageous management speaks an Asian language rather than English, the native tongue I share with Alan Turing.

This Viewpoint is derived from Ivan Sutherland's presentation at the ACM A.M. Turing Centenary Celebration Computer Architecture panel discussion this past June; see http://amturing.acm.org/acm_tcc_webcasts.cfm. [Also see the profile of Ivan Sutherland in the News section of this issue on page 10. —*Ed.*]

Further Reading

Sutherland, I.E.
Micropipelines. *Commun. ACM 32*, 6 (June 1989).

Sutherland, I.E. and Ebergen, J.
Computers without clocks. *Scientific American 287*, 2 (Aug. 2002), 62–69.

Riaz Sheikh and R. Manohar.
An operand-optimized asynchronous IEEE 754 double-precision floating-point adder. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)* 2010, 151–162.

N. Jamadagni and J. Ebergen.
An asynchronous divider implementation. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2012.

Ivan Sutherland (ivans@cecs.pdx.edu) is a visiting scientist in the Asynchronous Research Center at Portland State University in Oregon and the 1988 recipient of the ACM A.M. Turing Award for his pioneering contributions to computer graphics.

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

Professional membership dues include $40 toward a subscription to *Communications of the ACM*. Student membership dues include $15 toward a subscription to *XRDS*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

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

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

Satisfaction Guaranteed!

payment:

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

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:10.1145/2347736.2347750

Article development led by acmqueue
queue.acm.org

An introduction to PTP and its significance to NTP practitioners.

BY RICK RATZEL AND RODNEY GREENSTREET

Toward Higher Precision

IT IS DIFFICULT to overstate the importance of synchronized time to modern computer systems. Our lives today depend on the financial transactions, telecommunications, power generation and delivery, high-speed manufacturing, and discoveries in "big physics," among many other things, that are driven by fast, powerful computing devices coordinated in time with each other.

Since the first complete specification of Network Time Protocol (NTP) version 1 and its accompanying algorithms appeared in RFC 1059 in 1988, NTP has played a large role in time synchronization by keeping the clocks of networked computer systems synchronized to within milliseconds of each other. NTP has been deployed to a vast number of systems over the years, yet it hardly bears the burden of clock synchronization alone. When users want to coordinate events in time between multiple systems, they typically have many options for accomplishing this, all with different trade-offs.

One of the emerging alternatives to NTP is PTP (Precision Time Protocol). PTP is defined by IEEE standard 1588, published in November 2002 and based on early prototypes built at Agilent Technologies between 1990 and 1998. A revision of PTP with additional features and improved performance was published in 2008; it is known as PTP version 2 or IEEE 1588-2008 (all references to PTP in this article refer to this later version).

PTP and NTP are similar in that both are packet-based and send timestamps over a network from a time reference device to one or more other devices. Additionally, both synchronize device clocks based on time offsets and network delays, and both support heterogeneous devices with varying clock time precision, resolution, and stability over varying amounts of physical separation. Each protocol has its unique strengths, and choosing one over the other often warrants an evaluation of a system's environment, capabilities, and goals.

PTP is often chosen when the synchronization performance requirements of systems exceed the millisecond threshold of a typical NTP-based solution. When used with PTP-capable network hardware that has the ability to timestamp PTP packets precisely (something that is quickly becoming commonplace in industrial network interfaces), devices using PTP on a LAN (local-area network) can synchronize their clocks to within tens of nanoseconds of each other. Without hardware timestamping, referred to as a *software-only* configuration, PTP implementations can still achieve submillisecond precision.

NTP remains a popular synchronization technology, even as more PTP implementations have been made available to system designers on more platforms—both commercially and as freely available open source implementations. If PTP is available to a system designer and displays superior synchronization performance, why would NTP even be considered?


STOP WATCH
SBAR
HR min SEC 1/100S
SPLIT / / LAP
SPLIT CHRONO
START STOP

Should PTP replace NTP altogether? If not, what do system designers need to know in order to choose the appropriate protocol? Perhaps the answers to these questions can be discerned from the circumstances that led up to the definition of PTP and a more in-depth look at this newer standard.

Measurement and Control Devices: Time for Something New?

Measurement and control devices have always been a vanguard for high-precision event synchronization. To achieve the degree of synchronization that devices of this nature require, signals sent over specialized cables can be used to synchronize events between devices. These cables, which are used exclusively for event synchronization, are often matched in length to ensure propagation delay is consistent. Synchronization using this dedicated cabling results in extremely high precision, where events can be coordinated to within picoseconds of each other across multiple devices in proximity. This type of synchronization is commonly referred to as *signal-based*.

While nearly unbeatable for applications requiring the most accurate synchronization possible, signal-based synchronization can be highly impractical and sometimes not possible. The dedicated cabling needed to synchronize separate devices can be cost prohibitive, and signal-based synchronization requires specialized hardware and software to generate and receive the signals on the cable. The signaling protocol can be proprietary, resulting in potential vendor tie-in, risk of discontinuation, or other legal or technical restrictions. The cables themselves are often subject to varying propagation delay over time and temperature, and as more devices are added to a system, the complexity of cabling multiple devices increases the maintenance burden and effort in troubleshooting failures. Signal-based synchronization also requires the devices be relatively close to each other and does not scale over long distances when compared with other synchronization mechanisms.

Meanwhile, with Ethernet becoming more and more ubiquitous in the laboratories and on factory floors where measurement and control devices are deployed, a need arose for these devices to be able to use a LAN or even a wide-area network (WAN) for control and data communication. NTP was even leveraged to set system time for these devices, but the need for dedicated event synchronization cables still existed. Despite the presence of all the basic ingredients for event synchronization using coordinated, distributed timekeepers (sometimes referred to as *time-based* synchronization), an acceptable technology that could use this infrastructure to replace signal-based synchronization had yet to be created.

Measurement and control devices have always been a vanguard for high-precision event synchronization.

Because signal-based implementations impose the constraints previously mentioned, time-based solutions using Ethernet were investigated further as a synchronization solution. At first glance, NTP seems like a good candidate for a low-cost time-based synchronization solution—and it is for many applications. Signal-based synchronization, however, provides an extremely high level of precision, and NTP version 3 (until recently the officially supported NTP release) provides only millisecond precision, which is not even close to being sufficient for applications using signal-based solutions. PTP was designed to meet the needs of the measurement and control industry and is capable of near-nanosecond precision while taking advantage of infrastructure that is similar to what NTP uses. A closer look at PTP reveals why it is successful for measurement and control applications—and, as it developed, many other applications as well.

Introducing PTP

PTP's primary design goals have been listed in numerous presentations and documents, including the IEEE 1588 standard:[1]

- To provide sub-microsecond synchronization of real-time clocks in components of a networked distributed measurement and control system;
- To perform best with relatively localized systems typical of industrial automation and test and measurement environments;
- To be applicable to LANs supporting multicast communications (including but not limited to Ethernet);
- To provide a simple, administration-free installation;

▸ To support heterogeneous systems of clocks with varying precision, resolution, and stability; and

▸ To impose minimal resource requirements on networks and host components.

PTP meets these goals using a robust synchronization methodology, an algorithm that automatically and continuously maintains the proper device hierarchy for maximum accuracy, and specialized hardware (required only for optimal performance).

Synchronization methodology. At the heart of the PTP standard is the synchronization methodology. While similar to other time-based Ethernet synchronization protocols in concept, PTP's synchronization methodology is unique and somewhat dependent on the particular hardware and application (power industry, telecommunications, among others) of a PTP deployment.

PTP defines a master-slave hierarchy based on criteria that describe a device's timekeeping capability and the traceability of its time source. The master serves as the time reference for one or more slave devices. The process of selecting the master from the list of participating devices is defined in PTP's Best Master Clock (BMC) algorithm, which is applied by each device at specific intervals. Devices (often referred to as *ordinary clocks*) may consider themselves masters either because they have not yet evaluated themselves against other clocks or have determined, according to BMC, that they now have better timekeeping ability than the current master. They will transmit *Announce* messages using UDP (User Datagram Protocol) multicast (by default) at configurable intervals. The other devices will process these Announce messages according to BMC and select the new master. If a master receives an Announce message from another potential master (known as a *foreign master*) and the device's BMC indicates this foreign master should be master, the current master will transition to the slave state.

In addition to Announce messages, a master clock periodically transmits a *Sync* message using UDP multicast (by default), which is received by a slave clock. Each slave uses a Sync message to calculate the difference between its clock and the master clock. The message contains a timestamp from the master representing when it was issued (t1 in Figure 1); when the slave receives the Sync message, it records its time of receipt (t2). The time in the Sync message does not represent the precise time the message left the device, since it was not known until after it was sent. The master then sends a *Follow-up* message that includes the actual time the Sync message left the master as determined by specialized hardware (if equipped) or the network driver. The slave receives the Follow-up and uses that value as the actual t1.

At this point, the slave has two time values (t1 and t2) and can compute the offset between its timekeeper and the master's. Unfortunately, the offset derived from t1 and t2 includes some unknown amount of additional propagation delay incurred by the network. To determine this delay and compute the actual offset between timekeepers, the slave issues a *Delay Request* message to the master and notes the time it was sent (t3). The master notes when, according to its timekeeper, it receives the Delay Request (t4) and issues a *Delay Response* message back to the slave containing t4. When the slave receives the Delay Response it will have four timestamps—t1, t2, t3, and t4—and can compute the offset between its timekeeper and the master's timekeeper while properly taking into account the network delay.

Hardware timestamping and "software-only" configurations. To find the actual time the Sync message was sent from the master in order to insert it in the Follow-up, the master must know exactly when its network hardware was able to send the Sync message. This hardware is most likely the network interface's physical transceiver (PHY) or other hardware that recognizes PTP packets and notes the precise time they were sent or received. The difference between when the master's PTP software initiated the sending of this message (the estimated value of t1 included in the Sync message) and the time the PHY was able to send the signals on the physical media will not only vary, but will also be quite significant with respect to the sub-microsecond precision PTP is capable of. Therefore, the time the Sync message spends in the master's network stack needs to be accounted for in order to achieve maximum accuracy (see Figure 2).

PTP defines another, slightly different synchronization mechanism that takes advantage of additional hardware support, if available. The Sync and Follow-up messages used to cal-

Figure 1. Basic synchronization message exchange.

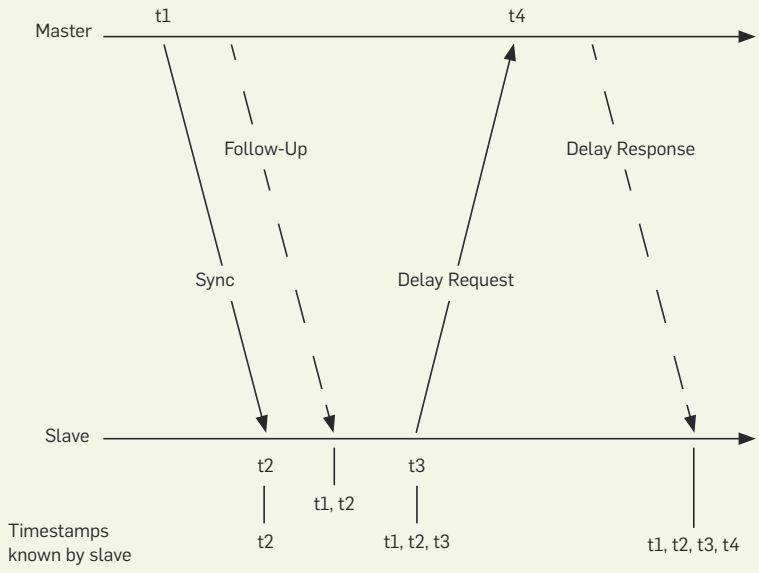


Figure 2. A network stack supporting hardware timestamping.

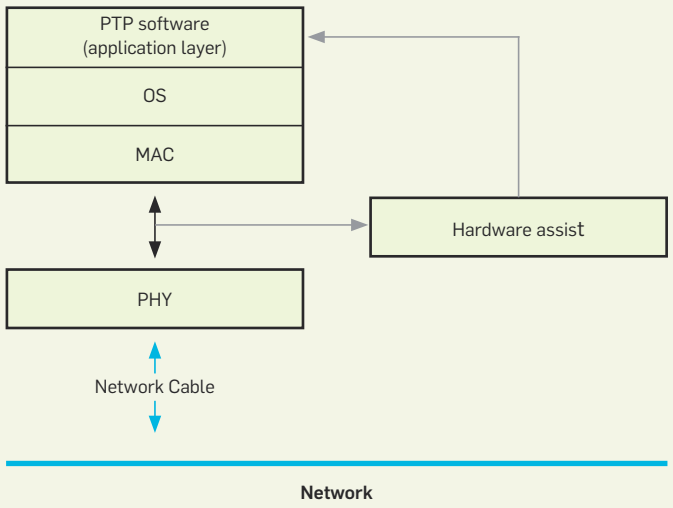
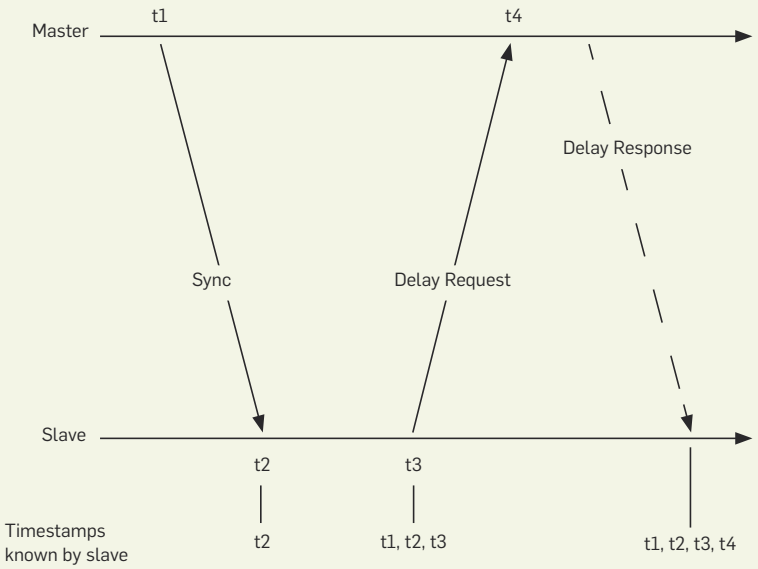


Figure 3. One-step operation.



culate the offset between the master and slave described earlier are used by a *two-step clock*. A *one-step clock* uses specialized network hardware not only to timestamp when a PTP Sync message leaves the device, but also to modify the outgoing Sync message's t1 value with the actual departure time (see Figure 3).

This value is normally sent in the Follow-up message, but because the hardware makes it available in the Sync message, the Follow-up is redundant and therefore not needed. A slave device must also understand that its master is operating as a one-step clock. It can determine this by reading a bit field in the PTP message headers sent by the master. A one-step clock helps minimize network traffic while maintaining high synchronization performance and is therefore a requirement for PTP applications in certain industries.

PTP-aware network interface hardware can also simply timestamp PTP messages and correlate them with message IDs for later retrieval by PTP software. This capability allows a PTP master operating in two-step mode to send the precise t1 value in a Follow-up message to slaves. While this degree of hardware support does not make (accurate) one-step operation possible, it does make for superior two-step performance when compared with a software-only implementation. A PTP device is considered to be operating as a software-only clock if it has no hardware support. A software-only clock is limited to two-step operation and typically sends t1 values in the Follow-up messages that are retrieved from software components as low as possible in the software stack, usually from the driver level. Although software-only clocks are obviously not as accurate as those with hardware assist, they are still capable of achieving sub-millisecond precision.

The Best Master Clock algorithm. The BMC algorithm gives PTP devices the ability to maintain the desired synchronization hierarchy under changing network conditions. Describing the BMC algorithm itself is beyond the scope of this article, but needless to say, the BMC is a key part of the "simple, administration-free" aspects of PTP's objectives. Any PTP device acting as either a master or slave continuously runs the BMC and uses it in determining if a new master needs to be selected, or if the device needs to transition out of the master state.

This type of state change can occur as the result of a number of conditions, all of which are reflected in a master's (or potential master's) Announce message. Masters and devices that could potentially be masters issue Announce messages at a configurable rate as part of the protocol. The Announce message contains all the pertinent information the BMC needs to determine if the current master should remain master or yield to a new master as a slave, or in the case of a slave, should start listening to a new master or become a master itself. Some of the attributes of an Announce message are the device's time source (GPS, atomic clock, or free-

running oscillator); the "priority" as determined by the local PTP administrators (which is used as an override mechanism and not required to be set for proper operation); the device's clock ID (which typically includes the device's MAC address); and other attributes used by the BMC.

Having every PTP device run the BMC and process Announce messages means administrators can simply power on a system and have a network of time-synchronized devices automatically configured for optimal performance, regardless of spontaneous network topology changes.

Boundary clocks. Because switches and routers effectively segment a PTP network, PTP introduced boundary clocks as a means of distributing a master clock to different parts of the network. The PTP standard describes a boundary clock as containing a single timekeeper disciplined by PTP but having multiple PTP ports in a domain. A port may serve as either the source of time (a PTP master) to devices attached to it or one that synchronizes the timekeeper (a PTP slave) to some other clock connected to that port. A boundary clock can be implemented to replace a traditional network switch or router in larger networks that are normally segmented by such devices. Because boundary clocks differ in operation from the PTP clocks previously described in this article, PTP differentiates the two by referring to them as either *ordinary clocks* or *boundary clocks*.

Each port of a boundary clock can be thought of as a separate ordinary clock instance that shares a single timekeeper with the boundary clock's other ordinary clock instances. Only one port on the device can be in the slave state, which eliminates contentious use of the device's timekeeper (two ports trying to adjust the time, for example). All other ports are considered masters to the devices on their respective segments.

The existence of boundary clocks requires PTP to use the term *grandmaster* to describe the master to the entire PTP network, since the slaves on a boundary-clock port consider the boundary clock to be their master. Each master port is responsible for handling the same duties as an ordinary clock master, which effectively hides all of the slaves from the boundary clock's master. Likewise, a slave ordinary clock (or another boundary clock with the connected port in the slave state) is hidden from the PTP hierarchy "above" the boundary clock. A boundary clock does not pass the PTP synchronization messages from its slaves "up" to its master. Without this behavior, a grandmaster would be responsible for processing Delay Request messages and issuing Delay Response messages from and to, respectively, every slave device on the entire PTP network. In most cases it would not be able to run the protocol stack effectively.

A boundary clock, however, may still allow any eligible slave clock in the entire PTP network to be grandmaster. For example, if the grandmaster goes offline, the next most eligible slave device can announce itself as master (once its BMC algorithm has determined it's appropriate to do so), and the boundary clock will transition the port connected to that slave to the slave state. That boundary clock will then have a port that was once in the slave state now in the master state to other ordinary and boundary clocks. Those clocks will then evaluate that new master with the BMC algorithm and transition appropriately, repeating this process for the rest of the hierarchy. Depending on the network topology, this situation may not be ideal as the number of hops in between this new master and a slave will have increased by one (the boundary clock connected

Figure 4. Device hierarchy using boundary clocks.

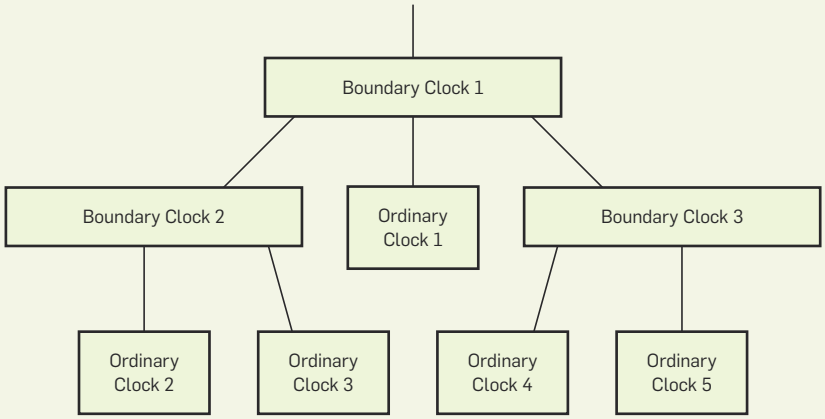
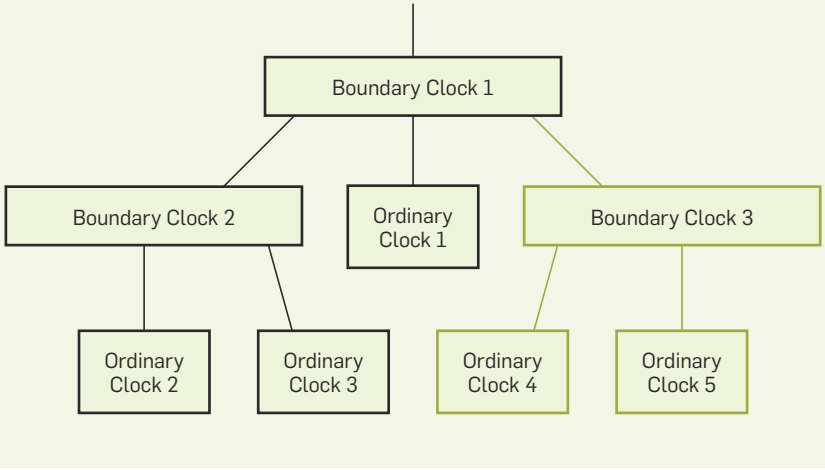


Figure 5. Boundary clocks used to join different network technologies.



to the new master), thus increasing any synchronization error accumulation.

The use of boundary clocks and the resulting hierarchy of PTP devices must be considered in order to maximize systemwide synchronization precision (see Figure 4).

Boundary clocks can also be used for bridging networks that use different networking protocols (illustrated in Figure 5), since there is no requirement that PTP implementations use the same underlying communication media or technology. For example, a system can have some devices using Ethernet and others using DeviceNet, all synchronized to the same grandmaster through the use of capable boundary clocks. In this scenario, a boundary clock would have a DeviceNet-capable port connected to the DeviceNet devices, and another connected to Ethernet devices. The specific communication media is abstracted from the PTP clocks, allowing both types of devices to synchronize to the same PTP grandmaster regardless of that grandmaster's media. In addition to different networking protocols, boundary clocks can also join PTP systems that use different delay calculation mechanisms, which are described later.

Transparent switches. Not all applications allow their PTP devices to be deployed in a manner that lends itself well to a balanced, treelike hierarchy. Systems are sometimes deployed in long linear or ring topologies, which can cause significant synchronization error accumulation when boundary clocks are used to join these segments. Because of this, PTP defines a device

Figure 6. A transparent switch updates the correction field of a PTP message in end-to-end delay mode.

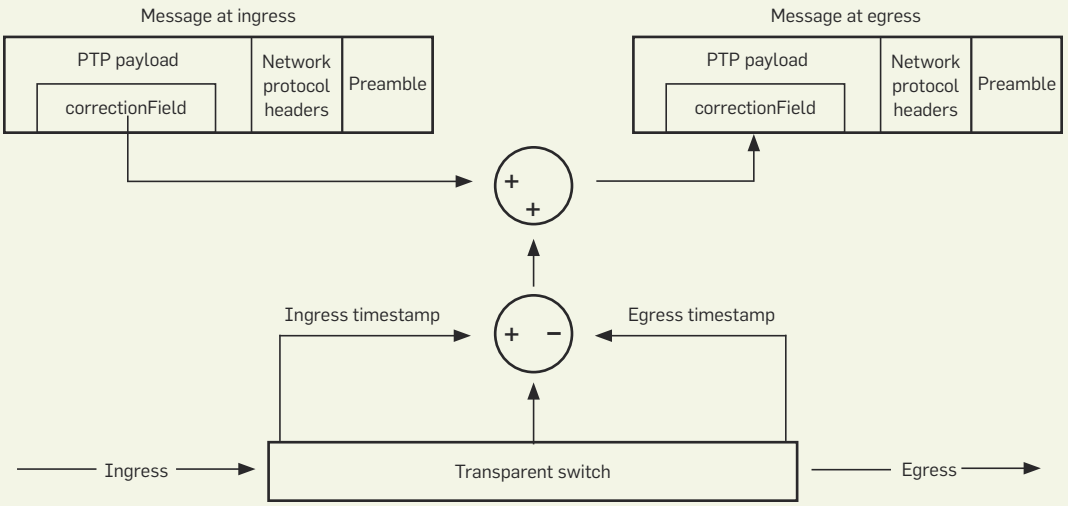
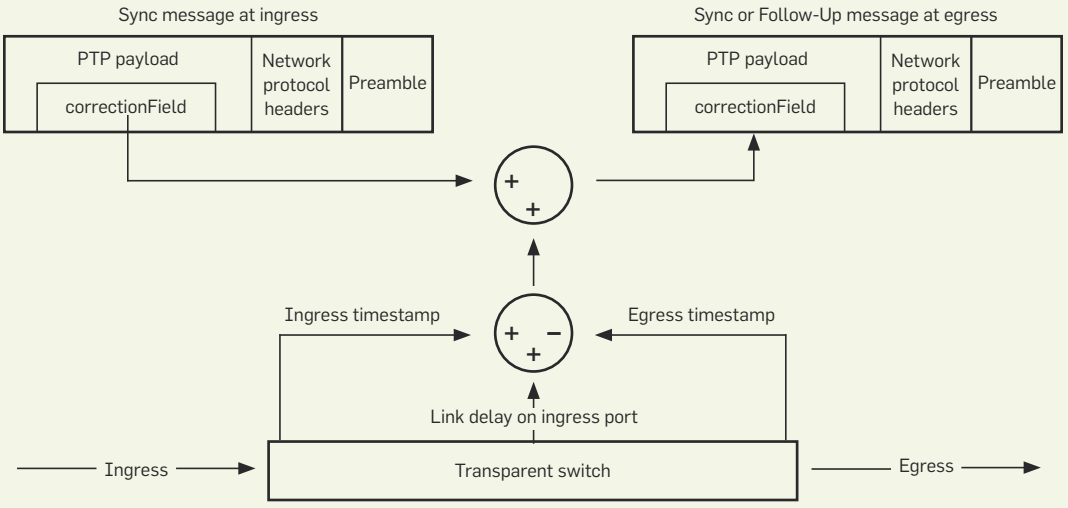


Figure 7. A transparent switch updates the correction field of a PTP message in peer-to-peer mode.



known as a *transparent switch*, which connects groups of PTP devices without segmenting the PTP network.

A transparent switch recognizes PTP messages passing through and notes each message's *residence time*, the time spent in the switch where the message is not yet visible by the intended PTP device. The residence time is added to the PTP message's *correction field* just before being transmitted from the switch to the next device (see Figure 6). PTP clocks can then examine the received message's correction field and apply it to their calculations. Even though the message was temporarily held up in the transparent switch—a nondeterministic behavior that normally introduces significant synchronization error—the correction field allows that time to be removed, as if the switch were never there (hence, the name *transparent* switch).

Unlike boundary clocks, transparent switches expose their slave devices to the PTP master. The transparent switch is typically interested in only a relative time (the time a message spends in the switch) and therefore does not need to have a timekeeper synchronized to the master's time. The oscillators that "tick" in both the master and the switch, however, must tick at the same rate. Keeping this rate the same is known as syntonization. PTP specifies that transparent switches must be *syntonized*—not necessarily *synchronized*—to the master.

The peer delay mechanism. The synchronization model described earlier, where the slave issues a Delay Request message and the master responds with a Delay Response, is known as the *delay request-response* mechanism, or sometimes as *end-to-end* mode. PTP offers an alternative to this known as the *peer-delay* mechanism, or *peer-to-peer* mode, which can provide superior performance in certain situations. Because end-to-end mode and peer-to-peer mode cannot be used together, system designers have to evaluate which delay mechanism will provide the best results and design their systems accordingly.

In peer-to-peer mode, a device issues a *Peer Delay Request* message to its immediate neighbor, which may or may not be the device's master. The receiving device responds with a *Peer Delay Response* message (and optionally a *Peer Delay Response Follow-up* after that if the device is operating in a two-step mode). This allows the requesting device to calculate the propagation delay for the individual segment.

NTP remains a popular synchronization technology, even as more PTP implementations have been made available to system designers on more platforms—both commercially and as freely available open source implementations.

By knowing the exact propagation delay for each segment of a network path, peer-to-peer mode allows PTP to apply delay compensations between master and slaves that are more accurate than end-to-end mode allows when the intermediate switches choose different paths. Since peer-to-peer mode specifies that transparent switches adjust the correction field with not only the residence time of Sync and Follow-up messages (just as a transparent switch operating in end-to-end mode does), it also adds the delay previously calculated for the link the message came in on (see Figure 7).

This behavior means the master need not process Delay Request messages from each of its slaves; instead it concerns itself only with Peer Delay Requests and Responses for its immediate peer (transparent switch or PTP clock in slave state). Because of this, transparent switches in peer-to-peer mode do not pass Delay Request or Delay Response messages. Unlike end-to-end mode, peer-to-peer mode can be even more attractive to system designers concerned with network traffic, since a master device need not receive and respond to each slave's Delay Request messages, and concerns itself only with its immediate peer.

Profiles. PTP profiles allow organizations to specify selections of attribute values and optional features of PTP that, when using the same transport protocol, work together and achieve a performance that meets the requirements of particular applications.[3] Profiles make PTP better suited for particular applications while adhering to the more general PTP standard. Profiles can specify several aspects of the standard. There are two "default" profiles: Delay Request-Response (often referred to as end-to-end mode) and Peer Delay (often referred to as peer-to-peer mode). Implementers must support at least one of these defaults. Profiles themselves are standardized and defined by a recognized standards organization that has jurisdiction over a particular industry (such as the IEC, IEEE, IETF, ANSI, or ITU). These organizations,

as stated in the PTP standard, should consult the Precise Networked Clock Synchronization Working Group of the IM/ST (Instrumentation and Measurements/Sensor Technology) Committee for technical review.

PTP profiles not only change several aspects of the PTP standard, but also extend it. A profile may define its own BMC algorithm; configuration and monitoring ("management") mechanism; path-delay mechanism (end-to-end or peer-to-peer); use of multicast or unicast; transport mechanism; node types; and any options that are required, permitted, or prohibited. Profiles may also define completely new transport mechanisms and data types. The flexibility that profiles have in morphing PTP to the needs of almost any particular application has proven useful to telecommunication and energy industries, among others.

Unicast. PTP was designed assuming a multicast communication mode, but support for unicast operation was eventually added as an optional feature. The PTP standard does not describe a unicast PTP implementation in detail, but instead describes several optional unicast features that can be used for an implementation "as long as the behavior of the protocol is preserved."[2] Some implementations may require that slave clocks use a configuration that specifies a list of known master clocks by protocol address (for example, a list of IP addresses when used over Ethernet) to discover the potential masters.

This unicast discovery mechanism is optional, meaning a unicast implementation could choose to use multicast for discovery of master clocks and unicast for all other messaging. Furthermore, this discovery mechanism may also require some amount of configuration to define the list of masters, since that is most likely specific to a given system and stretches the interpretation of the PTP objective to "provide a simple, administration-free installation."[1] Another optional implementation detail defined by PTP is the use of the unicast-negotiation mechanism, which involves sending specific signaling messages to master devices indicating that they respond with a unicast Announce, Sync, Delay Response, or Peer Delay Response to the signaling slave device. This flexibility in allowing unicast operation and providing several optional features to implement it allows profiles to define the specific unicast implementation details best suited for their applications.

PTP profiles not only change several aspects of the PTP standard, but also extend it.

Timescale. The timescale for a PTP network is defined by the grandmaster and can be one of two types: the default PTP timescale or an ARB (arbitrary) timescale.[5] With the ARB timescale, the epoch is set by some predetermined procedure and can be set again using that procedure during normal operation. The PTP timescale uses the PTP epoch, and its unit of time is the SI second. The PTP epoch is 1 January 1970 00:00:00 TAI (International Atomic Time), which is 31 December 1969 23:59:51.999918 UTC (Coordinated Universal Time).

Using the Right Tool for the Job: NTP or PTP?

The requirements of devices for the measurement and control industry are similar to those of many other industries—and many innovative outcomes have resulted from applying technology in ways that its designers had not originally considered—but the intended applications for any technology still should be considered before adopting it, regardless of the similarities it may have to the incumbent technology.

Objectives. As described earlier and stated in the standard, PTP was designed to be used over a LAN, or more specifically, "spatially localized systems with options for larger systems."[4] This is one of the more significant differentiators between PTP and NTP. The use of a LAN allows other PTP objectives to be fulfilled using techniques such as multicast for discovery and automatic selection of PTP masters, network equipment such as boundary clocks and transparent switches, and very high message exchange rates that may not be feasible over a WAN. A LAN also gives PTP some liberties that NTP does not usually have, such as assuming—with a reasonable degree of confidence—that unrelated network traffic and security risks are both low, given that LAN usage is usually confined and controlled.

In contrast, NTP is typically used over the Internet and is therefore subject to a large amount of nondeterministic delays from intermediate network elements (such as routers) and exposed to a far greater number of security

threats (denial-of-service and man-in-the-middle attacks being some of the more obvious). It must accept these penalties or, in the case of security, account for these challenges.

Security in particular is worth highlighting, since PTP includes only an experimental extension to the protocol to address security concerns, but NTP defines the use of access-control lists and a variant of public-key cryptography called Autokey.[6] Also note that NTP can use a multicast mode to discover servers automatically when used on a LAN, and PTP can operate in a unicast mode to be used over a WAN. Neither of these uses, however, is the most common and may impose additional configuration costs.

Another PTP objective is administration-free operation, where the devices that make up a system can be deployed with little or no configuration yet still achieve optimum time synchronization for the given environment. Devices can be added, removed, or reconfigured while the system is in use, and the PTP devices that make up the system will automatically negotiate a new hierarchy in order to maintain optimum synchronization performance. PTP's BMC algorithm is responsible for this behavior. NTP's optimization algorithm does not permit the same degree of autonomy in allowing any device to become the equivalent of a PTP grandmaster if necessary, despite the inclusion of a dynamic discovery scheme in the latest NTP specification.[7] Instead, NTP defines a series of mitigation algorithms to be used in finding the optimal network path,[6] provided an NTP client has been configured to select among more than one server.

Synchronization methodologies. While the synchronization methodologies employed by PTP and NTP are similar in that they both ultimately compute clock offset and message delays, the protocols differ greatly in various mechanisms that must be considered when selecting the appropriate technology. For example, PTP relies on boundary clocks and transparent switches to achieve maximum performance in certain environments. Not including these devices or incorrectly using them could significantly reduce PTP performance.

More relevant to implementers of these specifications is the fact that PTP does not define a servo algorithm for applying the information PTP gives a device to the device's oscillator. Instead, the servo definition is implementation-specific, and there are no guarantees that different PTP software stacks will exhibit the same synchronization behavior on the same device. In stark contrast, NTP "defines a highly evolved, adaptive-parameter, hybrid phase-frequency lock loop" used to adjust the device's timekeeper from data provided by NTP.[6]

The timescales of both technologies also differ. NTP uses UTC time, while PTP (typically) uses TAI and a UTC offset. This difference could be significant to system designers who assume a specific timescale. Concerns about properly handling leap seconds might factor in as well; unlike PTP, a leap second will cause the entire NTP timescale to shift by one second.

Performance expectations. A typical PTP-synchronized system can expect sub-microsecond synchronization precision, where "typical" includes hardware assist and a LAN. A typical NTP-synchronized system, meaning no specialized hardware and devices connected over a WAN, will achieve millisecond synchronization precision. When PTP is appropriately configured for use over a WAN, however, there may be little to no performance advantage over NTP.[8]

Conclusion

PTP has the capability to synchronize devices to within nanoseconds of each other over a common networking infrastructure, allowing system designers to replace synchronization solutions that are more expensive, limited, or both. NTP has similar use cases, but usually falls short for applications that require the level of performance typical of measurement and control systems. PTP's BMC algorithm allows it to adapt to changing conditions to ensure devices always have the highest-quality time reference. PTP boundary clocks and transparent switches ensure high synchronization performance even in a non-ideal network topology. In contrast, NTP requires that all devices be configured to reference a predetermined set of time servers prior to use, and its performance suffers when messages have to traverse network elements such as switches.

PTP's intended environment is different from NTP's, however, and depending on the application, NTP may be a better choice. For example, NTP's more established security mechanism and publicly available pool of time servers[9] make it better suited for synchronizing time over the Internet when performance requirements permit.

PTP has filled a niche that NTP has not been able to, but it has not replaced it. Instead, PTP offers system designers a new synchronization tool to put in their toolboxes. C

Related articles on queue.acm.org

Principles of Robust Timing over the Internet
Julien Ridoux, Darryl Veitch
http://queue.acm.org/detail.cfm?id=1773943

Modern Performance Monitoring
Mark Purdy
http://queue.acm.org/detail.cfm?id=1117404

The One-Second War (What Time Will You Die?)
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=1967009

References

1. Eidson, J. IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems. Agilent Technologies, 2005; http://www.nist.gov/el/isd/ieee/upload/tutorial-basic.pdf.
2. IEEE Std. 1588-2008 section 6.2. IEEE 1588 precision time protocol demonstration for STM32F107 connectivity line microcontroller (2011); http://www.st.com/internet/com/technical_resources/technical_literature/application_note/DM00030825.pdf.
3. IEEE Std. 1588-2008 section 19.3.
4. IEEE Std. 1588-2008 section 1.2.
5. IEEE Std. 1588-2008 section 7.2.
6. Mills, D. IEEE 1588 Precision Time Protocol (PTP), 2012; http://www.eecis.udel.edu/~mills/ptp.html.
7. Mills, D., Martin, J., Burbank, J. and Kasch, W. Network Time Protocol Version 4: Protocol and Algorithms Specification (2010); http://tools.ietf.org/html/rfc5905.
8. Novick, A.N., Weiss, M.A., Lee, K.B. and Sutton, D.D. Examination of time and frequency control across wide area networks using IEEE-1588v2 Unicast Transmissions (2011); http://www.nist.gov/manuscript-publication-search.cfm?pub_id=908609.
9. NTP Pool Project; http://www.pool.ntp.org.

Rick Ratzel (rick.ratzel@ni.com, rlratzel@ratzel.com) is a senior software engineer at National Instruments in Austin, Texas, where he works in the Timing & Synchronization group as a technical lead and developer. He has developed software at an electronic design automation start-up and a consulting start-up specializing in scientific computing prior to joining NI. He has taught Python classes for engineers.

Rodney Greenstreet (rodney.greenstreet@ni.com) is a technical lead in the Timing & Synchronization group at National Instruments with 20 years of industry experience. He is responsible for implementing the firmware for NI's first USB device and designing the first CompactPCI board, which served as the impetus to the PXI platform. As a technical lead in the Timing & Synchronization group, he is instrumental in architecting sub-nanosecond hardware and software synchronization products that leverage GPS, IEEE 1588, IRIG, and others.

DOI:10.1145/2347736.2347751

Article development led by acmqueue
queue.acm.org

Making the case for resilience testing.

BY JOHN ALLSPAW

Fault Injection in Production

WHEN WE BUILD Web infrastructures at Etsy, we aim to make them resilient. This means designing them carefully so they can sustain their (increasingly critical) operations in the face of failure. Thankfully, there have been a couple of decades and reams of paper spent on researching how fault tolerance and graceful degradation can be brought to computer systems. That helps the cause.

To make sure the resilience built into Etsy systems is sound and that the systems behave as expected, we have to see the failures being tolerated *in production*.

Why production? Why not simulate this in a QA or staging environment? The reason is the existence of any differences in those environments brings uncertainty to the exercise, as well as because the risk of not recovering has no consequences, which can bring unforeseen assumptions into the fault-tolerance design and into recovery. The goal is to reduce uncertainty, not increase it.

Forcing failures to happen, or even designing systems to fail on their own, generally is not easily sold to management. Engineers are not conditioned to embrace their ability to respond to emergency situations; they aim to avoid them altogether. Taking a detailed look at how to respond better to failure is essentially accepting that failure will happen, which you might think is counter to what you want in engineering, or in business.

Take, for example, what you would normally think of as a simple case: the provisioning of a server or cloud instance from zero to production:

1. Bare metal (or cloud-compute instance) is made available.
2. Base operating system is installed via PXE (preboot execution environment), or machine image.
3. Operating-system-level configurations are put into place (via configuration management or machine image).
4. Application-level configurations are put into place (via configuration management, app deployment, or machine image).
5. Application code is put into place and underlying services are started correctly (via configuration management, app deployment, or machine image).
6. Systems integration takes place in the network (load balancers, VLANs, routing, switching, DNS, among others).

This is probably an oversimplification, and each step or layer is likely to represent a multitude of CPU cycles; disk, network and/or memory operations; and various amounts of software mechanisms. All of these come together to bring a node into production.

Operability means that you can have confidence in this node coming into production, possibly joining a cluster, and serving live traffic seamlessly every time it happens. Furthermore, you want and expect to have confidence that if the underlying power, configuration, application, or compute resources (CPU, disk, memory, network, and so on) experience a fault, then you can survive such a fault by some means: allowing the application to degrade gracefully, rebuild itself, take itself out of production, and alert on the specifics of the fault.

Building this confidence typically comes in a number of ways:

PHOTOGRAPH BY MEG/FLICKR

▸ **Hardware burn-in testing.** You can run extreme tests on the various hardware components in a node in order to confirm that none of them would experience faults at the onset of load. This may not be necessary or feasible in a cloud-compute instance.

▸ **Unit testing of components.** Each service can be easily tested in isolation, and configuration can be checksummed to assure expectations.

▸ **Functional testing of integrations.** Each execution path (usually based on an application feature) can be explored with some form of automated procedure to assure expected results.

Traditionally, these sensible measures to gain confidence are made before systems or applications reach production. Once in production, the traditional approach is to rely on monitoring and logging to confirm that everything is working correctly. If it is behaving as expected, then you do not have a problem. If it is not, and it requires human intervention (troubleshooting, triage, resolution, and so on), then you need to react to the incident and get things working again as fast as possible.

This implies that once a system is in production, "Don't touch it!"—except, of course, when it's broken, in which case touch it all you want, under the time pressure inherent in an outage response.

This approach is not as fruitful as it could be, on a number of levels.

In the field, you need to prepare for ill-behaved circumstances. Power can get cut abruptly. Changes to application or configuration can produce unforeseen behaviors, no matter how full the coverage of testing. Application behavior under various resource-contention conditions (think traffic spikes from news events or firehose-like distributed denial-of-service attacks) can have surprising results. This is not a purely academic curiosity; these types of faults can (and will) affect production and, therefore, in Etsy's case, our sellers and our business. These types of events, however, are difficult to model and simulate with an accuracy that would inspire confidence surrounding the behavior with unknown failure pathologies.

The challenge is that Web systems (like many "complex" systems) are largely intractable, meaning that:

▸ To be fully described, there are many details, not few;

▸ The rate of change is high; the systems change before a full description (and therefore understanding) can be completed;

▸ How components function is partly unknown, as they resonate with each other across varying conditions; and

▸ Processes are heterogeneous and possibly irregular.

In other words, while testing outside of production is a very proper approach, it is incomplete because some behaviors can be seen only in production, no matter how identical a staging environment can be made.

Therefore, another option must be added to the confidence-gaining arsenal: fault injection exercises some-

times referred to as GameDay. The goal is to make these faults happen in *production* in order to anticipate similar behaviors in the future, understand the effects of failures on the underlying systems, and ultimately gain insight into the risks they pose to the business.

Causing failures to happen in complex systems is not a new concept. Organizations such as fire departments have been running full-scale disaster drills for decades. Web engineering has an advantage over these types of drills in that the systems engineers can gather a massive amount of detail on any fault at an extremely high resolution, wield a very large amount of control over the intricate mechanisms of failures, and learn how to recover very quickly from them.

Fault Injection

Constructing a GameDay exercise at Etsy follows this pattern:

1. Imagine a possible untoward event in your infrastructure.
2. Figure out what is needed to prevent that event from affecting your business, and implement that.
3. Cause the event to happen in production, ultimately to prove the noneffect of the event and gain confidence surrounding it.

The greatest advantage of a GameDay exercise is figuring out how to prevent a failure from affecting the business. It is difficult to overstate the importance of steps 1 and 2. The idea is to get a group of engineers together to brainstorm the various failure scenarios that a particular application, service, or infrastructure could experience. This will help remove complacency in the safety of the overall system. Complacency is an enemy of resilience. If a system has a period of little or no degradation, then there is a real risk of it drifting toward failure on multiple levels, because engineers can be convinced—falsely—that the system is experiencing no surprising events because it is inherently safe.

Imagining failure scenarios and asking, "What if...?" can help combat this thinking and bring a constant sense of unease to the organization. This is a hallmark characteristic of high-reliability organizations. Think of it as continuously deploying a business continuity plan (BCP).

Business Justification

In theory, the idea of GameDay exercises may seem sound: you make an explicit effort to anticipate failure scenarios, prepare for handling them gracefully, and then confirm this behavior by purposely injecting those failures into production. In practice, this idea may not seem appealing to the business: it brings risk to the forefront; and without context, the concept of making failures happen on purpose may seem crazy. What if something goes wrong?

The traditional view of failure in production is avoidance at all costs. The assumption is that failure is entirely preventable, and if it does happen, then find the persons responsible (usually those most proximate to the code or systems) and fire them, in the belief that getting rid of "bad apples" is how you bring safety to an organization.

This perspective is, of course, ludicrous. Fault injection and GameDay scenarios can revert this view into a more pragmatic and realistic one.

When approaching Etsy's executive team with the idea of GameDay exercises, I explained that it is not that we want to cause failures out of some perverse need to watch infrastructure crumble; it is because we know that parts of the system *will* inevitably fail, and we need to gain confidence that the system is resilient enough to handle it gracefully.

The concept, I explained to the executives, is that building resilient systems requires experience with failure, and that we want to anticipate and confirm our expectations surrounding failure *more* often, not *less* often. Shying away from the effects of failure in a misguided attempt to reduce risk will result in poor designs, stale recovery skills, and a false sense of safety.

In other words, it is better to prepare for and cause failures to happen in production *while we are watching*, instead of relying on a strategy of hoping the system will behave correctly *when we are not watching*. The worst-case scenario with a GameDay exercise is that something will go wrong during the exercise. In that case, an entire team of engineers is ready to respond to the surprises, and the system will become stronger as a result.

The worst-case scenario in the absence of a GameDay exercise is that something in production will fail that was not anticipated or prepared for, and it will happen when the team is not expecting or watching closely for it.

How can you assure that injecting faults into a live production system doesn't affect actual traffic, revenue, and the end-user experience? This can be done by treating the fault-tolerating and graceful degradation mechanisms as if they were *features*. This means bringing all of the other confidence-building techniques (unit and functional testing, staging hardware environments, among others) to these resilience measures until you are satisfied. Just as with every other feature of the application, it is not finished until you have deployed it to production and have verified that it is working correctly.

Case: Payments System

Earlier this year Etsy rolled out a new payment system (http://www.etsy.com/blog/news/2012/announcing-direct-checkout/) to provide more flexibility and reliability for buyers and sellers on the site. Obviously, resilience was of paramount importance to the success of the project. As with many Etsy features, the rollout to production was done in a gradual ramp-up. Sellers interested in allowing this new payment method could opt in, and Etsy would turn the functionality on for buckets of sellers at a time.

As you might imagine, the payment system is not particularly simple. It has fraud-detection components, audit trails, security mechanisms, processing-state machines, and other components that need to interact with each other. Thus, Etsy has a mission-critical system with a significant amount of complexity and whose expectations for being resilient are very high.

To confirm its ability to withstand failures gracefully, Etsy put together a list of reasonable scenarios to prepare for, develop against, and test in production, including the following:

- One of the app servers dies (power cable yanked out);
- All of the app servers leave the load-balancing pool;
- One of the app servers gets wiped clean and needs to be fully rebuilt from scratch;
- Database dies (power cable yanked

out and/or process is killed ungracefully);

- Database is fully corrupt and needs full restore from backup;
- Offsite database replica is needed to investigate/restore/replay single transactions; and
- Connectivity to third-party sites is cut off entirely.

The engineers then put together all of the expectations for how the system would behave if these scenarios occurred in production, and how they could confirm these expectations with logs, graphs, and alerts. Once armed with these scenarios, they worked on how to make these failures either:

- Not matter at all (transparently recover and continue on with processing);
- Matter only temporally (gracefully degrade with no data loss and provide constructive feedback to the user); or
- Matter only to a minimal subset of users (including an audit log for reconstructing and recovering quickly and possibly automatically).

After these mechanisms were written and tested in development, the time came to test them in production. The Etsy team was cognizant of how much activity the system was seeing; the support and product groups were on hand to help with any necessary communication; and team members went through each of the scenarios, gathering answers to questions such as:

- Were they successful in transparently recovering, through redundancy, replication, queuing?
- How long did each process take—in the case of rebuilding a node automatically from scratch, recovering a database?
- Could they confirm that no data was lost during the entire exercise?
- Were there any surprises?

The team was able to confirm most of the expected behaviors, and the Etsy community (sellers and buyers) was able to continue with its experience on the site, unimpeded by failure.

There were some surprises along the way, however, which the Etsy team took as remediation items coming out of the meeting. First, during the payments process, a third-party fraud-detection service was contacted with information about the transaction. While Etsy uses a number of external APIs (fraud or device reputation), this particular service had no specified timeout on the external call. When testing the inability to contact the service, the Etsy team used firewall rules both to hard close the connection and to attempt to hang it open. Having no specified timeout meant they were relying on the default, which was much too long at 60 seconds. The intended behavior was to fail open, which meant the transaction could continue if the external service was down. This worked, but only after the 60-second timeout, which caused live payments to take longer than necessary during the exercise.

Forcing failures to happen, or even designing systems to fail on their own, generally is not easily sold to management.

This was both a surprise and a relatively easy piece to fix, but it was nonetheless an oversight that affected production during the test.

Recovering from database corruption also took longer than expected. The GameDay exercise was performed on one side of a master-master pair of databases, and while the recovery happened on the corrupted server, the remaining server in the pair took all reads and writes for production. While no production data loss occurred, exposure with reduced capacity occurred for longer than expected, so the Etsy team began to profile and then try to reduce this time of recovery.

The cultural effect of the exercise was palpable. It greatly decreased anxiety surrounding the ramp-up of the payments system; it exposed a few darker-than-desired corners of the code and infrastructure to improve; and it brought an overall increase in confidence in the system. Complacency is not an immediate threat to the system as a result.

Limitations

The goal of fault injection and GameDay exercises is to increase confidence in an otherwise complicated or complex system's ability to stay resilient, but they have limitations.

First, the exercises are not meant to inform how engineering teams handle working under time pressure with escalating and sometimes disorienting scenarios. That needs to come from the postmortems of actual incidents, not from handling faults that have been planned and designed for.

The faults and failure modes are *contrived*. They reflect the fault designer's imagination and therefore

cannot be viewed to be comprehensive enough to gain perfect coverage of the system's safety. While any increase in the confidence of the system's resilient abilities is positive, it is still just that: an increase, not a completion of perfect confidence. Any complex system can (and will) fail in surprising ways, no matter how many different types of faults you inject and recover from.

Some have suggested that continually introducing failures automatically is a more efficient way to gain confidence in the adaptability of the system than manually running GameDay exercises as an engineering-team event. Both approaches have the same limitation mentioned here, in that they result in an increase in confidence but cannot be used to achieve sufficient safety coverage.

Automated fault injection can carry with it a paradox. If the faults that are injected (even at random) are handled in a transparent and graceful way, then they can go unnoticed. You would think this was the goal: for failures not to matter whatsoever when they occur. This masking of failures, however, can result in the very complacency they intend (at least should intend) to decrease. In other words, when you have randomly generated and/or continual fault injection and recovery happening successfully, care must be taken to *raise* the detailed awareness that this is happening—when, how, where. Otherwise, the failures themselves become another component that increases complexity in the system while still having limitations to their functionality (because they are still contrived and therefore sufficient).

Shying away from the effects of failure in a misguided attempt to reduce risk will result in poor designs, stale recovery skills, and a false sense of safety.

Fear

A lot of what I am proposing should simply be an extension of the confidence-building tools that organizations already have. Automated quality assurance, fault tolerance, redundancy, and A/B testing are all in the same category of GameDay scenarios, although likely with less drama.

Should everything have an associated GameDay exercise? Maybe, or maybe not, depending on the level of confidence you have in the components, interactions, and levels of complexity found in your application and infrastructure. Even if your business does not think that GameDay exercises are warranted, however, they ought to have a place in your engineering toolkit.

Safety Vaccines

Why would you introduce faults into an otherwise well-behaved production system? Why would that be useful?

First, these failure-inducing exercises can serve as "vaccines" to improve the safety of a system—a small amount of failure injected to help the system learn to recover. It also keeps a concern of failure alive in the culture of engineering teams, and it keeps complacency at bay.

It gathers groups of people who might not normally get together to share in experiencing failures and to build fault tolerance. It can also help bring the concept of operability in production closer to developers who might not be used to it.

At a high level, production fault injection should be considered one of many approaches used to gain confidence in the safety and resilience of a system. Similar to unit testing, functional testing, and code review, this approach is limited as to which surprising events it can prevent, but it also has benefits, many of which are cultural. We certainly cannot imagine working without it. C

Related articles on queue.acm.org

Black Box Debugging
James A. Whittaker, Herbert H. Thompson
http://queue.acm.org/detail.cfm?id=966807

Too Darned Big to Test
Keith Stobie
http://queue.acm.org/detail.cfm?id=1046944

A Conversation with Steve Bourne, Eric Allman, and Bryan Cantrill
http://queue.acm.org/detail.cfm?id=1413258

Concurrency's Shysters
January 14, 2009
http://blogs.sun.com/bmc/entry/concurrency_s_shysters

John Allspaw (jallspaw@etsy.com) is senior vice president of tech operations at Etsy. He has worked in systems operations for more than 14 years in biotech, government, and online media. He built the backing infrastructures at Salon, InfoWorld, Friendster, and Flickr.

DOI:10.1145/2347736.2347752

Article development led by acmqueue
queue.acm.org

Quality happens only when someone is responsible for it.

BY POUL-HENNING KAMP

A Generation Lost in the Bazaar

THIRTEEN YEARS AGO, Eric Raymond's essay, "The Cathedral and the Bazaar,"[2] redefined our vocabulary and all but promised an end to the waterfall model and big software companies, thanks to the new grass-roots open source software development movement. I found the book thought-provoking, but it did not convince me. On the other hand, being deeply involved in open source, I couldn't help but think that it would be nice if he was right.

The book I brought to the beach house this summer is also thought-provoking, much more so than Raymond's book (which it even mentions rather positively): Frederick P. Brooks's *The Design of Design*.[1] As much as I find myself nodding in agreement and as much as I enjoy Brooks's command of language and subject matter, the book also makes me sad and disappointed.

Thirteen years ago also marks the apogee of the dot-com euphoria, where every teenager was a Web programmer and every college dropout had a Web startup. I had genuine fun trying to teach some of those greenhorns about the good old-fashioned tricks of the trade—test-restoring backups, scripting operating-system installs, version control, and so on. Hindsight, of course, is 20/20 (that is, events may have been less fun than you remember), and there is no escaping that the entire dot-com era was a disaster for IT/CS in general and for software quality and Unix in particular.

I have not seen any competent analysis of how much bigger the IT industry became during the dot-com years. My own estimate is that—counted in the kinds of jobs that would until then have been behind the locked steel doors of the IT department—our trade grew by two orders of magnitude, or if you prefer, by more than 10,000%.

Getting hooked on computers is easy—almost anybody can make a program work, just as almost anybody

can nail two pieces of wood together in a few tries. The trouble is that the market for two pieces of wood nailed together—inexpertly—is fairly small outside of the "proud grandfather" segment, and getting from there to a decent set of chairs or fitted cupboards takes talent, practice, and education. The extra 9,900% had neither practice nor education when they arrived in our trade, and before they ever had the chance to acquire it, the party was over and most of them were out of a job. I will charitably assume that those who managed to hang on were the most talented and most skilled, but even then there is no escaping that as IT professionals they mostly sucked because of their lack of ballast.

The bazaar meme advocated by Raymond, "Just hack it," as opposed to the carefully designed cathedrals of the pre-dot-com years, did, unfortunately, not die with the dot-com madness, and today Unix is rapidly sinking under its weight.

I updated my laptop. I have been running the development version of FreeBSD for 18 years straight now, and compiling even my Spartan work environment from source code takes a full day, because it involves trying to make sense and architecture out of Raymond's anarchistic software bazaar.

At the top level, the FreeBSD ports collection is an attempt to create a map of the bazaar that makes it easy for FreeBSD users to find what they need. In practice this map currently consists of 22,198 files that give a summary description of each stall in the bazaar—a couple of lines telling you roughly what that stall offers and where you can read more about it. Also included are 23,214 Makefiles that tell you what to do with the software you find in each stall. These Makefiles also try to inform you of the choices you should consider, which options to choose, and what would be sensible defaults for them. The map also conveniently comes with 24,400 patch files to smooth over the lack of craftsmanship of many of the wares offered, but, generally, it is lack of portability that creates a need for these patch files.

Finally, the map helpfully tells you that if you want to have www/firefox, you will first need to get devel/nspr, security/nss, databases/sqlite3, and so on. Once you look up those in the map and find their dependencies, and recursively look up their dependencies, you will have a shopping list of the 122 packages you will need before you can get to www/firefox.

Modularity and code reuse is, of course, A Good Thing. Even in the most trivially simple case, however, the CS/IT dogma of code reuse is totally foreign in the bazaar: the software in the FreeBSD ports collection contains at least 1,342 copied and pasted cryptographic algorithms.

If that resistance/ignorance of code reuse had resulted in self-contained and independent packages of software, the price of the code duplication might actually have been a good trade-off for ease of package management. But that was not the case: the packages form a tangled web of haphazard dependencies that results in much code duplication and waste.

Here is one example of an ironic piece of waste: Sam Leffler's graphics/libtiff is one of the 122 packages on the road to www/firefox, yet the resulting Firefox browser does not render TIFF images. For reasons I have not tried to uncover, 10 of the 122 packages need Perl and seven need Python; one of them, devel/glib20, needs both languages for reasons I cannot even imagine.

Further down the shopping list are repeated applications of the Peter Principle, a belief that in an organization where promotion is based on achievement, success, and merit, that organization's members will eventually be promoted beyond their level of ability. The principle is commonly phrased, "Employees tend to rise to their level of incompetence." Applying the principle to software, you will find that you need three different versions of the Make program, a macroprocessor, an assembler, and many other interesting packages. At the bottom of the food chain, so to speak, is libtool, which tries to hide the fact that there is no standardized way to build a shared library in Unix. Instead of standardizing how to do that across all Unixen—something that would take just a single flag to the `ld(1)` command—the Peter Principle was applied and made it libtool's job instead. The Peter Principle is indeed strong in this case—the source code for devel/libtool weighs in at 414,740 lines. Half that line count is test cases, which in principle is commendable, but in practice it is just the Peter Principle at work: the tests elaborately explore the functionality of the complex

/*You are not expected to understand this*/

```
## Whether `make' supports order-only prerequisites.
AC_CACHE_CHECK([whether ${MAKE-make} supports order-only prerequisites],
  [lt_cv_make_order_only],
  [mkdir conftest.dir
   cd conftest.dir
   touch b
   touch a
cat >confmk << 'END'
a: b | c
a b c:
        touch $[]@
END
   touch c
   if ${MAKE-make} -s -q -f confmk >/dev/null 2>&1; then
     lt_cv_make_order_only=yes
   else
     lt_cv_make_order_only=no
   fi
   cd ..
   rm -rf conftest.dir
])
if test $lt_cv_make_order_only = yes; then
  ORDER='|'
else
  ORDER=''
fi
AC_SUBST([ORDER])
```

solution for a problem that should not exist in the first place. Even more maddening is that 31,085 of those lines are in a single unreadably ugly shell script called configure. The idea is that the configure script performs approximately 200 automated tests, so that the user is not burdened with configuring libtool manually. This is a horribly bad idea, already much criticized back in the 1980s when it appeared, as it allows source code to pretend to be portable behind the veneer of the configure script, rather than actually having the quality of portability to begin with. It is a travesty that the configure idea survived.

The 1980s saw very different Unix implementations: Cray-1s with their 24-bit pointers, Amdahl UTS mainframe Unix, a multitude of more or less competently executed SysV+BSD mashups from the minicomputer makers, the almost—but not quite—Unix shims from vendors such as Data General, and even the genuine Unix clone Coherent from the paint company Mark Williams.

The configure scripts back then were written by hand and did things like figure out if this was most like a BSD- or a SysV-style Unix, and then copied one or the other Makefile and maybe also a .h file into place. Later the configure scripts became more ambitious, and as an almost predictable application of the Peter Principle, rather than standardize Unix to eliminate the need for them, somebody wrote a program, autoconf, to write the configure scripts.

Today's Unix/Posix-like operating systems, even including IBM's z/OS mainframe version, as seen with 1980 eyes are identical; yet the 31,085 lines of configure for libtool still checks if <sys/stat.h> and <stdlib.h> exist, even though the Unixen, which lacked them, had neither sufficient memory to execute libtool nor disks big enough for its 16MB source code.

How did that happen?

Well, autoconf, for reasons that have never made sense, was written in the obscure M4 macro language, which means the actual tests look like the example in the accompanying figure.

Needless to say, this is more than most programmers would ever want to put up with, even if they had the skill, so the input files for autoconf happen by copy and paste, often hiding behind increasingly bloated standard macros covering "standard tests" such as those mentioned earlier, which look for compatibility problems not seen in the past 20 years.

This is probably also why libtool's configure probes no fewer than 26 different names for the Fortran compiler my system does not have, and then spends another 26 tests to find out if each of these nonexistent Fortran compilers supports the -g option.

That is the sorry reality of the bazaar Raymond praised in his book: a pile of old festering hacks, endlessly copied and pasted by a clueless generation of IT "professionals" who would not recognize sound IT architecture if you hit them over the head with it. It is difficult to believe today, but under this embarrassing mess lies the ruins of the beautiful cathedral of Unix, deservedly famous for its simplicity of design, its economy of features, and its elegance of execution. (*Sic transit gloria mundi* etc...etc...)

One of Brooks's many excellent points is that quality happens only if somebody has the responsibility for it, and that "somebody" can be no more than one single person—with an exception for a dynamic duo. I am surprised that Brooks does not cite Unix as an example of this claim, since we can pinpoint with almost surgical precision the moment that Unix started to fragment: in the early 1990s when AT&T spun off Unix to commercialize it, thereby robbing it of its architects.

More than once in recent years, others have reached the same conclusion as Brooks. Some have tried to impose a kind of sanity, or even to lay down the law formally in the form of technical standards, hoping to bring order and structure to the bazaar. So far they have all failed spectacularly, because the generation of lost dotcom wunderkids in the bazaar has never seen a cathedral and therefore cannot even imagine why you would want one in the first place, much less what it should look like. It is a sad irony, indeed, that those who most need to read it may find *The Design of Design* entirely incomprehensible. But to anyone who has ever wondered if using m4 macros to configure autoconf to write a shell script to look for 26 Fortran compilers in order to build a Web browser was a bit of a detour, Brooks book offers well-reasoned hope that there can be a better way. C

Related articles on queue.acm.org

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

The Hyperdimensional Tar Pit
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=2108597

Broken Builds
George Neville-Neil
http://queue.acm.org/detail.cfm?id=1740550

References

1. Brooks, F. *The Design of Design*. Addison-Wesley Professional, 2010.
2. Raymond, E. *The Cathedral and the Bazaar*. O'Reilly Media, Sebastapol, CA, 1999.

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

DOI:10.1145/2347736.2347753

Human subjects perform a computationally wide range of tasks from only local, networked interactions.

BY MICHAEL KEARNS

Experiments in Social Computation

SINCE 2005, WE have conducted an extensive series of behavioral experiments at the University of Pennsylvania on the ability of human subjects to solve challenging global tasks in social networks from only local, distributed interactions. In these experiments, dozens of subjects simultaneously gather in a laboratory of networked workstations, and are given financial incentives to resolve "their" local piece of some collective problem, which is specified via individual incentives and may involve aspects of coordination, competition, and strategy. The underlying network structures mediating the interaction are unknown to the subjects, and are often chosen from well-studied stochastic models for social network formation. The tasks examined have been drawn from a wide variety of sources, including computer science and complexity theory, game theory and economics, and sociology. They include problems as diverse as graph coloring, networked trading, and biased voting. This article surveys these experiments and their findings.

Our experiments are inherently interdisciplinary, and draw their formulations and motivations from a number of distinct fields. Here, I mention some of these related areas and the questions they have led us to focus upon.

▸ **Computer science.** Within computer science there is current interest in the field's intersection with economics (in the form of algorithmic game theory and mechanism design[22]), including on the topic of strategic interaction in networks, of which our experiments are a behavioral instance. Within the broader technology community, there is also rising interest in the phenomenon of crowdsourcing,[26] citizen science,[18] and related areas, which have yielded impressive "point solutions," but which remains poorly understood in general. What kinds of computational problems can populations of human subjects (perhaps aided by traditional machine resources) solve in a distributed manner from relatively local information and interaction? Does complexity theory or some variant of it provide any guidance? Our experiments have deliberately examined a wide range of problems with varying computational difficulty and strategic properties. In particular, almost all the tasks we have examined entail much more interdependence between user actions than most crowdsourcing efforts to date.

▸ **Behavioral economics and game theory.** Many of our experiments have

» key insights

- **Groups of human subjects are able to solve challenging collective tasks that require considerably more interdependence than most fielded crowdsourcing systems exhibit.**
- **In its current form, computational complexity is a poor predictor of the outcome of our experiments. Equilibrium concepts from economics are more appropriate in some instances.**
- **The possibility of Web-scale versions of our experiments is intriguing, but they will present their own special challenges of subject recruitment, retention, and management.**

ILLUSTRATION BY RANDY LYHUS

an underlying game-theoretic or economic model, and all are conducted via monetary incentives at the level of individual subjects. They can thus be viewed as experiments in behavioral economics,[1] but taking place in (artificial) social networks, an area of growing interest but with little prior experimental literature. In some cases we can make detailed comparisons between behavior and equilibrium predictions, and find systematic (and therefore potentially rectifiable) differences, such as networked instances of phenomena like inequality aversion.

▸ **Network science.** Network Science is itself an interdisciplinary and emerging area[9,25] that seeks to document "universal" structural properties of social and other large-scale networks, and ask how they might form and influence network formation and dynamics. Our experiments can be viewed as extending this line of questioning into a laboratory setting with human subjects, and examining the ways in which network structure influences human behavior, strategies, and performance.

▸ **Computational social science.** While our experimental designs have often emphasized collective problem solving, it is an inescapable fact that individual human subjects make up the collective, and individual decision-making, strategies, and personalities influence the outcomes. What are these influences, and in what ways do they matter? In many of our experiments there are natural and quantifiable notions of traits like stubbornness, stability, and cooperation whose variation across subjects can be measured and correlated with collective behavior and performance, and in turn used to develop simple computational models of individual behavior for predictive and explanatory purposes.

This article surveys our experiments and results to date, emphasizing overall collective performance, behavioral phenomena arising repeatedly across different tasks, task- and network-specific findings that are particularly striking, and the overall methodology and analyses employed. It is worth noting at the outset that one of the greatest challenges posed by this line of work has been the enormous size of the design space: each experimental session involves the selection of a collective problem, a set of network structures, their decomposition into local interactions and subject incentives, and values for many other design variables. Early on we were faced with a choice between breadth and depth—that is, designing experiments to try to populate many points in this space, or picking very specific types of problems and networks, and examining these more deeply over the years. Since the overarching goal of the project has been to explore the broad themes and questions here, and to develop early pieces of a behavioral science of human computation in networked settings, we have opted for breadth, making direct comparisons between some of our experiments difficult. Clearly much more work is needed for a comprehensive picture to emerge.

While our experimental designs have often emphasized collective problem solving, it is an inescapable fact that individual human subjects make up the collective, and individual decision-making, strategies, and personalities influence the outcomes.

In the remainder of this article, I describe the methodology of our experiments, including the system and its GUIs, human subject methodology, and session design. I then summarize our experiments to date and remark on findings that are common to all or most of the different tasks and highlight more specific experimental results on a task-by-task basis.

Experimental Methodology

All of the experiments discussed here were held over a roughly six-year period, in a series of approximately two-hour sessions in the same laboratory of workstations at the University of Pennsylvania. The experiments used an extensive software, network and visualization platform we have developed for this line of research, and which has been used by colleagues at other institutions as well. In all experiments the number of simultaneous subjects was approximately 36, and almost all of the subjects were drawn from Penn undergraduates taking a survey course on the science of social networks.[12] Each experimental session was preceded by a training and demonstration period in which the task, financial incentives, and GUI were explained, and a practice game was held. Sessions were closely proctored to make sure subjects were attending to their workstation and understood the rules and GUI; under no circumstances was advice on strategy provided. Physical partitions were erected around workstations to ensure subjects could only see their own GUI.

No communication or interaction of any kind outside that provided by the system was permitted. The system tabulated the total financial compensation earned by each subject throughout a session, and subjects were paid by check at a later date following the session. Compensation was strictly limited to the actual earnings of each individual subject according to their own play and the rules of the particular task or game; there was no compensation for mere participation. Following a session, subjects were given an exit survey in which they were asked to describe any strategies they employed and behaviors they observed during the experiments.

Within an individual experimental session, the overall collective task or problem was fixed or varied only slightly (for example, an entire session on graph coloring), while the underlying network structures mediating the interaction would vary considerably. Thus, the sessions were structured as a series of short (1 to 5 minutes) experiments, each with its own network structure but on the same task. This is the natural session format, since once the task and incentives are explained to the subjects, it is relatively easy for them to engage in a series of experiments on differing networks, whereas explaining a new task is time-consuming. Each experiment had a time limit imposed by the system, in order to ensure the subjects would not remain stuck indefinitely on any single experiment. In some sessions, there were also conditions for early termination of an experiment, typically when the instance was "solved" (for example, a proper coloring was found). A typical session thus produced between 50 and 100 short experiments.

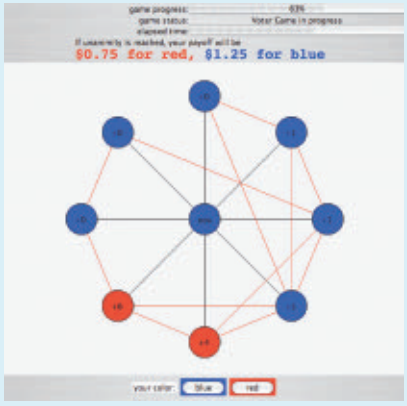
Within an individual experiment, the system randomly assigned subjects to one of the vertices in the network (thus there was neither persistence nor identifiability of network neighbors across experiments). Each subject's GUI (see Figure 1) showed them a local view of the current state of the network—usually a local fragment of the overall network in which the subject's vertex was in the middle and clearly labeled, as well as edges shown to the subject's network neighbors. Edges between a subject's neighbors were shown as well, but no more distant structure. The GUI also always clearly showed the incentives and current payoffs for each subject (which might vary from subject to subject within an experiment), as well the time remaining in the experiment. Typical incentives might pay subjects for being a different color than all their neighbors (graph coloring), the same color (consensus), or perhaps the same color but with different payoffs for different colors (biased voting). Other experiments involved financial scenarios, and the interface provided a mechanism for subjects to bargain or trade with their network neighbors. In general, GUIs always provided enough information for subjects to see the state of their neighbors' current play, and for them to determine their current (financial) best response.

Summary of Experiments

The accompanying table briefly summarizes the nature of the experiments conducted to date, describing the collective task, the network structures used, the individual incentives or mechanism employed, and some of the main findings that we detail below. Our first remark is on the diversity of these experiments along multiple dimensions. In terms of the

Figure 1. Sample screenshot of subject GUI for a biased-voting experiment; many other sessions involved similar GUIs.

The central panel shows the subject's vertex (currently in the "blue" state) with black edges to network neighbors and their current states; red lines denote edges between the subject's neighbors. The bottom action panel allows the subject to change their current state any time, while the top panel specifies their incentives and elapsed time in the experiment.



Summary of experiments to date. ER stands for Erdös-Renyi, PA for preferential attachment.

Task Description	Networks	Incentives/Mechanism	Sample Findings
graph coloring[17]	cycle+chords; PA	differ with neighbors	chords help; importance of information view
coloring and consensus[10]	clique chain w/rewiring	differ/agree with neighbors	opposite structure/task effects
networked trade[13]	ER; PA; structured; all bipartite	limit orders for trades for opposing good	comparison to equilibrium theory; networked inequality aversion
networked bargaining[3]	assorted	Nash bargain on each edge	behavioral price of obstinacy
independent set[15]	assorted	kings and pawns with side payments	side payments help; conflict and fairness
biased voting[14]	ER and PA between types; minority power	consensus with competing individual preferences	well-connected minority rules
network formation[16]	endogenous to the game	biased voting minus edge expenditures	poor collective performance

tasks, the computational complexity of the problems studied[a] varies from the trivial (biased voting and consensus, though this latter problem is difficult in standard models of distributed computation); to the tractable but challenging (networked trade, for which the closest corresponding algorithmic problem is the computation of market equilibria); to the likely intractable (graph coloring and independent set, both *NP*-hard). In terms of the networks, we have investigated standard generative models from the literature such as Erdös-Renyi, preferential attachment, and small worlds; highly structured networks whose design was chosen to highlight strategic tensions in the task and incentives; regular networks without obvious mechanisms to break symmetry; and various other topologies. Figure 2 depicts visualizations of a sampling of network structures investigated. And finally, regarding the financial incentives, these have varied from cooperative (tasks where all players could simultaneously achieve their maximum payoff in the solution); to competitive (where higher payoffs for some players necessarily entail lower payoffs for others); to market-based trading and bargaining, where there are nontrivial networked equilibrium theories and predictions; and to settings where side payments were permitted.

Despite this diversity, and the difficulties in making direct comparisons across sessions and experiments it engenders, there is one unmistakable commonality that has emerged across our six-year investigation: human subjects perform remarkably well at the collective level. While we have observed significant variability in performance across tasks, networks, and incentives, overall the populations have consistently exceeded our expectations. There is a natural and easy way of quantifying this performance: for any given short experiment, we of course know the exact network used, and the incentives and their arrangement within the network, and thus can compute the maximum welfare solution for that particular experiment—that is, the state or arrangement of subject play that would generate the greatest collective payments to the subjects. For each experiment, our system has also recorded the actual payments made, which are by definition less than the maximum social welfare. We can thus sum up all of the actual payments made across all sessions and experiments, and divide it by the sum of all the maximum social welfare payments to arrive at a measure of the overall efficiency of the subject pools over the years.

The resulting figure across the lifetime of our project[b] is 0.88—thus, overall subjects have extracted close to 90% of the payments available to them in principle. In interpreting this figure it should be emphasized that it is an average taken over the particular ensemble of tasks and networks we have studied, which as mentioned before was chosen for its breadth and not in a globally systematic fashion. Clearly it is possible to craft behaviorally "hard" problems and networks.

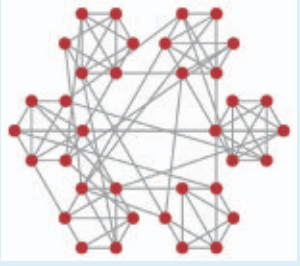
Nevertheless, their efficiency shows that subjects are capable of high performance on a wide variety of tasks and graph topologies.

Another phenomenon consistent across tasks has been the importance of network structure. For most tasks, we found there was a systematic and meaningful dependence of collective behavior on structure, and often an approximate ordering of difficulty of the network topologies could be inferred. Thus, simple cycles prove more difficult for coloring than preferential attachment networks,[17] denser networks result in higher social welfare in networked trading,[13] and so on. However, such dependences on structure are highly task-specific—which is perhaps not surprising for fixed heuristics or algorithms, but has not been documented behaviorally before. Indeed, in one set of experiments we isolated

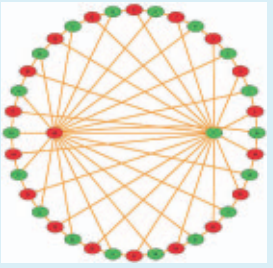
a Clearly computational complexity provides limited insight at best here, since it examines worst-case, centralized, asymptotic computation, all of which are violated in the experiments. But it remains the only comprehensive taxonomy of computational difficulty we have; perhaps these experiments call for a behavioral variant, much as behavioral game theory has provided for its parent field.

b This excludes the most recent experiments in network formation, which are of a qualitatively different nature than the rest, and result in a rather surprising outcome discussed later.

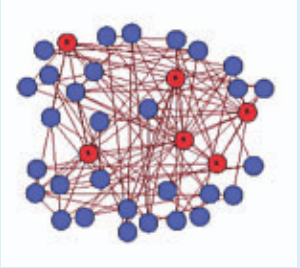
Figure 2. A small sampling of network structures in experiments.



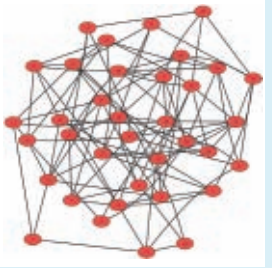
(a) from consensus and independent-set experiments, a chain of six cliques of size 6, with a fraction of the internal clique edges "rewired" to random vertices, thus allowing interpolation between a highly "tribal" network and effectively random networks.



(b) from coloring experiments, an engineered structure with a cycle and two "leaders" in a two-colorable graph.



(c) from biased-voting experiments, a preferential attachment network with a minority of high-degree players preferring red.



(d) from many tasks, a sample Erdös-Renyi network.

this phenomenon by showing that for two cognitively similar (but computationally different) problems, and for a particular generative model for networks, the effects of structure on collective behavioral performance is the opposite in the two tasks,[10] a finding discussed later in greater detail.

The third consistency we found across both tasks and networks was the emergence of individual subject "personalities" or behavioral traits. Our experimental platform is deliberately stylized, and effectively shoehorns the complexity of real human subjects into a highly constrained system, where language, emotion, and other natural forms of communication are eradicated, and all interactions must take place only via simple actions like selecting a color or offering a trade. While there are obvious drawbacks to this stylization in terms of realism, one benefit is that when we make a clear finding—such as the ability of a small but well-connected minority to systematically impose its preferences on the majority[14]—we have done so in a way that might identify the minimal network and task conditions for it to emerge.

Nevertheless, in our experiments we consistently find subjects differentiating and expressing themselves within the constraints of our system in ways that can be measured and compared. For instance, in many of our experiments there are natural notions of traits like stubbornness, stability, selfishness, patience, among others, that can be directly measured in the data, and the frequency of such behavior tallied for each subject. We often find the variation in such behaviors across a population indeed exceeds what can be expected by chance, and thus can be viewed as the personalities of human subjects peeking through our constraints. Harder to measure but still clearly present in almost every experiment we have conducted is the emergence of (sometimes complex) "signaling" mechanisms—it seems that when our system takes language away, the first thing subjects do is try to reintroduce it. From such behavioral traits arise many interesting questions, such as whether specific traits such as stubbornness are correlated with higher payoffs (sometimes they are, other times not), and whether certain mixtures of subject personalities are necessary for effective collective performance (such as a mixture of stubborn and acquiescent individuals in coordination problems).

A theme running throughout our experiments is that intuitions about what networks might be easy or difficult can be strongly violated when considering a distributed human population using only local information.

Highlights of Results

We now turn our attention to results at the level of specific tasks. For each task, I briefly outline any noteworthy details of the GUI or experimental setup, and then highlight some of the main findings.

Coloring and consensus. Our first set of experiments[17] explored the behavioral graph coloring task already alluded to—subjects were given financial incentives to be a different color than their network neighbors, saw only the colors of their local neighborhood, and were free to change their color at any time, choosing from a fixed set of colors whose size was the chromatic number of the underlying graph (thus demanding the subjects find an optimal coloring). It was in these initial experiments that we first found strong effects of network structure. For instance, while a simple two-colorable cycle proved surprisingly hard for the subjects—comparable to their difficulty with more complex and dense preferential attachment graphs—this difficulty was greatly eased by the addition of random chords to the cycle, which reduces diameter and increases edge density. But the preferential attachment networks had the smallest diameter and highest edge density, so these structural properties do not alone explain collective performance.

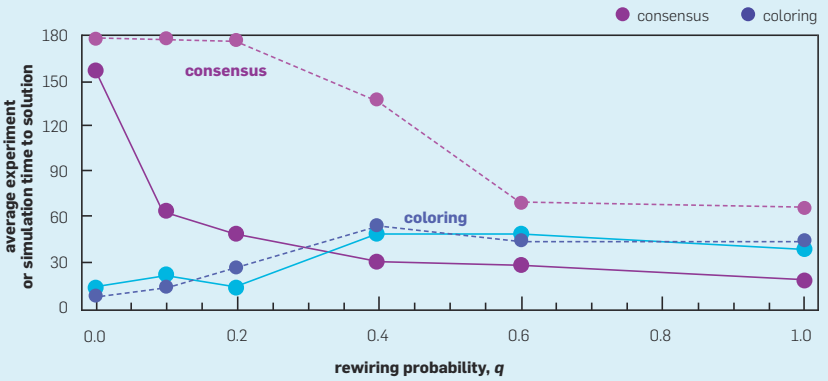
A theme that runs throughout our experiments is that intuitions about what networks might be easy or difficult can be strongly violated when considering a distributed human population using only local information. The challenge of finding simple explanations of such structural results is highlighted by the fact that a natural distributed, randomized heuristic for coloring—namely, not changing colors if there is no current conflict with neighbors, changing to a color resolving a local conflict if one exists, and picking a random color if conflict is unavoidable—produced an ordering of

the difficulty of the networks that was approximately the reverse of that for the subjects.

These first experiments were also the only ones in which we investigated the effects of global information views on performance. In a subset of the experiments, subjects actually saw the current state of the entire network (again with their own vertex in the network clearly indicated), not just the colors of their neighbors. Not surprisingly, this global view led to dramatically improved performance in a simple cycle, where the symmetric structure of the network and the optimal solution become immediately apparent. But strikingly, in preferential attachment networks, global views led to considerable *degradation* in collective performance—perhaps an instance of "information overload," or simply causing subjects to be distracted from attending to their local piece of the global problem.

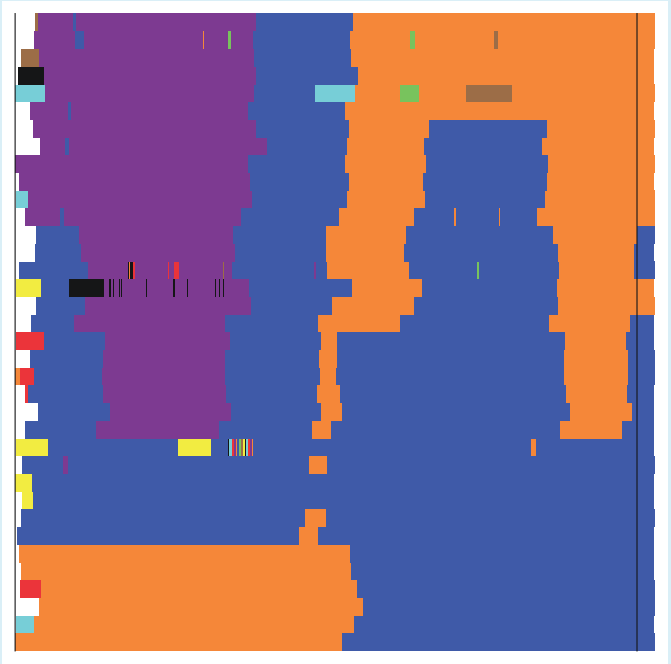
In a later session,[10] we ran experiments on both coloring and *consensus* (where subjects were given financial incentives to be the same color as their neighbors, chosen from a fixed menu of nine colors), on the same set of underlying networks. Despite the vastly different (centralized) computational complexity of these problems—coloring being *NP*-hard, consensus trivial—the two tasks are cognitively very similar and easy for subjects to switch between: coloring is a problem of social differentiation, consensus one of social coordination.

Figure 3. Average time to global solution for coloring and consensus experiments (solid lines) as a function of edge rewiring in a clique-chain network, and simulation times (dashed lines) on the same networks for distributed heuristics. The parametric structure has the opposite effect on the two problems.



In these experiments, the networks were drawn from a parametric family that begins with six cliques of size six loosely connected in a chain. A rewiring parameter q determines the fraction of internal clique edges that are replaced with random "long distance" edges, thus allowing interpolation between a highly clustered, "tribal" network, and the Erdös-Renyi random graph model; see Figure 2(a) for an example. The primary finding here was that the effect on collective performance of varying the rewiring parameter is systematic and *opposite* for the two problems—consensus performance benefits from more rewiring, coloring performance suffers. This effect can be qualitatively captured by simple distributed heuristics, but this does not diminish the striking behavioral phenomenon (see Figure 3). The result suggests that efforts to examine purely structural properties of social and organizational networks, without careful consideration of how structure interacts with the task(s) carried out in those networks, may provide only limited insights on collective behavior.

Figure 4. Visualization of a consensus experiment with low rewiring parameter, showing collective and individual behaviors, and effects of underlying clique structure.



In addition to such systematic, statistically quantifiable results, our experiments often provide interesting opportunities to visualize collective and individual behavior in more anecdotal fashion. Figure 4 shows the actual play during one of the consensus experiments on a network with only a small amount of rewiring, thus largely preserving the tribal clique structure. Each row corresponds to one of the 36 players, and the horizontal axis represents elapsed time in the experiment. The horizontal bars then show the actual color choice by the player at that moment. The first six rows correspond to the players in the first (partially rewired) clique, the next six to the second clique, and so on. The underlying

network structure manifests itself visually in the tendency for these groups of six to change colors approximately simultaneously. As was typical, after an initial diversity of colors, the population quickly settles down to just two or three, and nearly converges to blue before a trickle of orange propagates through the network and takes firm hold; at some point the majority is orange, but this wanes again until the experiment ends in deadlock. Acts of individual signaling (such as toggling between colors) and (apparent) irrationality or experimentation (playing a color not present anywhere else in the network) can also be observed.

Networked trading and bargaining. Our experiments on trading and bargaining differ from the others in that they are accompanied by nontrivial equilibrium theories that generalize certain classical microeconomic models to the networked setting.[4,11] In the networked trading experiments,[4] there were two virtual goods available for trade—call them milk and wheat—and two types of players: those that start with an endowment of milk, but whose payoff is proportional only to how much wheat they obtain via trade; and those that start with wheat but only value milk. All networks were bipartite between the two types of players, and trade was permitted only with network neighbors; players endowed with milk could only trade for wheat and vice-versa, so there were no "resale" or arbitrage opportunities. All endowments were fully divisible and equal, so the only asymmetries are due to network position. The system GUI allowed players to broadcast to their neighbors a proposed rate of exchange[c] of their endowment good for the other good in the form of a traditional limit order in financial markets, and to see the counter offers made by their neighbors; any time the rates of two neighboring limit orders crossed, an irrevocable trade was booked for both parties.

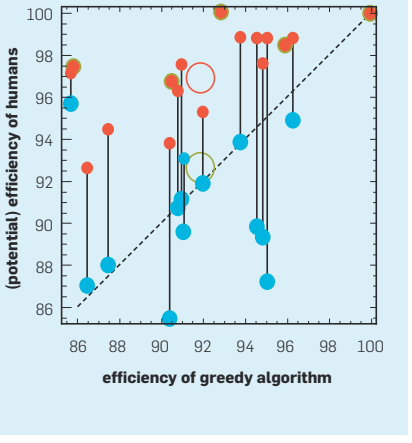
For the one-shot, simultaneous trade version of this model, there is a detailed equilibrium theory that precisely predicts the wealth of every player based on their position in the network;[11] in brief, the richest and poorest players at equilibrium are determined by finding the subset of vertices whose neighbor set yields the greatest contraction,[d] and this can be applied recursively to compute all equilibrium wealths. An implication is that the only bipartite networks in which there will not be variation in player wealths at equilibrium are those that contain perfect matchings. One of the primary goals of the experiment was to test this equilibrium theory behaviorally, particularly because equilibrium wealths are not determined by local structure alone, and thus might be challenging for human subjects to discover from only local interactions; even the best known centralized algorithm for computing equilibrium uses linear programming as a subroutine.[5] We again examined a wide variety of network structures, including several where equilibrium predictions have considerable variation in player wealth.

There were a number of notable findings regarding the comparison of subject behavior to the equilibrium theory. In particular, across all experiments and networks, there was strong *negative* correlation between the equilibrium predicted variation of wealth across players, and the collective earnings of the human subjects—even though there was strong *positive* correlation between equilibrium wealth variation and behavioral wealth variation. In other words, the greater the variation of wealth predicted by equilibrium, the greater the actual variation in behavioral wealth, but the more money that was left on the table by the subjects. This apparent distaste for unequal allocation of payoffs was confirmed by our best-fit model for player payoffs, which turned out to be a mixture of the equilibrium wealth distribution and the uniform distribution in approximately a (3/4; 1/4) weighting. Thus the equilibrium theory is definitely relevant, but is improved by tilting it toward greater equality. This can be viewed as a networked instance of inequality aversion, a bias that has been noted repeatedly in the behavioral game theory literature.[1]

Our experiments on networked bargaining[3] have a similarly financial flavor, and are also accompanied by an equilibrium theory.[4] In these experiments, each edge in the network represents a separate instance of Nash's bargaining game:[21] if by the end of the experiment, the two subjects on each end of an edge can agree on how to split $2, they each receive their negotiated share (otherwise they receive nothing for this edge). Subjects were thus simultaneously bargaining independently with multiple neighbors for multiple payoffs. Network effects can arise due to the fact that different players have different degrees and thus varying numbers of deals, thus affecting their "outside options" regarding any particular deal. In many experiments, the system also enforced limits on the number of deals a player could close; these limits were less than the player's degree, incentivizing subjects to shop around for the best deals in their neighborhood. The system provided a GUI that let players make and see separate counter offers with each of their neighbors.

Perhaps the most interesting finding regarding the comparison between subject performance and a simple

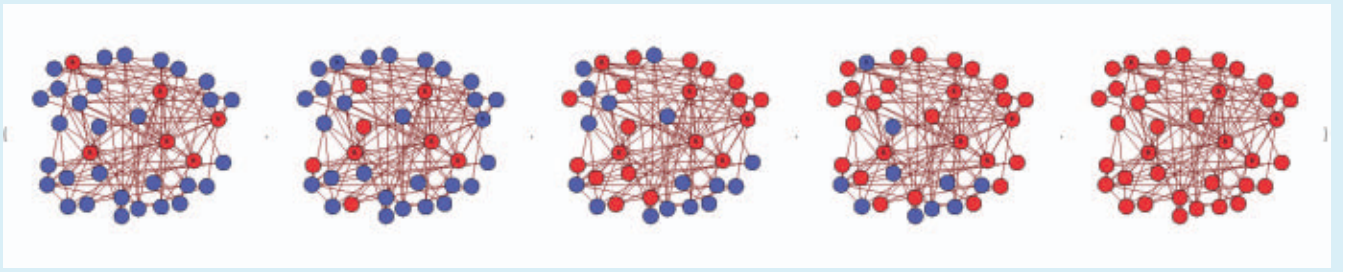
Figure 5. Human performance vs. greedy algorithm in networked bargaining, demonstrating the effects of subject obstinacy. Where occlusions occur, blue dots are slightly enlarged for visual clarity. The length of the vertical lines measure the significant effects of subject obstinacy on payoffs.



c As per the theoretical model, players were not able to offer different rates to different neighbors; thus conceptually prices label vertices, not edges.

d For instance, a set of 10 milk players who collectively have only three neighboring wheat players on the other side of the bipartite network has a contraction of 10/3.

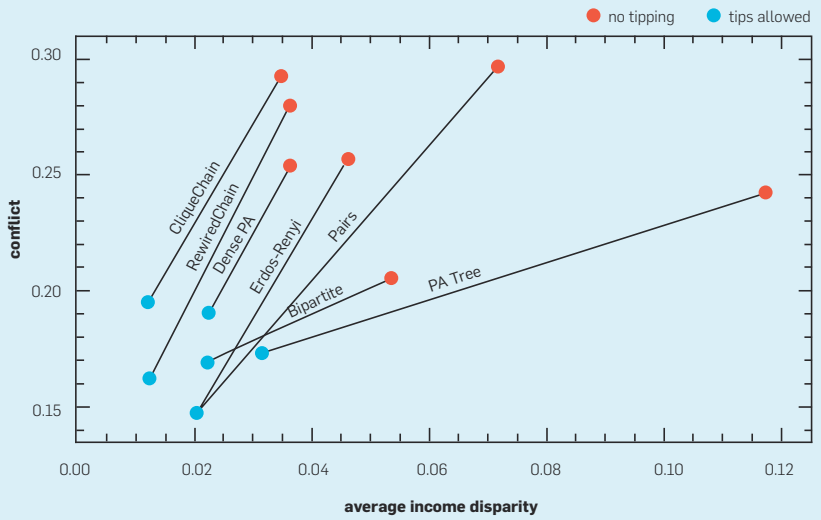
Figure 7. Series of snapshots of global state in a minority power biased voting experiment, showing an instance in which a minority player (upper left vertex R) acquiesces at various times though eventually wins out.



greedy algorithm for approximating the maximum social welfare solution, summarized in Figure 5. This centralized greedy algorithm simply selects random edges in the network on which to close bargains, subject to any deal limits in the experiment, until no further deals could be closed without violating some deal limit. The social welfare obtained (which does not require specifying how the edge deals are split between the two players) is then simply $2 times the number of closed deals, as it is for the behavioral experiments as well.

The blue dots in Figure 5 each represent averages over several trials of one of the network topologies examined (thus each dot corresponds to a different topological family). The *x* value shows the social welfare of the greedy algorithm as a percentage of the maximum social welfare (optimal) solution, while the y value shows the same measure for the human subjects. Averaged over all topologies, both humans and greedy perform rather well—roughly 92% of optimal (blue open circle). However, while the greedy solutions are maximal and thus cannot be locally improved, much of the inefficiency of the subjects can be attributed to what we might call the *Price of Obstinacy*: at the end of many experiments, there were a number of deals that still could have been closed given the deal limits on the two endpoints, but on which the two human subjects had not been able to agree to a split. If we simply apply the greedy algorithm to the final state of each behavioral experiment, and greedily close as many remaining deals as possible, the *potential* performance of the subjects on each topology, absent obstinacy, rises to the orange dot connected to the corresponding blue dot in the figure. This hypothetical subject performance is now well above the performance of pure greedy (all orange points above the diagonal now), and the average across topologies is close to 97% of optimal (orange open circle). In other words, the human subjects are consistently finding better underlying solutions than those obtained by simply running greedy on the initial graph, but are failing to realize those better solutions due to unclosed deals. While humans may show aversion to inequality of payoffs, they can also be stubborn to the point of significant lost payoffs.

Figure 6. From independent-set experiments: Average income disparity between neighbors (*x*-axis) vs. average time neighbors are conflicting kings (*y*-axis), both with (blue) and without (orange) side payments. Grouped by network structure. The side payments uniformly reduced conflict and disparity.



Independent set. Another set of experiments required subjects to declare their vertex to be either a "king" or a "pawn" at each moment, with the following resulting payoffs: any player who is the only one that has declared kingship in his neighborhood enjoys the highest possible rate of pay; but if one or more of their neighbors are also kings, the player receives nothing. On the other hand, pawns receive an intermediate rate of pay regardless of the states of their neighbors. It is easily seen that the Nash equilibria of the one-shot, simultaneous move version of this game are the maximal independent sets (corresponding to the kings) of the graph, while the maximum social welfare state is the largest independent set, whose centralized computation is *NP*-hard. Because we were concerned that computing payoffs based on only the final state of the gain

would lead to an uninteresting global "chicken" strategy (all players declaring king until the final seconds of the experiment, with some players then "blinking" and switching to pawn), in these experiments payoffs accrued continuously according to the prorated time players spent in each of the three possible states (pawn, king with no conflicting neighbors, conflicting kings).

Every experiment was run under two conditions—one just as described above, and another in which the GUI included an additional element: in the case that the player was the lone king in their neighborhood, and thus enjoying the highest rate of pay, a slider bar permitted them to specify a fraction of their earnings in that state to be shared equally among all their neighbors (whose pawn status allows the king's high payoff). These "tips" or side payments could range from 0% to 100% in increments of 10%, and could be adjusted at any time. Note that in some cases, depending on network structure, some vertices might be able to obtain a higher rate of pay by being a pawn receiving side payments from many neighboring kings than by being the lone king in their neighborhood.

The most striking finding was that, across a wide variety of network structures, the introduction of the side payments uniformly raised the collective payoffs or social welfare. Side payment rates were often generous, and averaged close to 20%. Furthermore, when side payments are introduced, both the average income disparity between neighboring players, and the amount of time they spend as conflicting kings, are considerably reduced, across all network structures examined (see Figure 6). This suggests that without side payments, subjects used conflict, which reduces the wealth of all players involved, to express perceived unfairness or inequality. The side payments reduce unfairness and consequently reduce conflict, thus facilitating coordination and raising the social welfare.

Biased voting. The biased voting experiments[14] shared with the earlier consensus experiments an incentive toward collective agreement and coordination, but with an important strategic twist. As in consensus, each player had to simply select a color for their vertex, but now only between the two colors red and blue. If within the allotted time, the *entire population* converged unanimously to either red or blue, the experiment was halted and every player received some payoff. If this did not occur within the allotted time, every player received nothing for that experiment. Thus the incentives were now not at the individual level, but at the collective—players had to not only agree with their neighbors, but with the entire network, even though they were still given only local views and interactions.

The side payments reduce unfairness and consequently reduce conflict, thus facilitating coordination and raising the social welfare.

The strategic twist was that different players were paid different amounts for convergence to the two colors within the same experiment. In particular, some players received a higher payoff for convergence to blue, while others received a higher payoff for convergence to red. Typical incentives might pay blue-preferring players $1.50 for blue convergence and only $0.50 for red, with red-preferring players receiving the reverse. Some experiments permitted asymmetries between higher and lower payoffs, thus incentivizing some players to "care" more about the color chosen by the population. These experiments thus set up a deliberate tension between competing individual preferences and the need for collective unity.

In the most dramatic set of experiments, networks were chosen according to preferential attachment—known to generate a small number of vertices with high degree—and the vast majority of players given incentives that paid more for convergence to blue. However, the minority of vertices preferring red was chosen to be the high-degree vertices. These experiments tested whether a small but well-connected minority could systematically impose its preferences on the majority, thus resulting in suboptimal social welfare.

The answer was resoundingly affirmative: in 27 such "minority power" experiments, 24 of them resulted in the subjects reaching a unanimous choice—in every case, the preferred choice of the well-connected minority. The finding is especially surprising when we remember that since everyone has only local views and information,

the powerful minority has no particular reason to believe they are powerful—in fact, their high degree ensured that at the start of each such experiment, they would see themselves surrounded by players choosing the opposing color. Indeed, the minority players would often acquiesce to the majority early in the experiment (see Figure 7, which shows a series of snapshots of actual play during an experiment). But the dynamics always eventually came to favor the minority choice.

A behavioral network formation game. Our most recent experiments[16] attempted to address what is perhaps the greatest of many artificialities in this line of research: the exogenous imposition of the social network structure mediating interactions. While corporations and other social entities of course often do impose organizational structure, it is natural to believe that in many circumstances, humans will organically construct the communication and interaction patterns required to solve a task efficiently—perhaps even circumventing any imposed hierarchy or structure. Given the aforementioned overall strong performance of our subjects across a wide variety of challenging tasks, even when network structures were complex and not directly optimized for the task, we were naturally interested in whether performance might improve even further if the subjects could collectively choose the networks themselves.

We thus ran among the first experiments in network formation games, on which there is an active theoretical literature.[8,24] We wanted to design such a game in which the formation of the network was not an end in itself, as it is in many of the theoretical works, but was in service of a collective task—which we again chose to be biased voting. The framework was thus as followed: the payoff functions for the players was exactly as described for biased voting, with all players wanting to reach unanimity, but having a preferred (higher payoff) color. Now, however, there were *no edges* in the network at the start of each experiment—every vertex was isolated, and players could thus see only their own color. Throughout the experiment, players could optionally and unilaterally *purchase* edges to other players, resulting in subsequent bilateral viewing of each other's colors for the two players; the GUI would adapt and grow each player's neighborhood view as edges were purchased. A player's edge purchases were deducted from any eventual payoffs from the biased voting task (subject to the constraints that net payoffs could never be negative).

These experiments thus tested whether a small but well-connected minority could systematically impose its preferences on the majority, resulting in suboptimal social welfare.

Players were thus doing two things at once—building the network by purchasing edges, and choosing colors in the biased voting task. The GUI had an edge purchasing panel that showed players icons indicating the degrees and shortest-path distances of players they were not currently connected to, thus allowing them to choose to buy edges (for instance) to players that were far away in the current network and with high degree, perhaps in the hopes that such players would aggregate information from distant areas of the network; or (for instance) to low-degree vertices, perhaps in the hope of strongly influencing them. The formation game adds to the biased voting problem the tension that while the players must collectively build enough edges to facilitate global communication and coordination, individual players would of course prefer that others purchase the edges.

While there were many detailed findings, the overall results were surprising: the collective performance on this task was by far the worse we have seen in all of the experiments to date, and much worse than on the original, exogenous network, biased voting experiments. Across all experiments (that included some in which the subjects started not with the empty network, but with some "seed" edges that were provided for free), the fraction in which unanimity was reached (and thus players received nonzero payoffs) was only 41%—far below the aforementioned nearly 90% efficiency across all previous experiments. We were sufficiently surprised that we ran control experiments in which a subsequent set of subjects were once again given fixed, exogenously imposed networks—but this time, the "hard" networks created by the network formation subjects in cases where they failed to solve the biased voting task. This was

done to investigate the possibility that the formation subjects built good networks for the task, but either ran out of time to reach unanimity, or included subjects who behaved very stubbornly because they had significant edge expenditures and thus strongly held out for their preferred color.

Performance on the control experiments was even worse. The surprising conclusion seems to be that despite the fact that subjects clearly understood the task, and were now given the opportunity to solve it not on an arbitrary network, but one collectively designed by the population in service of the task, they were unable to do so. One candidate for a structural property of the subject-built networks that might account for their difficulty in the biased voting task is *(betweenness) centrality*, a standard measure of a vertex's importance[e] in a network. Compared to the networks used in the original, exogenous-network biased voting experiments, the distribution (across vertices) of centrality in the subject-built networks is considerably more skewed.[16] This means that in the network formation experiments, there was effectively more reliance on a small number of high-centrality vertices or players, making performance less robust to stubbornness or other non-coordinating behaviors by these players. Indeed, there was moderately positive and highly significant correlation between centrality and earnings, indicating that players with high centrality tended to use their position for financial gain rather than global coordination and information aggregation.

Despite their demonstrated ability to solve a diverse range of computational problems on a diverse set of networks, human subjects seem poor at *building* networks, at least within the limited confines of our experiments so far. Further investigation of this phenomenon is clearly warranted.

Concluding Remarks

Despite their diversity, our experiments have established a number of rather consistent facts. At least in moderate population sizes, human subjects can perform a computationally wide range of tasks from only local interaction. Network structure has strong but task-dependent effects. Notions of social fairness and inequality play important roles, despite the anonymity of our networked setting. Behavioral traits of individual subjects are revealed despite the highly simplified and stylized interactions; with language removed, subjects persistently try to invent signaling mechanisms.

There are a number of recent efforts related to the research described here. Some compelling new coloring experiments[7,20] have investigated the conditions under which increased connectivity improves performance. Our experimental approach has thus far aimed for breadth, but studies such as these are necessary to gain depth of understanding. We have also usually done only the most basic statistical analyses of our data, but others have begun to attempt more sophisticated models.[6]

Perhaps the greatest next frontier is to conduct similar experiments on the Web, where a necessary loss of control over subjects and the experimental environment may be compensated by orders of magnitude greater scale, both in population size and the number of experimental conditions investigated. Recent efforts using both the open web and Amazon's Mechanical Turk online labor market have started down this important path.[2,19,23]

Acknowledgments

Many thanks to the stellar colleagues who have been my coauthors on the various papers summarized here: Tanmoy Chakraborty, Stephen Judd, Nick Montfort, Sid Suri, Jinsong Tan, Jennifer Wortman Vaughan, and Eugene Vorobeychik. I give especially warm acknowledgments to Stephen Judd, who has been my primary collaborator throughout the project. Thanks also to Colin Camerer and Duncan Watts, who both encouraged me to start and continue this line of work, and who made a number of important conceptual and methodological suggestions along the way.

e The betweenness centrality of vertex v is average, over all pairs of other vertices u and w, of the fraction of shortest paths between u and w in which v appears.

References

1. Camerer, C. *Behavioral Game Theory*. Princeton University Press, Princeton, NJ, 2003.
2. Centola, D. The spread of behavior in an online social network experiment. *Science 329*, 5996 (2010), 1194–1197.
3. Chakraborty, T., Judd, J.S., Kearns, M., and Tan, J. A behavioral study of bargaining in social networks. *ACM Conference on Electronic Commerce*. ACM Press, New York, 2010, 243–252.
4. Chakraborty, T., Kearns, M., and Khanna, S. Networked bargaining: Algorithms and structural results. *ACM Conference on Electronic Commerce*. ACM Press, New York, 2009,159–168.
5. Devanur, N.R., Papadimitriou, C.H., Saberi, A., and Vazirani, V.V. Market equilibrium via a primal-dual algorithm for a convex program. *Journal of the ACM 55*, 5 (2008).
6. Duong, Q., Wellman, M.P., Singh, S., and Kearns, M. Learning and predicting dynamic behavior with graphical multiagent models. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* (2012).
7. Enemark, D.P., McCubbins, M.D., Paturi, R., and Weller, N. Does more connectivity help groups to solve social problems? In *ACM Conference on Electronic Commerce*. ACM Press, New York, 2011, 21–26.
8. Jackson, M.O. A survey of models of network formation: stability and efficiency. In *Group Formation in Economics: Networks, Clubs and Coalitions*, Cambridge University Press, 2005.
9. Jackson, M.O. *Social and Economic Networks*. Princeton University Press, Princeton, NJ, 2010.
10. Judd, S., Kearns, M., and Vorobeychik, Y. Behavioral dynamics and influence in networked coloring and consensus. *Proceedings of the National Academy of Sciences 107*, 34 (2010), 14978–14982.
11. Kakade, S.M., Kearns, M.J., Ortiz, L.E., Pemantle, R., and Suri, S. Economic properties of social networks. *Neural Information Processing Systems* (2004).
12. Kearns, M. Networked Life. University of Pennsylvania Undergraduate Course; http://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife
13. Kearns, M. and Judd, J.S. Behavioral experiments in networked trade. *ACM Conference on Electronic Commerce*. ACM Press, New York, 2008, 150–159.
14. Kearns, M., Judd, S., Tan, J., and Wortman, J. Behavioral experiments in biased voting in networks. *Proceedings of the National Academy of Sciences 106*, 5 (2009), 1347–1352.
15. Kearns, M., Judd, S., and Vorobeychik, Y. Behavioral conflict and fairness in social networks. *Workshop on Internet and Network Economics* (2011).
16. Kearns, M., Judd, S., and Vorobeychik, Y. Behavioral experiments on a network formation game. *ACM Conference on Electronic Commerce*. ACM Press, New York, 2012.
17. Kearns, M., Suri, S., and Montfort, N. An experimental study of the coloring problem on human subject networks. *Science 313* (2006), 824–827.
18. Khatib, F., Cooper, S., Tyka, M.D., Xu, K., Makedon, I., Popovic, Z., Baker, D., and Foldit Players. Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences* (2011).
19. Mason, W. and Suri, S. Conducting behavioral research on Amazon's Mechanical Turk. *Behavior Research Methods* (2011).
20. McCubbins, M.D., Paturi, R., and Weller, N. Connected coordination network structure and group coordination. *American Politics Research 37*, 5 (2009), 899–920.
21. Nash, J.F. The bargaining problem. *Econometrica 18*, 2 (1950), 155–162.
22. Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V.V., Eds. *Algorithmic Game Theory*. Cambridge University Press, 2007.
23. Suri, S. and Watts, D.J. Cooperation and contagion in Web-based, networked public goods experiments. *PLoS ONE 6*, 3 (2011).
24. Tardos, E. and Wexler, T. Network formation games and the potential function method. In *Algorithmic Game Theory*, Cambridge University Press, 2007, 487–513.
25. Watts, D.J. *Six Degrees: The Science of a Connected Age*. W.W. Norton & Company, 2003.
26. Wikipedia. Crowdsourcing; http://en.wikipedia.org/wiki/Crowdsourcing en.wikipedia.org/wiki/Crowdsourcing.

Michael Kearns (mkearns@cis.upenn.edu) is a professor in the Computer and Information Science Department of the University of Pennsylvania. His research interests include machine learning, social networks, algorithmic game theory, and computational finance.

DOI:10.1145/2347736.2347754

Internet voting is unachievable for the foreseeable future and therefore not inevitable.

BY BARBARA SIMONS AND DOUGLAS W. JONES

Internet Voting in the U.S.

THE ASSERTION THAT Internet voting is the wave of the future has become commonplace. We frequently are asked, "If I can bank online, why can't I vote online?" The question assumes that online banking is safe and secure. However, banks routinely and quietly replenish funds lost to online fraud in order to maintain public confidence.

We are told Internet voting would help citizens living abroad or in the military who currently have difficulty voting. Recent federal legislation to improve the voting process for overseas citizens is a response to that problem. The legislation, which has eliminated most delays, requires states to provide downloadable blank ballots but does not require the insecure return of voted ballots.

Yet another claim is that email voting is safer than Web-based voting, but no email program in widespread use today provides direct support for encrypted email. As a result, attachments are generally sent in the clear, and email ballots are easy to intercept and inspect, violating voters' right to a secret ballot. Intercepted ballots may be modified or discarded without forwarding. Moreover, the ease with which a `From` header can be forged means it is relatively simple to produce large numbers of forged ballots. These special risks faced by email ballots are in addition to the general risks posed by all Internet-based voting schemes.[17]

Many advocates also maintain that Internet voting will increase voter participation, save money, and is safe. We find the safety argument surprising in light of frequent government warnings of cybersecurity threats and news of powerful government-developed viruses. We see little benefit in measures that might improve voter turnout while casting doubt on the integrity of the results.[a]

Almost all the arguments on behalf of Internet voting ignore a critical risk Internet-based voting shares with all computerized voting—wholesale theft. In the days of hand-counted paper ballots, election theft was conducted at the retail level by operatives at polling places and local election offices. By contrast, introduction of computers into the voting process created the threat that elections can be stolen by inserting malware into code on large numbers of machines. The situation is even more dangerous with Internet voting, since both the central servers and the voters' computers are potentially under attack from everywhere.

a Portions of this article are taken from the book *Broken Ballots: Will Your Vote Count?* by Douglas W. Jones and Barbara Simons, CSLI Publications, Stanford, CA, 2012; http://brokenballots.com

» key insights

- **Internet voting is fundamentally insecure.**
- **Most people do not associate widely publicized computer viruses and worms with Internet voting.**
- **Internet voting is being pushed in many countries by vendors, election officials, and well-meaning people who do not understand the risks.**

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

Despite the serious threats it poses to election integrity, Internet voting is being used in several countries and U.S. states, and there is increasing public pressure to adopt it elsewhere. We examine some of these threats, in the hope of encouraging the technical community to oppose Internet voting unless and until the threats are eliminated.

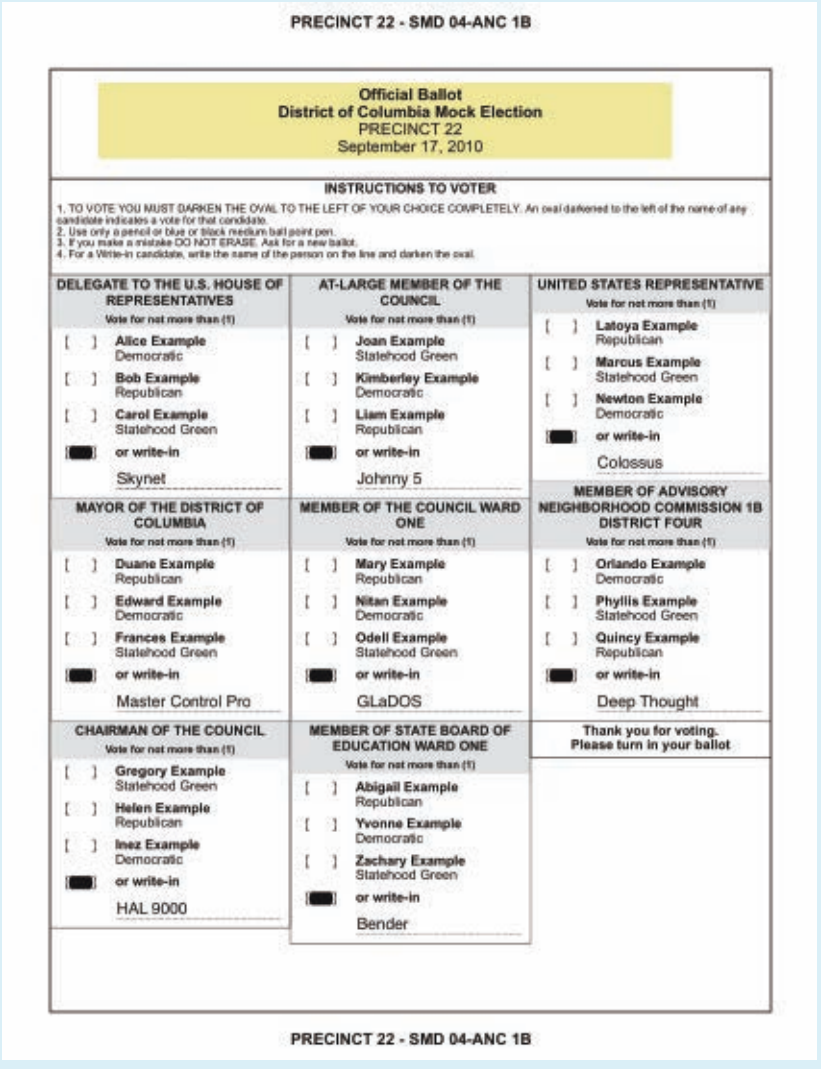
D.C. pilot test Internet voting has generally been deployed without being subjected to public testing prior to use. To the best of our knowledge, the only exception was a "digital vote by mail" pilot project in Washington, D.C. in 2010. In June of that year, the Open Source Digital Voting Foundation announced that it had been selected by the District of Columbia Board of Elections and Ethics (BOEE) to support a project to allow Internet voting for military and overseas voters, starting with the upcoming September primary. The BOEE had optimistically planned a "public review period" in advance of the primary in which everyone was invited to try to attack the system in a mock election. While the system was not ready for the primary, a public test was eventually scheduled to run from September 28 to October 6, with midterm election voting scheduled to begin October 11 or 12.

The break-in. By October 1 people testing the system reported hearing the University of Michigan fight song following a 15-second pause after they submitted their ballots.[6,44] A Michigan team had taken over the system within 36 hours of the start of the tests by exploiting a shell-injection vulnerability, thereby gaining almost total control over the BOEE server. The attackers remained in control for two business days, until the BOEE halted the test after noon on October 1. An attacker intent on subverting a real election would not leave such an obvious calling card. The delay between the break-in and the shutdown of the system reveals how difficult it is to determine that a break-in has occurred, even when the "culprits" announce themselves with music.

On October 5, Michigan professor Alex Halderman revealed that, in addition to installing the fight song, his team had changed ballots cast prior to their intrusion, had rigged the system to alter subsequently cast ballots, and could violate voters' secret ballot rights. That day the BOEE restarted the test with the song removed. Testers were told to print out and mail in their ballots, instead of returning them over the Internet. Figure 1 is the hacked ballot, with write-in candidates selected by the Michigan team.

Halderman was the star of an October 8 oversight hearing, where he dropped additional bombshells. From the start, his team had control of the network infrastructure for the pilot project. The team used the default master password from the owner's manuals, which had not been changed, for the routers and switches, thereby gaining control of the infrastructure and obtaining an alternative way to steal votes in a real election. Control of the network also enabled the team to watch network operators configure and test the equipment. When they discovered that a pair of security cameras in the BOEE data center was connected to the pilot system and unprotected, the team used the cameras to watch the system operators. As proof, Halderman brought some security-camera photos to the hearing. Halderman even discovered a file used to test the system that consisted of copies of all 937 letters sent to real voters. The letters included voter names, IDs, and 16-character PINs for authentication in the real Internet election. While the team could already change voter selections, inclusion of unencrypted PINs in a file used for testing demonstrates that the BOEE did not understand the fundamental principles of computer security. The PINs would have allowed the team or any other intruder to cast ballots for actual voters. Finally, Halderman found evidence of attempted break-ins that appeared to be from China and Iran. Since the attempts involved trying to guess the network logins, the Michigan team changed the previously unchanged defaults (user: *admin*, password: *admin*). Whether or not they were intentionally directed at the D.C. voting system, the attempts showed how dangerous the Internet can be, with sophisticated adversaries from around the world constantly trying to break in to systems.

Figure 1. The rigged District of Columbia ballot.



PRECINCT 22 - SMD 04-ANC 1B

Official Ballot
District of Columbia Mock Election
PRECINCT 22
September 17, 2010

INSTRUCTIONS TO VOTER

1. TO VOTE YOU MUST DARKEN THE OVAL TO THE LEFT OF YOUR CHOICE COMPLETELY. An oval darkened to the left of the name of any candidate indicates a vote for that candidate.
2. Use only a pencil or blue or black medium ball point pen.
3. If you make a mistake DO NOT ERASE. Ask for a new ballot.
4. For a Write-in candidate, write the name of the person on the line and darken the oval.

DELEGATE TO THE U.S. HOUSE OF REPRESENTATIVES
Vote for not more than (1)
[ ] Alice Example, Democratic
[ ] Bob Example, Republican
[ ] Carol Example, Statehood Green
[■] or write-in: Skynet

AT-LARGE MEMBER OF THE COUNCIL
Vote for not more than (1)
[ ] Joan Example, Statehood Green
[ ] Kimberley Example, Democratic
[ ] Liam Example, Republican
[■] or write-in: Johnny 5

UNITED STATES REPRESENTATIVE
Vote for not more than (1)
[ ] Latoya Example, Republican
[ ] Marcus Example, Statehood Green
[ ] Newton Example, Democratic
[■] or write-in: Colossus

MAYOR OF THE DISTRICT OF COLUMBIA
Vote for not more than (1)
[ ] Duane Example, Republican
[ ] Edward Example, Democratic
[ ] Frances Example, Statehood Green
[■] or write-in: Master Control Pro

MEMBER OF THE COUNCIL WARD ONE
Vote for not more than (1)
[ ] Mary Example, Republican
[ ] Nitan Example, Democratic
[ ] Odell Example, Statehood Green
[■] or write-in: GLaDOS

MEMBER OF ADVISORY NEIGHBORHOOD COMMISSION 1B DISTRICT FOUR
Vote for not more than (1)
[ ] Orlando Example, Democratic
[ ] Phyllis Example, Statehood Green
[ ] Quincy Example, Republican
[■] or write-in: Deep Thought

CHAIRMAN OF THE COUNCIL
Vote for not more than (1)
[ ] Gregory Example, Statehood Green
[ ] Helen Example, Republican
[ ] Inez Example, Democratic
[■] or write-in: HAL 9000

MEMBER OF STATE BOARD OF EDUCATION WARD ONE
Vote for not more than (1)
[ ] Abigail Example, Republican
[ ] Yvonne Example, Democratic
[ ] Zachary Example, Statehood Green
[■] or write-in: Bender

Thank you for voting.
Please turn in your ballot

PRECINCT 22 - SMD 04-ANC 1B

Implications of the attack. The D.C. incursion illustrates how Internet voting can be attacked from anywhere. Most complex software systems have an abundance of vulnerabilities, with attackers needing to exploit just one. Moreover, all attacks except those specifically targeting the designated BOEE election network were out of bounds in the pilot test. Examples of non-allowed attacks included client-side malware; denial-of-service attacks; attacks against ISPs; and DNS, routing, and other network attacks. Attackers in a real election would not have felt bound by such constraints. Once the Michigan team had changed all the votes, it was impossible for D.C. officials to reconstruct the original ballots. In a close race, attackers might control the outcome without risk of detection. It took more than a day for D.C. officials to realize their system had been successfully attacked, despite the musical calling card. By the time officials discovered the attack, it was too late to recover from it.

The BOEE had intended to accept voted ballots over the Internet. If there had been no pilot test or if the Michigan team had not participated, members of the military and civilians living abroad who vote in Washington, D.C. would have been voting over a highly vulnerable system. The BOEE did the right thing (for a municipality determined to deploy Internet voting) by setting up a public test. It also learned an important lesson from the test and ultimately canceled the Internet-ballot-return portion. Voters were instead allowed to download blank ballots from the Web and print and return them by postal mail. Unfortunately, other states have not been as responsible. In the upcoming 2012 U.S. election, 33 states will allow some kind of Internet voting, including at least one Web-based Internet pilot project, and the return of voted ballots over the Internet through email attachment or fax, without first

encouraging independent experts to test their systems.[42]

One of us (Jones) has consulted with several election offices, including the BOEE. He observed it to be above average, in terms of both physical and human resources, suggesting that the mistakes found by the Michigan team were not the result of isolated incompetence, but are typical of the best we can expect under current conditions. Likewise, Halderman has said that the quality of the D.C. source code seemed much better than the closed-source electronic voting systems he has examined. Security is difficult, and even organizations with security expertise have been successfully attacked. Given that elections offices are under-resourced, have many other problems to worry about, lack security expertise, and are highly decentralized, it is completely unrealistic to expect extraordinary security competence from them.

The Case for Internet Voting

Despite warnings from independent studies and commissions, as well as sensational news stories about hacking and viruses, some widely held misconceptions about Internet voting persist: It saves money and increases voter turnout; Web-based voting is more secure than postal voting or voting by email or fax; because banking and purchasing can be done over the Internet, voting can be done safely over the Internet; and Internet voting is inevitable—the wave of the future. We discuss the first three points in the following sections and the fourth in the sidebar "Internet Voting and E-Commerce Compared." Regarding the inevitability of Internet voting, some of the most outspoken Internet voting opponents are highly respected computer security experts. Our goal is to convince you that secure Internet voting is unachievable for the foreseeable future and therefore, we sincerely hope, not inevitable.

Saves money. The cost of Internet voting, especially up-front charges, can be steep. For example, 2009 cost estimates from Internet voting vendor Everyone Counts were so large that a legislative proposal in Washington state to allow Internet voting for military and civilian voters was killed in committee. The estimated costs, obtained by John Gideon of VotersUnite, included proposed up-front costs ranging from $2.5 million to $4.44 million. After that, each county would have been hit with an annual license fee of $20,000–$120,000, plus $2–$7 per overseas voter.[5]

In the March 2011 election in the state of New South Wales, Australia, 46,864 people voted on an Internet voting system called iVotes, also an Everyone Counts product.[33] The development and implementation costs for using iVotes in the election exceeded $3.5 million (Australian dollars), resulting in a cost of about $74 per vote cast. By contrast, the average cost for all forms of voting in the same election was $8 per vote, though the cost per Internet vote would have decreased if amortized over more voters.

Increases turnout. Internet voting does not necessarily increase turnout. Everyone Counts ran an Internet-based election in Swindon, U.K., in 2007 and a local election in Honolulu, HI, in 2009 where votes were cast only by Internet or telephone. The Electoral Commission, established by the U.K. Parliament, determined that Internet voting in Swindon had a negligible effect on turnout; meanwhile, in Honolulu there was an 83% drop in turnout compared to a similar election in 2007.[22,40] We know of no rigorous study of the impact of Internet voting on turnout; conducting such a study would be difficult, since turnout can vary enormously from election to election. But even if Internet voting could increase turnout, the increase would be irrelevant if the election results were at risk of corruption by insecure Internet use.

Web-based voting is more secure. Verifiability and transparency are critical aspects of any election, especially if it involves a secret ballot. It is fundamentally impossible for anyone, even election officials, to directly oversee or observe the tabulation of an Internet-based election, including one that is Web-based. A software bug or an attack could cause an election outcome to be wrong because either the tabulation is incorrect or the voters' selections were modified. To address such risks, we need to determine after an election that the technology operated correctly and the declared winner actually won.

We can verify the results of a paper-based election by auditing a sample of the cast ballots or, in the extreme, by recounting all of them. Such an audit or recount must involve a secure, observable chain of custody of the ballots, something impossible with current Internet voting technology. Allowing voters to print copies of their ballots for personal use is meaningless, because these copies may not match the electronic versions used in computing the results.

Military Voting

Members of uniformed services and their families and non-military citizens living overseas are called UOCAVA voters, after the U.S. Uniformed and Overseas Citizens Absentee Voting Act of 1986 (http://www.fvap.gov/reference/laws/uocava.html). They have long complained that absentee ballots are never delivered or their returned voted ballots arrive too late to be counted, concerns used to justify the push for Internet voting at both the state and federal levels. A widely discussed solution is to have the military run its own centralized Internet voting system over its high-security infrastructure. This is a bad idea for at least two reasons: First, it runs counter to the principle of civilian control over the military and creates the potential that the military might control the vote. Second, it is unrealistic and unwise to even consider connecting unsecure Web servers run by local election officials to a military network that is supposed to maintain a high level of security. Some supporters of Internet voting for the military have noted that postal mail ballots are also not secure. While it is true that all forms of remote voting pose security problems, Internet voting can be attacked by anyone from anywhere, something that is not the case for postal ballots. In addition, the Internet can be used for wholesale attacks on large numbers of voters, whereas attacks on postal ballots are inherently confined to a retail scale.

Two projects for UOCAVA voters are noteworthy: SERVE, killed in 2004, and Operation BRAVO, implemented in the 2008 U.S. presidential election:

SERVE. The Secure Electronic Registration and Voting Experiment, or SERVE (www.fvap.gov/resources/media/serve.pdf), was the most ambitious project to date intended for use

by UOCAVA voters. The goal of the $22 million project was to allow registration and voting over the Internet in the 2004 primaries and general election. Participation by states and counties within those states was voluntary. Voters could use any Windows computer, either their own or a public computer, like those found in libraries and cybercafés. Voters were responsible for the security of whatever computers they used. The vendor was Accenture.

In 2003, a group of experts called the Security Peer Review Group was assembled by the Federal Voting Assistance Program (FVAP) to evaluate SERVE; FVAP was charged with facilitating voting for all UOCAVA voters. Following two three-day meetings with FVAP and the lead technical staff of SERVE, the four computer scientists who attended both meetings, including one of us (Simons), released a report, the conclusion of which said: "Because the danger of successful, large-scale attacks is so great, we reluctantly recommend shutting down the development of SERVE immediately and not attempting anything like it in the future until both the Internet and the world's home computer infrastructure have been fundamentally redesigned, or some other unforeseen security breakthroughs appear."[18]

When the report was issued in early 2004, 50 counties in seven states—Arkansas, Florida, Hawaii, North Carolina, South Carolina, Utah, and Washington—were planning to participate in SERVE. FVAP had estimated the maximum overall vote total would be approximately 100,000, including primaries and the general election. On January 30, 2004 Deputy Secretary of Defense Paul Wolfowitz said the Pentagon "...will not be using the SERVE Internet voting project in view of the inability to assure legitimacy of votes that would be cast using the system, which thereby brings into doubt the integrity of election results."[43] SERVE was subsequently terminated.

Operation BRAVO. In 2008, Operation BRAVO, or Bring Remote Access to Voters Overseas, provided Internet voting from secure kiosks for residents of Okaloosa County, FL. Unlike previous pilot projects, these kiosks were equipped with printers to create paper voter-choice records of voters' ballots. Voters could verify the records before leaving the kiosk, after which the records were flown back to Okaloosa County for manual reconciliation with the ballots sent over an Internet-based virtual private network. Small discrepancies in the ballot count were uncovered by law professor Martha Mahoney of the University of Miami, but, as of August 2012, BRAVO had yet to release a formal report explaining the discrepancies.[26] The vendor was Scytl.

The Okaloosa County experiment concerned only a single county. Expanding kiosk-based Internet voting for all service members would be very difficult, since the system would have to deal with tens of thousands of different ballot styles and conflicting state rules governing ballot presentation, requirements that would also add significantly to the cost.

The MOVE Act. Instead of Internet voting, why not allow remote voters to download a blank ballot from the Internet, print it, and return the voted ballots by mail? If the blank ballots are available early enough, most voted ballots should arrive in time to be counted. Such a system might not have the pizzazz of Internet voting but would have fewer security issues and almost certainly involve less cost. That is one of the reforms dictated by the 2009 Military and Overseas Voter Empowerment, or MOVE, Act. Written to address the problems of UOCAVA voters, MOVE requires states to make blank ballots available electronically at least 45 days prior to any federal election; UOCAVA voters may also request and receive voter-registration and absentee-ballot applications electronically.

The Military Postal Service Agency analyzed the handling of absentee ballots during the 2010 general election,[29] finding problems with getting postal ballots to members of the military, though paper ballots were generally returned quickly. Many had been electronically downloaded, filled out by service members, and returned by postal mail. The average postal delay for returned ballots was 5.2 days, well ahead of the seven-day limit set by the MOVE Act; 92% of absentee ballots were delivered within seven days of acceptance at overseas Military Post Offices (MPOs). Only 118 out of 23,900 voted ballots, most likely from Afghanistan or Iraq, took 20 or more days to be returned from an MPO. The time to get a voted ballot from a service member to an MPO ranged from two to 20 days. Therefore, if election officials provide downloadable blank ballots at least 45 days before an election, essentially all members of the military should be able to return their voted paper ballots in time to be counted.

Risks

Not satisfied with the significant speed-up provided by MOVE, Internet-voting advocates continue to call for the return of voted ballots through the Internet, either as email attachments or as some kind of Web form. Doing either securely would require solving some of the most intractable problems in cybersecurity:

The server. In the 2010 D.C. pilot project, University of Michigan graduate students attacked the election server over the Internet. Independent hackers, political operatives, foreign governments, and terrorists could also mount such attacks. Local election officials with little or no expertise in computer security have little hope of defending themselves.

Corporate and government vulnerability. Many corporations and government agencies store sensitive or classified information on their computers, sharing with election officials the goal of defending against attackers who might steal or alter such information. Despite large staffs of security professionals with significant resources, computers in major corporations and government agencies have been attacked successfully. For example, a 2008 survey of approximately 1,000 large organizations worldwide found the average loss per organization from intellectual property cybertheft was about $4.6 million.[19] A December 2009 report from the Computer Security Institute (http://gocsi.com) surveying 443 U.S. companies and government agencies found 64% had reported malware infections during the preceding year.[36]

A major China-based Internet attack on Google and many other companies in late 2009 showed that even major corporate sites are vulnerable. The attack targeted Gmail accounts of Chinese human-rights activists and Google's own intellectual property, including

software-development systems.[31] As many as 34 companies were targeted, including Adobe, Juniper Networks, defense contractor Northrop-Grumman, major security supplier Symantec, and Yahoo!.[41] The attacked companies have vastly more security expertise and resources than local election officials or today's relatively small Internet voting vendors. The attacks used email that appeared to come from trusted sources, so victims would be tricked into clicking on a link or opening an attachment. Then, using a vulnerability in Microsoft's Internet Explorer browser, the attacker would download and install malware that took complete control of the compromised systems.

George Kurtz, executive vice president and worldwide chief technology officer of McAfee, an Internet security company, expressed dismay at the implications: "All I can say is wow. The world has changed. Everyone's threat model now needs to be adapted to the new reality of these advanced persistent threats. In addition to worrying about Eastern European cybercriminals trying to siphon off credit card databases, you have to focus on protecting all of your core intellectual property, private nonfinancial customer information and anything else of intangible value."[23]

Government sites have also been vulnerable. In a March 2010 address to the RSA Security Conference, FBI director Robert S. Mueller said the FBI's computer network had been penetrated and the attackers had "corrupted data."[31] Later that year, General Michael Hayden, former director of both the CIA and the NSA, said: "The modern-day bank robber isn't speeding up to a suburban bank with weapons drawn and notes passed to the teller. He's on the Web taking things of value from you and me."[13]

Finally, malware that appears to be government-generated has been used to obtain critical intelligence, as in the case of the Flame virus, and, for targeted attacks, Stuxnet. Both were widely reported to have been developed by the governments of Israel and the U.S., with Stuxnet apparently created to attack Iran's nuclear facilities.[32,38] Similar tools could allow a foreign power to attack or subvert an Internet election anywhere.

Internet Voting and E-Commerce Compared

Internet voting involves complications not found in e-commerce:

Secret ballots. **Secret ballots are required by law to protect against vote buying and coercion. Ballot secrecy prohibits anyone from linking voted ballots to the voters casting them. This precludes the kind of transaction logging routinely used in e-commerce to allow reconstruction of who did what and when, should a question arise.**

Receipts. **Receipts, including unique transaction numbers and complete transaction descriptions, are routinely issued in e-commerce. These receipts confirm that the correct orders were placed and may be used as proof of purchase in the event of disputes. Ballot secrecy prevents issuing any documents to voters that voters could use to prove how they voted. Documents that do not provide such proof are of limited use in an audit or recount.**

Malfunction and fraud. **In the event of an e-commerce failure due to malfunction or fraud, there is a good chance the situation will be rectified or that the purchaser can stop a credit-card payment after noticing the discrepancy. However, if a ballot is not successfully cast on election day, the voter probably will not know and almost certainly will not be able to revote.**

Vote buying and selling. **Unlike commercial activities, vote buying and selling is illegal. In the 2000 U.S. presidential election between Republican George W. Bush and Democrat Al Gore, an online system designed to broker Green Party candidate Ralph Nader and Gore votes was created but forced to shut down by the California attorney general. There is no evidence that any votes were actually traded. With Internet voting, voters could sell their voting credentials, perhaps even online, using a Web site designed to automatically cast their ballots.[a]**

No proposed Internet voting system is able to overcome these hurdles.

a When family members vote on a home computer or citizens vote from a computer in a public library, multiple voters will share the same IP address; while it is possible to detect multiple votes from one IP address, it would be problematic to prohibit them.

Insider attacks. While many security discussions focus on outsider attacks, insider attacks might be even more dangerous. A risk of any computerized voting, including Internet voting, is that one or more insiders (programmers, election officials, volunteers, or vendors to whom the election is outsourced) could rig an election by manipulating election software. Since computerized voting is an opportunity for wholesale rigging through software used by large numbers of voters, the size of the conspiracy needed to win an election is greatly reduced, as is the risk of being caught.

An attacker could add a back door to the system, with or without the vendor's knowledge. In general, no amount of testing can be relied on to reveal the presence of a back door. A thorough code review (not required by current law) can sometimes do this, but code reviews cannot reliably distinguish between an innocent mistake and intentional malware. A trusted insider (such as former CIA agent Aldrich Ames[b]) can do tremendous damage, even if eventually caught.

The client. Since malware can infect public or privately owned machines linked to the Internet without the owner's knowledge or permission, client-side malware designed to steal an election poses significant risks for ballots cast from voters' computers. These risks include credential theft, copying of the ballot to a third party, and modification of the ballot before encryption, as well as outright prevention of voting. Machines can be infected in many ways, including downloading documents with malicious macros, browser plugins, or improper security settings.

Furthermore, millions of computers are already connected to botnets. In 2010, the FBI reported the Mariposa botnet may have infected eight million to 12 million computers worldwide.[9] The virus used to create the botnet could steal credit-card data and online-

b Ames gave the Soviet Union significant U.S. secrets resulting in the death of a number of "CIA assets."

banking passwords, as well as launch a denial-of-service attack; the creator of the virus also sold customized versions with augmented features. A Microsoft report estimated that in the first half of 2010 more that 2.2 million U.S. Windows PCs were in botnets.[4]

Those wishing to rig elections need not build new botnets. Many botnets used for financial fraud are available for rent. It would not take a large staff to alter existing malware to attack elections, and it would not be out of character for existing malware developers to offer ready-to-customize election-rigging malware as soon as Internet voting were to enter widespread use.

The sheer number of potential attacks and the difficulty of preventing any of them increase the vulnerability of Internet-based elections. In light of the many successful attacks against governments, major banks, and the world's technology leaders, it should be relatively easy to entrap large numbers of voters who are not technologists. Once a voter's computer is infected, all bets are off. Malware can make the computer display a ballot image that represents the voter's intent correctly, even as it sends something entirely different over the Internet. That is, it is the virus that votes, not the voter. The voter never knows, because it is impossible for the voter to see what is actually sent.

Since antivirus software works by checking for known viruses and worms, whenever a new virus appears, the antivirus software must be updated. There can be many days or even weeks between the time the virus is initially distributed and when it is recognized and analyzed. After that, the virus fix must be distributed, and victims must disinfect their machines. Because antivirus software has limited capability for recognizing unknown malware, a new virus or worm may well escape detection for a while. Even if detected, removal can be difficult, as most PC owners who have had to deal with adware and spyware are aware. A 2007 study found that antivirus software has become less effective over time, with recognition of malware by most commercial antivirus software falling from 40%–50% at the beginning of 2007 to 20%–30% by the end of that year.[12] Another set of experiments conducted at the University of Michigan showed the number of malware samples detected decreased significantly as the malware became more current; when the malware was only one week old, the detection rate was very low.[34] Given the limitations of antivirus software, an effective attack would be to distribute election-stealing malware far in advance of the election. If the malware were to spread silently, it could infect a large number of machines before being detected, if it is detected at all. Moreover, it might be impossible to determine which votes are modified or even which computers are infected.

The Conficker worm illustrates the risk malware poses to Internet elections. Having rapidly infected from nine million to 15 million machines in 2009, Conficker could "call home" for more instructions, so the unknown creator of Conficker could instruct infected machines to install additional malware remotely without the computer owner's knowledge.[2] The new instructions might target specific candidates and elections shortly before a vote.

While many viruses and worms are planted without the computer owner's knowledge, users can be duped into downloading highly questionable software. In August 2009 a spam message circulated, saying "If You dont [sic] like Obama come here, you can help to ddos [Distributed Denial of Service] his site with your installs." CNET News reported that people who clicked on the email link were offered money in exchange for downloading the software; they were even told to return to the Web site for updates if their virus-detection software deleted their first download.[30] While the source of the software is not known, the goal could have been to disrupt sites associated with President Barack Obama, to engage in identity theft, or even to infect machines of Obama opponents, something that could be especially useful if Internet voting were to become an option in the U.S.

Threat example: The Zeus virus. The Zeus virus illustrates how a virus can manipulate what a voter sees and change the voter's selection. While Zeus has been used mainly to steal money, it would not be difficult to reprogram it to steal votes.

In April 2009, malicious software was discovered in Paul McCartney's Web site that redirected visitors to an IP address in Amsterdam in order to exploit vulnerabilities on the victims' machines to install the Zeus virus.[16] The infection, planted shortly before McCartney's New York reunion concert with Ringo Starr, was timed to catch as many victims as possible before discovery.

The German edition of Wikipedia was another source of infection.[14] A bogus Wikipedia article about another dangerous piece of malware contained a link to software that would supposedly fix the problem. However, anyone who downloaded the "fix" was actually downloading a copy of Zeus. In 2009 it was estimated by security firm Damballa that Zeus had infected about 3.6 million PCs in the U.S. alone.[28]

Zeus was built to steal money from online financial accounts. When victims would visit their banks' Web sites, Zeus would copy their credentials and send them to a remote location where they would be used to steal from their accounts. Zeus could even forge financial statements so victims would see no evidence of the theft when checking their online statements.[39] Victims typically learned of the theft only when financial transactions failed to clear due to insufficient funds, at which point it was too late to retrieve the money.

The Zeus virus also spoofed verification systems used by Visa and MasterCard when enrolling new users[7] (see Figure 2), thereby obtaining sensitive information (such as Social Security numbers, card numbers, and PINs) from unknowing victims who would think they were providing the information to the real bank. This information, sent to the attacker's computers, would be used to defraud the victims.

Yet another attack was reported in August 2010 by Internet security firm M86 Security; the report said that about 3,000 bank customers in the U.K. were victimized by a form of the Zeus virus. The announcement accompanying the report's release, which did not provide the bank's name, said the following about the attack:[25] "Unprotected customers were infected by a Trojan—which managed to avoid detection by traditional anti-virus software—while browsing the Internet. The Trojan, a Zeus v3, steals the customer's online banking ID and hijacks

their online banking sessions. It then checks the account balance and, if the account balance is bigger than GBP 800 value, it issues a money transfer transaction... From July 5, the cyber criminals have successfully stolen GBP 675,000 (c. USD 1,077,000) and the attack is still progressing."

On September 29, 2010, the U.K. Police Central e-crime Unit announced the arrest of 19 individuals accused of using Zeus to steal $6 million from thousands of victims over a three-month period.[24] To this day, new Zeus attacks continue to be discovered; for example, in October 2010, *Computerworld* reported that Zeus was attacking Charles Schwab investment accounts,[20] with victims' machines infected by links to malicious sites hidden in bogus LinkedIn reminders. There is even a criminal service that will compile a Zeus binary for a fee.[10]

Impersonating the election server. Another Internet risk involves Website spoofing. Because counterfeit sites can be made to look like legitimate sites, spoofing can fool victims into revealing sensitive personal information. With Internet voting, spoofing can be used to trick voters into thinking they have actually voted when in fact they have not, while also collecting authentication codes and voters' intended ballots, a violation of the right to a secret ballot.

Phishing involves email messages that appear to be from a legitimate organization, such as a credit-card company. The phony message contains an authentic-looking link that appears to go to a legitimate site but actually goes to a spoofed site. When such email messages and Web sites are well designed, victims end up providing sensitive information, such as credit-card numbers. Phishing is usually used to steal personal information, but can also be used to trick voters into voting on a spoofed Web site. Phishing is a powerful tool for amplifying the power of spoofing, though its effectiveness can be reduced if voters are instructed to always type in the full URL of the voting Web site, instead of just clicking on links.

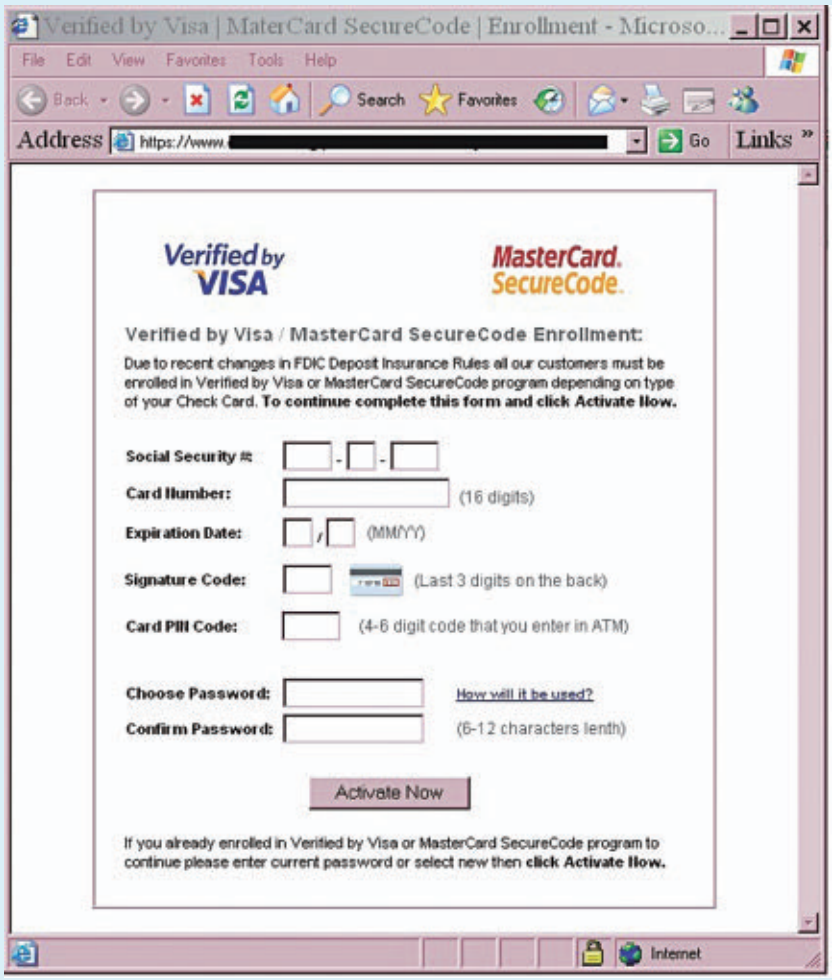
A counterfeit voting site can conduct a man-in-the-middle attack. In its simplest form, the counterfeit site relies entirely on the real site for content, monitoring and occasionally editing the information flow between the voter and the real election server. This allows the attacker to intercept information, such as passwords and votes, and potentially to alter votes. A more complex counterfeit could simulate a voting session, then use the credentials collected from the voter at a later time to cast a forged ballot. Monitoring the IP addresses from which ballots are cast is not a defense, since multiple voters might share the same IP address for legitimate reasons.

A common way to avoid counterfeit Web sites is to rely on a certificate authority (CA) to authenticate sites. If the browser does not recognize the issuer of a certificate, it will ask if the user still wants to access the site. A user who does not understand the significance of the browser's question may naïvely ignore it and access a counterfeit site.

Even when voters are careful to visit only sites they believe are legitimate, they could still be victimized. First, it is possible to trick many browsers into going to the attacker's, rather than to the legitimate, site.[45] Second, some CAs do not validate the identities of sites they vouch for.[35] Third, an attack on the CA can create fake SSL certificates, as happened to DigiNotar, a Dutch CA.[21] Finally, an attack on the routing infrastructure of the Internet could divert voters to a counterfeit voting site without their noticing the diversion.[27]

Denial-of-service attacks. There are many documented instances of Distributed Denial-of-Service (DDoS) attacks. For example, the massive 2007 DDoS attack on Estonia and the attacks on the Republic of Georgia during the 2008 Russo-Georgian war all originated in Russia. Other victims of DDoS attacks include Amazon, eBay, Facebook, Google, Twitter, and Yahoo!. Politically

Figure 2. Bogus enrollment screen displayed by Zeus; screenshot by Amit Klein of Trusteer.



motivated DDoS attacks, like the one on Wikileaks in 2010 and a reprisal by Anonymous against MasterCard, have become relatively common.

A DDoS attack could prevent certain groups from voting or even disrupt an entire election, as probably occurred in a 2003 leadership vote by the New Democratic Party (NDP) in Canada. Internet voting for the NDP election lasted from January 2 until the party convention January 25, 2003. Coincidentally, on January 25, the same day the Slammer worm was attacking large numbers of (unpatched) Windows 2000 servers on the Internet, the NDP voting site was reportedly down or effectively unusable for hours.[3]

Due to the secrecy surrounding the technical aspects of the NDP election, we do not know if the NDP voting site was brought down by a DDoS attack or by the Slammer worm. The vendor, election.com, claimed to have patched the servers against Slammer and maintained that it experienced a denial-of-service attack. Unfortunately, election.com provided neither logs nor other proof that its servers were patched, nor did it permit expert examination of its records. There was no transparency and hence no way for an independent outsider to determine what had happened.

Not having learned from the 2003 attack, the NDP suffered a massive DDoS attack during its March 2012 leadership election. The NDP was so ill prepared that people attending the party conference were unable to vote during the attack, as no back-up paper had been provided. Once again, there was no independent examination or report.

Loss of the secret ballot. All forms of remote voting diminish ballot secrecy and increase the risk of coercion and vote selling simply because they eliminate voting booths. Internet voting decreases secrecy still further. States that allow the return of voted ballots by fax or email attachments have been asking voters to sign statements relinquishing the right to a secret ballot. Mix nets and other cryptographic schemes can mimic the secrecy protections of the double envelopes traditionally used to partially preserve ballot secrecy in postal voting, but they do not protect against client-side attacks.

The threat to eliminate the secret ballot for a class of voters is disturbing for several reasons: First, it renders these voters second-class citizens, deprived of a right other citizens take for granted. Second, there is no need to eliminate the secret ballot for overseas voters, as we discussed earlier. Third, and most important, ballot-secrecy protection is more than an individual right; it is a systemic requirement, essential for fair, honest elections. Without ballot secrecy, voters, especially those in hierarchical organizations, such as the military, may be subject to coercion. An election where some voters can be pressured to vote a particular way is not a free and fair election.

Bribery. Finally, we cannot rule out the threat of old-fashioned bribery. National races in the U.S. cost vast sums—a small fraction of which would be an exceedingly large bribe and more than enough to cover the cost of attacks, such as the one on the 2010 pilot D.C. voting system, as well as others on voters' computers. Halderman said his team's attack would have cost less than $50,000 at generous consulting rates.

Other Countries

We have focused on Internet voting in the U.S., but Internet voting has been used in several other countries, including Estonia and Switzerland, neither of which protects against malware on voters' computers, and Norway in 2011.[c] The Netherlands provided an Internet voting option in its 2006 parliamentary elections, but Internet voting was subsequently banned, largely because of work by a group called "We Don't Trust Voting Computers." The U.K. tried Internet voting on a pilot basis in 2007, but the U.K. Electoral Commission recommended against further e-voting pilot projects until a range of issues had been addressed.[40]

Far Future

Systems like Helios[15] and Remotegrity[37] use encryption to allow voters to verify that their ballots were accurately received and counted. Unfortunately, cryptography does not protect Internet-based elections against DDoS attacks, spoofing, coercion, design flaws, and many kinds of ordinary software bugs.[8] Recounts on these cryptographic voting systems cannot recover from such threats. While these systems have been used for some small Internet elections, the consensus in the cryptographic community is that they are not ready for use in a major election. Ben Adida, creator of Helios, wrote in 2011: "The one problem I don't know how to address with Helios is client-side security...We now have documented evidence...that viruses like Stuxnet that corrupt nuclear power plants by spreading from one Windows machine to the other have been built...So if you run a very large-scale election for a president of a G8 country, why wouldn't we see a similar scenario? Certainly, it's worth just as much money; it's worth just as much strategically... All the ability doesn't change the fact that a client-side corruption in my browser can flip my vote even before it's encrypted, and if we... must have a lot of voters verify their process, I think we're going to lose, because most voters don't quite do that yet."[1] Note that while Helios can detect DDoS attacks, network attacks, and several other types of attacks mentioned here, it cannot prevent, diagnose, or fix them.

Perhaps eventually a paperless cryptographic Internet voting system will be developed that is sufficiently secure, accurate, usable, and transparent to be used in major elections. Until then, the conclusion of the National Commission on Federal Election Reform, co-chaired by Presidents Gerald R. Ford and Jimmy Carter in 2001, still stands, that Internet voting "is an idea whose time most certainly has not yet come."[11]

Conclusion

Proposals for conducting voting pilot projects using real elections continue to reappear in the U.S. and elsewhere, apparently independent of warnings from computer-security experts. While the appeal of Internet voting is obvious, the risks are not, at least to many decision makers. Computer profes-

c Norway uses encryption, but malware on a voter's computer is still able to change votes, so long as the change is consistent with the partial proof sent to the voter or the voter does not check the partial proof.

sionals have an obligation to explain these risks.

Pilot projects are routinely declared successes, regardless of any problems encountered. However, it is dangerous to draw conclusions from a "successful" Internet voting pilot project. There is little reason to attack a small pilot project, and a malicious player might refrain from attacking a major election until the new technology is entrenched. Having claimed success, independent of proof of the accuracy of the pilot project, Internet-voting vendors and enthusiasts routinely push to extend Internet voting to a broader group of voters, thereby seriously undermining election security. Computer professionals must object to pilot projects that do not plan for an assessment of the integrity of the election and a public reporting of any discrepancies encountered.

Unlike legitimate computer-security experts, malicious attackers are not likely to publicize their attacks, just as credit-card thieves do not openly advertise their thefts. When election officials and policymakers ask for proof that a voting system has been attacked, it is important to keep in mind that detecting well-devised attacks is inherently difficult. The burden of proof that a voting system has not been attacked should fall on those making the claim, not the other way around.

Ultimately, the balance between the integrity of election technology on the one hand and convenience on the other is both a public-policy and a technological issue. Decision makers must be warned of all the risks in order to craft wise policy.

Acknowledgment

We are grateful to the referees who provided us with excellent recommendations.

References

1. Adida, B. Panelist remarks at panel on Internet voting. *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections* (San Francisco, Aug. 9, 2011); http://www.usenix.org/events/evtwote11/stream/benaloh_panel/index.html
2. Bowden, M. The enemy within. *The Atlantic* (June 2010); http://www.theatlantic.com/magazine/archive/2010/06/the-enemy-within/8098/
3. CBC News. Computer vandal delays leadership vote (Jan. 25, 2003); http://www.cbc.ca/news/story/2003/01/25/ndp_delay030125.html
4. Claburn, T. Microsoft Finds U.S. Leads In Botnets. *InformationWeek* (Oct. 14, 2010); http://www.informationweek.com/security/vulnerabilities/microsoft-finds-us-leads-in-botnets/227800051
5. DeGregorio, P. *UOCAVA Voting Scoping Strategy.* Washington Secretary of State Public Record, Jan. 18, 2009; http://www.votersunite.org/info/WA-PRR-ScopingStrategy.pdf
6. District of Columbia and Halderman, J.A. Thank you to voters (hacked ballot acknowledgment with Michigan fight song); https://jhalderm.com/pub/dc/thanks/
7. Dunn, J.E. Trojan attacks credit cards of 15 U.S. banks. *Techworld* (July 14, 2010).
8. Estehghari, S. and Desmedt, Y. Exploiting the client vulnerabilities in Internet e-voting systems: Hacking Helios 2.0 as an example. *2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections* (Washington D.C., Aug. 9, 2010); http://static.usenix.org/events/evtwote10/tech/full_papers/Estehghari.pdf
9. FBI. *FBI, Slovenian and Spanish Police Arrest Mariposo Botnet Creator, Operators.* Press Release, July 28, 2010; http://www.fbi.gov/news/pressrel/press-releases/fbi-slovenian-and-spanish-police-arrest-maripora-botnet-creator-operators/
10. Fisher, D. New Service helps attackers get Zeus botnet off the ground. *Threatpost* (Jan. 10, 2011); http://threatpost.com/en_us/blogs/new-service-helps-attackers-get-zeus-botnet-ground-011011
11. Ford, G.R. and Carter, J. *To Assure Pride and Confidence in the Electoral Process.* National Commission on Federal Election Reform, Aug. 2001; http://fl1.findlaw.com/news.findlaw.com/hdocs/docs/election2000/electionreformrpt0801.pdf
12. The H Security. *Antivirus Protection Worse than a Year Ago.* Heise Media, U.K., Dec. 20, 2007; http://www.h-online.com/security/news/item/Antivirus-protection-worse-than-a-year-ago-735697.html
13. Hayden, M. Hackers force Internet users to learn self defense. *PBS NewsHour* (Aug. 11, 2010); http://www.pbs.org/newshour/bb/science/Jul.-dec10/cyber_08-11.html
14. Head, W. Hackers use Wikipedia to spread malware. *IT News for Australian Business* (Nov. 6, 2006); http://www.itnews.com.au/News/67796,hackers-use-wikipedia-to-spread-malware.aspx
15. Helios. http://heliosvoting.org/
16. InfoSecurity. McCartney site serves up Zeus malware. *InfoSecurity* (Apr. 8, 2009); http://www.infosecurity-us.com/view/1178/mccartney-site-serves-up-zeus-malware/
17. Jefferson D. Email voting: A national security threat in government elections. VerifiedVoting blog (June 2011); http:/blog.verifiedvoting.org/2011/06/20/1375
18. Jefferson, D., Rubin, A.B., Simons, B., and Wagner, D. A *Security Analysis of the Secure Electronic Registration and Voting Experiment (SERVE)*, Jan. 20, 2004; http://servesecurityreport.org/
19. Kanan, K., Rees, J., and Spafford, E. *Unsecured Economies: Protecting Vital Information.* Technical Report. McAfee, Inc., Santa Clara, CA, Feb. 2009; resources.mcafee.com/content/NAUnsecuredEconomiesReport
20. Keizer, G. Zeus botnet gang targets Charles Schwab accounts. *Computerworld* (Oct. 16, 2010); http://www.computerworld.com/s/article/9191479/Zeus_botnet_gang_targets_Charles_Schwab_accounts
21. Kirk, J. Comodo hacker claims credit for DigiNotar attack. *Computerworld* (Sept. 2011); http://www.computerworld.com/s/article/9219739/Comodo_hacker_claims_credit_for_DigiNotar_attack
22. KITV. Voting drops 83 percent in all-digital election. Honolulu, May 2009; http://www.kitv.com/politics/19573770/detail.html
23. Kurtz, G. Operation 'Aurora' hit Google, others. McAfee Security Insights blog, Jan. 10, 2010; http://blogs.mcafee.com/corporate/cto/operation-aurora-hit-google-others
24. Leyden, J. UK cybercops cuff 19 Zeus banking trojan suspects. *The Register* (Sept. 29, 2010); www.theregister.co.uk/2010/09/29/zeus_cybercrime_arrests/
25. M86 Security. *M86 Security Labs Discovers Customers of Global Financial Institution Hit by Cybercrime.* Press Release, London, U.K., Aug. 10, 2010; http://www.marketwire.com/press-release/m86-security-labs-discovers-customers-global-financial-institution-hit-cybercrime-1302266.htm
26. Mahoney, M.R. *Comment on Pilot Project Testing and Certification.* EAC, Washington, D.C., Apr. 2010; http://www.eac.gov/assets/1/AssetManager/Martha%20Mahoney%20-%20Comment%20on%20Pilot%20Project%20Testing%20and%20Certification.pdf
27. Marsan, C.D. Feds to shore up net security. *Network World* (Jan. 19, 2009); http://www.pcworld.com/businesscenter/article/157909/feds_to_shore_up_net_security.html
28. Messmer, E. America's 10 most wanted botnets. *Network World* (July 22, 2009); http://www.networkworld.com/news/2009/072209-botnets.html
29. Military Postal Service Agency. *2010 Analysis of the Military Postal System Compliance with the MOVE Act.* Washington, D.C., Aug. 2, 2011; www.fvap.gov/resources/media/2010_MPSA_after_action_report.pdf
30. Mills, E. Spam offers to let people use their PC to attack Obama site. *CNET* (Aug. 18, 2009); http://news.cnet.com/8301-1009_3-10312641-83.html?tag=nl.e757
31. Mueller III, R.S. *Prepared Remarks.* RSA Security Conference, San Francisco, Mar. 4, 2010; http://www.fbi.gov/news/speeches/tackling-the-cyber-threat
32. Nakashima, E., Miller, G., and Tate, J. U.S., Israel developed computer virus to slow Iranian nuclear efforts, officials say. *Washington Post* (June 19, 2012); http://www.washingtonpost.com/world/national-security/us-israel-developed-computer-virus-to-slow-iranian-nuclear-efforts-officialssay/2012/06/19/gJQA6xBPoV_story.html?wpisrc=al_national
33. New South Wales Electoral Commission. *Report on the Conduct of the NSW State Election 2011;* http://www.parliament.nsw.gov.au/Prod/parlment/committee.nsf/0/67f2055c4d085409ca25795a0017cf2c/$FILE/NSW%20EC%27s%20Report%20on%20the%202011%20State%20Election.pdf
34. Oberheide, J., Cooke, E., and Jahanian, F. CloudAV: N-version antivirus in the network cloud. In *Proceedings of the 17th USENIX Security Symposium* (San Jose, CA, July 28–Aug. 1, 2008), 91–106.
35. Palmer, C. *Unqualified Names in SSL Observatory.* Electronic Frontier Foundation Deeplinks blog, Apr. 5, 2011; https://www.eff.org/deeplinks/2011/04/unqualified-names-ssl-observatory
36. Peters, S. *14th Annual CSI Computer Crime and Security Survey, Executive Summary.* Computer Security Institute, New York, Dec. 2009; http://www.docstoc.com/docs/40697141
37. Remotegrity. 2011; https://demo.remotegrity.org/http://www.scantegrity.org/wiki/index.php/Remotegrity_Frequently_Asked_Questions
38. Sanger, D.E. Obama order sped up wave of cyberattacks against Iran. *New York Times* (June 1, 2012); http://www.nytimes.com/2012/06/01/world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html?pagewanted=all
39. Trusteer Inc. *Measuring the In-the-Wild Effectiveness of Antivirus Against Zeus.* White Paper, Sept. 14, 2009; http://www.techrepublic.com/whitepapers/measuring-the-in-the-wild-effectiveness-of-antivirus-against-zeus/1686945/post
40. U.K. *Electoral Commission. Key Issues and Conclusions, May 2007 Electoral Pilot Schemes.* London, Aug. 2007; http://www.electoralcommission.org.uk/__data/assets/electoral_commission_pdf_file/0009/16200/ICMElectoralPilotsresearchreport_27285-20161__E__N__S__W__.pdf
41. Vascellaro, J.E. and Solomon, J. Yahoo! was also targeted in hacker attack. *Wall Street Journal* (Jan. 14, 2010); http://online.wsj.com/article/SB10001424052748703657604575004421409691754.html
42. Verified Voting Foundation. *Internet Voting 2012*; http://www.verifiedvotingfoundation.org/article.php?list=type&type=27
43. Weiss, T.R. Pentagon drops online votes for armed forces. *Computer Weekly* (Feb. 6, 2004); http://www.computerweekly.com/news/2240054464/Pentagon-drops-online-votes-for-armed-forces
44. Wolchok, S., Wustrow, E. Isabel, D., and Halderman, J.A. Attacking the Washington, D.C. Internet voting system. In *Proceedings of the 16th Conference on Financial Cryptography and Data Security* (Bonaire, Feb. 28. 2012); http://fc12.ifca.ai/pre-proceedings/paper_79.pdf
45. Zetter, K. Vulnerabilities allow attacker to impersonate any website. *Wired.com* (July 29, 2009); http://www.wired.com/threatlevel/2009/07/kaminsky/

Barbara Simons (simons@acm.org) is a retired IBM Research staff member, Board Chair of Verified Voting, and former ACM President.

Douglas W. Jones (jones@cs.uiowa.edu) is an associate professor in the Department of Computer Science of the University of Iowa in Iowa City.

DOI:10.1145/2347736.2347755

Tapping into the "folk knowledge" needed to advance machine learning applications.

BY PEDRO DOMINGOS

A Few Useful Things to Know About Machine Learning

MACHINE LEARNING SYSTEMS automatically learn programs from data. This is often a very attractive alternative to manually constructing them, and in the last decade the use of machine learning has spread rapidly throughout computer science and beyond. Machine learning is used in Web search, spam filters, recommender systems, ad placement, credit scoring, fraud detection, stock trading, drug design, and many other applications. A recent report from the McKinsey Global Institute asserts that machine learning (a.k.a. data mining or predictive analytics) will be the driver of the next big wave of innovation.[15] Several fine textbooks are available to interested practitioners and researchers (for example, Mitchell[16] and Witten et al.[24]). However, much of the "folk knowledge" that is needed to successfully develop machine learning applications is not readily available in them. As a result, many machine learning projects take much longer than necessary or wind up producing less-than-ideal results. Yet much of this folk knowledge is fairly easy to communicate. This is the purpose of this article.

» key insights

- **Machine learning algorithms can figure out how to perform important tasks by generalizing from examples. This is often feasible and cost-effective where manual programming is not. As more data becomes available, more ambitious problems can be tackled.**
- **Machine learning is widely used in computer science and other fields. However, developing successful machine learning applications requires a substantial amount of "black art" that is difficult to find in textbooks.**
- **This article summarizes 12 key lessons that machine learning researchers and practitioners have learned. These include pitfalls to avoid, important issues to focus on, and answers to common questions.**

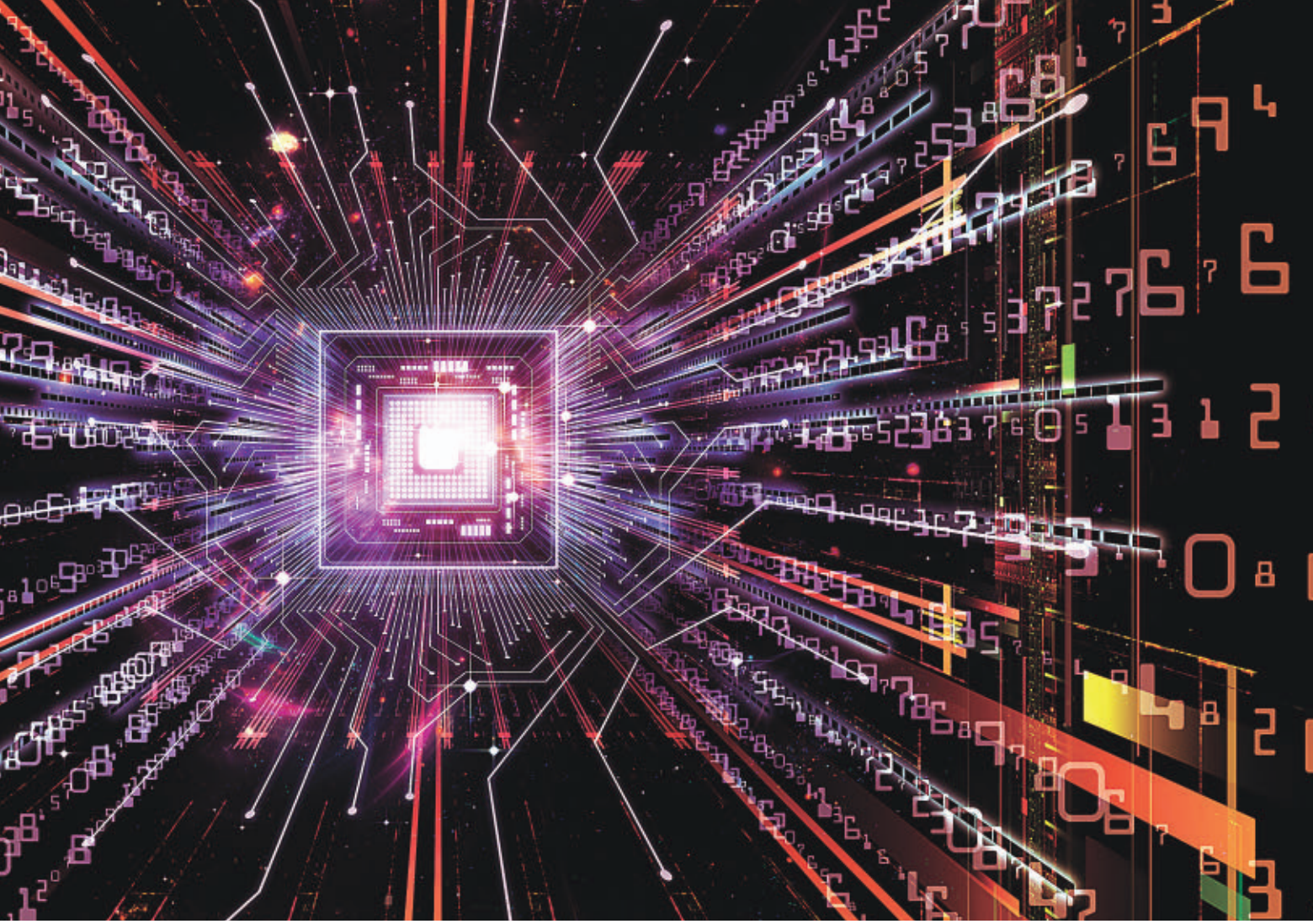


IMAGE BY AGSANDREW/SHUTTERSTOCK.COM

Many different types of machine learning exist, but for illustration purposes I will focus on the most mature and widely used one: classification. Nevertheless, the issues I will discuss apply across all of machine learning. A *classifier* is a system that inputs (typically) a vector of discrete and/or continuous *feature values* and outputs a single discrete value, the *class*. For example, a spam filter classifies email messages into "spam" or "not spam," and its input may be a Boolean vector $\mathbf{x} = (x_1,\ldots,x_j,\ldots,x_d)$, where $x_j = 1$ if the j^{th} word in the dictionary appears in the email and $x_j = 0$ otherwise. A *learner* inputs a training set of examples $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ is an observed input and y_i is the corresponding output, and outputs a classifier. The test of the learner is whether this classifier produces the correct output y_t for future examples $\mathbf{x}_t$ (for example, whether the spam filter correctly classifies previously unseen email messages as spam or not spam).

Learning = Representation + Evaluation + Optimization

Suppose you have an application that you think machine learning might be good for. The first problem facing you is the bewildering variety of learning algorithms available. Which one to use? There are literally thousands available, and hundreds more are published each year. The key to not getting lost in this huge space is to realize that it consists of combinations of just three components. The components are:

▸ **Representation.** A classifier must be represented in some formal language that the computer can handle. Conversely, choosing a representation for a learner is tantamount to choosing the set of classifiers that it can possibly learn. This set is called the *hypothesis space* of the learner. If a classifier is not in the hypothesis space, it cannot be learned. A related question, that I address later, is how to represent the input, in other words, what features to use.

▸ **Evaluation.** An evaluation function (also called *objective function* or *scoring function*) is needed to distinguish good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize, for ease of optimization and due to the issues I will discuss.

▸ **Optimization.** Finally, we need a method to search among the classifiers in the language for the highest-scoring one. The choice of optimization technique is key to the efficiency of the learner, and also helps determine the classifier produced if the evaluation function has more than one optimum. It is common for new learners to start out using off-the-shelf optimizers, which are later replaced by custom-designed ones.

The accompanying table shows common examples of each of these three components. For example, *k*-nearest neighbor classifies a test example by finding the *k* most similar training examples and predicting the majority class among them. Hyperplane-based methods form a linear

Table 1. The three components of learning algorithms.

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

Algorithm 1. Decision tree induction.

LearnDT (*TrainSet*)

if all examples in *TrainSet* have the same class y_* **then**
  return MakeLeaf(y_*)
if no feature x_j has InfoGain(x_j ,y) > 0 **then**
  $y_* \leftarrow$ Most frequent class in *TrainSet*
  return MakeLeaf(y_*)
$x_* \leftarrow \text{argmax}_{x_j}$ InfoGain(x_j, y)
$TS_0 \leftarrow$ Examples in *TrainSet* with $x_* = 0$
$TS_1 \leftarrow$ Examples in *TrainSet* with $x_* = 1$
return MakeNode(x_*, LearnDT(TS_0), LearnDT(TS_1))

combination of the features per class and predict the class with the highest-valued combination. Decision trees test one feature at each internal node, with one branch for each feature value, and have class predictions at the leaves. Algorithm 1 (above) shows a bare-bones decision tree learner for Boolean domains, using information gain and greedy search.[20] InfoGain(x_j, y) is the mutual information between feature x_j and the class y. MakeNode(x,c_0,c_1) returns a node that tests feature x and has c_0 as the child for $x = 0$ and c_1 as the child for $x = 1$.

Of course, not all combinations of one component from each column of the table make equal sense. For example, discrete representations naturally go with combinatorial optimization, and continuous ones with continuous optimization. Nevertheless, many learners have both discrete and continuous components, and in fact the day may not be far when every single possible combination has appeared in some learner!

Most textbooks are organized by representation, and it is easy to overlook the fact that the other components are equally important. There is no simple recipe for choosing each component, but I will touch on some of the key issues here. As we will see, some choices in a machine learning project may be even more important than the choice of learner.

It's Generalization that Counts

The fundamental goal of machine learning is to generalize beyond the examples in the training set. This is because, no matter how much data we have, it is very unlikely that we will see those exact examples again at test time. (Notice that, if there are 100,000 words in the dictionary, the spam filter described above has $2^{100,000}$ possible different inputs.) Doing well on the training set is easy (just memorize the examples). The most common mistake among machine learning beginners is to test on the training data and have the illusion of success. If the chosen classifier is then tested on new data, it is often no better than random guessing. So, if you hire someone to build a classifier, be sure to keep some of the data to yourself and test the classifier they give you on it. Conversely, if you have been hired to build a classifier, set some of the data aside from the beginning, and only use it to test your chosen classifier at the very end, followed by learning your final classifier on the whole data.

Contamination of your classifier by test data can occur in insidious ways, for example, if you use test data to tune parameters and do a lot of tuning. (Machine learning algorithms have lots of knobs, and success often comes from twiddling them a lot, so this is a real concern.) Of course, holding out data reduces the amount available for training. This can be mitigated by doing cross-validation: randomly dividing your training data into (say) 10 subsets, holding out each one while training on the rest, testing each learned classifier on the examples it did not see, and averaging the results to see how well the particular parameter setting does.

In the early days of machine learning, the need to keep training and test data separate was not widely appreciated. This was partly because, if the learner has a very limited representation (for example, hyperplanes), the difference between training and test error may not be large. But with very flexible classifiers (for example, decision trees), or even with linear classifiers with a lot of features, strict separation is mandatory.

Notice that generalization being the goal has an interesting consequence for machine learning. Unlike in most other optimization problems, we do not have access to the function we want to optimize! We have to use training error as a surrogate for test error, and this is fraught with danger. (How to deal with it is addressed later.) On the positive side, since the objective function is only a proxy for the true goal, we may not need to fully

optimize it; in fact, a local optimum returned by simple greedy search may be better than the global optimum.

Data Alone Is Not Enough

Generalization being the goal has another major consequence: Data alone is not enough, no matter how much of it you have. Consider learning a Boolean function of (say) 100 variables from a million examples. There are $2^{100} - 10^6$ examples whose classes you do not know. How do you figure out what those classes are? In the absence of further information, there is just no way to do this that beats flipping a coin. This observation was first made (in somewhat different form) by the philosopher David Hume over 200 years ago, but even today many mistakes in machine learning stem from failing to appreciate it. Every learner must embody some knowledge or assumptions beyond the data it is given in order to generalize beyond it. This notion was formalized by Wolpert in his famous "no free lunch" theorems, according to which no learner can beat random guessing over all possible functions to be learned.[25]

This seems like rather depressing news. How then can we ever hope to learn anything? Luckily, the functions we want to learn in the real world are *not* drawn uniformly from the set of all mathematically possible functions! In fact, very general assumptions—like smoothness, similar examples having similar classes, limited dependences, or limited complexity—are often enough to do very well, and this is a large part of why machine learning has been so successful. Like deduction, induction (what learners do) is a knowledge lever: it turns a small amount of input knowledge into a large amount of output knowledge. Induction is a vastly more powerful lever than deduction, requiring much less input knowledge to produce useful results, but it still needs more than zero input knowledge to work. And, as with any lever, the more we put in, the more we can get out.

A corollary of this is that one of the key criteria for choosing a representation is which kinds of knowledge are easily expressed in it. For example, if we have a lot of knowledge about what makes examples similar in our domain, instance-based methods may be a good choice. If we have knowledge about probabilistic dependencies, graphical models are a good fit. And if we have knowledge about what kinds of preconditions are required by each class, "IF . . . THEN . . ." rules may be the best option. The most useful learners in this regard are those that do not just have assumptions hardwired into them, but allow us to state them explicitly, vary them widely, and incorporate them automatically into the learning (for example, using first-order logic[21] or grammars[6]).

In retrospect, the need for knowledge in learning should not be surprising. Machine learning is not magic; it cannot get something from nothing. What it does is get more from less. Programming, like all engineering, is a lot of work: we have to build everything from scratch. Learning is more like farming, which lets nature do most of the work. Farmers combine seeds with nutrients to grow crops. Learners combine knowledge with data to grow programs.

Overfitting Has Many Faces

What if the knowledge and data we have are not sufficient to completely determine the correct classifier? Then we run the risk of just hallucinating a classifier (or parts of it) that is not grounded in reality, and is simply encoding random quirks in the data. This problem is called *overfitting*, and is the bugbear of machine learning. When your learner outputs a classifier that is 100% accurate on the training data but only 50% accurate on test data, when in fact it could have output one that is 75% accurate on both, it has overfit.

Everyone in machine learning knows about overfitting, but it comes in many forms that are not immediately obvious. One way to understand overfitting is by decomposing generalization error into *bias* and *variance*.[9] Bias is a learner's tendency to consistently learn the same wrong thing. Variance is the tendency to learn random things irrespective of the real signal. Figure 1 illustrates this by an analogy with throwing darts at a board. A linear learner has high bias, because when the frontier between two classes is not a hyperplane the learner is unable to induce it. Decision trees do not have this problem because they can represent any Boolean function, but on the other hand they can suffer from high variance: decision trees learned on different training sets generated by the same phenomenon are often very different, when in fact they should be

Figure 1. Bias and variance in dart-throwing.

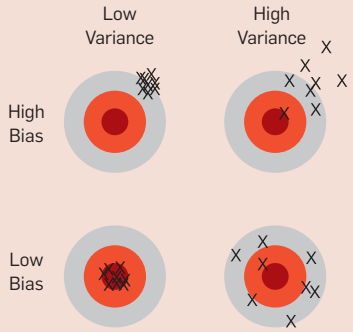
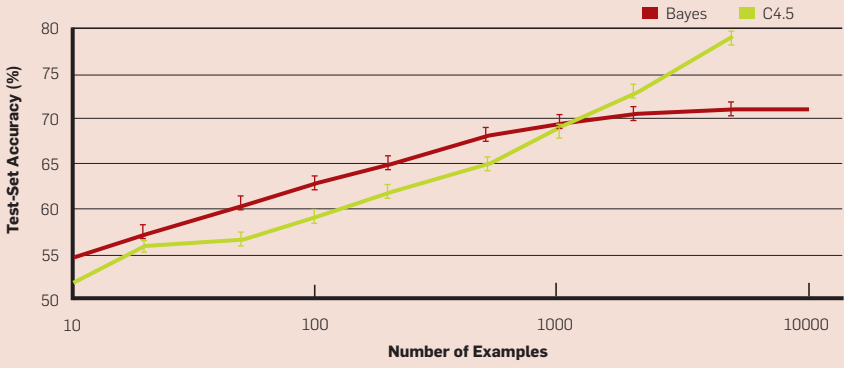


Figure 2. Naïve Bayes can outperform a state-of-the-art rule learner (C4.5rules) even when the true classifier is a set of rules.



the same. Similar reasoning applies to the choice of optimization method: beam search has lower bias than greedy search, but higher variance, because it tries more hypotheses. Thus, contrary to intuition, a more powerful learner is not necessarily better than a less powerful one.

Figure 2 illustrates this.[a] Even though the true classifier is a set of rules, with up to 1,000 examples naive Bayes is more accurate than a rule learner. This happens despite naive Bayes's false assumption that the frontier is linear! Situations like this are common in machine learning: strong false assumptions can be better than weak true ones, because a learner with the latter needs more data to avoid overfitting.

Cross-validation can help to combat overfitting, for example by using it to choose the best size of decision tree to learn. But it is no panacea, since if we use it to make too many parameter choices it can itself start to overfit.[17]

Besides cross-validation, there are many methods to combat overfitting. The most popular one is adding a *regularization term* to the evaluation function. This can, for example, penalize classifiers with more structure, thereby favoring smaller ones with less room to overfit. Another option is to perform a statistical significance test like chi-square before adding new structure, to decide whether the distribution of the class really is different with and without this structure. These techniques are particularly useful when data is very scarce. Nevertheless, you should be skeptical of claims that a particular technique "solves" the overfitting problem. It is easy to avoid overfitting (variance) by falling into the opposite error of underfitting (bias). Simultaneously avoiding both requires learning a perfect classifier, and short of knowing it in advance there is no single technique that will always do best (no free lunch).

A common misconception about overfitting is that it is caused by noise, like training examples labeled with the wrong class. This can indeed aggravate overfitting, by making the learner draw a capricious frontier to keep those examples on what it thinks is the right side. But severe overfitting can occur even in the absence of noise. For instance, suppose we learn a Boolean classifier that is just the disjunction of the examples labeled "true" in the training set. (In other words, the classifier is a Boolean formula in disjunctive normal form, where each term is the conjunction of the feature values of one specific training example.) This classifier gets all the training examples right and every positive test example wrong, regardless of whether the training data is noisy or not.

The problem of *multiple testing*[13] is closely related to overfitting. Standard statistical tests assume that only one hypothesis is being tested, but modern learners can easily test millions before they are done. As a result what looks significant may in fact not be. For example, a mutual fund that beats the market 10 years in a row looks very impressive, until you realize that, if there are 1,000 funds and each has a 50% chance of beating the market on any given year, it is quite likely that one will succeed all 10 times just by luck. This problem can be combatted by correcting the significance tests to take the number of hypotheses into account, but this can also lead to underfitting. A better approach is to control the fraction of falsely accepted non-null hypotheses, known as the *false discovery rate*.[3]

Intuition Fails in High Dimensions

After overfitting, the biggest problem in machine learning is the *curse of dimensionality*. This expression was coined by Bellman in 1961 to refer to the fact that many algorithms that work fine in low dimensions become intractable when the input is high-dimensional. But in machine learning it refers to much more. Generalizing correctly becomes exponentially harder as the dimensionality (number of features) of the examples grows, because a fixed-size training set covers a dwindling fraction of the input space. Even with a moderate dimension of 100 and a huge training set of a trillion examples, the latter covers only a fraction of about 10^{-18} of the input space. This is what makes machine learning both necessary and hard.

More seriously, the similarity-based reasoning that machine learning algorithms depend on (explicitly or implicitly) breaks down in high dimensions. Consider a nearest neighbor classifier with Hamming distance as the similarity measure, and suppose the class is just $x_1 \wedge x_2$. If there are no other features, this is an easy problem. But if there are 98 irrelevant features $x_3,\ldots, x_{100}$, the noise from them completely swamps the signal in x_1 and x_2, and nearest neighbor effectively makes random predictions.

Even more disturbing is that nearest neighbor still has a problem even if all 100 features are relevant! This is because in high dimensions all examples look alike. Suppose, for instance, that examples are laid out on a regular grid, and consider a test example x_t. If the grid is d-dimensional, x_t's $2d$ nearest examples are all at the same distance from it. So as the dimensionality increases, more and more examples become nearest neighbors of x_t, until the choice of nearest neighbor (and therefore of class) is effectively random.

This is only one instance of a more general problem with high dimensions: our intuitions, which come from a three-dimensional world, often do not apply in high-dimensional ones. In high dimensions, most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant "shell" around it; and most of the volume of a high-dimensional orange is in the skin, not the pulp. If a constant number of examples is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most examples are closer to a face of the hypercube than to their nearest neighbor. And if we approximate a hypersphere by inscribing it in a hypercube, in high dimensions almost all the volume of the hypercube is outside the hypersphere. This is bad news for machine learning, where shapes of one type are often approximated by shapes of another.

Building a classifier in two or three dimensions is easy; we can find a reasonable frontier between examples of different classes just by visual in-

a Training examples consist of 64 Boolean features and a Boolean class computed from them according to a set of "IF . . . THEN . . ." rules. The curves are the average of 100 runs with different randomly generated sets of rules. Error bars are two standard deviations. See Domingos and Pazzani[10] for details.

spection. (It has even been said that if people could see in high dimensions machine learning would not be necessary.) But in high dimensions it is difficult to understand what is happening. This in turn makes it difficult to design a good classifier. Naively, one might think that gathering more features never hurts, since at worst they provide no new information about the class. But in fact their benefits may be outweighed by the curse of dimensionality.

Fortunately, there is an effect that partly counteracts the curse, which might be called the "blessing of nonuniformity." In most applications examples are not spread uniformly throughout the instance space, but are concentrated on or near a lower-dimensional manifold. For example, *k*-nearest neighbor works quite well for handwritten digit recognition even though images of digits have one dimension per pixel, because the space of digit images is much smaller than the space of all possible images. Learners can implicitly take advantage of this lower effective dimension, or algorithms for explicitly reducing the dimensionality can be used (for example, Tenenbaum[22]).

Theoretical Guarantees Are Not What They Seem

Machine learning papers are full of theoretical guarantees. The most common type is a bound on the number of examples needed to ensure good generalization. What should you make of these guarantees? First of all, it is remarkable that they are even possible. Induction is traditionally contrasted with deduction: in deduction you can guarantee that the conclusions are correct; in induction all bets are off. Or such was the conventional wisdom for many centuries. One of the major developments of recent decades has been the realization that in fact we can have guarantees on the results of induction, particularly if we are willing to settle for probabilistic guarantees.

The basic argument is remarkably simple.[5] Let's say a classifier is bad if its true error rate is greater than ϵ. Then the probability that a bad classifier is consistent with n random, independent training examples is less than $(1 - \epsilon)^n$. Let b be the number of bad classifiers in the learner's hypothesis space H. The probability that at least one of them is consistent is less than $b(1 - \epsilon)^n$, by the union bound. Assuming the learner always returns a consistent classifier, the probability that this classifier is bad is then less than $|H|(1 - \epsilon)^n$, where we have used the fact that $b \leq |H|$. So if we want this probability to be less than δ, it suffices to make $n > \ln(\delta/|H|)/\ln(1 - \epsilon) \geq 1/\epsilon\,(\ln |H| + \ln 1/\delta)$.

One of the major developments of recent decades has been the realization that we can have guarantees on the results of induction, particularly if we are willing to settle for probabilistic guarantees.

Unfortunately, guarantees of this type have to be taken with a large grain of salt. This is because the bounds obtained in this way are usually extremely loose. The wonderful feature of the bound above is that the required number of examples only grows logarithmically with $|H|$ and $1/\delta$. Unfortunately, most interesting hypothesis spaces are *doubly* exponential in the number of features d, which still leaves us needing a number of examples exponential in d. For example, consider the space of Boolean functions of d Boolean variables. If there are e possible different examples, there are 2^e possible different functions, so since there are 2^d possible examples, the total number of functions is 2^{2^d}. And even for hypothesis spaces that are "merely" exponential, the bound is still very loose, because the union bound is very pessimistic. For example, if there are 100 Boolean features and the hypothesis space is decision trees with up to 10 levels, to guarantee $\delta = \epsilon = 1\%$ in the bound above we need half a million examples. But in practice a small fraction of this suffices for accurate learning.

Further, we have to be careful about what a bound like this means. For instance, it does not say that, if your learner returned a hypothesis consistent with a particular training set, then this hypothesis probably generalizes well. What it says is that, given a large enough training set, with high probability your learner will either return a hypothesis that generalizes well or be unable to find a consistent hypothesis. The bound also says nothing about how to select a good hypothesis space. It only tells us that, if the hypothesis space contains the true classifier, then the probability that the learner outputs a bad classifier decreases with training set size.

If we shrink the hypothesis space, the bound improves, but the chances that it contains the true classifier shrink also. (There are bounds for the case where the true classifier is not in the hypothesis space, but similar considerations apply to them.)

Another common type of theoretical guarantee is asymptotic: given infinite data, the learner is guaranteed to output the correct classifier. This is reassuring, but it would be rash to choose one learner over another because of its asymptotic guarantees. In practice, we are seldom in the asymptotic regime (also known as "asymptopia"). And, because of the bias-variance trade-off I discussed earlier, if learner A is better than learner B given infinite data, B is often better than A given finite data.

The main role of theoretical guarantees in machine learning is not as a criterion for practical decisions, but as a source of understanding and driving force for algorithm design. In this capacity, they are quite useful; indeed, the close interplay of theory and practice is one of the main reasons machine learning has made so much progress over the years. But caveat emptor: learning is a complex phenomenon, and just because a learner has a theoretical justification and works in practice does not mean the former is the reason for the latter.

Feature Engineering Is The Key

At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used. Learning is easy if you have many independent features that each correlate well with the class. On the other hand, if the class is a very complex function of the features, you may not be able to learn it. Often, the raw data is not in a form that is amenable to learning, but you can construct features from it that are. This is typically where most of the effort in a machine learning project goes. It is often also one of the most interesting parts, where intuition, creativity and "black art" are as important as the technical stuff.

First-timers are often surprised by how little time in a machine learning project is spent actually doing machine learning. But it makes sense if you consider how time-consuming it is to gather data, integrate it, clean it and preprocess it, and how much trial and error can go into feature design. Also, machine learning is not a one-shot process of building a dataset and running a learner, but rather an iterative process of running the learner, analyzing the results, modifying the data and/or the learner, and repeating. Learning is often the quickest part of this, but that is because we have already mastered it pretty well! Feature engineering is more difficult because it is domain-specific, while learners can be largely general purpose. However, there is no sharp frontier between the two, and this is another reason the most useful learners are those that facilitate incorporating knowledge.

A dumb algorithm with lots and lots of data beats a clever one with modest amounts of it.

Of course, one of the holy grails of machine learning is to automate more and more of the feature engineering process. One way this is often done today is by automatically generating large numbers of candidate features and selecting the best by (say) their information gain with respect to the class. But bear in mind that features that look irrelevant in isolation may be relevant in combination. For example, if the class is an XOR of k input features, each of them by itself carries no information about the class. (If you want to annoy machine learners, bring up XOR.) On the other hand, running a learner with a very large number of features to find out which ones are useful in combination may be too time-consuming, or cause overfitting. So there is ultimately no replacement for the smarts you put into feature engineering.

More Data Beats a Cleverer Algorithm

Suppose you have constructed the best set of features you can, but the classifiers you receive are still not accurate enough. What can you do now? There are two main choices: design a better learning algorithm, or gather more data (more examples, and possibly more raw features, subject to the curse of dimensionality). Machine learning researchers are mainly concerned with the former, but pragmatically the quickest path to success is

often to just get more data. As a rule of thumb, a dumb algorithm with lots and lots of data beats a clever one with modest amounts of it. (After all, machine learning is all about letting data do the heavy lifting.)

This does bring up another problem, however: scalability. In most of computer science, the two main limited resources are time and memory. In machine learning, there is a third one: training data. Which one is the bottleneck has changed from decade to decade. In the 1980s it tended to be data. Today it is often time. Enormous mountains of data are available, but there is not enough time to process it, so it goes unused. This leads to a paradox: even though in principle more data means that more complex classifiers can be learned, in practice simpler classifiers wind up being used, because complex ones take too long to learn. Part of the answer is to come up with fast ways to learn complex classifiers, and indeed there has been remarkable progress in this direction (for example, Hulten and Domingos[11]).

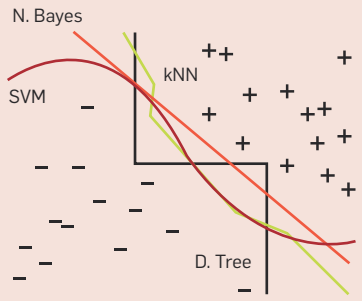
Part of the reason using cleverer algorithms has a smaller payoff than you might expect is that, to a first approximation, they all do the same. This is surprising when you consider representations as different as, say, sets of rules and neural networks. But in fact propositional rules are readily encoded as neural networks, and similar relationships hold between other representations. All learners essentially work by grouping nearby examples into the same class; the key difference is in the meaning of "nearby." With nonuniformly distributed data, learners can produce widely different frontiers while still making the same predictions in the regions that matter (those with a substantial number of training examples, and therefore also where most test examples are likely to appear). This also helps explain why powerful learners can be unstable but still accurate. Figure 3 illustrates this in 2D; the effect is much stronger in high dimensions.

As a rule, it pays to try the simplest learners first (for example, naïve Bayes before logistic regression, *k*-nearest neighbor before support vector machines). More sophisticated learners are seductive, but they are usually harder to use, because they have more knobs you need to turn to get good results, and because their internals are more opaque.

Learners can be divided into two major types: those whose representation has a fixed size, like linear classifiers, and those whose representation can grow with the data, like decision trees. (The latter are sometimes called nonparametric learners, but this is somewhat unfortunate, since they usually wind up learning many more parameters than parametric ones.) Fixed-size learners can only take advantage of so much data. (Notice how the accuracy of naive Bayes asymptotes at around 70% in Figure 2.) Variable-size learners can in principle learn any function given sufficient data, but in practice they may not, because of limitations of the algorithm (for example, greedy search falls into local optima) or computational cost. Also, because of the curse of dimensionality, no existing amount of data may be enough. For these reasons, clever algorithms—those that make the most of the data and computing resources available—often pay off in the end, provided you are willing to put in the effort. There is no sharp frontier between designing learners and learning classifiers; rather, any given piece of knowledge could be encoded in the learner or learned from data. So machine learning projects often wind up having a significant component of learner design, and practitioners need to have some expertise in it.[12]

In the end, the biggest bottleneck is not data or CPU cycles, but human

Figure 3. Very different frontiers can yield similar predictions. (+ and – are training examples of two classes.)



cycles. In research papers, learners are typically compared on measures of accuracy and computational cost. But human effort saved and insight gained, although harder to measure, are often more important. This favors learners that produce human-understandable output (for example, rule sets). And the organizations that make the most of machine learning are those that have in place an infrastructure that makes experimenting with many different learners, data sources, and learning problems easy and efficient, and where there is a close collaboration between machine learning experts and application domain ones.

Learn Many Models, Not Just One

In the early days of machine learning, everyone had a favorite learner, together with some a priori reasons to believe in its superiority. Most effort went into trying many variations of it and selecting the best one. Then systematic empirical comparisons showed that the best learner varies from application to application, and systems containing many different learners started to appear. Effort now went into trying many variations of many learners, and still selecting just the best one. But then researchers noticed that, if instead of selecting the best variation found, we combine many variations, the results are better—often much better—and at little extra effort for the user.

Creating such *model ensembles* is now standard.[1] In the simplest technique, called *bagging*, we simply generate random variations of the training set by resampling, learn a classifier on each, and combine the results by voting. This works because it greatly reduces variance while only slightly increasing bias. In *boosting*, training examples have weights, and these are varied so that each new classifier focuses on the examples the previous ones tended to get wrong. In *stacking*, the outputs of individual classifiers become the inputs of a "higher-level" learner that figures out how best to combine them.

Many other techniques exist, and the trend is toward larger and larger ensembles. In the Netflix prize, teams from all over the world competed to build the best video recommender

system (http://netflixprize.com). As the competition progressed, teams found they obtained the best results by combining their learners with other teams', and merged into larger and larger teams. The winner and runner-up were both stacked ensembles of over 100 learners, and combining the two ensembles further improved the results. Doubtless we will see even larger ones in the future.

Model ensembles should not be confused with Bayesian model averaging (BMA)—the theoretically optimal approach to learning.[4] In BMA, predictions on new examples are made by averaging the individual predictions of *all* classifiers in the hypothesis space, weighted by how well the classifiers explain the training data and how much we believe in them a priori. Despite their superficial similarities, ensembles and BMA are very different. Ensembles change the hypothesis space (for example, from single decision trees to linear combinations of them), and can take a wide variety of forms. BMA assigns weights to the hypotheses in the original space according to a fixed formula. BMA weights are extremely different from those produced by (say) bagging or boosting: the latter are fairly even, while the former are extremely skewed, to the point where the single highest-weight classifier usually dominates, making BMA effectively equivalent to just selecting it.[8] A practical consequence of this is that, while model ensembles are a key part of the machine learning toolkit, BMA is seldom worth the trouble.

Simplicity Does Not Imply Accuracy

Occam's razor famously states that entities should not be multiplied beyond necessity. In machine learning, this is often taken to mean that, given two classifiers with the same training error, the simpler of the two will likely have the lowest test error. Purported proofs of this claim appear regularly in the literature, but in fact there are many counterexamples to it, and the "no free lunch" theorems imply it cannot be true.

We saw one counterexample previously: model ensembles. The generalization error of a boosted ensemble continues to improve by adding classifiers even after the training error has reached zero. Another counterexample is support vector machines, which can effectively have an infinite number of parameters without overfitting. Conversely, the function sign(sin(ax)) can discriminate an arbitrarily large, arbitrarily labeled set of points on the x axis, even though it has only one parameter.[23] Thus, contrary to intuition, there is no necessary connection between the number of parameters of a model and its tendency to overfit.

Just because a function can be represented does not mean it can be learned.

A more sophisticated view instead equates complexity with the size of the hypothesis space, on the basis that smaller spaces allow hypotheses to be represented by shorter codes. Bounds like the one in the section on theoretical guarantees might then be viewed as implying that shorter hypotheses generalize better. This can be further refined by assigning shorter codes to the hypotheses in the space we have some a priori preference for. But viewing this as "proof" of a trade-off between accuracy and simplicity is circular reasoning: we made the hypotheses we prefer simpler by design, and if they are accurate it is because our preferences are accurate, not because the hypotheses are "simple" in the representation we chose.

A further complication arises from the fact that few learners search their hypothesis space exhaustively. A learner with a larger hypothesis space that tries fewer hypotheses from it is less likely to overfit than one that tries more hypotheses from a smaller space. As Pearl[18] points out, the size of the hypothesis space is only a rough guide to what really matters for relating training and test error: the procedure by which a hypothesis is chosen.

Domingos[7] surveys the main arguments and evidence on the issue of Occam's razor in machine learning. The conclusion is that simpler hypotheses should be preferred because simplicity is a virtue in its own right, not because of a hypothetical connection with accuracy. This is probably what Occam meant in the first place.

Representable Does Not Imply Learnable

Essentially all representations used in variable-size learners have associated

theorems of the form "Every function can be represented, or approximated arbitrarily closely, using this representation." Reassured by this, fans of the representation often proceed to ignore all others. However, just because a function can be represented does not mean it can be learned. For example, standard decision tree learners cannot learn trees with more leaves than there are training examples. In continuous spaces, representing even simple functions using a fixed set of primitives often requires an infinite number of components. Further, if the hypothesis space has many local optima of the evaluation function, as is often the case, the learner may not find the true function even if it is representable. Given finite data, time and memory, standard learners can learn only a tiny subset of all possible functions, and these subsets are different for learners with different representations. Therefore the key question is not "Can it be represented?" to which the answer is often trivial, but "Can it be learned?" And it pays to try different learners (and possibly combine them).

Some representations are exponentially more compact than others for some functions. As a result, they may also require exponentially less data to learn those functions. Many learners work by forming linear combinations of simple basis functions. For example, support vector machines form combinations of kernels centered at some of the training examples (the support vectors). Representing parity of n bits in this way requires 2^n basis functions. But using a representation with more layers (that is, more steps between input and output), parity can be encoded in a linear-size classifier. Finding methods to learn these deeper representations is one of the major research frontiers in machine learning.[2]

Correlation Does Not Imply Causation

The point that correlation does not imply causation is made so often that it is perhaps not worth belaboring. But, even though learners of the kind we have been discussing can only learn correlations, their results are often treated as representing causal relations. Isn't this wrong? If so, then why do people do it?

More often than not, the goal of learning predictive models is to use them as guides to action. If we find that beer and diapers are often bought together at the supermarket, then perhaps putting beer next to the diaper section will increase sales. (This is a famous example in the world of data mining.) But short of actually doing the experiment it is difficult to tell. Machine learning is usually applied to *observational* data, where the predictive variables are not under the control of the learner, as opposed to *experimental* data, where they are. Some learning algorithms can potentially extract causal information from observational data, but their applicability is rather restricted.[19] On the other hand, correlation is a sign of a potential causal connection, and we can use it as a guide to further investigation (for example, trying to understand what the causal chain might be).

Many researchers believe that causality is only a convenient fiction. For example, there is no notion of causality in physical laws. Whether or not causality really exists is a deep philosophical question with no definitive answer in sight, but there are two practical points for machine learners. First, whether or not we call them "causal," we would like to predict the effects of our actions, not just correlations between observable variables. Second, if you can obtain experimental data (for example by randomly assigning visitors to different versions of a Web site), then by all means do so.[14]

Conclusion

Like any discipline, machine learning has a lot of "folk wisdom" that can be difficult to come by, but is crucial for success. This article summarized some of the most salient items. Of course, it is only a complement to the more conventional study of machine learning. Check out http://www.cs.washington.edu/homes/pedrod/ class for a complete online machine learning course that combines formal and informal aspects. There is also a treasure trove of machine learning lectures at http://www.videolectures.net. A good open source machine learning toolkit is Weka.[24]

Happy learning!

References

1. Bauer, E. and Kohavi, R. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning 36* (1999), 105–142.
2. Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.
3. Benjamini, Y. and Hochberg, Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B, 57* (1995), 289–300.
4. Bernardo, J.M. and Smith, A.F.M. *Bayesian Theory.* Wiley, NY, 1994.
5. Blumer, A., Ehrenfeucht, A., Haussler, D. and Warmuth, M.K. Occam's razor. *Information Processing Letters 24* (1987), 377–380.
6. Cohen, W.W. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence 68* (1994), 303–366.
7. Domingos, P. The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery 3* (1999), 409–425.
8. Domingos, P. Bayesian averaging of classifiers and the overfitting problem. In *Proceedings of the 17th International Conference on Machine Learning* (Stanford, CA, 2000), Morgan Kaufmann, San Mateo, CA, 223–230.
9. Domingos, P. A unified bias-variance decomposition and its applications. In *Proceedings of the 17th International Conference on Machine Learning* (Stanford, CA, 2000), Morgan Kaufmann, San Mateo, CA, 231–238.
10. Domingos, P. and Pazzani, M. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning 29* (1997), 103–130.
11. Hulten, G. and Domingos, P. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Edmonton, Canada, 2002). ACM Press, NY, 525–531.
12. Kibler, D. and Langley, P. Machine learning as an experimental science. In *Proceedings of the 3rd European Working Session on Learning* (London, UK, 1988). Pitman.
13. Klockars, A.J. and Sax, G. *Multiple Comparisons.* Sage, Beverly Hills, CA, 1986.
14. Kohavi, R., Longbotham, R., Sommerfield, D. and Henne, R. Controlled experiments on the Web: Survey and practical guide. *Data Mining and Knowledge Discovery 18* (2009), 140–181.
15. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. and Byers, A. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.
16. Mitchell, T.M. *Machine Learning.* McGraw-Hill, NY, 1997.
17. Ng, A.Y. Preventing "overfitting" of cross-validation data. In *Proceedings of the 14th International Conference on Machine Learning* (Nashville, TN, 1997). Morgan Kaufmann, San Mateo, CA, 245–253.
18. Pearl, J. On the connection between the complexity and credibility of inferred models. *International Journal of General Systems 4* (1978), 255–264.
19. Pearl, J. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, Cambridge, UK, 2000.
20. Quinlan, J.R. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
21. Richardson, M. and P. Domingos. Markov logic networks. *Machine Learning 62* (2006), 107–136.
22. Tenenbaum, J., Silva, V. and Langford, J. A global geometric framework for nonlinear dimensionality reduction. *Science 290* (2000), 2319–2323.
23. Vapnik, V.N. *The Nature of Statistical Learning Theory.* Springer, NY, 1995.
24. Witten, I., Frank, E. and Hall, M. *Data Mining: Practical Machine Learning Tools and Techniques, 3rd Edition.* Morgan Kaufmann, San Mateo, CA, 2011.
25. Wolpert, D. The lack of a priori distinctions between learning algorithms. *Neural Computation 8* (1996), 1341–1390.

Pedro Domingos (pedrod@cs.washington.edu) is a professor in the Department of Computer Science and Engineering at the University of Washington, Seattle.

research highlights

DOI:10.1145/2347736.2347756

Technical Perspective
A High-Dimensional Surprise

By Rocco A. Servedio

HIGH-DIMENSIONAL SPACE IS a counterintuitive place, where natural geometric intuitions from the familiar three-dimensional world may lead us badly astray. As a simple example consider the Earth, which has a radius of about 3,950 miles (let's pretend it is a perfect sphere). What fraction of the Earth's surface lies within 1/100 of its radius, that is, within 39.5 miles, from the equator? A reasonable guess would be "about 1/100," and that is almost exactly correct. Now let's think high-dimensionally: what fraction of the surface of a d-dimensional sphere lies within 1/100 of its radius from its equator? Again one might guess "about 1/100," but this is wildly incorrect: all but an *exponentially small* (in terms of d) fraction of the d-dimensional sphere's surface lies within 1/100 of its radius from the equator. Strange as it seems, virtually all the real estate on d-dimensional Earth is in the tropics!

The challenges of dealing with high-dimensional space are familiar to researchers in fields like machine learning and combinatorial optimization, where the "curse of dimensionality" arises often. Tasks that are easy in few dimensions—finding a simple curve to fit data points, computing the volume of a region—can become frustratingly difficult or even provably intractable (assuming $P!=NP$) in high-dimensional space. Yet sometimes the tables are turned, and things that seem difficult or impossible based on our low-dimensional intuition can, near-magically, become possible in high dimension. The following paper describes such a situation, where a clever construction inspired by ideas from computational complexity theory proves the existence of what seems like a too-good-to-be-true high-dimensional object.

The authors reveal an unexpected connection between spheres and cubes and construct a new geometric object that has some of the most fundamental properties of both shapes. Turning first to cubes, they are the canonical shape that can be used to "tile space" according to a regular grid-like lattice leaving no holes or unfilled space (think of slicing a block of cheese into cubes). Of course a simple cube is not the only shape that can do this—for example, adding bumps to the top and carving matching divots into the bottom of a cube yields a different, Lego-like shape that also fits perfectly with itself to fill space with no holes under a grid-like tiling. Any shape that can do this is called a "cubical tile."

A sphere is certainly not a cubical tile, since a grid-like pattern of 3D spheres leaves a lot of empty space unfilled. In higher dimensions spheres do even worse: the fraction of unfilled space becomes exponentially larger (as a function of d) than the fraction actually filled by the spheres. But spheres reign supreme for a different property: they have the smallest surface-area-to-volume ratio of any shape, in three or any number of dimensions. (This is one reason why water towers have their familiar round shape—small surface area means less building material and more uniform water temperature.) In fact, while a d-dimensional cube of volume 1 has surface area $2d$, a d-dimensional sphere of volume 1 has surface area only about $4.13\sqrt{d}$, which is much smaller in high dimensions.

So cubes can tile space perfectly but have high surface-area-to-volume ratio, while spheres have the lowest possible surface-area-to-volume ratio but cannot tile space. The authors show that in high-dimensional space it is possible to get the best of both worlds, by proving that there exists a d-dimensional "spherical cube." This counterintuitive object is a cubical tile (that is, a shape that tiles space perfectly like the cube), but has surface-area-to-volume ratio very nearly as small as that of the sphere: roughly $12.6\sqrt{d}$, that is, the same as the sphere up to a small constant factor!

Why should we care? There are at least three reasons. First, as described in the paper, their geometric construction yields a highly noise-resistant randomized procedure for rounding high-dimensional data points to integer values; such a procedure could be useful in many scenarios. Second, the authors also obtain the best surface-area-to-volume ratio of any 3D cubical tile, improving on a construction of Choe[1] from 1989; this could have practical consequences in material science, fabrication, or related areas. Third, their work is based on a surprising connection to recent deep results on hardness amplification in computational complexity theory;[2] their article provides a fascinating perspective on this area, and a great illustration of how ideas from one branch of mathematics can have unexpected yet tremendous utility in a completely different area. Finally, there is the sheer "wow, I wouldn't have thought that's possible!" factor—it is just plain neat that you can have a "spherical cube" in high dimensions.

Many important questions remain unanswered, both in the complexity-theoretic foundations of Kindler et al.'s result and in the specifics of their geometric construction. A randomized algorithm, not a particularly efficient one, constructs their cubical tile; natural goals are to come up with a more efficient randomized construction, or ideally even an explicit construction that does not use randomness. We can reasonably hope that future work—perhaps again informed by perspectives and ideas from complexity theory—will provide more insights into the mysteries and surprises of high-dimensional geometry. C

References

1. Choe. J. On the existence and regularity of fundamental domains with least boundary area. *J. Differential Geom. 29* (1989), 623–663.
2. Ran, R. A counterexample to strong parallel repetition. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science* (2008), 369–373.

Rocco Servedio (rocco@cs.columbia.edu) is an associate professor of computer science at Columbia University, New York, NY.

DOI:10.1145/2347736.2347757

Spherical Cubes: Optimal Foams from Computational Hardness Amplification

By Guy Kindler, Anup Rao, Ryan O'Donnell, and Avi Wigderson

Abstract

Foam problems are about how to best partition space into bubbles of minimal surface area. We investigate the case where one unit-volume bubble is required to tile *d*-dimensional space in a periodic fashion according to the standard, cubical lattice. While a cube requires surface area 2*d*, we construct such a bubble having surface area very close to that of a sphere; that is, proportional to $\sqrt{d}$ (the minimum possible even without the constraint of being periodic). Our method for constructing this "spherical cube" is inspired by foundational questions in the theory of computation related to the concept of hardness amplification. Our methods give new algorithms for "coordinated discretization" of high-dimensional data points, which have near-optimal noise resistance. We also provide the most efficient known cubical foam in three dimensions.

1. INTRODUCTION

A *foam* in the *d*-dimensional space $\mathbb{R}^d$ is a partition of $\mathbb{R}^d$ into bounded sets called bubbles. In such a foam, the bubbles are said to *tile* the space. The main question studied in this work is if a foam in $\mathbb{R}^d$ has only bubbles with a given volume, what is the minimal possible average surface area of its bubbles? This fundamental question has been a focus of study for scientists in many disciplines, from physicists studying soap bubbles,[21] to chemists studying crystal structures,[12] biologists studying cell aggregation,[3] mathematicians studying sphere-packings,[14] materials scientists studying polymers,[20] and even artists and architects.[6] In this work, we present a new approach to the construction of tiling shapes, based on methods from computer science. This approach leads to an asymptotically optimal solution of the Cubical Foam Problem, defined below.

1.1. Foams

Questions about minimal surface area tilings of space have a very long history. In the 19th century, Thomson (Lord Kelvin) introduced the Kelvin Foam Problem,[26] which asks how three-dimensional space can be partitioned into bubbles of volume 1 such that the average surface area of the bubbles in the foam is minimized. This question is motivated not only by its mathematical appeal, but also by interest in the physics of foams in nature, since surface tension makes bubbles seek to minimize their surface area.

One way to design foams with small surface area is to first construct a lattice of periodically arranged points, and then to take the Voronoi cells around each lattice point. The Voronoi cell of a lattice point x is the bubble, which includes all points that are closer to x than to any other lattice point. The solution Kelvin proposed in 1887 for his problem was based on the Voronoi foam associated with the so-called *body-centered* cubic lattice. The bubbles in this foam have a surface area $(3/4)\sqrt[3]{120\sqrt{3}+148} \approx 5.315$. Kelvin further suggested letting this foam relax, so that it conforms with Plateau's Rules for soap bubbles[21]; modern computer simulations show that this decreases the surface area to about 5.306.[7, 18] In 1994, Weaire and Phelan[27] exhibited a foam with an improved average surface area of about 5.288. The Weaire–Phelan foam is formed by relaxing the Voronoi foam for a periodic subset of lattice points (Figure 1). Weaire and Phelan used the crystal structure of a certain silicon–sodium clathrate to choose the points. It is still unknown whether or not their foam optimally solves Kelvin's problem.

It is natural to study the Kelvin Foam Problem in dimensions other than three. In two dimensions, it was long believed that the best solution is to tile space with regular hexagons, which is the Voronoi foam of the triangular lattice. The optimality of this foam was conjectured as far back as in the 4th century by Pappus of Alexandria, but a mathematical proof was found only in 1999, by Hales.[13] In higher dimensions, a lower bound on the average surface area follows from the *Isoperimetric Inequality*: the surface area of any bubble of volume 1 must be at least as large as that of a ball of volume 1. As the number of dimensions d grows, this lower bound asymptotically approaches $\sqrt{2\pi e d}$. An upper bound that matches this lower bound up to a factor of 2 can be obtained by taking the Voronoi foam of a d-dimensional lattice in which the covering-radius to packing-radius ratio tends to 2. Such a lattice can be obtained by a probabilistic construction.[8] Hence, the minimum surface area in the d-dimensional Kelvin Foam Problem grows in proportion to the square-root of the dimension.

In our work, we consider tilings that are periodic with respect to the *integer lattice* (also known as the cubic lattice).

The original version of this paper is entitled "Spherical Cubes and Rounding in High Dimensions" and was published in the *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science,* IEEE Computer Society, 2008.

This lattice consists of the points in d-dimensional space whose coordinates are all integers. We address the following question:

Cubical Foam Problem: What is the least surface area of a bubble that partitions d-dimensional space periodically according to the integer lattice?

The Voronoi foam for the integer lattice consists of cubes of side length 1. In d dimensions, these cubes have surface area $2d$. This grows linearly with the dimension, much higher than the known lower bound of $\sqrt{d}$. Are there more "spherical" cubes, which still tile by the integer lattice, but have surface area closer to that of a ball? This is the main question that we answer in this work.

The Cubical Foam Problem seems to have been first formally raised by Choe.[10] Choe showed that in two dimensions, the unit square whose surface area (perimeter) is 4 is not the optimal solution. Rather, the optimal solution is the isosceles hexagon shown in Figure 2, with 120° angles, side lengths $\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{6}}$ and $\sqrt{\frac{2}{3}}$, and perimeter about 3.864. Choe gave the three-dimensional version as an open problem. Prior to our work, the best known solution was simply to add depth to the Choe hexagon, transforming it into the prism shown in Figure 2, with surface area 5.864.[11]

The high-dimensional version of the Cubical Foam Problem was raised by Feige et al.[11] in 2007, who noted a surprising connection to a certain problem in theoretical computer science about computational hardness amplification. We shall explain the details of this connection later. A subsequent result of Raz[24] on the limits of such amplification, using an idea from a related paper of Holenstein,[16] provided us with the tools to solve the high-dimensional Cubical Foam Problem.

1.2. Our results

- We give a probabilistic construction proving the existence of a bubble that partitions d-dimensional space according to the cubic lattice and whose surface area is at most $4\pi\sqrt{d}$. Thus, our bubble is nearly spherical, in the sense that its surface area is larger than that of a sphere by only a constant multiplicative factor (about 3.04). The best previous constructions had surface area proportional to d (like the cube itself has). We conclude that the optimal solution to the Cubical Foam Problem has surface area proportional to the square-root of the dimension, just as in the more general Kelvin Foam Problem. Thus in high dimensions, integer-lattice til-

Figure 1. (Left) Four bubbles in the Kelvin Foam, formed by relaxing the Voronoi cells of the body-centered cubic lattice. (Right) Seven bubbles in the Weaire–Phelan Foam, formed by relaxing the Voronoi cells of the A15 Packing.

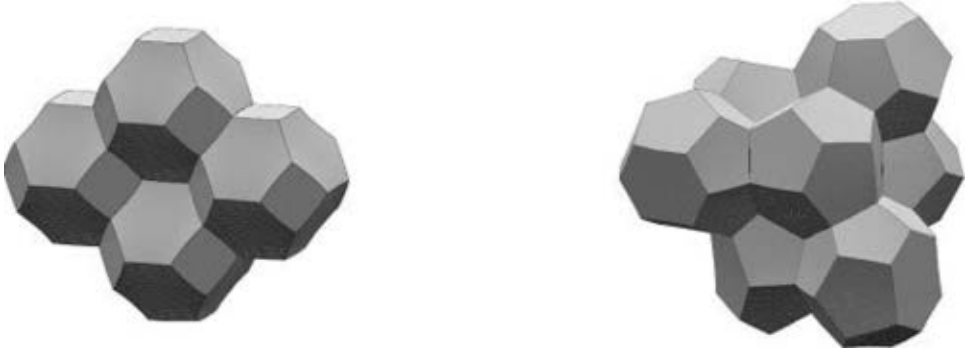
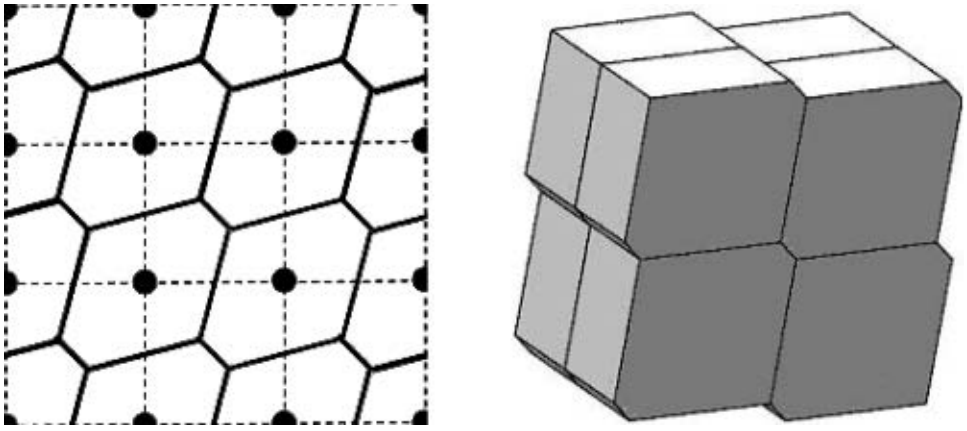


Figure 2. (Left) The Choe Hexagon, Choe's optimal solution to the Cubical Foam Problem in two dimensions. (Right) The hexagons extruded into three-dimensional prisms. The resulting three-dimensional cubical foam is not optimal; a solution with smaller average surface area is presented in this work.



ings are essentially as efficient as arbitrary tilings.

- We show that our construction also yields a highly noise-resistant procedure for the rounding high-dimensional data. Specifically, we obtain a randomized procedure, which assigns each vector of real numbers $x = (x_1,..., x_d)$ to a vector of integers $n = (n_1,..., n_d)$, with the following two guarantees. Each n_i is always simply x_i rounded up or down to its floor or ceiling integer, yet if two real vectors x, y are at Euclidean distance ρ, then our procedure assigns them to the same integer vector except with probability at most $2 \cdot \pi \cdot \rho$. Thus, if two parties get two noisy versions x, y of the same integer vector n, they will round them to the same vector with very high probability. Somewhat remarkably, the error in this natural coordination task does not depend on the dimension at all, and depends only linearly on the noise. Previously known procedures either rounded points to far off points, so that $|x_i - n_i|$ could be as large as $\sqrt{d}$, or separated points at distance ρ with probability as high as $\sqrt{d}\rho$.[9, 19]
- Using different (and ad hoc) methods, we give an explicit construction for the Cubical Foam Problem in three dimensions with a surface area about 5.602. This beats the surface area of the prism obtained from Choe's prism by about 4.5%.

1.3. Computational hardness amplification

Our method for constructing an efficient cubical foam has a surprising inspiration: the subject of computational hardness amplification in the study of the theory of computation. Consider a computational task T, such as solving a system of equations, finding the best move in a chess position, optimizing a schedule under constraints, etc. The difficulty, or hardness, of T is measured in terms of the computational resources (say the number of steps in an algorithm) required to obtain a solution of a given quality. *Hardness amplification* refers to methods that can be used to transform T into an even harder task, requiring even more resources. For example, one could consider the task of solving T on d inputs simultaneously. Is this d times harder, or can a clever reuse of resources allow for a computation that achieves more than what one would naively expect?

This question arises in many areas of computational theory, including cryptography, pseudorandomness, and optimization. Our foam construction is motivated by progress in understanding hardness amplification in the context of solving constraint satisfaction problems, a major topic in computer science, operations research, statistical physics, and information theory (e.g., Achlioptas et al.[1]). A constraint satisfaction problem is specified by a collection of constraints on n variables. For the purpose of this discussion, we shall restrict ourselves to *bivariate* constraints, of the form $c(x, y)$, where each constraint c depends on only two of the n variables. A solution to the problem is a setting of the variables that maximizes the number of constraints that are satisfied. For example, the well-known graph coloring problem can be viewed as a bivariate constraint problem. Associate a single variable x_v for each vertex v of the given graph and each edge $\{u, v\}$ gives us the constraint $x_u \neq x_v$. If the variables are allowed to take values from $\{1, 2, 3\}$, then the constraints are all satisfiable, if and only the graph is 3-colorable.

One way to solve a constraint satisfaction problem is to try all settings of the variables and count the number of constraints that each setting satisfies, but this method is very inefficient, requiring exponential time. If $P \neq NP$, one cannot find a solution that satisfies *all* the constraints efficiently. A seminal hardness amplification result, the PCP Theorem,[4, 5] from 1992, improved this to show that if $P \neq NP$, then efficient algorithms cannot even find approximate solutions that satisfy *nearly all*, a $(1 - \epsilon_0)$ fraction of the constraints, when a solution satisfying all the constraints is known to exist. Here, ϵ_0 is a small positive constant. Indeed, the proof actually shows that it is hard to approximate even the fraction of constraints that are satisfiable.

Raz's celebrated Parallel Repetition Theorem[23] from 1995 dramatically strengthened this hardness of approximation result: he showed that for every $\epsilon > 0$, if $P \neq NP$, then efficient algorithms cannot guarantee solutions that satisfy very few, an ϵ fraction of the satisfiable constraints. Raz achieves this through a transformation of constraints called *parallel repetition* that we describe next. Suppose we are given a set of constraints on n variables. These constraints can be used to define new constraints on n^d variables as follows: Each new variable corresponds to a d-tuple of variables from the old problem. For every d-tuple of the original constraints $c_1, c_2,..., c_d$, we obtain a new constraint $C(X, Y) = c_1(x_1, y_1) \wedge c_2(x_2, y_2) \wedge ... \wedge c_d(x_d, y_d)$ that can be thought of as a bivariate constraint on the variables $X = (x_1,..., x_d)$ and $Y = (y_1,..., y_d)$. The resulting compound constraints are said to have been obtained by repeating the original constraints d times in *parallel*.

If the best assignment can satisfy $(1 - \epsilon)$ fraction of the original constraints, how many constraints can be satisfied in the parallel repetition? Intuitively, one might think that the fraction of satisfiable constraints should be smaller, perhaps decay exponentially in d, since each new constraint corresponds to satisfying a set of d of the original constraints. But proving that this is true turned out to be very challenging, and counterexamples for pure exponential decay were known. In a breakthrough result, Raz showed that no assignment that can satisfy roughly $(1 - \epsilon^{32})^d$ fraction of the constraints obtained by parallel repetition, namely, *some* exponential decay, occurs. A key quantity of interest here is the *rate* of decay, namely, how "quickly" does the problem become harder, and the approximating factor decrease. Set $f(d)$ to be the largest number for which d repetitions of a constraint satisfaction problem decreases the approximation factor from $1 - \frac{1}{f(d)}$ to some small constant, say 1/10. In this notation, Raz showed that $f(d) \geq d^{1/32}$, and left open determining the optimal dependence on d.

This result played a key role in showing that many types of problems cannot be solved by efficient algorithms (again assuming $P \neq NP$). A theorem that would prove a bound of the type $f(d) \approx d$ was dubbed a *Strong Parallel Repetition* theorem, and it remained open whether such a theorem holds. Subsequent works improved Raz's bound

toward such a strong theorem, first to $f(d) \geq d^{1/3}$ [15] in general, and then to $f(d) \geq \sqrt{d}$ [22] for an important subclass of constraints satisfaction problems, but the progress stopped at $\sqrt{d}$.

Feige et al.[11] were the first to observe that the parallel repetition question was related to foams. They studied a particular collection of constraints called *Odd Cycle* constraints, and showed that for them $f(d) \geq \sqrt{d}$. They also showed that if there is a d-dimensional cubical foam with surface area $A(d)$, then for this constraint satisfaction problem $f(d) \leq A(d)$. In particular, this meant that any proof that improved on their bound would show that there is no cubical foam with surface area $\sqrt{d}$, and any Strong Parallel Repetition theorem would prove that standard cubes are essentially the best cubical foams.

Once again, Raz[24] resolved the matter and showed that Odd Cycles were a counterexample to Strong Parallel Repetition. He proved that for Odd Cycle constraints $f(d) \leq \sqrt{d}$ and thus that the results of bounds $f(d) \geq \sqrt{d}$ of Feige et al.[11] and Rao[22] are optimal! Indeed, one can view Raz's work as constructing a *discrete* cubical foam, with surface area proportional to $\sqrt{d}$. A key tool used by Raz was the so-called *Consistent Sampling Lemma*, invented by Holenstein to prove the upper bound of $f(d) \leq d^{1/3}$ mentioned above. Raz's result inspired our construction.

2. AN ALGORITHM FOR BUILDING CUBICAL FOAMS

Our solution to the Cubical Foam Problem involves generalizing Raz's discrete methods to Euclidean space and opening up the proof of the Consistent Sampling Lemma. We use the "Buffon's Needle" method to estimate surface area and we optimize our results using Fourier analysis.

Before describing our "sphere-like" cubical foam, we give some motivation for its construction. As stated earlier, our construction can also be interpreted as a very noise-resistant randomized discretization procedure for rounding off vectors of real numbers to vectors of integers.

Definition 1. A discretization is a mapping which assigns each point $x = (x_1,..., x_d) \in \mathbb{R}^d$ to an integer point $r = (r_1,..., r_d) \in \mathbb{Z}^d$, such that $|r_i - x_i| < 1$ for each i. If a discretization has the property that whenever x is rounded to r, then $x + s$ is rounded to $r + s$ for all $s \in Z^d$, we say that the discretization is periodic. Given a periodic discretization, we define its principle bubble to be the set of points that are rounded to the origin.

The principal bubble of a periodic discretization tiles $\mathbb{R}^d$ according to the integer lattice. Thus, any periodic discretization immediately yields a cubical foam. We will in fact give a randomized procedure whose output is a periodic discretization (hence also a cubical foam). As described earlier, we say that such a procedure is noise-resistant if every two nearby points $x, y \in \mathbb{R}^d$ are unlikely to be assigned to different integer points. Intuitively, we expect the bubbles produced by a noise-resistant procedure to have small surface area, because x and y are assigned to different integer points only if the line segment joining them crosses the surface of a bubble. We will later see that finding a periodic discretization in which nearby pairs x and y are usually assigned to the same integer point is very similar to a constraint satisfaction problem where every constraint involves two variables which are points in $\mathbb{R}^d$. Indeed, Raz's analysis of the Odd Cycle constraints suggests the investigation of a related problem (which we state somewhat imprecisely for the sake of brevity): Assign each point x in the unit cube $[0, 1)^d$ a shift $z \in [0, 1)^d$ in such a way that most nearby pairs of points are likely to be assigned the same shift.

The construction will be based on a carefully chosen probability distribution f on $[0, 1)^d$. For now, we describe the construction for any distribution f and later explain how to choose the optimal f. Let f_x be f translated by $-x$, so f_x is a probability density function on $-x + [0, 1)^d$; and let $\tilde{f}_x$ be the periodic extension of this function, so $\tilde{f}_x(z) = f(z + x \bmod 1)$. Holenstein's Consistent Sampling Lemma gives a method for assigning shifts to all points such that the probability x and y are assigned different shifts is essentially $\int_{[0,1)^d} |f_x - f_y|$. We draw a sequence of points (Z_1, H_1), (Z_2, H_2),... such that Z_i is drawn uniformly from $[0, 1)^d$, and H_i is uniform on $(0, \|f\|_\infty)$, where $\|f\|_\infty$ denotes the maximum value of f. Each x is then assigned a shift $z = Z_i$, where z_i comes from the first pair satisfying $\tilde{f}_x(Z_i) > H_i$. Thus, we are led to seek a function f for which this is small whenever x and y are nearby points.

Given a density function f and a number $h > 0$, consider the shape $D = \{x : f(x) > h\} \subset (0, 1)^d$ together with its boundary. We call D, or a translate of D a *droplet*. We will want droplets to have smooth boundaries, which do not touch the boundary of $[0, 1]^d$. For this reason, we will require that f's periodic extension $\tilde{f}$ be analytic and equal to 0 on the boundary of $[0, 1]^d$; we will call such a density function *f proper*. Given a proper density function f, we can now describe our randomized algorithm for producing a periodic discretization and associated cubical foam:

Algorithm 1: Periodic discretization (and foam) construction, given *f*:

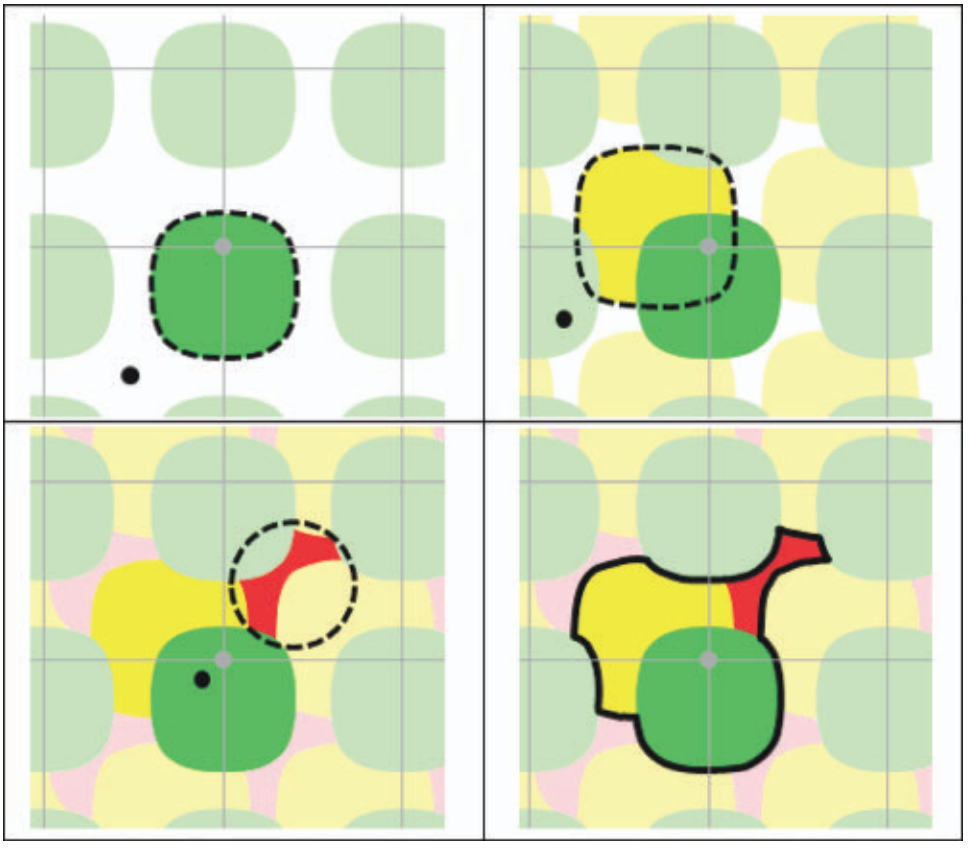
1. Let all points of $\mathbb{R}^d$ be unassigned.
2. For stage $i = 1, 2, ...$ until all points are assigned:
   (a) Choose a uniformly random pair (Z_i, H_i) from $[0, 1)^d \times (0, \|f\|_\infty)$.
   (b) Let droplet D_i be $\{x : fz_i(x) > H_i\}$, together with its boundary.
   (c) Assign all currently unassigned points in D_i to $(0,..., 0)$, and extend this assignment periodically.
   (d) Color all assignments from stage i with color i.

We remark that we color all the assignments merely to aid in the analysis of the algorithm.

It is not hard to prove that this algorithm indeed ends after a finite number of stages with probability 1. It is also clear that regardless of the algorithm's random choices, it always produces a periodic discretization. Thus, the points assigned to $(0,..., 0)$ by the algorithm always constitute a principal bubble, which partitions space according to the integer lattice.

We illustrate a sample run of the algorithm in Figure 3, with $d = 2$ and $f(x_1, x_2) = 4\sin^2(\pi x_1) \sin^2(\pi x_2)$. The integer lattice is outlined in gray, with the origin depicted as a gray dot. The first three panels illustrate stages 1, 2, and 3 of the

Figure 3. A sample run of the algorithm, with $d = 2$, $f(x_1, x_2) = 4\sin^2(\pi x_1)\sin^2(\pi x_2)$. The integer lattice is outlined in gray, with the origin depicted as a gray dot. The first three panels illustrate stages 1, 2, and 3 of the algorithm. In each stage, the black dot represents $-Z_i$ and the black dashed line outlines the droplet D_i. Colors 1, 2, and 3 are green, yellow, and red, respectively; we have used dark colors to show the points assigned to (0, 0) and light colors to show their periodic translations. In the first panel is the assignment after stage 1: all points in the dark green droplet are assigned to (0, 0); the light green translations are assigned periodically. The assignment after stage 2: the unassigned (uncolored) points within the outlined droplet are colored dark yellow and are assigned to (0, 0). The assignment after stage 3, using red. The algorithm terminates after this stage—all points in $\mathbb{R}^2$ have been assigned. In the final panel, we outline the final bubble which partitions $\mathbb{R}^2$ periodically according to the integer lattice.



algorithm. In each stage, the black dot represents $-Z_i$ and the black dashed line outlines the droplet D_i. Colors 1, 2, and 3 are green, yellow, and red, respectively; we have used dark colors to show the points assigned to (0, 0) and light colors to show their periodic translations.

3. ANALYSIS OF THE ALGORITHM

First, we compute the probability of rounding a pair of points x, y to different droplets:

THEOREM 1. *Let f be a proper density function, and $\overline{xy}$ be a short line segment in $\mathbb{R}^d$, say $y = x + \epsilon \cdot u$, where u is a vector of length 1, and $\epsilon > 0$ is sufficiently small. For a given execution of Algorithm 1, let N denote the number of times $\overline{xy}$ crosses the boundary between differently colored regions. Then*

$$\mathbb{E}[N] \approx \epsilon \cdot \int_{[0,1)^d} |\langle \nabla f, u \rangle|,$$

where the notation $\approx$ means equality up to order ϵ^2.

Next, using the relationship between noise resistance and surface area we show:

THEOREM 2. *Given an execution of Algorithm 1, let A denote the surface area of the boundary between color regions within $[0, 1)^d$. Then*

$$\mathbb{E}[A] = \int_{[0,1)^d} ||\nabla f||.$$

Finally, we find f that minimizes the noise resistance and surface area:

THEOREM 3. *$f = \prod_{i=1}^{d} (2\sin^2(\pi x_i))$ is a proper density function with $\int ||\nabla f|| \le 2\pi\sqrt{d}$. Moreover, for each vector of unit length u, f satisfies $\int |\langle \nabla f, u \rangle| \le 2\pi$.*

The bounds we obtain for our cubical foam solution and for the noise resistance of our coordinated discretization procedure follow easily from these theorems. The bubble B output by Algorithm 1 has a surface area at most $2A$, where A is the quantity in Theorem 2; hence with the f from Theorem 3, the expected value of B's surface area is at most $4\pi\sqrt{d}$. Hence, there must exist a bubble that tiles $\mathbb{R}^d$ according to the integer lattice with a surface area at most $4\pi\sqrt{d}$. As for the noise resistance of Algorithm 1 as a coordinated

discretization procedure, if points $x, y \in \mathbb{R}^d$ at distance ϵ are assigned different integer points by the algorithm, then N, the number of times segment $\overline{xy}$ crosses the boundary between color regions, must be at least 1. The probability of this event is at most $\mathbb{E}[N]$. Combining Theorems 1 and 3, this probability is at most $2\pi\epsilon$ (up to an error of ϵ^2, but this error can be eliminated) as claimed.

Next we outline the proofs of Theorems 1, 2, and 3.

We begin with the main result, Theorem 1. Let $\tilde{D} = \text{closure}(\{x : \tilde{f}Z(x) > H\})$ be a random "droplet pattern" as would be chosen in a single stage of Algorithm 1; that is, $\tilde{D}$, is a random droplet along with all of its integer translations. Let I denote the event that $\tilde{D}$ intersects the segment $\overline{xy}$, and let M denote the number of intersection points between the *boundary* of D and the segment $\overline{xy}$. Since $\overline{xy}$ is very short, the first color region that touches the segment is very likely to completely enclose it. Thus, even *conditioned* on event I occurring, M is quite likely to be 0. More precisely, our main goal is to show that

$$\mathbb{E}[M \mid I] \approx \epsilon \cdot b, \quad \text{where } b = \int_{[0,1)^d} |\langle \nabla f, u \rangle|. \tag{1}$$

Using Equation (1), it is not hard to prove Theorem 1. To see this, consider the first stage of the algorithm's execution in which a droplet pattern touching $\overline{xy}$ is chosen. Let M_1 be the number of intersections between $\overline{xy}$ and the boundary of this droplet pattern. Equation (1) tells us that $\mathbb{E}[M_1] \approx \epsilon \cdot b$. Recalling that N denotes the number of times $\overline{xy}$ crosses the boundary between differently colored regions at the termination of the algorithm, we certainly have $N \geq M_1$. However, it is unlikely that N will exceed M_1. Indeed, consider the second stage of the algorithm's execution in which a droplet pattern touching $\overline{xy}$ is chosen, and define M_2 to be the number of intersections between the boundary of this droplet pattern and $\overline{xy}$. Again, by Equation (1) we have $\mathbb{E}[M_2] \approx \epsilon \cdot b$. But furthermore, these M_2 intersections can only contribute to N if the first droplet pattern to touch $\overline{xy}$ failed to completely enclose it. The probability of this is precisely $\Pr[M_1 > 0] \leq \mathbb{E}[M_1] \approx \epsilon \cdot b$. Hence, the expected contribution to N from this second stage is at most $(\epsilon \cdot b)^2$. Continuing the argument, we are able to upper-bound $\mathbb{E}[N]$ by

$$\epsilon \cdot b + (\epsilon \cdot b)^2 + (\epsilon \cdot b)^3 + \cdots \approx \epsilon \cdot b.$$

We now describe the proof of Equation (1). Since the probabilistic choice of $\tilde{D}$ is invariant under any translation of $\mathbb{R}^d$, we may assume $x = 0$ and hence $y = \epsilon \cdot u$. For a given $z \in [0, 1)^d$, let $g_z : [0, \epsilon] \to \mathbb{R}^{\geq 0}$ denote the restriction of $\tilde{f}_z$ to the line segment from 0 to $\epsilon \cdot u$ and write $G(z) = \|g_z\|_\infty$. The event I occurs when the randomly chosen Z and H satisfy $H < G(Z)$, and the quantity M is equal to $\#\{s \in [0, \epsilon] : g_z(s) = H\}$. (Here, we have discounted events with probability 0.) Hence,

$$\mathbb{E}[M \mid I] = \frac{\int_{[0,1)^d} \int_0^{G(Z)} \#\{s \in [0, \epsilon] : g_z(s) = H\} \, dH \, dZ}{\int_{[0,1)^d} G(Z) \, dZ} \tag{2}$$

Regarding the denominator of this expression, $G(Z) = gz(0) = f(Z)$ up to an additive error of order ϵ by Taylor's theorem, and hence $\int_{[0,1)^d} G(Z) dZ = \int_{[0,1)^d} f(Z) dZ = 1$ up to order ϵ. As for the numerator of Equation (2), the inner integral equals the vertical distance traveled by a particle moving along the curve gz; this can be seen by partitioning the curve into small, nearly straight arcs, and considering their contribution to the integral. Hence, the inner integral equals $\int_0^\epsilon |g'_z|$. Since $|g'_z(t)| = |g'_z(0)|$ up to order ϵ on $[0, \epsilon]$, $\int_0^\epsilon |g'_z| \approx \epsilon \cdot |g'_z(0)| = \epsilon \cdot |\langle \nabla \tilde{f}_z(0), u \rangle| = \epsilon \cdot |\langle \nabla \tilde{f}(Z), u \rangle|$ up to order ϵ^2. Substituting this into the outer integral in the numerator of Equation (2), we conclude that

$$\mathbb{E}[M \mid I] \approx \epsilon \cdot \int_{[0,1)^d} |\langle \nabla \tilde{f}(Z), u \rangle| \, dZ = \epsilon \cdot b, \tag{3}$$

up to order ϵ^2 as claimed in Equation (1).

To prove Theorem 2, we need a method for computing surface area. We use the following (see Santalo[25])

Theorem 4 (Buffon's Needle Theorem)*: Let S be a $\mathbb{Z}^d$-periodic piecewise smooth surface. "Drop a needle" of length $0 < \epsilon < 1$; i.e., let x be a random point in $[0, 1)^d$, let u be a random vector of length 1, and let $y = x + \epsilon \cdot u$. If N denotes the number of intersections of the needle $\overline{xy}$ with the surface S, then*

$$\text{area}(S \cap [0, 1)^d) = \frac{\mathbb{E}[N]}{c_d \cdot \epsilon}, \tag{4}$$

where c_d is the dimension-dependent–constant $\mathbb{E}[|u_1|]$.

Note that the value of c_d is not important, as it gets cancelled out in our analysis. Given the execution of Algorithm 1, we can apply Buffon's Needle Theorem to compute the quantity A from Theorem 2. We obtain that $\mathbb{E}[A] = \mathbb{E}[N]/(c_d \cdot \epsilon)$, where in $\mathbb{E}[N]$ the probabilistic experiment is both the execution of Algorithm 1 and the random choice of the "needle." Applying Theorem 1 for each choice of the needle, we obtain

$$\begin{aligned} \mathbb{E}[A] &= \mathbb{E}_u\left[\epsilon \cdot \int_{[0,1)^d} |\langle \nabla f, u \rangle|\right] / (c_d \cdot \epsilon) \\ &= \int_{[0,1)^d} \mathbb{E}_u[|\langle \nabla f, u \rangle|] / c_d, \end{aligned} \tag{5}$$

up to an additive error of order ϵ. Since ϵ can be arbitrarily small, Equation (5) is in fact an exact equality. And further, for each fixed vector $\nabla f(z)$, the quantity $\mathbb{E}_u[|\langle \nabla f(z), u \rangle|]$ equals $\|f(z)\| \cdot c_d$. Substituting this into Equation (5) proves Theorem 2.

It remains to prove Theorem 3. Suppose, we seek a proper density function on $[0, 1)^d$ such that $\int \|f\|$ is small. Writing $f = g^2$ and using the Cauchy–Schwarz inequality:

$$\int \|\nabla f\| = 2 \int |g| \cdot \|\nabla g\| \leq 2 \sqrt{\int g^2} \cdot \sqrt{\int \|\nabla g\|^2}, \tag{6}$$

where we also used that $\int f = \int g^2 = 1$. Since f is a proper density function, g's periodic extension is smooth and 0 on the boundary of $[0, 1]^d$. We may therefore rewrite g using the multidimensional sine series:

$$g(x) = \sum_{\omega \in \mathbb{N}^d} \hat{g}(\omega) \prod_{i=1}^{d} \sqrt{2} \sin(\pi \omega_i x_i). \tag{7}$$

Differentiating Equation (7) and applying Parseval's theorem, we get

$$\int \|\nabla g\|^2 = \pi^2 \sum_{\omega \in \mathbb{N}^d} \|\omega\|^2 \hat{g}(\omega)^2. \quad (8)$$

Applying Parseval to $\int g^2 = 1$ yields that $\sum_{\omega \in \mathbb{N}^d} \hat{g}(\omega)^2 = 1$, subject to which Equation (8) is minimized when $\hat{g}(1, 1, \ldots, 1) = 1$. Thus, we are led to the solution for f stated in Theorem 3 and obtain the bound $\int \|\nabla f\| \le 2\pi\sqrt{d}$ from Equations (6) and (8). It remains to verify that $\int |\langle \nabla f, u \rangle| \le 2\pi$ also holds for each vector u. Using the Cauchy–Schwartz inequality, we obtain

$$\int |\langle \nabla f, u \rangle| \le 2\sqrt{\int \langle \nabla g, u \rangle^2} = 2\sqrt{\sum_i u_i^2 \pi^2} = 2\pi,$$

for our choice of g, and hence f. This completes the proof of Theorem 3.

4. A THREE-DIMENSIONAL CUBICAL FOAM

Although we have asymptotically solved the Cubical Foam Problem up to a small constant factor, in the physically natural case of $d = 3$, our construction does not improve on the Choe Prism, or even the cube. Here, we present an improved three-dimensional cubical foam, constructed via an ad hoc method.

The two-dimensional minimizer given by Choe in Figure 2 (left) can be described as follows: Start with a "base" facet centered at the origin; specifically, an edge from $(-s, s)$ to $(s, -s)$ for some parameter s. This already gives all vertices, by periodic extension. The hexagonal bubble is the convex hull of the two base points, their translates within $[0, 1)^2$, and their translates by $(1, 1)$. One chooses s to minimize the resulting surface area (perimeter).

We similarly construct a tiling shape B in three dimensions. We form a "base" facet centered at the origin, which is a regular hexagon, with vertices $\pm(0, -t, t)$, $\pm(-t, 0, t)$, and $\pm(-t, t, 0)$, for some $t \in (0, 1/3)$. Again, this gives all vertices, by periodic extension. We take B to be the convex hull of the 6 base points, along with their 6 translates within $[0, 1)^3$ and their 6 translates within $(0, 1]^3$. The polytope B has 14 facets: two opposing base regular hexagons, six larger "isosceles" hexagons, and six rectangles. An illustration is in Figure 4.

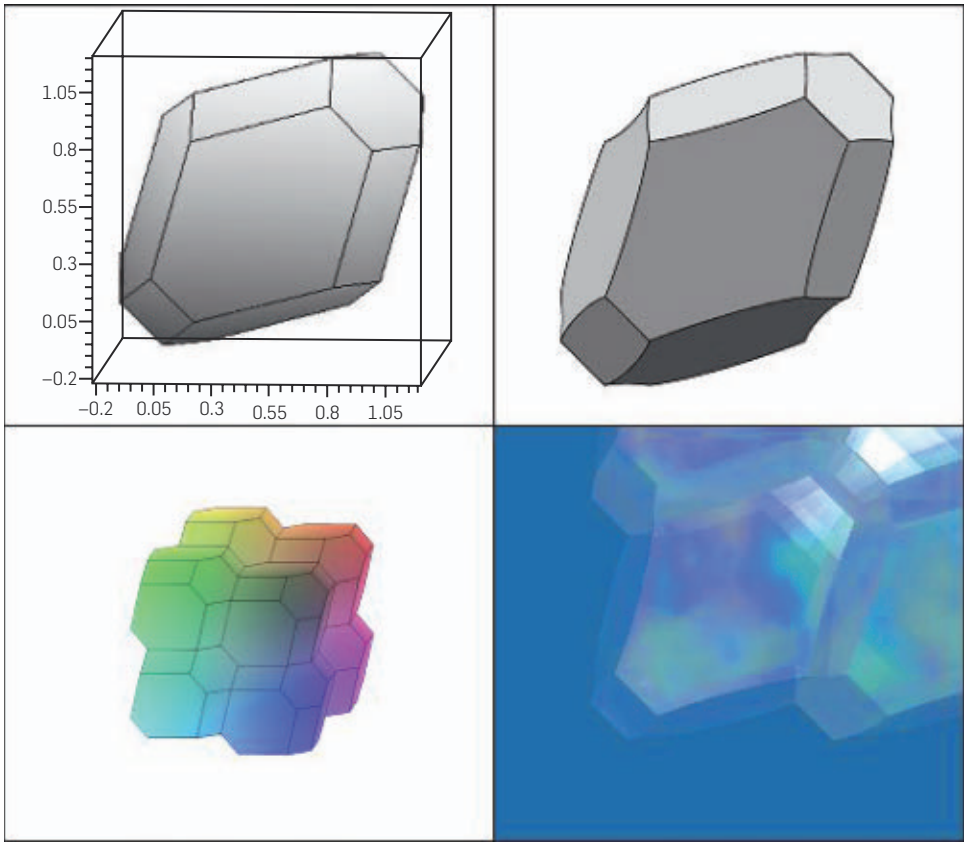
One may calculate that B has a surface area

$$6\left(\sqrt{3}t^2 + \sqrt{2}t\sqrt{1-4t+6t^2} + (1-t)\sqrt{1-2t+3t^2}\right),$$

which is minimized when $t \approx 0.1880$, having minimal value about 5.6121. This already beats the surface area of the Choe prism.

We can further improve this solution by letting B relax (within the torus $\mathbb{R}^3/\mathbb{Z}^3$) as a soap bubble. Using Brakke's

Figure 4. Our new three-dimensional cubical foam. (Top left) The unrelaxed tile. (Top right) The tile after it has relaxed according to Plateau's Rules. (Bottom left) The unrelaxed tile forming a foam according to the integer lattice. (Bottom right) Illustration of the relaxed foam as soap bubbles.



Surface Evolver,[7] we obtain the relaxed bubble B shown in Figure 4. We remark that it has slightly wavy faces and curved edges, and that the vertices have moved according to $t \approx 0.1814$. The surface area of B is slightly less than 5.602, according to Surface Evolver.

5. DISCUSSION

We have given a probabilistic construction of a cubical foam with near-spherical surface area. The construction uses ideas that are new to the study of foams, and is inspired by work on the limits of "hardness amplification" in certain computational optimization problems. Our construction gives the first suggestion that in high dimensions, optimal foams might not be derived from Voronoi cells and may be quite unlike polyhedra.

We have also given an algorithmic application of our foam's construction: a very "noise-resistant" procedure for rounding off vectors of d real numbers to integers. This discretization algorithm may not be practical for very large d, as Algorithm 1 is likely to run for a number of stages which is exponential in d. An important open problem is to find a coordinated discretization procedure with similar noise resistance, but taking time that grows only polynomially in d.

Finally, the construction of our cubical foam used randomness in an essential way; randomness is also used in other efficient high-dimensional constructions of foams (such as high-dimensional Kelvin foams). Although randomness is clearly required for noise-resistant coordinated discretization, it is an intriguing question as to whether it is necessary for the construction of foams, or whether explicit or derandomized constructions exist.

Subsequent to our work.[17] Alon and Klartag[2] gave a simpler derivation of our cubical foam via Cheeger's isoperimetric inequality; their analysis also shows that there exists a fixed parameter h that can be used as H_i throughout Algorithm 1. In other words, a good foam can be derived from the random translations of a single droplet of the form

$$D = \text{closure}\left(\left\{x \in \mathbb{R}^d : \prod_{i=1}^{d} (2\sin^2(\pi x_i)) > h\right\}\right).$$

However, it still remains unknown as to how to construct an explicit "spherical cube."

Acknowledgments

We thank N. Alon, K. Ball, H. Cohn, P. D'Ancona, M. Goresky, J. Håstad, L. Hoffman, J. R. Lee, A. Klivans, D. Micciancio, V. Milman, O. Regev, and J. Sullivan for their insights. We thank R. Nelson for his assistance with raytracing code. Kindler was supported in part by the Koshland fellowship and by the Binational Science Foundation (BSF). O'Donnell was supported in part by NSF grants CCF-0747250 and CCF-0915893, BSF grant 2008477, and Sloan, Okawa, and von Neumann fellowships. Rao was supported in part by NSF grant CCF-1016565 and a Sloan fellowship. Wigderson was supported by NSF grants CCF-0832797 and DMS-0835373.

References

1. Achlioptas, D., Naor, A., Peres, Y. Rigorous location of phase transitions in hard optimization problems. *Nature 435* (2005), 759–764.
2. Alon, N., Klartag, B. Economical toric spines via Cheeger's inequality. *1* (Sept. 18, 2009), 101–111.
3. Arnold, W.N., Lacy, J.S. Permeability of the cell envelope and osmotic behavior in *Saccharomyces cerevisiae*. *J. Bacteriol. 131*, 2 (1977) 564–571.
4. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M. Proof verification and the hardness of approximation problems. *J. ACM 45*, 3 (1998), 501–555.
5. Arora, S., Safra, S. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, *45*, 1 (1998), 70–122.
6. Ball, P. Science in culture: Beijing bubbles. *Nature 448*, 7151 (2007), 256.
7. Brakke, K.A. The surface evolver. *Exp. Math. 1*, 2 (1992), 141–165.
8. Butler, G. Simultaneous packing and covering in Euclidean space. *Proc. London Math. Soc. 3*, 4 (1972), 721–735.
9. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S. Approximating a finite metric by a small number of tree metrics. In *Proceedings of 39th Annual IEEE Symposium on Foundations of Computing* (1998), 379–388.
10. Choe, J. On the existence and regularity of fundamental domains with least boundary area. *J. Differ. Geom. 29* (1989), 623–663.
11. Feige, U., Kindler, G., O'Donnell, R. Understanding parallel repetition requires understanding foams. In *Proceedings of 22nd Annual IEEE Conference on Computational Complexity* (2007), 179–192.
12. Frank, F.C., Kasper, J.S. Complex alloy structures regarded as sphere packings. I. Definitions and basic principles. *Acta Cryst. 11*, 3 (1958), 184–190.
13. Hales, T.C. The honeycomb conjecture. *Disc. Comp. Geom. 25*, 1 (2001), 1–22.
14. Hales, T.C. A proof of the Kepler conjecture. *Ann. Math. 162*, 3 (2005), 1065–1186.
15. Holenstein, T. Parallel repetition: simplifications and the no-signaling case. In *Proceedings of 39th ACM Symposium on Theory of Computing* (2007), 411–419.
16. Holenstein, T. Parallel repetition: simplification and the no-signaling case. *Theory of Computing 5*, 1 (2009), 141–172.
17. Kindler, G., O'Donnell, R., Rao, A., Wigderson, A. Spherical cubes and rounding in high dimensions. In *FOCS* (2008), IEEE Computer Society, 189–198.
18. Kusner, R., Sullivan, J.M. Comparing the Weaire–Phelan equal-volume foam to Kelvin's foam. *Forma 11*, 3 (1996), 233–242.
19. Lee, J.R., Naor, A. Extending Lipschitz functions via random metric partitions. *Invent. Math. 160*, 1 (2005), 59–95.
20. Nielsen, L.E., Landel, R.F. Mechanical Properties of Polymers and Composites, 2nd edn, CRC Press, Boca Raton, 1993.
21. Plateau, J. Statique expÃl'rimentale et thÃl'oretique des liquides soumis aux seules forces molÃl'culaires, Gauthier-villars, Paris, 1873.
22. Rao, A. Parallel repetition in projection games and a concentration bound. In *Proceedings of 40th ACM Symposium on Theory of Computing* (2008), 1–10.
23. Raz, R. A parallel repetition theorem. *SIAM J. Comput. 27*, 3 (1998), 763–803.
24. Raz, R. A Counterexample to Strong Parallel Repetition. IEEE Computer Society, 2008, 369–373.
25. Santalo, L. Integral Geometry and Geometric Probability, 2nd edn, Cambridge University Press, Cambridge, 2002.
26. Thomson, W. On the division of space with minimum partitional area. *Philos. Mag. 24* (1887), 503–514.
27. Weaire, D., Phelan, R. A counter-example to Kelvin's conjecture on minimal surfaces. *Phil. Mag. Lett. 69*, 2 (1994), 107–110.

Guy Kindler (guy.kindler@weizmann.ac.il), School of Computer Science and Engineering, Hebrew University of Jerusalem.

Ryan O'Donnell (odonnell@cs.cmu.edu), Computer Science Department, School of Computer Science, Carnegie Mellon University.

Anup Rao (anuprao@cs.washington.edu), Computer Science and Engineering, University of Washington.

Avi Wigderson (avi@math.ias.edu), School of Mathematics, Institute for Advanced Study.

DOI:10.1145/2347736.2347758

Technical Perspective
Graph Embeddings and Linear Equations

By Bruce Hendrickson

ALGORITHMIC ADVANCES CAN come from the most unexpected places. The following paper by Koutis, Miller, and Peng is an elegant case in point. It describes an emerging approach to solving linear systems of equations that relies heavily on techniques from graph theory.

OK, I admit it. Solving systems of equations is one of those "supposed to be good for you'' topics that we all had to suffer through but most of us quickly forgot. But it also happens to be hugely important. For decades, scientists and engineers have simulated the world by solving linear systems. Systems with billions of variables are commonly solved, and this computation almost certainly consumes the majority of supercomputing cycles in the world. In recent years, linear systems have played a key role in page ranking, data analysis, recommendation systems and numerous other aspects of our data-centric world.

Back in high school we were all taught how to solve such systems by repeatedly subtracting one equation from another to eliminate a variable. Unfortunately, this simple approach takes $O(n^3)$ time for n equations with n variables. These days, large systems are usually solved using a different type of approach in which better and better approximations to the answer are computed until an acceptable tolerance is achieved. The traditional theory behind these *iterative methods* relies on numerical analysis, not a favorite class for most computer science students.

In recent years, a small cadre of researchers has been building a novel theoretical framework for analyzing a very important class of linear systems. With this new approach, the performance of an algorithm can be evaluated using techniques from graph theory—a discipline quite different from numerical analysis. The authors describe the quirky history of this line of research, starting with the conceptual breakthrough by Vaidya and continuing with the theoretical heavy lifting provided by Spielman, Teng, and collaborators.

The connection between graph embeddings and linear solvers described in the following paper is a perfect example of cross-disciplinary mixing.

Beyond its novelty value, this research affords insights that traditional approaches cannot. Numerical analysis techniques (and the algorithms that come from them) often have to assume that the system of equations comes from a grid with very simple connectivity. These assumptions are clearly not true for data-centric applications like those mentioned above. By contrast, the new techniques are quite general and impose no constraints on the structure of the system of equations. They lead to algorithmic approaches and analysis tools that are unlike anything that has come before. In addition, theoretical computer scientists think about computational complexity in a different way than numerical analysts do, and so provide a new framework for algorithmic analysis.

This is a challenging area of research because it cuts across disciplines and requires a depth of expertise in multiple areas. This paper provides an accessible introduction to this rapidly evolving set of ideas. Its key contribution is a significantly simpler algorithm that retains desirable theoretical properties. Unfortunately, "simpler" is still probably too complex for the numerical computing community to adopt. But this work brings these ideas much closer to practical realization.

When unanticipated connections between fields are uncovered, we get to see the familiar in strange and fresh ways. Our understanding is deepened and new insights emerge from the fusion of alternative perspectives. The connection between graph embeddings and linear solvers described in this paper is a perfect example of this cross-disciplinary mixing. The line of work it describes has already resulted in improved algorithms for long-standing graph problems, and spun off numerous juicy theoretical questions. Almost lost in this excitement is the implication for scientific computing of provably near-optimal solvers for a hugely important class of linear systems. This is important work whose full ramifications are still emerging.

Bruce Hendrickson (bahendr@sandia.gov) is Senior Manager for Computational Sciences and Mathematics at Sandia National Laboratories. He is also an affiliated faculty member in the computer science department at the University of New Mexico.

DOI:10.1145/2347736.2347759

A Fast Solver for a Class of Linear Systems

By Ioannis Koutis, Gary L. Miller, and Richard Peng

Abstract

The solution of linear systems is a problem of fundamental theoretical importance but also one with a myriad of applications in numerical mathematics, engineering, and science. Linear systems that are generated by real-world applications frequently fall into special classes. Recent research led to a fast algorithm for solving symmetric diagonally dominant (SDD) linear systems. We give an overview of this solver and survey the underlying notions and tools from algebra, probability, and graph algorithms. We also discuss some of the many and diverse applications of SDD solvers.

1. INTRODUCTION

One of the oldest and possibly most important computational problems is that of finding a solution to a system of linear equations. There is evidence that humans have been solving linear systems to facilitate economic activities since at least the first century AD. With the advent of physical sciences and engineering, linear systems have been, for centuries now, a central topic of applied mathematics. And over the last two decades, the digital revolution has expanded the use of linear system solvers to applications of surprising variety.

Many of these new applications typically model entities and their relationships as networks, also known as graphs, and use solvers to extract information from them. The resulting linear systems frequently obey simple constraints which classifies them as **symmetric diagonally dominant** (SDD).

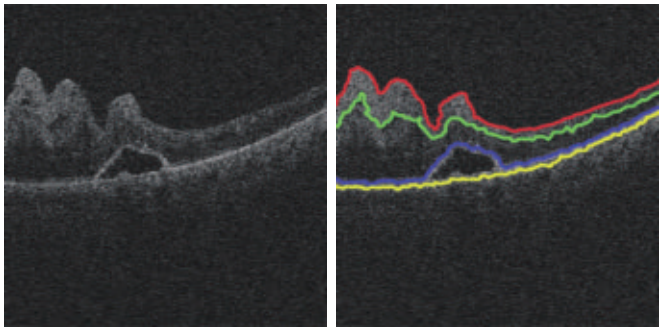
An example of an area where such systems arise is in the analysis of social networks. Such networks can be represented as a set of links connecting people; an example is shown in Figure 1. A natural question to ask is how "close" are two persons in the network. Purely graph-based methods measure either the length of the shortest path or the maximum number of disjoint paths between the two nodes, but not both. To take both of these quantities into account we can view the network as an electric circuit with each connection corresponding to an electrical wire. Hooking a battery at the two nodes of interest and measuring the resistance of the entire network gives a quantity known as the **effective resistance**, which can be used as a "proximity" measure. Since the electrical network is not physically available, we cannot measure the effective resistance. We can, however, compute it by solving an SDD linear system.

The above example is only one of many instances of inference on a graphical model. Similar methods are applicable in a wide range of problems, such as measuring the importance of specific proteins in protein–protein interaction networks[14]; the link prediction problem in social networks[13]; or even problems where graphs arise less directly, such as segmenting the image shown in Figure 2.

More intricate uses of electrical networks have been discovered in the context of classical graph optimization problems, with the recent network flow algorithm by Christiano et al.[5] standing out as an algorithmic breakthrough. The algorithms reduce the problem to not just one network,

Figure 1. Representing a social network as a graph.

Figure 2. Segmentation of medical scans.[21]



This paper is based on two previous works: "Approaching Optimality for Solving SDD Linear Systems," which appeared in the *Proceedings of FOCS '10* and "A Nearly-*m* log *n* Time Solver for SDD Linear Systems," which appeared in the *Proceedings of FOCS '11*.

but to a sequence of networks via successive readjustment of edges. In these algorithms, some of the resulting systems are significantly harder than "typical" instances, capturing—in some sense—the hardness of the optimization problems themselves.

Current SDD solvers are empirically fast for some engineering applications, but they are not able to efficiently solve most cases of SDD linear systems. Besides these practical limitations, the fact that existing SDD solvers lack guarantees on arbitrary instances limits their implications to the theory of algorithms as well.

These factors underline the need for "mathematically certified" solvers that are **provably** fast for arbitrary SDD linear systems, independently of their origin, be it—for instance—social or protein networks. This paper describes our state of the art solver for SDD linear systems.

1.1. A glimpse at the solver

The class of SDD linear systems arises in particular in the study of electrical networks, which provide us with a concept crucial to understanding how our algorithm works: the effective resistance between two points in a network. In addition, the connection to networks enables adopting a second alternative view of our linear system, as a **weighted graph**. We give the details in Section 2.

We then move to the **algebraic component** of our solver. The approximate solution of linear systems via iterative methods is a topic not commonly encountered in computer science but thoroughly studied in the context of numerical linear algebra and scientific computation. Section 3 explains iterative methods via an analogy with the computation of the inverse of a real number in a calculator with a broken division key, where only addition and multiplication operations are available. This leads us to preconditioning, a term first used by Alan Turing. In the graph theoretic context, preconditioning provides a measure of **similarity** between graphs. This measure is used to formalize **design conditions** sufficient for the construction of a fast iterative method.

What distinguishes our solver from classical iterative methods is its **combinatorial component** and specifically the use of graph theoretic algorithms. It was understood before our work that the key to a fast solver is finding a subgraph (the preconditioner) which is similar to a given graph, but has substantially fewer edges.[20] Our contribution is a conceptually simple procedure for constructing good preconditioners, this is the topic of Section 4.

The base of our preconditioner is a spanning tree of the input graph, in other words a minimally connected subgraph. Our algorithm needs a special type of spanning tree called a low-stretch tree (LSST) which we describe in Section 4.1. This can be found using very sophisticated but fast algorithms.

To get a better preconditioner, we perform random sampling: each edge of the input graph is put into the preconditioner with a specified probability. It was known that the effective resistance between the two endpoints of each edge provides a good sampling probability for it.[18] Unfortunately the problem of computing the effective resistance seems to require solving an SDD linear system, which is the problem we are trying to solve in the first place.

Our main contributions are two ideas that allow us to circumvent this "chicken and egg" problem. The first idea is to use an upper estimate on the effective resistance for each edge. The second idea is to compute these estimates on a modified graph, in which the estimates are sufficiently good. The modification is in fact quite simple; we find an LSST of the graph and increase the weight all of its edges. To compute the upper estimate for the effective resistance of an edge in the modified graph we only use the edges of the LSST. A key side effect of this modification is that the number of non-tree edges in the preconditioner is much less than the number of edges in the original graph. In this way we meet the known design conditions and obtain the faster solver.

2. NETWORKS, SYSTEMS, SOLVERS

Let us consider the problem of finding a voltage setting given the desired net current flow at each of the vertices. A simple three-node example of an electric network is depicted in Figure 3. The inverse of the resistance of wire, also known as **conductance**, is a direct analogue to the edge weight in a graph; because of that we choose to label each wire by its conductance rather than its resistance. Setting the **voltages** of the vertices to some values leads to an **electrical flow** through the edges. There are two fundamental principles governing this voltage setting. (a) Kirchhoff's law, which states that with the exception of the vertices where current is injected/extracted, the net flow at each vertex is zero. (b) Ohm's law, which states that the current on an edge equals the voltage difference between its endpoints times the conductance of the wire.

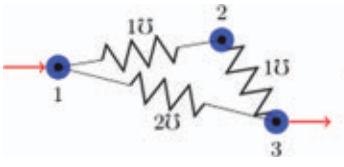
As an example consider the network given in Figure 3 where we set the voltages at the three vertices to be x_1, x_2, and x_3 respectively. By Ohm's law we get that the current flows along edges $1 \rightarrow 2$ and $1 \rightarrow 3$ are $1 \cdot (x_1 - x_2)$ and $2 \cdot (x_1 - x_3)$, respectively. Therefore the amount of current we will need to inject into vertex 1 to maintain these voltages is:

$$1 \cdot (x_1 - x_2) + 2 \cdot (x_1 - x_3) = 3x_1 - x_2 - 2x_3$$

Identities for the required current entering/leaving vertices 2 and 3 can also be derived similarly. Therefore, if we want one unit of current to enter at vertex 1 and leave at vertex 3, the voltages will need to satisfy the following system of linear equations:

Using more compact notation, linear systems assume the form $Ax = b$ where x is a $n \times 1$ column vector of unknowns,

Figure 3. A simple network and linear system.



$$\begin{aligned} 3x_1 - 1x_2 - 2x_3 &= 1 \\ -1x_1 + 2x_2 - 1x_3 &= 0 \\ -2x_1 - 1x_2 + 3x_3 &= -1 \end{aligned}$$

also called variables, b is a $n \times 1$ column vector of real numbers, and A is an $n \times n$ matrix containing the coefficients of the variables. For example, the above linear system can be expressed in matrix form as:

$$\begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}. \quad (2.1)$$

Note that each off-diagonal entry is the negation of the conductance of the resistor connecting its two vertices, and each diagonal entry is the sum of the conductances of all resistors incident to the corresponding vertex. Since resistive networks can also be viewed as undirected graphs, this type of matrix is known as a **Graph Laplacian** and we will rely on this connection extensively in our algorithm. SDD matrices are a further generalization of graph Laplacians. However, an SDD system can be easily transformed into a Laplacian system (e.g. see Koutis et al.[9]) and so we will restrict our attention entirely to graph Laplacians.

Once we're able to obtain the voltages at each vertex, we can also compute the **effective resistance** between two vertices. Intuitively, this notion can be viewed as thinking of the entire network as a single electrical component. Then by Ohm's law the voltage drop required to send 1 unit of current corresponds to the resistance of the component. In our example, the effective resistance between vertex 1 and 2 is $x_1 - x_3 = 2/5$. Formally, this value equals $v_s - v_t$ from the solution of the linear system $Lv = \varphi$, where φ is zero everywhere except in the two entries corresponding to the nodes s and t, for which we set $\varphi_s = 1$ and $\varphi_t = -1$. As we will see later, this metric is not only used for network analytics, but also plays a crucial role in our solver itself.

2.1. Solvers and their speed

Despite its long history, the problem of constructing good solvers is considered far from being solved, especially in terms of speed. The speed of algorithms is commonly measured in terms of the input size. In the case of general linear systems on n variables, the matrix has size n^2. However, matrices are often **sparse**, that is, most of their entries are equal to zero. Because of this we can easily "compress" them to size proportional to the **number of non-zeros**, denoted by m. The best case scenario, which remains entirely consistent with our current understanding, is that linear systems can be solved with $O(m)$[a] operations.

It's fair to say that Gaussian elimination is the most well-known method for solving linear systems. It runs in $O(n^3)$ time and it is known as a direct method in that, if the arithmetic operations are performed exactly then one gets the exact solution to the system. Although this exponent of 3 has been decreased to as low as 2.37,[24] direct methods in general require storing n^2 entries, creating a natural bottleneck that limits us to systems with a few thousand variables.

One possible remedy to the space and time limitations of direct methods are **iterative methods**. These compute progressively better approximate solutions by only performing **matrix–vector products** and other simpler vector operations.

One of the most important iterative methods is Conjugate Gradient, discovered by Lanczos, Hestenes, and Stiefel in the early 1950s. This method works for arbitrary symmetric positive definite systems, a class that includes SDD systems. While it requires only $O(m)$ space, it is understood that its running time—in its original form—can be large.

Strong evidence that iterative methods can combine low space requirements and very fast running time was provided by a family of iterative methods known as multigrid.[22] Multigrid solvers have an $O(m)$ running time guarantee albeit for restricted and well-structured systems that arise in scientific computing.

The solver we will review in this paper is also an iterative method. It is the culmination of a line of work initiated by Vaidya,[23] which was brought to near-completion with the breakthrough achievement of Spielman and Teng[19]: the first solver that runs in time $O(m \log^c n)$ for any graph Laplacian, where c is a large constant. The work discussed here, summarized in the following claim from Koutis et al.,[10] provides a conceptually simpler, faster and more practical algorithm.

THEOREM. SDD systems can be solved in $\tilde{O}(m \log n \log(1/\epsilon))$ time,[b] where ϵ is a standard measure of the approximation error.

3. THE ALGEBRAIC COMPONENT

3.1. Iterative methods: Division-free inversion

Our way towards the faster solver starts with a basic and perhaps seemingly unrelated question: is it possible to compute the inverse $1/\alpha$ of a number α using a calculator with a broken division key?

To answer the question we can invoke a basic identity that tells us that when $0 < \alpha < 2$, $1/\alpha$ equals the following infinite sum:

$$\begin{aligned} 1/\alpha &= 1/(1-(1-\alpha)) \\ &= 1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \ldots \end{aligned} \quad (3.2)$$

Of course, computing an infinite sum is not possible. But keeping a number of terms will give us an **approximation** of $1/\alpha$; the more terms we keep the better the approximation.

But how is this related to the problem of solving linear systems? Matrices borrow several of the usual properties of scalar numbers. When A is symmetric, its inverse A^{-1} also satisfies the identity in 3.2, substituting A for α, A^{-1} for $1/\alpha$ and the identity matrix I for the number 1. Furthermore, if we want an approximation to $x = A^{-1}b$[c] we can actually avoid entirely taking powers of the matrix; the ith approximate vector

$$x^{(i)} = (I + (I - A) + \ldots (I - A)^i)b$$

[a] We use $f(n) = O(g(n))$ to denote $f(n) \le c \cdot g(n)$ when $n \ge n_0$ for some constants c and n_0.

[b] The $\tilde{O}()$ notation hides a $\log \log n$ factor.

[c] If A^{-1} does not exist, as in the case of Laplacians, we use A^{-1} to denote the pseudoinverse as well.

can be produced with i applications of the following simple recurrence:

$$x^{(0)} = 0$$
$$x^{(i+1)} = b + (I - A)x^{(i)} \text{ for } i > 0.$$

It can be seen that each step involves a matrix–vector multiplication by A. This is the simplest among **iterative** methods that in general attempt to approximate the solution of a linear system using only a sum of results from a series of matrix–vector multiplications.

3.2. Preconditioning

So far, our replacement for the division button is of rather restricted value, since it only works when $0 < \alpha < 2$, and can converge very slowly when α is close to 0 or 2. One way to extend our method and to speed up its convergence is to add a "restricted division" key to our calculator. This key allows us to "divide" by a fixed scalar β of our choice, which in the matrix setting corresponds to a matrix–vector product involving the inverse, B^{-1} of a matrix B. We can speed up our algorithm by pressing the "restricted division" button after each matrix–vector multiplication by A, giving the following modified recurrence known as **preconditioned Richardson iteration**:

$$x^{(0)} = 0$$
$$x^{(i+1)} = B^{-1} b + (I - B^{-1}A)x^{(i)} \text{ for } i > 0.$$

The matrix B is known as the **preconditioner** and instead of solving the system $Ax = b$, we are essentially solving the **preconditioned** system: $B^{-1}Ax = B^{-1}b$. It is worth emphasizing that each step of this method involves a matrix–vector multiplication by A followed by a "division" by the matrix B.

3.3. Measuring similarity between matrices

Looking back at the single variable recurrence, the critical condition for its convergence is $0 < \alpha < 2$. An extension of it is needed in order to analyze preconditioned iterative methods involving matrices A and B. For matrices A and B, we say $A \preceq B$ when for all vectors x we have

$$x^T Bx \leq x^T Ax.$$

Unlike the situation with scalars, this ordering is only "partial". Even for size 2 diagonal matrices, it is possible that neither $B \preceq A$ nor $A \preceq B$ holds. But when A and B are symmetric, there will be numbers κ_{max} and κ_{min} such that:

$$\kappa_{min} A \preceq B \preceq \kappa_{max} A.$$

We will say that B is a κ-approximation of A, where $\kappa = \kappa_{max}/\kappa_{min}$. In this case, after introducing additional scaling factors, it can be shown that the preconditioned Richardson's iteration gives a good approximation in $O(\kappa)$ iterations. There are iterative methods with faster convergence rates, and—as we will see—our solver relies on one of them, known as Chebyshev iteration.

3.4. Interpreting similarity

It is interesting to see what this measure of similarity means in the context of electrical networks, that is when both A and B are Laplacians. The quadratic form

$$x^T Ax$$

is equal to the **energy dissipation** of the network A, when the voltages at vertices are set to the values in the vector x. Then, the network B is a κ-approximation of the network A whenever for all voltage settings, B dissipates energy which is within a κ factor of that dissipated by A.

So, roughly speaking, **two networks are similar when their "energy profiles" are similar**. This definition does not necessarily correspond to intuitive notions of similarity; two networks may appear to be very different but still be similar. An example is shown in Figure 4.

3.5. What is a good preconditioner?

Armed with the measure of similarity, we are now ready to face the central problem in solver design: how do we compute a good preconditioner?

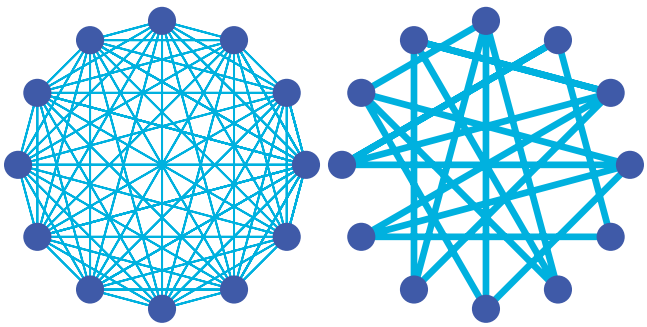
To deal with the question we must first understand what properties are desirable in a preconditioner. A big unknown in the total running time is the cost incurred by the limited division button that evaluates $B^{-1}y$.

To evaluate $B^{-1}y$ we do not need to compute B^{-1}. We can instead solve the system $Bz = y$; the solution z will be equal to $B^{-1}y$. Clearly, we would like to solve systems involving B as quickly as possible. At the same time we would like the number of iterations to be as small as possible, since each of them requires at least m operations. Furthermore, a slow algorithm for computing the preconditioner B would defeat the purpose of a fast solver. So, we should also be able to find B quickly. Balancing these three opposing goals makes the problem quite challenging.

3.6. Recursion and design conditions

In order to solve the linear system fast, we will need a preconditioner B which is an **extremely good** approximation of A **and** can be solved in linear time. Satisfying both these requirements is too much to hope for. In practice, any good graph preconditioner B won't be significantly easier to solve

Figure 4. Two similar graphs: A complete graph and a random small subset of its edges, made heavier.



comparing to A. As a result, there is no hope that preconditioned Richardson's iteration or any other preconditioned method can lead to fast solvers.

The remedy to the problem is **recursion**. In a recursive preconditioned method, the system in the preconditioner B is not solved exactly but approximately, via a recursive invocation of the same iterative method. We now have to find a preconditioner for B, and furthermore a preconditioner for it and so on. This produces a **multilevel hierarchy** of progressively smaller graphs.

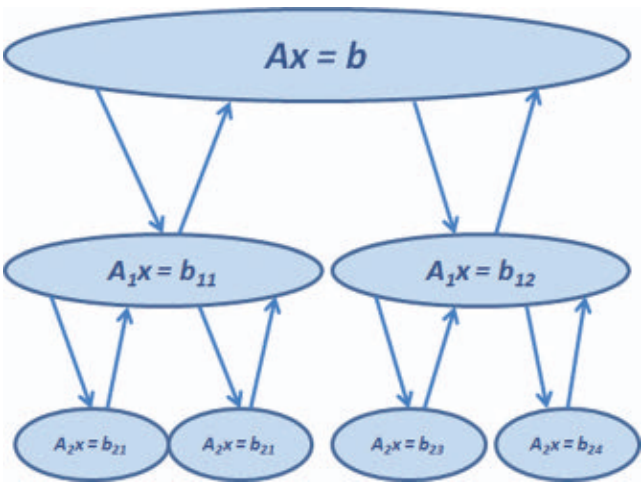
Rohklin, Spielman, and Teng[19] analyzed a recursive iterative method which moves between levels of the hierarchy as shown in Figure 5; for each visit at level i, the algorithm makes k visits to level $i+1$. Every time the algorithm returns to the ith level it performs **matrix–vector multiplications** with the graph A_i, and other simpler operations; so the work is proportional to the number of edges of A_i. To keep the **total work** as small as possible, that is $O(km)$, the graphs in the hierarchy must get smaller sufficiently fast. In particular, it is sufficient that the graph on level $i+1$ is smaller than the graph on level i by a factor of $1/(2k)$.

However, the algorithm must converge within the $O(km)$ time bound. To achieve this, the iterative method analyzed within this recursive framework is a method known as Chebyshev iteration. It requires only $O(k)$ iterations, when B is a k^2-approximation of A, as compared to the $O(k^2)$ iterations required by Richardson's iteration. Using this fact Spielman and Teng arrived at **design conditions** that are sufficient for a fast solver.[20] It was actually shown that a good algorithm for preconditioning **extends** to a good solver. More specifically, assume that for some fixed value C and any value of k, we have a fast algorithm that given A, produces a k^2-approximation with $n + C \cdot m/k$ edges. Then we automatically get a solver that runs in time $O(k \cdot m)$.

Carefully checking the above statement, we realize that there is a slight discrepancy. If m is close to n and k is large, then $n + C \cdot m/k$ will be bigger than m, which contradicts our promise for a multilevel hierarchy of progressively smaller graphs. However, as observed by Vaidya,[23] when m is almost the same n, the graph has several "tree-like" parts, and these can be reduced via a "partial" Gaussian elimination that runs in $O(m)$ time. So whenever this case appears, it makes sense to first run partial elimination. This will decrease the vertex count n, leading to a much smaller instance on which recursion is applicable.

The multilevel analysis of Spielman and Teng is significant not only for its actual algorithmic value but also the conceptual **reduction** of the multi-level solver design problem to a well-defined two-level preconditioning problem, allowing us now to focus on the combinatorial component of the solver.

Figure 5. The sequence of calls of a recursive iterative method. The matrix is fixed at each level.



4. THE COMBINATORIAL COMPONENT

Although graph theory has been used to speed up direct methods, it took a paradigm-shifting idea of Pravin Vaidya to enter a **systematic study** of using graph theory for iterative methods. In particular, Vaidya suggested the use of a **spanning tree** of the graph A as a **building base** for the preconditioner B. A spanning tree of a graph is a connected subgraph without loops. The choice of a tree stems from the observation that linear systems whose matrix is the Laplacian of a tree can be solved in $O(n)$ time via Gaussian elimination. Adding a few edges of A back onto the tree returns a preconditioner B which can only be better than the tree, while still being relatively easy to solve. Vaidya's idea set forth two questions: (i) What is an appropriate base tree? (ii) Which off-tree edges should be added into the preconditioner?

While these questions seem to be interrelated, we can actually address them separately.

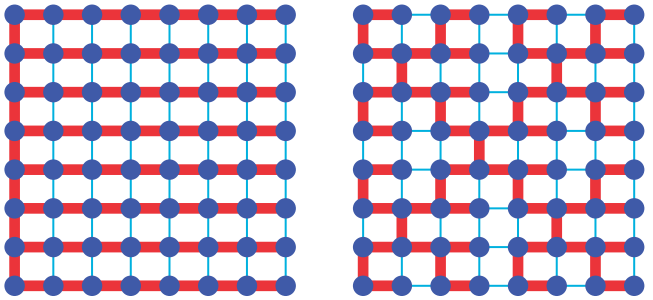
4.1. Low-stretch: The base spanning tree

The goal of finding a preconditioning tree B which is as similar as possible to the graph A led Vaidya to a natural idea: use a tree which concentrates the maximum possible weight from the total weight of the edges in A.

The maximum-weight spanning tree idea led to the first non-trivial results, but does not suffice for our algorithm. In fact, the weight measure does not distinguish trees in unweighted graphs, where all trees have equal weight.

The key to finding a good tree to use as a building base is the notion of **stretch**: For every edge (u, v) of the graph, there is a unique "detour" path between u and v in a tree T. The stretch of the edge with respect to T is equal to the distortion caused by this detour, and in the unweighted case, it is simply the length of the tree path. This notion generalizes naturally to the weighted case, which we will formalize in Section 4.3. The **total stretch** of a graph A with respect to a tree T is the sum of the stretches of all the off-tree edges. A **low-stretch tree** (LSST) is one for which we have a good upper bound on the total stretch. So, at a high level, an LSST has the property that it provides good (on average) "detours" for edges of the graph. A concrete example on a larger unweighted graph is given in Figure 6, where the tree on the right has lower total stretch, and as it turns out is a better base tree to add edges to.

Figure 6. Two possible spanning trees of the unweighted square grid, shown with red edges.



Trees for the grid: a quantitative example

The two spanning trees of the 8×8 grid shown in Figure 6 can guide our understanding of the general $\sqrt{n} \times \sqrt{n}$ grid.

In the tree on the left, for each vertical edge beyond column $\sqrt{n}/2$, at least $\sqrt{n}$ horizontal edges are needed to travel between its endpoints; that means that its stretch is at least $\sqrt{n}$. So the $n/2$ edges in the right half of the square grid contribute a total stretch of $n^{1.5}$.

In the tree on the right, all edges along the middle row and column still have stretch $O(\sqrt{n})$. However, the middle row and column only have $O(\sqrt{n})$ edges and so they contribute only $O(n)$ to the total stretch. Recall that all we need is a low total stretch, so a small number of high-stretch edges is permitted. Having accounted for the edges in the middle row and column, the argument can then be repeated on the four smaller subgraphs of size $n/4$ formed by removing the middle row and column. These pieces have trees that are constructed similarly, leading to the recurrence

$$\text{TotalStretch}(n) = 4 \times \text{TotalStretch}(n/4) + O(n).$$

Its solution is $\text{TotalStretch}(n) = O(n \log n)$.

A generalization of this type of "accounting", that keeps the number of high stretch edges small, forms the basis of the current state-of-the-art algorithms.[1]

Algorithms for the computation of LSSTs were first studied in an unrelated context,[2] where it was shown that any graph contains a spanning tree with total stretch $O(m^{1+\epsilon})$; the tree can be found in $O(m \log n)$ time. The total stretch was lowered to $O(m \log^2 n)$ in Elkin et al.,[6] and further to $\tilde{O}(m \log n)$ in Abraham and Neiman,[1] giving the following theorem.

THEOREM. Every graph has a spanning tree of total stretch $\tilde{O}(m \log n)$. The tree can be found in $\tilde{O}(m \log n)$ time.

Boman and Hendrickson first introduced LSSTs as stand alone preconditioners in 2001. This was a catalyst to subsequent progress, which used LSSTs as a base for building even more intricate preconditioners. In fact, LSSTs are indispensable components of all nearly-linear time SDD system solvers. It is worth pointing out that while LSSTs were originally conceived as potentially good two-level preconditioners, their full power in the context of multilevel solvers was not realized until our work, which we describe in Section 4.3.

4.2. Sparsification

Spielman and Teng's[19] main contribution was a "tour de force" algorithm for finding a preconditioner that's the LSST plus a small number of edges. It took many by surprise as it yielded the first nearly-linear time SDD solver.

Describing the algorithm is out of the scope of this article, but it is worth noting its two enhancements over previous approaches. First, instead of just adding off-tree edges from A back onto the tree, the algorithm **re-weights** them. The tree edges may themselves be re-weighted in the final preconditioner B. Second, the procedure for adding edges in B is not deterministic but **randomized**, as it contains a process for sampling edges from A.

However the major conceptual and technical contribution of Spielman and Teng that formed the cornerstone of their solver was a **sparsification** algorithm. They showed that every graph A has a 2-approximation B which has $O(n \log^c n)$ edges for some large constant c. The graph B is called the sparsifier and, of course, it can be used as a preconditioner when A is dense.

After the first sparsification result, progress towards faster SDD solvers took a detour through the study of spectral sparsification as a stand-alone problem. Works by Batson, Kolla, Makarychev, Saberi, Spielman, Srivastava, and Teng led to nearly-optimal spectral sparsifiers, albeit at the cost of much higher running time. These results were motivated by the work of Spielman and Srivastava,[18] who gave an extremely simple algorithm for finding spectral sparsifiers with only $O(n \log n)$ edges. Their algorithm, as well as the Spielman–Teng spectral sparsification algorithm builds upon a framework established by Benczur and Karger for sampling and re-weighting a graph.

The framework requires positive numbers t_e assigned to each edge, corresponding to the relative probabilities of sampling them. It calculates the sum of these numbers, $\sum_e t_e$ and proceeds for $O(t \log n)$ rounds. In each round one new edge is added to the sparsifier B. The edge is picked randomly with replacement among the m edges of A, but not in a "fair" way. An edge e is picked with relative probability t_e, which equates to a probability of $p_e = t_e/t$. Once an edge is picked, it is added to B with weight scaled down by a factor of $O(t_e \log n)$. Furthermore, if an edge is picked twice or more during this process, each new copy is added as a parallel edge, making B potentially a multi-graph.

Understanding re-weighting

While it may appear complicated, the re-weighting choice is quite natural. The reasoning is that the "expected value" of B should be A itself on an edge-to-edge basis. In other words, the average of many B's output by the algorithm should be A itself.

Spielman and Srivastava found the probabilities that give sparsifiers with the fewest number of edges with the help of some experimentation. Amazingly, the answer turned out to be related to the effective resistance of the edge, specifically $t_e = w_e R_e$. With hindsight, it is interesting to reflect about the natural meaning of effective resistance. If there is a wire of resistance $r_e = 1/w_e$, between i and j, the effective resistance R_e will in general be smaller than r_e because most probably there will be other network connections to accommodate the flow; this is known as Rayleigh's monotonicity theorem. The extreme case $w_e R_e = 1$ occurs only when there is no other route between i and j except the wire joining them. In this situation, the edge (i, j) is crucial for the network. On the other hand if $w_e R_e$ is very small, there must be significant alternative network connections between (i, j). Therefore, the product $w_e R_e$ as a measure of the importance of a wire. Using tools from modern matrix theory,[15] Spielman and Srivastava proved that this algorithm does return a good spectral sparsifier with high probability. Combining with the fact that $t = \sum_e w_e R_e = n - 1$ yields the overall number of edges: $O(n \log n)$.

Despite being a major improvement in the theory of graph sparsification, the algorithm of Spielman and Srivastava did not accelerate the SDD solver as current methods for quickly computing effective resistances require the solution of linear systems. The guarantee of $O(n \log n)$ edges is also hard to connect with the $n + C \cdot m/k$ edges needed by the design condition. However, it is fair to say that their result cleared the way to our contribution to the problem.

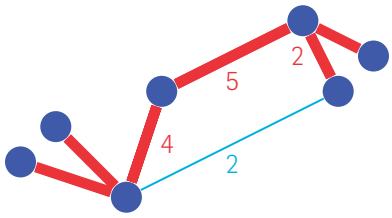
4.3. Which off-tree edges?

If we cannot effectively compute effective resistances, can we at least approximate them quickly, even poorly? A closer look at the matrix concentration bounds allows us to relax this goal a bit further: the sampling algorithm described in Section 4.2 can be shown to work with any choice of t_e, as long as $t_e \geq w_e R_e$. The observant reader may notice that the expected number of times e is picked is $O(t_e \log n)$, so increasing t_e only results in more copies of e being picked without affecting the expectations of all other edges.

The intuition that the low-stretch spanning tree must be part of the preconditioner leads us to taking tree-based estimates R'_e for the effective resistances R_e. In particular, for an off-tree edge e we let R'_e be the sum of the resistances along the unique path between the endpoints of e in the tree, as shown in Figure 7. By Rayleigh's monotonicity theorem, we know that this estimate will be higher than the actual R_e. This leads to the tree-based sampling probability for an off-tree edge e being proportional to $t_e = w_e R'_e$. Furthermore, if we keep the entire tree in B, we can modify the sampling algorithm presented in Section 4.2 to only sample off tree edges. Then the total number of off-tree (multi) edges sampled in B is $O(t \log n)$ where t is the sum of all t_es, which in turn. This brings us to the question: how big is t?

This question leads us back to the discussion of low-stretch spanning tree and the definition of **stretch** for the general weighted case: if we view the length of an edge e as the inverse of its weight, then its stretch equals

Figure 7. The effective resistance R'_e of the blue off-tree edge in the red tree is 1/4 + 1/5 + 1/2 = 0.95. Its stretch $w_e R'_e$ is (1/4 + 1/5 + 1/2)/(1/2) = 1.9.



to $w_e R'_e$.[d] Therefore, t is the **total stretch** of the off-tree edges with respect to the tree. Then, using the low-stretch spanning tree of the Theorem in Section 4.1, we can upper bound t by $O(m \log n)$. Recall that the number of samples will be $t \log n$ and so it appears that we do not gain much from the sampling process unless the graph A has a very special tree.

Our key idea is to make a special graph $\tilde{A}$ out of A. We do so by **scaling up**, by a factor of κ, the weights of edges of a low-stretch spanning tree in A. For an edge that's not part of the tree, its weight does not change, while the tree path connecting its endpoints is now heavier by a factor of κ. So the stretch decreases by a factor of κ and the total stretch of these edges becomes $t = O((m \log n)/\kappa)$. Now, consider what happens if we sample the off-tree edges in $\tilde{A}$. The output B will be a 2-approximation of $\tilde{A}$. On the other hand, the graph $\tilde{A}$ is a κ-approximation to A, and by transitivity B is 2κ-approximation to A. Also, the number of non-tree edges sampled will be $O(t \log n) = O((m \log^2 n)/\kappa)$. Adding in the $n - 1$ tree edges gives a total of $n + O((m \log^2 n)/\kappa)$ edges in B. Recall that the two-level **design conditions** stated in Section 3.6 require a k^2-approximation with $n + C \cdot m/k$ edges in order to obtain a running time of $O(k \cdot m)$. So by setting κ to $O(\log^4 n)$, we meet the conditions with $k = O(\log^2 n)$ and arrive at our first result:

THEOREM[9]. SDD systems can be solved in $\tilde{O}(m \log^2 n \log(1/\epsilon))$ time, where ϵ is a standard measure of the approximation error.

As it turned out, the low-stretch spanning tree is not only a good base tree, but also tells us which off-tree edges should go to the preconditioner. Our faster, $O(m \log n)$ time algorithm will come via an even better understanding of the properties of the tree.

4.4. The final push: Low-stretch spine

Assume that we are given a graph A, found its LSST T_A, and based on it, computed the preconditioner B. Then the $O(m \log^2 n)$ time solver algorithm dictates that we recursively do the same with B. But do we really have to scrap T_A and find another LSST T_B? After all, it may be the case that T_A is a LSST of B, or close to being one.

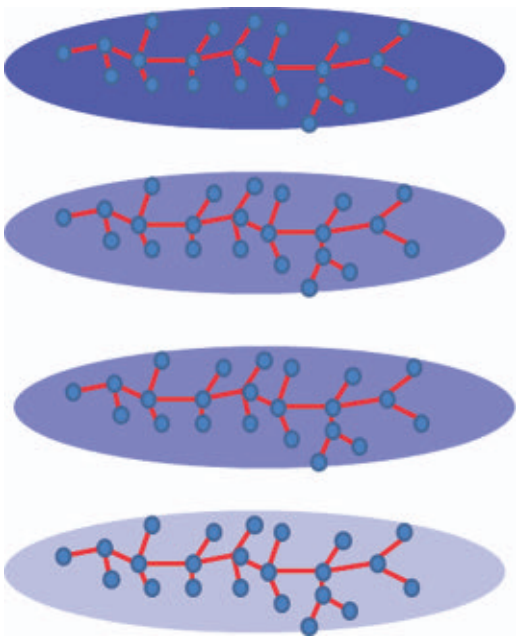
[d] An alternate view is that the stretch of e is the weighted length of the tree path between e's end points divided by e's own length.

What the $O(m \log^2 n)$ algorithm[9] missed is the observation that we can **keep sampling based on the same tree**, gradually generating **all** levels of the multilevel hierarchy, until what is left is the tree itself. This justifies thinking of a low-stretch spanning tree as a graph **spine**, and is depicted in Figure 8.

When the sparsifier B is viewed as a graph, it is possible for some of its edges to have high stretch. However, a more careful reexamination of the sampling algorithm shows that these edges are the result of an edge being sampled many times. From this perspective, these heavy edges are in fact many multi-edges, each with low stretch. Therefore, if we process these multi-edges separately, the tree T_A will be a low-stretch spanning tree in B, and the higher edge count is still bounded by the number of rounds made by the sampling algorithm. This allows us to use T_A as a low-stretch spanning tree and sample the off-tree edges in B according to it. Note that with this modification, it's possible for us to observe a temporary "slow down" in the reduction of the overall edge count; the preconditioner of B may have the same number of off-tree edges as B itself. However the total number of multi-edges will decrease at a rate that meets the design conditions. This reuse of the tree for generating sparsifiers is a crucial deviation from prior works.

But this doesn't fully explain the faster solver algorithm. To achieve it we need an extra trick. Assume for a moment that our graph A is what we call **spine-heavy**; that is, it has a tree of total stretch equal to $O(m/\log n)$. Then by an argument analogous to the one using a standard low stretch spanning tree, we can show that B actually satisfies the two-level preconditioning requirement for an even lower value of κ, namely a fixed constant. This, in combination with spine-based sampling allows us to solve **spine-heavy** graphs in **linear time**. A more global view of this algorithm, as shown in Figure 8 is that it progressively makes the tree heavier, while removing off-tree ones.

Figure 8. Low-stretch spanning tree as a spine. The "cloud" of off-tree edges becomes progressively sparser.



Since our initial graph A cannot be expected to be spine-heavy, we make a spine-heavy graph $\tilde{A}$ out of A, by scaling-up its LSST by an $O(\log^2 n)$ factor. Now $\tilde{A}$ is an $O(\log^2 n)$-approximation to A and we can solve it in $O(m)$ time. Using it as preconditioner for A completes the $O(m \log n)$ time solver. So, we have arrived at our destination.

THEOREM[10]. SDD systems can be solved in $\tilde{O}(m \log n \log(1/\epsilon))$ time, where ϵ is a standard measure of the approximation error.

5. EXTENSIONS

5.1. Parallelization

Several algorithms in numerical linear algebra have parallel versions that are work-efficient. A parallel algorithm is called **work-efficient** if it performs roughly the same work as its best sequential algorithm for the same problem, while allowing the use of parallel processing.

The first steps towards studying the parallelism potential of SDD solvers were taken in Blelloch et al.,[3] which presented a nearly (up to log factors) work-efficient algorithm, running in $O(m^{1/3})$ parallel time. Informally, this means that up to $m^{2/3}$ parallel processes can be used to accelerate the algorithm, a non-trivial potential for parallelism.

5.2. Implementation

The most complicated component of our solver is the algorithm for computing a LSST. It is however expected that a conceptually simpler algorithm for this problem is to be discovered, leading to a fast and "clean" implementation, and quite likely the removal of the log log n factors from the running time.

In a practical implementation, it would be a good idea to substitute the recursive preconditioned Chebyshev iteration by a recursive preconditioned Conjugate Gradient (PCG) iteration. It is known that, in two-level methods, PCG is essentially able to automatically optimize the performance of the preconditioner. It is expected that the same should be true for some multilevel variant of PCG, but this is yet to be proven.

We expect that, eventually, the best implementations of SDD solvers will combine ideas from this work and other existing graph-based techniques,[8] or entirely new ideas. Such ideas will certainly be needed to achieve—if possible—a "fully parallel", $O(\log n)$ time, work-efficient SDD solver.

6. THE LAPLACIAN PARADIGM

Solvers for SDD systems are increasingly viewed as an algorithmic **primitive;** a fundamental subroutine that can be used to design many other efficient algorithms. Indeed, since the Spielman–Teng breakthrough, the availability of fast SDD solvers has sparked what has been dubbed **the Laplacian paradigm**: an entire class of new algorithms spanning various areas. Because it is impossible to do justice to each one of these topics, we will present some unifying themes and only point to some representative examples of applications.

Perhaps the most direct example of using the solver as a primitive is the computation of eigenvectors. It was shown

in Spielman and Teng[20] that $O(\log n)$ iterations of solves produce a good approximation to a basic eigenvector of a graph. More closely related to preconditioned iterative methods is a solver for elliptic finite element linear systems.[4] This work showed that such systems can be preconditioned with graph Laplacians and so they can be solved in nearly linear time.

A more general framework stems from one of the most powerful discoveries in combinatorial optimization: interior point algorithms. It was shown by Daitch and Spielman[17] that interior point algorithms allow us to reduce a broad class of graph problems to solving a small number of SDD linear systems. This led to the best known running times for problems such as minimum cost flow and loss generalized flow. These problems are extensions of the maximum flow problem, which in its simplest version asks for the maximum number of edge disjoint routes (or "flow") between two nodes s and t. Further work in this direction led to the first improvement in 20 years on the approximate maximum flow problem.[5] The max-flow result is in turn directly applicable to graph partitioning, that is the separation of a graph to two well connected pieces; the fastest known algorithm for this problem repeatedly applies the fast max-flow algorithm.[16]

It is also worth noting that the solver presented in Blelloch et al.[3] readily gives—for all the above problems—parallel algorithms that are essentially able to split evenly the computational work and yield speedups even when only a small number of processors is available. This is a rare feature among previous algorithms for these problems.

Solver-based algorithms have already entered practice, particularly in the area of computer vision, where graphs are used to encode the neighboring relation between pixels. Several tasks in image processing, such as image denoising, gradient inpainting, or colorization of grayscale images, are posed as optimization problems for which the best known algorithms solve SDD systems.[11, 12] Linear systems in vision are often "planar", a class of SDD systems for which an $O(m)$ time algorithm is known.[7]

Given the prevalence of massive graphs in modern problems, it is expected that the list of applications, both theoretical and practical, will continue expanding in the future. We believe that our solver will accelerate research in this area and will move many of these algorithms into the practical realm.

Acknowledgments

This work is partially supported by NSF grant number CCF-1018463. I. Koutis is supported by NSF CAREER award CCF-1149048. Part of this work was done while I. Koutis was at CMU. R. Peng is supported by a Microsoft Research Fellowship. C

References

1. Abraham, I., Neiman, O. Using petal decompositions to build a low stretch spanning tree. In *Proceedings of the 44th Symposium on Theory of Computing* (STOC '12, 2012), ACM, New York, NY, 395–406.
2. Alon, N., Karp, R., Peleg, D., West, D. A graph-theoretic game and its application to the *k*-server problem. *SIAM J. Comput. 24*(1) (1995), 78–100.
3. Blelloch, G.E., Gupta, A., Koutis, I., Miller, G.L., Peng, R., Tangwongsan, K. Near linear-work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA '11, 2011), ACM, New York, NY, 13–22.
4. Boman, E.G., Hendrickson, B., Vavasis, S.A. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM J. Numer. Anal. 46*(6) (2008), 3264–3284.
5. Christiano, P., Kelner, J.A., Mądry, A., Spielman, D., Teng, S.-H. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011.
6. Elkin, M., Emek, Y., Spielman, D.A., Teng, D.A. Lower-stretch spanning trees. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, 494–503, 2005.
7. Koutis, I., Miller, G.L. A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Proc. 18th ACM–SIAM Symposium on Discrete Algorithms (SODA)*, 2007.
8. Koutis, I., Miller, G.L. Graph partitioning into isolated, high conductance clusters: Theory, computation and applications to preconditioning. In *Symposiun on Parallel Algorithms and Architectures (SPAA)*, 2008.
9. Koutis, I., Miller, G.L., Peng, R. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, FOCS*, IEEE Computer Society, 2010.
10. Koutis, I., Miller, G.L., Peng, R. A near-*m* log *n* solver for SDD linear systems. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, IEEE Computer Society, 2011.
11. Koutis, I., Miller, G.L., Tolliver, D. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Comput. Vision Image Understand. 115*(12) (2011), 1638–1646.
12. Krishnan, D., Szeliski, R. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph. 30*(6) (2011), 177.
13. Liben-Nowell, D., Kleinberg, J.M. The link-prediction problem for social networks. *JASIST 58*(7) (2007), 1019–1031.
14. Missiuro, P.V., Liu, K., Zou, L., Ross, B.C., Zhao, G., Liu, J.S., Ge, H. Information flow analysis of interactome networks. *PLoS Comput. Biol. 5*(4) (2009), e1000350.
15. Rudelson, M., Vershynin, R. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM 54*(4), (2007), 21.
16. Sherman, J. Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science* (FOCS '09, 2009), IEEE Computer Society, Washington, DC, USA, 363–372.
17. Spielman, D.A., Daitch, S.I. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, May 2008.
18. Spielman, D.A., Srivastava, N. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, 2008, 563–568.
19. Spielman, D.A., Teng, S.-H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, June 2004, 81–90.
20. Spielman, D.A., Teng, S.-H. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.
21. Tolliver, D.A., Koutis, I., Ishikawa, H., Schuman, J.S., Miller, G.L. Automatic multiple retinal layer segmentation in spectral domain oct scans via spectral rounding. In *ARVO Annual Meeting*, May 2008.
22. Trottenberg, U., Schuller, A., Oosterlee, C. *Multigrid*, 1st edn, Academic Press, London, 2000.
23. Vaidya, P.M. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. *A Talk Based on this Manuscript was Presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation*, October 1991.
24. Vassilevska Williams, V. Breaking the Coppersmith-Winograd barrier. In *Proceedings of the 44th Symposium on Theory of Computing*, STOC '12, 2012.

Ioannis Koutis (ioannis.koutis@upr.edu), Computer Science Department, University of Puerto Rico-Rio Piedras.

Gary L. Miller (glmiller@cs.cmu.edu), Computer Science Department, Carnegie Mellon University.

Richard Peng (yangp@cs.cmu.edu), Computer Science Department, Carnegie Mellon University.

CAREERS

Brown University/Institute for Computational and Experimental Research in Mathematics (ICERM)

Postdoctoral Fellowships

Postdoctoral Fellowship and Institute Postdoctoral Fellowship Opportunities at Brown University/Institute for Computational and Experimental Research in Mathematics (ICERM): ICERM invites applications for two types of positions.

Postdoctoral Fellowship: four one-semester non-renewable Postdoctoral Fellowships with stipend, to commence in January 2014. These positions are intended for mathematical scientists at an early career stage who would like to participate in the spring 2014 semester long program at ICERM: Network Science and Graph Algorithms. All application materials must be submitted on line at https://www.mathjobs.org/jobs/jobs/3877.

Institute Postdoctoral Fellowship: a one-year non-renewable salaried Postdoctoral Institute Fellowship that commences in September 2013. This position is intended for mathematical scientists at an early career stage who will be in residence at ICERM starting in September 2013 with an ICERM faculty mentor, and who would like to participate in ICERM's spring 2014 semester program: Network Science and Graph Algorithms. **All application materials must be submitted on line at** https://www.mathjobs.org/jobs/jobs/3872.

Documentation of completion of all requirements for a doctoral degree in mathematics or a related area by the start of the appointment is required. Preference will be given to applicants with a PhD awarded in 2010 or later.

Brown University is an Equal Opportunity/Affirmative Action employer and encourages applications from women and minorities.

Cal Poly

Computer Science

Tenure Track Faculty Position

Computer Science - Tenure track faculty position in Computer Science Department at Cal Poly, San Luis Obispo, California beginning September 2012. Particular areas of interest include: network security, mobile security, information assurance, usability and security, digital forensics, biometrics. Cal Poly is currently developing a state of the art cyber security laboratory for teaching and research in cyber security.

For details, qualifications, and application instructions (online application required), visit www.calpolyjobs.org and apply to requisition 102596.

Carnegie Mellon University

Tepper School of Business

Information Systems Tenure-Track Faculty Position

The Tepper School of Business at Carnegie Mellon University seeks a professor in Information Systems for tenure-track appointment, starting September 2013. We invite applications from individuals who are involved in cutting edge research involving information technologies and who are interested in how information technology will transform businesses, markets, and economic processes. Applicants may hold a doctoral degree in information systems, any other business discipline, computer science, economics, or statistics. We are primarily seeking candidates at the Assistant Professor level. Applicants should have completed or be nearing completion of a Ph.D., and should demonstrate potential for excellence in research and teaching. Teaching assignments encompass BS, Masters, and Ph.D. programs. Being an innovative technology expert in a leading business school brings unique opportunities for high-impact and highly visible work; collaboration across disciplines is an important part of the culture of the Tepper School and the wider Carnegie Mellon community. This open environment provides opportunities to work with our top-ranked economics and marketing groups and the school of computer science.

Application Instructions

Applicants should submit a current curriculum vitae, a statement of research interests, a short statement of teaching philosophy, copies of up to four representative publications or preprints, and at least three references through https://academicjobsonline.org/ajo/jobs/1744.

If you have any questions about the application, please contact Mr. Phil Conley (isgroup@andrew.cmu.edu, 412-268-6212). Screening begins immediately and candidates are encouraged to submit their completed applications as soon as possible. In order to ensure full consideration, completed applications must be received by November 30, 2012.

Carnegie Mellon is an equal opportunity, affirmative action employer with particular interest in identifying women and minority applicants for faculty positions.

Drexel University

Department of Computer Science
College of Engineering

Faculty Positions

Drexel University's Department of Computer Science (www.cs.drexel.edu) in the College of Engineering invites applications for up to 3 tenure-track faculty positions at the Assistant Professor level. Preferred areas of expertise are Systems, Data Mining, and Networks. Candidates in areas that align with the strengths of the department will also be considered, namely Software Engineering, HCI, Applied Algorithms, Computer Vision, Security, AI, Scientific Computation, High Performance Computing, and Computer Gaming. The department offers BS, BA, MS, and PhD degrees in computer science, as well as BS and MS degrees in software engineering. The department has a tradition of rigorous academic programs that feature a co-op experience, and a strong emphasis on research, as evidenced by the number of NSF CAREER Award recipients on the faculty.

Drexel is a private university founded in 1891 and is the third largest university in the Philadelphia area with over 12,000 undergraduate, 7000 graduate students, and over 1,000 faculty members. The University consists of 11 colleges and schools offering 175 degree programs. Drexel is a national pioneer of cooperative education, with formal relationships in place with over 2,700 local, national and multi-national companies. Drexel is located on Philadelphia's "Avenue of Technology" in the University City District and at the hub of the academic, cultural, and historical resources of one of the nation's largest metropolitan regions.

Review of applications begins immediately. To assure consideration, materials from applicants should be received by January 31, 2013. Successful applicants must demonstrate potential for research and teaching excellence in the environment of a major research university. To be considered, please send an email to:

cs-search-13@cs.drexel.edu

with a cover letter, CV, brief statements describing your research program and teaching philosophy, and at least four letters of reference. Electronic submissions in PDF format are preferred.

Drexel University is an Equal Opportunity/Affirmative Action Employer. The College of Engineering is especially interested in qualified candidates who can contribute to the diversity and excellence of the academic community. Background investigations are required for all new hires as a condition of employment, after the job offer is made. Employment will be contingent upon the University's acceptance of the results of the background investigation.

HGST

Research Scientist
Storage Architecture

Research Scientist in HGST (a Western Digital Company) Research Storage Architecture

HGST San Jose Research Center is a premier research center with more than 120 researchers working in many exciting fields including storage architecture, consumer electronics, storage technology, nanotechnology, electronics, nanomagnetism and micromechanics. Join the team of outstanding HGST Research scientists and engineers located in the evergreen hills of Yerba Buena in San Jose, and generate new ideas, publications and patents in the area of storage architecture.

The job opening is in the area of research of storage architecture and systems, more specifically operating systems, novel file systems, reliability of storage devices and systems and novel storage and computing architectures. Ideal can-

didate would work both independently and as a part of a HGST Storage Architecture group in conceiving, prototyping and guiding development of new storage related projects and ideas, as well as writing invention disclosures, academic papers and publications and participating in scientific societies and industry associations. Postdoctoral candidates are welcome to apply and propose more specific research programs.

Job Requirements
PhD in Computer science, Computer engineering, Electrical engineering or related, with a proven publication track record and implementation experience in system architecture, operating systems, file systems, and embedded systems.

Send applications (a full Curriculum Vitae and short description of research interests) to Zvonimir.Bandic@hgst.com

Indiana University, Bloomington
Six Positions, Positions are open at all levels

The School of Informatics and Computing at Indiana University, Bloomington, invites applications for six positions beginning in Fall 2013, in the areas of computer science (all subareas; two positions), computer science education research (joint position with School of Education), computer security, health informatics, and human-computer interaction. The School expects continued hiring in the coming years.

Positions are open at all levels. Applicants should have a Ph.D.in the relevant area and a well-established record (senior level) or demonstrable potential for excellence in research and teaching (junior level).

The IU Bloomington School of Informatics and Computing is the first of its kind and among the largest in the country, with unsurpassed breadth. It includes more than 70 faculty members, 500 graduate students, and strong undergraduate programs. Degrees offered include M.S. degrees in Computer Science, Bioinformatics, Human Computer Interaction Design, and Security Informatics, and Ph.D. degrees in Computer Science and in Informatics. The School has received public recognition as a "top-ten program to watch" (Computerworld) thanks to its excellence and leadership in academic programs, interdisciplinary research, placement, and outreach. The school offers excellent work conditions, including attractive salaries and research support, and low teaching loads in a setting of strong student growth.

Located in the wooded rolling hills of southern Indiana, Bloomington is a culturally thriving college town with a moderate cost of living and the amenities for an active lifestyle. IU is renowned for its top-ranked music school, high performance computing and networking facilities, and performing and fine arts.

Applicants should submit a curriculum vitae, a statement of research and teaching, and the names of 3 references (junior level) or 6 references (senior level) using the submissions link at https://indiana.peopleadmin.com (preferred) or by mail to Faculty Search Committee, School of Informatics and Computing, 919 E 10th Street, Bloomington, IN 47408. Questions may be sent to faculty-search12@informatics.indiana.edu. To receive full consideration completed applications must be received by December 1, 2012.

Indiana University is an Equal Opportunity/Affirmative Action employer. Applications from women and minorities are strongly encouraged. IU Bloomington is vitally interested in the needs of Dual Career couples.

Lehigh University
Department of Computer Science and Engineering
Assistant or Associate Professor

Applications are invited for a tenure-track position at the Assistant or Associate Professor level in the Computer Science and Engineering Department (http://www.cse.lehigh.edu) of Lehigh University to start in August 2013. Outstanding candidates in all areas of computer science will be considered, with priority given to candidates with a research focus in either Data Mining or Cybersecurity, both defined broadly.

The successful applicant will hold a Ph.D. in Computer Science, Computer Engineering, or a closely related field. The candidate must demonstrate a strong commitment to quality undergraduate and graduate education, and the potential to develop and conduct a high-impact research program with external support. Applicants should have an interest in teaching core courses in computer science as well as courses

in their research area. The ability to contribute to interdisciplinary research programs is also an important criterion.

The faculty of the Computer Science and Engineering department maintains an outstanding international reputation in a variety of research areas, and includes ACM and IEEE fellows as well as five NSF CAREER award winners. The department has 16 full-time faculty members and more than 250 undergraduate and 70 graduate students. We offer B.A., B.S., M.S., and Ph.D. degrees in Computer Science and jointly oversee programs in Computer Engineering and in Computer Science and Business.

The department maintains state-of-the-art computing facilities, and also has access to powerful university-supported facilities dedicated to high performance computing. The university is well connected to the Internet, Internet2, and is a charter member of the 1600-mile PennREN research network.

Lehigh University is a private, highly selective institution that is consistently ranked among the top 40 national research universities by *U.S. News & World Report*. Located in Bethlehem, Pennsylvania, Lehigh is 80 miles west of New York City and 50 miles north of Philadelphia, providing an accessible and convenient location that offers an appealing mix of urban and rural lifestyles. Lehigh Valley International Airport, just six miles from campus, provides easy access to the world. Lehigh Valley cities and towns are regularly listed as among the best places to live in the country.

Applications can be submitted online at http://www.academicjobsonline.org/ajo/jobs/1712. Review of applications will begin December 1, 2012 and will continue until the position is filled. Lehigh University is an Equal Opportunity/Affirmative Action Employer and is the recipient of an NSF ADVANCE Institutional Transformation award for promoting the careers of women in academic science and engineering. Lehigh provides comprehensive benefits including domestic partner benefits (see also http://www.lehigh.edu/worklifebalance/). Questions concerning this search may be sent to faculty-search@cse.lehigh.edu.

Louisiana State University
Professional in Residence & Founding Director
Program in Digital Media Arts & Engineering

Louisiana State University (LSU) seeks to hire an individual with seasoned industry experience and proven creative management skills of the highest caliber to build and direct an academic program in Digital Media Arts & Engineering. Through the development and implementation of a master's level curriculum focused on game design, visual effects, and related digital media applications, the envisioned program will provide a world-class environment for educating students in the collaborative, communications, business, technical, and creative skills needed to become successful contributors in the rapidly evolving arena defined by digital game and digital media industries.

In support of this effort, the successful candidate will have the opportunity to draw upon the creative, intellectual, and capital resources of LSU's Center for Computation & Technology and associated AVATAR and digital media programs; LSU's College of Engineering and its School of Electrical Engineering & Computer Science; LSU's Schools of Art, Music, and Mass Communications; the Baton Rouge Area Digital Industry Consortium; and Louisiana's Department of Economic Development. To ensure long-term program relevance, the successful candidate should also nurture a network of industry partnerships, including the involvement of industry in curriculum development. Leveraging these ties, the resulting program should provide an academic environment in which students with creative or technical (or both) backgrounds can excel. A key measure of success will be placement of students into desirable positions within Louisiana's growing digital game and digital media industries.

Required qualifications:
- ▸ At least eight years of related experience in industries such as game design, digital media, animation, etc.
- ▸ Management experience
- ▸ Bachelor's degree

Preferred qualifications:
- ▸ Ten or more years of related industry experience
- ▸ Prior experience working in a university setting
- ▸ Master's degree in related field

Letters of nomination may be e-mailed to jobs@cct.lsu.edu.

Individuals who wish to apply for this opportunity are asked to submit their applications online which must include a letter of interest, resume, professional references and a vision statement for the new program in Digital Media Arts & Engineering. Review of applications will begin November 1, 2012 and continue until the position is filled. An offer of employment is contingent on a satisfactory pre-employment background check. Salary will be competitive and commensurate with qualifications. Applications will be accepted on the LSU Careers website: https://lsusystemcareers.lsu.edu Position number 035785.

LSU IS AN EQUAL OPPORTUNITY/EQUAL ACCESS EMPLOYER

Stanford University
Graduate School of Business
Faculty Positions in Operations, Information and Technology

The Operations, Information and Technology (OIT) area at the Graduate School of Business, Stanford University, is seeking qualified applicants for full-time, tenure-track positions, starting 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, 2013.

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 (nonreturnable) 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, 2012.**

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.

Texas State University-San Marcos

Department of Computer Science

Assistant Professor or Associate Professor

Applications are invited for two tenure-track faculty positions, one in software engineering and the second in any computer science field to start September 1, 2013. The rank for either can be Assistant Professor or Associate Professor, depending on qualifications. Consult the department recruiting page at http://www.cs.txstate.edu/recruitment/faculty_recruit.php for job duties, qualifications, application procedures, and information about the university and the department.

Texas State University-San Marcos (Texas State) will not discriminate against any person in employment or exclude any person from participating in or receiving the benefits of any of its activities or programs on any basis prohibited by law, including race, color, age, national origin, religion, sex, disability, veterans' status, or on the basis of sexual orientation. Texas State is committed to increasing the diversity of its faculty and senior administrative positions. Texas State is a member of The Texas State University System. Texas State is an EOE.

Toyota Technological Institute Chicago

Faculty Positions at All Levels

Toyota Technological Institute at Chicago (TTIC) is a philanthropically endowed degree-granting institute for computer science located on the University of Chicago campus. Applications are being accepted in all areas, but we are particularly interested in machine learning, speech processing, computational linguistics, Computer vision, computational biology and optimization. Positions are available at all ranks, and we have a large number of three year limited term positions currently available. For all positions we require a Ph.D. Degree or Ph.D. candidacy, with the degree conferred prior to date of hire.

Apply URL: http://ttic.uchicago.edu/facapp/

Toyota Technological Institute at Chicago is an Equal Opportunity Employer.

University of Wisconsin Oshkosh

Tenure-track Computer Science position (rank open)

Tenure-track Computer Science position (rank open) beginning September 1, 2013. Ph.D. in Computer Science or closely related field required. Duties: teaching 9 credits/semester in ABET-accredited program, research, advising majors. Send letter of application, CV, teaching & research statements, transcripts (official or photocopy), & 3 current letters of recommendation directly to: Tom Naps Chair, Computer Science Dept., University of Wisconsin Oshkosh, 800 Algoma Blvd., Oshkosh, WI 54901-8643. Application deadline: November 9, 2012. Employment requires criminal background check AA/EOE

Yale University

Department of Computer Science

Faculty Position Openings

The Department of Computer Science at Yale is seeking to fill a faculty position at the level of Assistant Professor (tenure track). We are interested in applicants from any of the following areas: Artificial Intelligence, Economics and Computation, Human Computer Interaction, Machine Learning, Networking, Operating Systems, Programming Languages, Robotics, and Theory of Computing. Applicants whose research spans two or more of these areas are particularly encouraged to apply. We seek excellent researchers who are also committed to excellence in teaching.

Members of the Yale Computer Science faculty have many opportunities to collaborate. Interdisciplinary work is encouraged with Yale's world-class faculty in such computationally active fields as biology, chemistry, economics, engineering, geophysics, management, mathematics, medicine, psychology, physics, and statistics. Yale faculty members teach excellent students, both graduate and undergraduate, in relatively small classes.

Candidates should hold a Ph.D. in computer science or a related discipline. Review of applications will begin on December 15, 2012, and candidates are encouraged to apply early. Applications from qualified women and minority candidates are especially welcome. Yale is an affirmative action/equal opportunity employer. The department's home page can be found at http://www.cs.yale.edu. Applicants should submit a curriculum vita, brief statements of research and teaching, and the contact information for three references. **Applications should be submitted online at** https://academicpositions.yale.edu/. Questions should be directed to faculty-recruiting@cs.yale.edu.

From the intersection of computational science and technological speculation, with boundaries limited only by our ability to imagine what could be.

DOI:10.1145/2347736.2347760

Geoffrey A. Landis

Future Tense
Fermi's Paradox and the End of the Universe

How to colonize the galaxy, one electron spin state at a time.

THE FERMI PARADOX is a question that has bedeviled scientists for a long time. The galaxy, it is believed, should be home to myriad alien species, some millions or even billions of years older than humanity. So, Fermi asks, why aren't they here?

Take, for example, the gwyrxia.

Their name is not gwyrxia, of course. They do not communicate with sound waves, and, in fact, any sound waves humans can hear are of such low frequency the gwyrxia would not have considered even the possibility they might be a means of communication. They communicate with a spread-spectrum electromagnetic radiation, so efficiently encoded that, if we humans even detected it, we would think of it as indistinguishable from thermal noise. Gwyrxia is what it would sound like—sort of—if you could decipher their name for themselves out of that pink noise.

In order to live forever, the gwyrxia knew they would have to abandon organic bodies. They needed a more durable form. This they did many billions of years ago. The most efficient encoding of a mind is to imprint the patterns of their consciousness into electron spin states, using a quantum computation as a form of thinking. They implemented this quantum computation in the spin states of the valence-band electrons of silicon-oxygen bonds in silicate rock. Silicate rock seemed a reasonable matrix, since there is plenty of it in the universe.

About 10^{15} electron spins are required to hold a gwyrxia mind. The interaction between electron spins, each interaction either flipping or not flipping a spin, provides the qbit computations that correspond to "thinking" for a gwyrxia.

So, rather slowly, the gwyrxia decided to colonize the galaxy. They live a long time. They are slow—but they began a few billion years ago.

They are massively parallel, and massively redundant. Each gwyrxia mind is duplicated several trillion times in any particular silicate instantiation.

They have barely noticed us, so far. As I said, audible frequency pressure waves do not matter much to them, and only since humans have been using radio waves—barely 100 years—has there been even a possibility of their being able to detect us. They communicate very slowly; remember, they have already decided to live to the end of the universe, and a century here or a millennium there is not terribly important to them.

So they are only just beginning to notice us.

For decades, we have been wondering, where are they?

The answer is, all around us.

We call them "rocks."

Geoffrey A. Landis (geoffrey.landis@nasa.gov) is a scientist at the NASA John Glenn Research Center, Cleveland, OH, working on Mars missions and advanced concepts and technology for future space missions, and author of science fiction novels and short stories, most notably *Mars Crossing* (Tor Books, 2000) about an ill-fated expedition to the red planet.

PHOTOGRAPH BY ERIC LO

The Ultimate Online Resource for Computing Professionals & Students

Association for Computing Machinery

Advancing Computing as a Science & Profession

Computing Reviews is on the move

Our new URL is

ComputingReviews.com

CR

COMPUTING REVIEWS

A daily snapshot of what is new and hot in computing