COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

10/08 VOL.51 NO.10

The Many Facets of Natural Computing

Debating the Use of E-Voting Machines

Code Spelunking

Topology of Dark Networks

A Closer Look at GPUs

Green Computing

Will the Future of Software be Open Source?

Association for Computing Machinery

FDG 09

www.foundationsofdigitalgames.org

Onboard the Disney Wonder, departing from Port Canaveral, FL
April 26th - April 30th, 2009

The Fourth International Conference on the

Foundations of Digital Games

Conference Chair
Jim Whitehead
University of California, Santa Cruz,
Santa Cruz, CA

Program Chair
R. Michael Young
NC State University,
Raleigh,NC

Submission Deadlines

Long Papers
December 19, 2008

Posters
December 19, 2008

Doctoral Consortium Proposals
December 29, 2008

Corporate Sponsors

Microsoft Research

FDG '09 – the International Conference on Foundations of Digital Games – is a focal point for academic efforts in all areas of research and education involving computer and console games, game technologies, game play and game design.

Topics of Interest

FDG 2009 seeks high-quality, original submissions that report on work across all areas of games research that promote new game capabilities, designs, applications and modes of play. We encourage participation from researchers and practitioners from around the globe. All submissions will be reviewed by a distinguished international program committee. Further, we encourage submissions in specific theme areas including artificial intelligence, computer science and games education, databases, games studies & game design, graphics & interfaces and networks & security.

Conference Program

Previously known as Academic Days in Game Development and Computer Science Education (GDCSE 08), this year's conference expands its scope to cover its previous educational focus as well as the breadth of game research and education.

The conference's program includes presentations of full length papers which are selected by the program committee after in-depth analysis and discussion. A set of poster presentations will stimulate discussion on work in progress. A doctoral consortium will encourage new researchers early in their graduate career by providing input on their ideas from the conference program committee. The conference experience also includes keynote and invited talks by industry and academic leaders, panels on exciting issues and tutorials focusing on the incorporation of game technology into research and academic programs.

Springer
the language of science

springer.com

Noteworthy Titles

How to Solve It: Modern Heuristics

Z. Michalewicz, D.B. Fogel

This book is the only source that provides comprehensive, current, and correct information on problem solving using modern heuristics. It covers classic methods of optimization, including dynamic programming, the simplex method, and gradient techniques, as well as recent innovations such as simulated annealing, tabu search, and evolutionary computation. This second edition contains two new chapters, one on co evolutionary systems and one on multicriterial decision-making. Also some new puzzles are added and various subchapters are revised.

2nd ed. Revised and Extended. 2004. XVIII, 554 p. 174 illus. Hardcover
ISBN 978-3-540-22494-5 ► **$59.95**

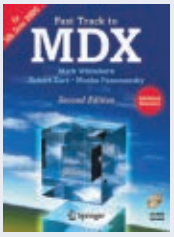
Web Data Mining

Exploring Hyperlinks, Contents, and Usage Data

B. Liu

This book provides a comprehensive text on Web data mining. Key topics of structure mining, content mining, and usage mining are covered. The book brings together all the essential concepts and algorithms from related areas such as data mining, machine learning, and text processing to form an authoritative and coherent text. The book offers a rich blend of theory and practice. It is suitable for students, researchers and practitioners interested in Web mining. Lecturers can readily use it for classes on data mining, Web mining, and Web search. Internet support with lecture slides and project problems is available online.

2007. XX, 532 p. 177 illus. (Data-Centric Systems and Applications) Hardcover
ISBN 978-3-540-37881-5 ► **$59.95**

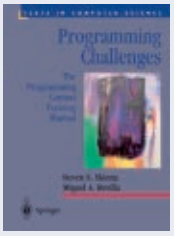


Fast Track to MDX

M. Whitehorn, R. Zare, M. Pasumansky

It was the clarity, precision and sheer readability of the first edition that made it a best-seller. With that firmly in mind the authors have left the original 18 chapters intact (apart from minor updates); meaning that the second edition remains the best introduction to MDX for SQL Server 2000 that is available. What they have done is to add three brand new chapters. These introduce the topic of recursion in MDX, walk the reader through the process of creating recursive expressions and finally demonstrate how recursion can be used to effectively solve a series of business problems

2nd ed. 2006. XXVI, 310 p. 199 illus. With CD-ROM., Softcover
ISBN 978-1-84628-174-7 ► **$54.95**



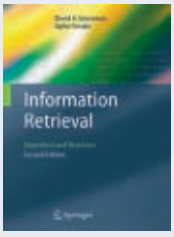
Programming Challenges

The Programming Contest Training Manual

S.S. Skiena, M.A. Revilla

This book uses international programming competition-type problems to motivate the study of algorithms, programming, and other topics in computer science. It includes more than 100 programming challenges, as well as the theory and key concepts necessary for approaching them. Problems are organized by topic, and supplemented by complete tutorial material. Readers gain a concrete understanding of both algorithmic techniques and advanced coding topics.

2003. XIX, 359 p. 65 illus. (Texts in Computer Science) Softcover
ISBN 978-0-387-00163-0 ► **$54.95**



Information Retrieval

Algorithms and Heuristics

D.A. Grossman, O. Frieder

This book is not yet another high level text. Instead, algorithms are thoroughly described, making this book ideally suited for both computer science students and practitioners who work on search-related applications. This book provides a current, broad, and detailed overview of the field and is the only one that does so. Examples are used throughout to illustrate the algorithms. This edition is a major expansion of the one published in 1998. Besides updating the entire book with current techniques, it includes new sections on language models, cross-language information retrieval, peer-to-peer processing, XML search, mediators, and duplicate document detection.

2nd ed. 2004. XX, 332 p. (The Information Retrieval Series, Vol. 15) Softcover
ISBN 978-1-4020-3004-8 ► **$59.95**

Data Compression

The Complete Reference

D. Salomon

Data compression is one of the most important tools in modern computing, and there has been tremendous progress in all areas of the field. This fourth edition of Data Compression provides an all-inclusive, thoroughly updated, and user-friendly reference for the many different types and methods of compression (especially audio compression, an area in which many new topics covered in this revised edition appear).

4th ed. 2007. XXVIII, 1092 p. 428 illus. 6 in color. Hardcover
ISBN 978-1-84628-602-5 ► **$89.95**

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

013830x

Departments

5 Editor's Letter
Let Us—Together—Make CACM Exciting
By Moshe Y. Vardi

7 Publisher's Corner
The Softer Side of *Communications*
By Scott E. Delman

8 Letters To The Editor
Prep Students for Irreversible Software Trends

10 CACM Online
***Communications* Site to Launch in January**
By David Roman

106 Careers

News

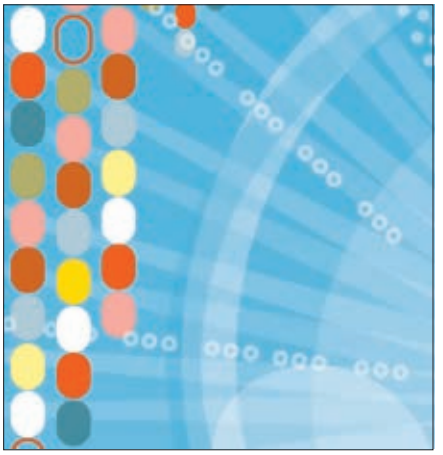
11 **Green Computing**
Are you ready for a personal energy meter?
By Patrick Kurp

14 **Searching the Deep Web**
While the Semantic Web may be a long time coming, Deep Web search strategies offer the promise of a semantic Web.
By Alex Wright

16 **Clean Elections**
With end-to-end auditable voting, a voter can verify whether his or her vote was tallied correctly and whether all of the votes were properly tabulated.
By Cyrus Farivar

19 **An Inspiring Legacy**
Admired and respected by his students and colleagues, Randy Pausch will be remembered as a devoted teacher and innovative researcher.
By Leah Hoffmann

Viewpoints



21 Historical Reflections
Will the Future of Software be Open Source?
Tracing the course of influential computing developments and considering possible paths to new paradigms.
By Martin Campbell-Kelly

24 Computing Ethics
Computer Experts: Guns-for-Hire or Professionals?
Considering the responsibilities of those who build systems fundamental to significant social functions, institutions, and values.
By Deborah G. Johnson

27 From the Front Lines
DOA with SOA
Diagnosing the symptoms of failing to accommodate critical software architecture properties that often result in the demise of projects.
By Alex E. Bell

29 Point/Counterpoint
The U.S. Should Ban Paperless Electronic Voting Machines
Debating the public policy issues involved in proposed efforts toward improving voting systems while considering the range of technical and societal challenges.
By David L. Dill/Daniel Castro

Practice

36 **Code Spelunking Redux**
Is it getting any easier to understand other people's code?
by George V. Neville-Neil

43 **Document Design Matters**
How do we apply the concept of resource orientation by designing representations to support interactions?
By Erik Wilde and Robert J. Glushko

50 **A Closer Look at GPUs**
As the line between GPUs and CPUs begins to blur, it's important to understand what makes GPUs tick.
By Kayvon Fatahalian and Mike Houston

LEFT: ILLUSTRATION BY CELIA JOHNSON; RIGHT: ILLUSTRATION BY JOHN HERSEY

10/08
VOL. 51 NO. 10

Contributed Articles

58 **The Topology of Dark Networks**
Knowing the structure of criminal and terrorist networks could provide the technical insight needed to disrupt their activities.
By Jennifer Xu and Hsinchun Chen

66 **Crossroads for Canadian CS Enrollment**
What should be done to reverse falling CS enrollment in the Canadian education system?
By Jacob Slonim, Sam Scully, and Michael McAllister

Review Articles

72 **The Many Facets of Natural Computing**
Natural computing builds a bridge between computer science and natural sciences.
By Lila Kari and Grzegorz Rozenberg

About the Cover: The ant algorithm for discrete optimization is one of many examples of computational techniques inspired by nature and discussed in this month's cover story "The Many Facets of Natural Computing" beginning on page 72. The image is courtesy of Alex Wild, ant photographer and biologist at the University of Illinois at Urbana-Champaign. For more of his work, visit www.alexanderwild.com/.

ILLUSTRATION BY LEANDER HERZOG

Research Highlights

86 Technical Perspective
Computational Photography on Large Collections of Images
By Marc Levoy

87 **Scene Completion Using Millions of Photographs**
By James Hays and Alexei A. Efros

95 Technical Perspective
New Developments in Graph Partitioning
By Éva Tardos

96 **Geometry, Flows, and Graph-Partitioning Algorithms**
By Sanjeev Arora, Satish Rao, and Umesh Vazirani

Last Byte

112 Q&A
A Complex Thinker
Daphne Koller discusses probabilistic relational modeling, artificial intelligence, and her new work with biologists.
By Leah Hoffmann

Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition. To ensure timely publication, ACM created *Communications*' Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

Large Scale Project Team Building: Beyond the Basics
Mary C. Jones

Understanding Evolution in Technology Ecosystems
Gediminas Adomavicius, Jesse Bockstedt, Alok Gupta, and Robert J. Kauffman

Understanding the Influence of Network Positions and Knowledge Processing Styles
Seokwoo Song, Sridhar Nerur, and James T. C. Teng

RFID in the Supply Chain: Panacea or Pandora's Box?
Brian L. Dos Santos and Lars S. Smith

Switching Between Consumer Technologies
Chen Ye, Kevin C. Desouza, Sridhar R. Papagari Sangareddy and Sanjeev Jha

Governing Diversity in the Digital Ecosystem
Mary Darking, Edgar A. Whitney and Paolo Dini

Myths and Paradoxes in Japanese IT Offshoring
Amrit Tiwana, Ashley A. Bush, Hiroshi Tsuji, Kenichi Yoshida and Akito Sakurai

Technical Opinion
Which Data Warehouse Architecture is Best?
Thilini Ariyachandra and Hugh J. Watson

COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

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.

STAFF

GROUP PUBLISHER
Scott E. Delman
publisher@cacm.acm.org

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

Art Director
Andrij Borys
Associate Art Director
Alicia Kubista
Assistant Art Director
Mia Angelica Balaquiot
Production Manager
Lynn D'Addesio
Director of Media Sales
Jonathan Just
Advertising Coordinator
Graciela Jacome
Marketing & Communications Manager
Brian Hebert
Public Relations Coordinator
Virgina Gold
Publications Assistant
Emily Eng

Columnist
Alok Aggarwal; Phillip G. Armour
Martin Campbell-Kelly;
Michael Cusumano; Peter J. Denning;
Shane Greenstein; Mark Guzdial;
Peter Harsha; Leah Hoffmann;
Deborah Johnson; Mari Sako;
Pamela Samuelson; Gene Spafford;
Cameron Wilson

CONTACT POINTS

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

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ADVERTISING

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

Director of Media Sales
Jonathan M. Just
jonathan.just@acm.org

Media Kit
acmmediasales@acm.org

Subscriptions
Annual subscription cost is included in the society member dues of $99.00 (for students, cost is included in $42.00 dues); the nonmember annual subscription rate is $100.00.

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://cacm.acm.org/advertising or by contacting ACM Media Sales at (212) 626-0654.

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

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Association for Computing Machinery

Printed in the U.S.A.

DOI: 10.1145/1400181.1400182

Moshe Y. Vardi

Let Us—Together—Make CACM Exciting

It's been four months since we launched the "new CACM." By now, I hope it is quite clear to our readers that the revamped flagship publication of ACM has undergone a rather dramatic transformation in both appearance and content. In my July editorial I began explaining the new editorial model of *Communications*, particularly the Research Highlights section that provides a bird's-eye view of the breadth and depth of computing research.

But there are several other innovative editorial features to this publication. The Practice section targets professionals in the software industry with an emphasis on software engineering. Articles published in this section frame and define technical problems and challenges ahead while helping readers sharpen their own thinking and ability to pursue innovative solutions. Practice articles do not focus on industry news or the latest solutions. Rather, they explore disruptive technologies that are just on the verge of breaking through. The editorial board of *Communications*' Practice section also serves as the editorial board of *ACM Queue*, enabling us to benefit from their experience in creating great computing-practice content.

The third novel feature is the greatly expanded News section, now publishing a selection of brief news updates and in-depth news articles on a broad range of topics. The goal of this section is to cover major developments in computing in a broadly accessible manner. As a monthly publication, *Communications* cannot offer the immediacy of the mass media, but it can cover topics in an increased depth, as befits its sophisticated readership.

The fourth new editorial element is the section featuring Review Articles that describe new developments of broad significance to the field and highlight unresolved questions and future directions. Unlike survey articles that provide a detailed introduction to a technical area, a Review article in *Communications* offers a high-level perspective. This is consistent with our goal of offering its readers a broad perspective on new developments in computing.

Finally, the new editorial model inherits two successful features from its previous model. The Viewpoints section is dedicated to opinions and views that pertain to issues of broad interest to the community, typically, but not exclusively, of a nontechnical nature. This section consists of both regular columns, some that have been appearing in *Communications* for years and some that are new to the magazine, as well as unsolicited opinion pieces and point-counterpoint editorial debates.

Finally, Contributed Articles features unsolicited articles from the community covering the abundant spectrum of the computing field—its open challenges, technical visions and perspectives, educational aspects, societal impact, significant applications, and research results of high significance and broad interest. *Communications*' editorial policy toward contributed articles has changed significantly with the July issue, as we now require that such articles be aimed at the broad computing and IT community.

Unlike some other flagship publications, most of *Communications*' content is produced not by professional writers but by members of the computing community. To succeed, *Communications* must become a venue of choice for computing content of the highest quality. While many of *Communications*' articles are solicited by members of its editorial board, unsolicited articles are crucial to our success. It is important, however, to recognize *Communications*' unique role. It does not aim to compete with ACM's dozens of scholarly journals or nearly 150 conferences. *Communications* enjoys a uniquely broad readership; ACM's 90,000 members consist of students, educators, researchers, software developers, CTOs, and CIOs. Such a diverse and international readership calls for articles that are aimed at a broad rather than specialized audience. Prospective authors are advised to consult this issue, or the three previous editions, for great examples of this new editorial focus. Interested authors are advised to review the recently revised Author Guidelines (www.cacm.acm.org/guidelines) for detailed information on how to position, prepare, and submit manuscripts.

Let me end by repeating the last paragraph of my January 2008 essay, "CACM: Past, Present, and Future." We live in a consumer society, so it is easy to evaluate products from a consumer perspective: "Is *Communications* a satisfactory product?" "Am I getting my money's worth for my ACM membership?" ACM, however, is not a consumer-product vendor, it is a professional society. We are not ACM customers, we are ACM members. *Communications* is not a product, it is a project. For this project to succeed, the membership of ACM must collectively undertake it. Let us—together—make *Communications* the exciting publication it should be. Please write to me at eic@cacm.acm.org.

Moshe Y. Vardi, EDITOR-IN-CHIEF

The 1st ACM SIGGRAPH Conference and Exhibition in Asia
www.siggraph.org/asia2008

New Horizons

ACM SIGGRAPH launches the premiere SIGGRAPH Asia in Singapore

Programs include:

- Papers
- Sketches and Posters
- Courses
- Art Gallery
- Computer Animation Festival
- Educators Program
- Emerging Technologies
- Exhibition

Calling all creative researchers, artists, digital innovators and animators!

This is your opportunity to present your stellar work or attend the 1st ACM SIGGRAPH conference and Exhibition in Asia.

Queries?

Contact Conference and Exhibitions Management at asia2008@siggraph.org or Tel: +65 6500 6700

SIGGRAPHASIA2008

Conference and Exhibition on Computer Graphics and Interactive Techniques

Singapore, 10-13 December 2008

Held in
UNIQUELY Singapore
visitsingapore.com

DOI:10.1145/1400181.1400183

Scott E. Delman

The Softer Side of *Communications*

Much has already been written about the expanded editorial scope of the new *Communications*. For most of these changes we have Editor-in-Chief Moshe Y. Vardi and his distinguished Editorial Board to thank.

They have worked tirelessly over the past year to develop a pipeline of high-quality articles that will keep the magazine at the level you have seen over the past few months. But there have also been other significant changes with the magazine. Among them is *Communications*' completely new look.

If you have not already noticed, please pick up a few issues from the first half of 2008 and compare them to any of the issues appearing since July, including this one. What you will see is the result of a carefully planned redesign that involved a partnership between ACM staff, the internationally recognized design firm Pentagram Design, and the magazine's new art and composition team Andrij Borys Associates.

The first thing you may notice from a design perspective is the striking cover of the magazine, including beautiful artwork, new text typography, and the generous use of white space. The combination of these three elements creates a cover design that is fresh, modern, uncluttered, and easily identifiable. The cover art itself is carefully selected or created by professional artists who in many instances are also computer scientists using complex programs and algorithms to generate the art. While we will deviate from this convention, we thought it was appropriate for the magazine to showcase work from within the community. Moreover, there is simply no denying the striking beauty of many of the images we saw from these artists.

Every effort is being made to keep the integrity of the magazine's graphics at a consistently high level.

For those of you interested in typefaces, we now use a combination of Foundry Gridnik, Arnhem, Flama, and Klavika. Foundry Gridnik is used for headlines throughout the magazine, including the cover logo. Arnhem is the typeface for the body text. The many colored decks and pull quotes throughout the issue are set in Flama type and Klavika is the typeface for tables and figures. Each of these fonts is unique and I would encourage anyone interested to learn more about their history. For example, Gridnik was originally created by Dutch designer Wim Crouwel in the late 1960s in Holland as a single-weight typewriter face and a version of this font appears on low-value stamps from the Dutch Post Office. The font was aptly named Gridnik for the designer's devotion to grids and systems.

To improve readability, we have moved to a three-column format for nearly all sections of the magazine and decreased the font size of article text from 11.5 to 9.5. This new format and point size allow us to publish more words on each page than the traditional two-column format, which has become increasingly important in an age of swelling printing costs and verbose writing, while at the same time not compromising the readability of articles. It is a careful balance, but I believe we have gotten it right.

Where possible, we will use images, photographs, and artwork provided by our authors, but every effort is being made to keep the integrity of the magazine's graphics at a consistently high level, so our art and composition team maintains diligent control over this process. In addition, unlike many scholarly publications, *Communications* magazine will alter figures and tables for improved visual affect. We work closely with a company called SPi Global Solutions on the typesetting of our full-length research papers. SPi has extensive experience working with the scientific and scholarly community and is among the world's largest providers of typesetting services to the journal publishing industry. The result of this collaboration can be seen throughout the Research Highlights section (beginning on page 85) reflecting a format that is as appealing as any scholarly journal in existence today.

All of these changes work together to create what we at ACM believe is a more appealing magazine for our readership. We hope you will agree.

Scott E. Delman, GROUP PUBLISHER

DOI:10.1145/1400181.1400184

Prep Students for Irreversible Software Trends

STEPHEN J. ANDRIOLE *and Eric Roberts debated how to educate the next generation of technology professionals in the Viewpoint Point/Counterpoint "Technology Curriculum for the Early 21st Century" (July 2008). Here, they offer their final words on each other's thoughts, beginning with Andriole.*

The assumption that the number of "programming" jobs will increase over time or that the location or nature of programming work will not change challenges every assumption about the trajectory of change in the industry. Could Eric Roberts actually believe, as he said in his "Counterpoint" (July 2008) that the video gaming industry will create enough new programming jobs to offset the loss of opportunities in U.S. and global corporations that long ago yielded to the commoditization of software and the relentless march toward software delivery through installed and hosted packaged applications? It's clearly difficult for a $10 billion industry to replace a $200 billion industry.

Moreover, Roberts did not discuss my call (in my side of the debate) for greater emphasis on integration, interoperability, applications architecture, communications architecture, and data architecture. I was calling for innovation and design, not implementation. It appears that as programming continues to commoditize, even as it standardizes on quasi-open standards, the demand for architectural services will be huge. There is also demand for optimization and metrics identification and management that represents even more opportunities for computing professionals. I am talking about the relative need for software applications, design, development, and support, all of which are changing as a result of technological and market forces.

I have never advocated abandoning software engineering or declared programming "irrelevant." (I feel like a political candidate misquoted for the purposes of the opposing candidate's argument.) "Programming" must tilt toward architecture and design and stop assuming jobs will be available for professionals writing nonstandard, proprietary-extreme applications and that these jobs will yield to commoditization, standardization, and alternative software-delivery models.

Will there be a shortage of computing professionals? Yes, but why? Perhaps it will be the result of the mismatch between what we teach and what employers need. The data we've collected reflects the shortage of professionals who understand design, architecture, integration, interoperability, open standards, ERP/CRM/NSM implementation and support, and the ability to optimize standards and architectures for business value.

In July, I said that the world is changing and we owe it to our students that we change with it in the right directions. Here's a simple test: Gartner Group recently identified 14 alternative technology-delivery models, including business-process utilities, capacity-on-demand, communications-as-a-service, community source, grid computing, infrastructure utility, remote-management services, software-as-a-service, software streaming, software-based appliances, storage-as-a-service, user-owned devices, utility computing, and Web platforms. Are we educating our students in these areas? Do academic programs support these delivery models? We must be more relevant to the changing world off-campus.

My work as a consultant and venture capitalist has exposed me to software design and development in many industries, including niche markets like video gaming. But corporate computing across all vertical industries—and its providers—represents professional opportunities for our students who later go to work for, say, Accenture, Cisco Systems, CSC, Deloitte, EMC, IBM, Intel, Hitachi, HP/EDS, Microsoft, Oracle, PWC, SAP, and Sun Microsystems. As the number of developers and providers shrinks through consolidation and commoditization (EDS is today part of HP; all major business-intelligence vendors have been acquired; and the number of hardware vendors continues to decline), opportunities for our students are changing dramatically. We need to prepare them, not stubbornly cling to the way things were.

Stephen J. Andriole, Villanova, PA

Roberts responds:

Stephen J. Andriole's "Viewpoint" (July 2008) began by saying "the assumption that the number of 'programming' jobs will increase over time…challenges every assumption about the trajectory of change in the industry." While this view may indeed challenge his assumptions, it does not contradict the available evidence. As I said in my "Counterpoint," the U.S. Bureau of Labor Statistics is projecting a dramatic increase in programming-intensive jobs from 2006 to 2016, to the point that two of the top five fastest-growing job categories over that time are "network systems and data communications analyst" and "computer software engineers, applications," both of which require significant programming expertise.

Andriole also said I misrepresented his arguments, in particular that he "never advocated abandoning software engineering." But he did precisely that, explicitly proposing the reduction of the five academic disciplines identified by the Joint ACM/IEEE-CS Task Force on Computing Curricula—computer engineering, computer science, information systems, information technology, and software engineering—to "three flavors: computer engineering, computer science, and information systems." Software engineering is conspicuously absent. He also said he never deemed programming "irrelevant." Despite the fact that the word he quoted—irrelevant—never appeared in my "Counterpoint," it is important to look at what he did say. Reading the paragraph that asks

"Who programs?," it is difficult to believe that he sees much relevance in this activity.

Andriole missed my central point—that every subdiscipline in the broad array of computing specialties is experiencing growth in demand. There are undeniably more jobs in computing that do not require sophisticated programming in the classical sense. What he failed to recognize is that there are also more jobs in computing that do require such talents. We must expand educational programs that focus on the IT areas he championed but not at the expense of more traditional CS programs that produce too few graduates to meet industry demand.

The most disturbing aspect of Andriole's comments is his charge that those of us in traditional computing disciplines are somehow jeopardizing the opportunities available to those newly entering the field. Each year, the CS graduates of my own institution [Stanford University] have extraordinary opportunities precisely because of the education they have received. Our bachelor's degree recipients get multiple offers, sometimes commanding salaries in excess of $100,000 per year. Providing students such opportunities is hardly a disservice. If we could somehow produce 10 times the number of CS graduates, they would get the same sort of offers. The demand for those with skills is strong and growing. Shifting our educational focus away from critically needed skills would be the height of irresponsibility.

Eric Roberts, Stanford, CA

Correction: Expected Loss Conditional

In our article, "Information Security and Risk Management" (Apr. 2008), the notation for the expected value of a severe loss was incorrectly written as the expected loss conditional on the loss being severe. We thank Jonathan Katz of the University of Maryland, College Park, for bringing it to our attention. While we correctly defined the concept of the expected severe loss (page 65), the notation on the left-hand side of the second equation in the figure (page 66) was in error.

As in the article, let T denote the threshold value of a severe loss and X be a random variable representing the possible loss values. Let Y be a random variable representing a loss due to a severe breach. Then, the possible values of Y are zero, plus the values of X that are greater of equal to T. The probability of Y=x equals the probability that X=x for those values of X greater than or equal to T. Further, the probability that Y=0 equals the probability that X is strictly less than T. The left-hand side of the equation (page 66) should have been written as E[Y]. For the example given, the random variable Y can take on the values 0, 8, and 9 with probabilities of 0.8, 0.1 and 0.1, respectively. Then, E[Y] = 0*.8 + 8*.1 + 9*.1 = 1.7.

Lawrence D. Bodin, Lawrence A. Gordon, Martin P. Loeb, College Park, MD

To Boost Enrollment Fix the CS Image Problem

As an IS practitioner and parent of three college-age sons, my perspective on the reasons for decreased IS and CS enrollment is a bit different from that of Wayne Wei Huang et al. in "Outsourcing and the Decrease of IS Program Enrollment" (June 2008). Although parents are a primary source of career guidance for students, they also derive their opinions from conversations with practitioners, as well as from news reports. The personal conversations are likely to carry much more weight than the news reports or the pronouncements of academic advisors who have a stake in selling their programs.

Many IS professionals today do not offer a very positive picture of the industry when asked about their own careers. The oft-quoted statistics concerning the relatively small number of outsourced jobs misses the point. Outsourcing has changed the image of the profession from valued corporate partner to commodity service performed by the lowest bidder. The change in perception affects every IS professional in organizations undergoing outsourcing and its associated layoffs; everyone worries their jobs are next. Following my own experience in the profession, none of my sons is pursuing IS or CS as a major, studying it instead only as an adjunct to other majors.

Until practitioners address the profession's image problem, academic programs will continue to have trouble attracting the numbers of students needed to maintain a healthy profession in the U.S.

Jack Bush, Watkinsville, GA

Inspire Excitement

Both Peter J. Denning in "Voices of Computing" (Aug. 2008) and Rick Rashid in "Inspiring a New Generation of Computer Scientists" (July 2008) were right: There are too few computer programmers coming up, and we must go to the schools and recruit them. If students don't want jobs that pay upward of $60,000 annually, fine, there are plenty of people who need them and can be inspired to want them. Here are four ideas that could help recruit students:

Excitement of programming. Each of us can recall a program or system we developed that was exciting and that solved an important problem for our employer or client. Convey the excitement of programming;

A good life. Make the connection between well-paying jobs and a good life. Use images and videos to show the kinds of places where programmers can afford to work, live, and play. Tell students how they can give back to their communities but only if they have the wherewithal to do it;

Responsible for own success. Give students the tools they need to succeed. Lots of books explain what it takes to be a good student. Make sure schools have them, then get students to read them. Tell them the truth—that while schoolteachers and administrators, boys and girls clubs advisors, church officials, parents, and others can help, you are ultimately responsible for your own success; and

Share with administrators. Talk over these and other ideas with school administrators before giving your own presentations, eliciting and trying to incorporate their ideas. They have a strong stake in your success.

Charles Elliott, Philadelphia, PA

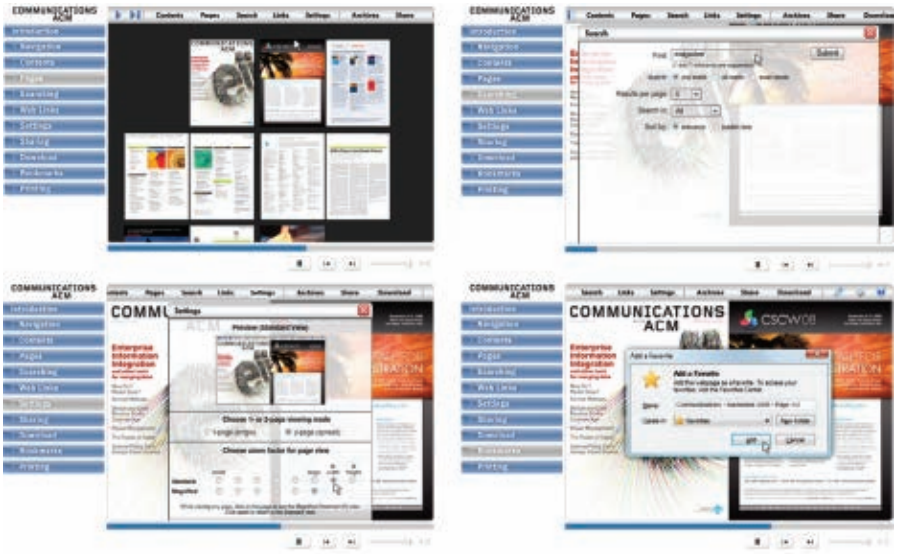
Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

DOI:10.1145/1400181.1400185 David Roman

Communications Site To Launch in January

That's Entertainment
The redesigned *Communications* Web site, slated to launch in January, will post material from this magazine, as well as from other ACM publications, including *Computers in Entertainment* (**cie.acm.org**). Look for video interviews with and written articles by noted researchers, academics, animators, game developers, film producers, and performance artists in the entertainment field to be part of the new *Communications* site.

Your Favorite Blog(s)
Dozens of blogs will be mined for the new *Communications* Web site. We welcome your input and invite you to share your personal favorite(s). There are a few rules: The blog(s) must be interesting, valuable, and written or managed by an ACM member. For details, go to **www.surveymonkey.com/recommend-a-blog.**



Video Tour
A new online video demonstrates how to navigate *Communications'* Digital Edition, a digital counterpart to the monthly magazine. Included are how the current issue can be searched, shared, and saved and how its settings can be personalized. The Digital Edition, which is emailed to all ACM members, is available at **cacm.acm.org**; the video is at **mags.acm.org/communications/current/include/mmhelp/mmhelp.html**.

***Queue*, Too**
Communications isn't the only ACM magazine redesigning its Web site. Look for *ACM Queue* to introduce a new site in the weeks ahead at **www.acmqueue.com**.

ACM Member News

ACM SIGCOMM AWARD WINNER
University of Massachusetts-Amherst professor of computer science Don Towsley was recently awarded the highest honor from ACM SIGCOMM, ACM's Special Interest Group on Data Communications, for his contributions to the modeling, analysis, and control of communication networks in the computer networking field. Towsley's research and leadership have helped shape an era of networking research and practice built upon a deeper scientific basis than in the past.

Towsley has made numerous innovative and pioneering contributions to foundational modeling and analysis techniques, which have enabled computer scientists and engineers to better understand today's computer networks, network protocols, and networked applications.

ACM VOTING EXPERT JOINS U.S. ADVISORY COMMITTEE
Computer scientist and founder of ACM's U.S. Public Policy Committee Barbara Simons was recently appointed to the board of advisors for the Election Assistance Commission, the U.S. federal body that oversees voting technology standards. A past president of the ACM, Simons fills a vacancy on the board. Her seat is one of four positions out of a total of 37 members allocated for science and technology professionals.

SC08 CONFERENCE
The leading international conference on high performance computing, networking, storage, and analysis, SC08 will be held in Austin, TX, from Nov. 15–21. The keynote speaker is Michael Dell, and the invited speakers include National Cancer Institute associate director Kenneth H. Buetow, University of California, Berkeley professor of computer science David Patterson, and Mary Wheeler, director of the Center for Subsurface Modeling at the University of Texas at Austin. For more information, visit sc08.supercomputing.org.

Society | DOI:10.1145/1400181.1400186

Patrick Kurp

Green Computing

Are you ready for a personal energy meter?

ANDY HOOPER INSISTS he's not a utopian, but his vision of the future of computing shares some resemblances with the dreams of science-fiction writers.

He foresees a not-too-distant time when the world's sources of computing power are concentrated in remote server warehouses strategically located near the sources of renewable energy that power them, such as wind and solar farms. And the usage of the power sources could shift across the globe, depending on where energy is most abundant.

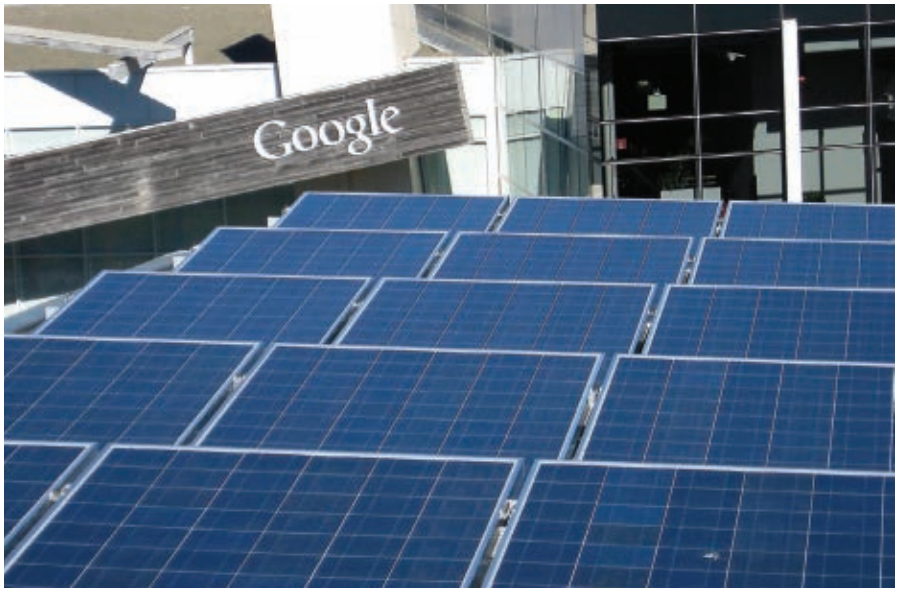
"The system we now employ is hugely wasteful," says Hopper, a professor of computer technology at the University of Cambridge and head of its Computer Laboratory. "We lose energy by relying on the national grid. I propose a system that is more efficient, much less expensive, and that would have an immediate impact on the world's energy consumption. It's always cheaper to move data than energy."

Hopper is among the more conspicuous and outspoken pioneers in the green computing movement—a multifaceted, global effort to reduce energy consumption and promote sustainability. Proposed and existing strategies range from the practical to the fanciful, and include government regulations, industry initiatives, environmentally friendly computers made of recyclable materials, and Hopper's suggestion of a personal energy meter.

Much of the green computing movement's focus today is on data centers, which have been lambasted as "the SUVs of the tech world" for their enormous and wasteful consumption of electricity. The approximately 6,000 data centers in the United States, for instance, consumed roughly 61 billion kilowatt-hours (kWh) of energy in 2006, according to Lewis Curtis, a strategic infrastructure architect at Microsoft. The total cost of that energy, $4.5 billion, was more than the cost of electricity used by all the color televisions in the U.S. in 2006, Curtis says.

The Department of Energy (DOE) reports that data centers consumed 1.5% of all electricity in the U.S. in 2006, and their power demand is growing 12% a year. If data centers' present rate of consumption continues, Curtis warns, they will consume about 100 billion kWh of energy at an annual cost of $7.4 billion by 2011.

The federal government wants data centers' energy consumption to be reduced by at least 10% by 2011. That translates into an energy savings equivalent to the electricity consumed by a million average U.S. households, according to Paul Sheathing, a spokesman for DOE's Office of Energy Efficiency and Renewable Energy.



PHOTOGRAPH COURTESY OF GOOGLE

"There's no simple path to green computing, but there are some low-hanging fruit," Curtis notes in "Green: The New Computing Coat of Arms?", a paper he co-authored with Joseph Williams, the CTO of WW Enterprise Sales at Microsoft. "You can spin the dial on some straightforward actions, such as orienting racks of servers in a data center to exhaust their heat in a uniform direction, thus reducing overall cooling costs.... A comprehensive plan for achieving green computing really does require an architectural approach."

David Wang, the data center architect for Teradata, has specialized in thermal management solutions for the Miamisburg, OH-based data warehousing company since 1996. "I've raised the issue [of green computing] because, for me, it's both a business question and an ethical question," Wang says. "Look at the basic fact, the one that has to be addressed: Power consumption at the server level has increased along with performance increase, and business needs have grown even faster."

More attention must be devoted to data centers' ever-increasing power density and heat removal, Wang says. "In the past, the sole focus was on IT equipment processing power and associated equipment spending. The infrastructure—power, cooling, data center space—was always assumed to be available and affordable," he says. "Now the infrastructure is becoming a limiting factor."

Microsoft, Google, and Yahoo are addressing the environmental concerns about their data centers' carbon footprint, the measure of the environmental impact of an individual or organization's lifestyle or operation, measured in units of carbon dioxide produced.

Google uses customized evaporative cooling to significantly reduce its data centers' energy consumption.

In recent years, Microsoft and other companies have built data centers in central Washington to take advantage of the hydroelectric power produced by two dams in the region. The Microsoft facility, which consumes up to 27 megawatts of energy at any given time, is powered by hydroelectricity.

"This way, because we're so close to the source, we're not losing any energy and the energy we do use is pure and clean," says Francois Ajanta, Microsoft's director of environmental strategy.

Another Microsoft data center, located in Dublin, Ireland, is expected to become operational in 2009 and, thanks to Ireland's moderate climate, the 51,000-square-meter facility will be air cooled, making it 50% more energy-efficient than other comparably sized data centers.

Google "has committed to being carbon-neutral for 2007 and beyond," says Bill Weihl, Google's director of energy strategy. "Our carbon footprint is calculated globally and includes our direct fuel use, purchased electricity, and business travel—as well as estimates for employee commuting, construction, and server manufacturing at our facilities around the world."

According to Google, its data centers use half the industry's average amount of power. Google attributes this improved energy usage to the cooling technologies, such as ultra-efficient evaporative cooling, that the company has customized for itself.

Yahoo's data centers also went carbon-neutral last year, in part because of its use of carbon offsets.

Government regulations and industry initiatives are also tackling data centers' energy usage. The U.S. Environmental Protection Agency (EPA), for instance, should have its phase-one version of Energy Star standards for servers ready by year's end. Eventually, the server rating will measure energy use at peak demand, but for the purpose of getting an Energy Star rating under way, the EPA will first release a Tier 1 standard, which will measure the efficiency of the server's power supply and its energy consumption while idle.

Meanwhile, a global consortium of computer companies, including AMD, Dell, IBM, Sun Microsystems, and VMware, organized The Green Grid in 2007, with the goal of improving energy efficiency in data centers and business computing systems. To achieve that goal, The Green Grid collaborates with individual companies, government agencies, and industry groups to provide recommendations on best practices, metrics, and technologies that will improve data centers' energy efficiency.

Earth-Friendly Computers

As with any evolving idea, people will need to think differently and more deeply when it comes to green comput-

Data Mining

Consumers' Invisible Profiles

Health and life insurance companies in the U.S. are increasingly using consumers' prescription drug data to determine what type of coverage, if any, to offer applicants, the *Washington Post* reports.

The insurance companies hire health information services companies—such as Ingenix, which had $1.3 billion in sales last year—to help create consumer profiles. The health information services companies mine the databases of prescription drug histories that are kept by pharmacy benefit managers (PBMs), which help insurers to process drug claims. (Ingenix even has its own servers located in some PBM data centers.) The health information services companies also access patient databases held by clinical and pathological laboratories.

The health information services companies say that consumers have authorized the release of their records and that their approach saves insurance companies money and time. Privacy advocates note that consumers do sign consent forms authorizing the release of data, but they have to if they want insurance, and that many people are unaware of the existence of health information services companies.

ing. It is not unusual, for instance, for companies to replace their older computers with new, more energy-efficient ones in an effort to become more earth-friendly.

This practice might not always be the most environmental solution, says Teradata's Wang. "What I propose is that we look at the entire life cycle of a computer, the whole picture, from manufacturing through day-to-day operation," says Wang. "Every step consumes energy, and buying a new, more efficient computer may not always be the answer."

Some computer manufacturers are retooling their products from a life-cycle point of view and making the decision to buy a new, energy-efficient computer much easier. Dell is accelerating its programs to reduce hazardous substances in its computers, and its new OptiPlex desktops are 50% more energy-efficient than similar systems manufactured in 2005, thanks to more energy-efficient processors, new power management features, and other factors.

Likewise, Hewlett-Packard recently unveiled what it calls "the greenest computer ever"—the rp5700 desktop PC. The rp5700 exceeds U.S. Energy Star 4.0 standards, has an expected life of at least five years, and 90% of its materials are recyclable. The computer is easy to disassemble and meets the European Union's RoHS standards for the restriction of the use of certain hazardous substances in electrical and electronic equipment. Moreover, 25% of the rp5700's packaging materials are made of recycled material.

For the Future of the Planet

In an effort to ensure "computing can have a positive effect on our lives and the world," Hopper and Andrew Rice, an assistant director of research at the University of Cambridge's Computer Laboratory, have identified four principal goals in their paper "Computing for the Future of the Planet." The first goal is an optimal digital infrastructure in which computing's overall energy consumption is reduced and the efficient use of energy in the manufacture, operation, and disposal of computing devices is maximized.

The second goal is "to sense and optimize the world around us with reference to a global world model," which would "inform us about the energy consumption and other effects of our activities on the natural environment."

The third goal is a new emphasis on predicting and responding to future events by modeling their behavior. According to Hopper and Rice, "The traditional role of computing as an execution platform for these models will continue to be important and must grow in performance to service both the increasing demands of higher-fidelity models and also to accommodate any new overheads incurred by correctness checking."

Lastly, Hopper and Rice are "interested in the possible benefit of digital alternatives to our physical activities," such as electronic versions of printed newspapers, music downloads rather than physical CDs, and online shopping as opposed to visiting stores and supermarkets. According to Hopper and Rice, "One might argue that a total shift from physical to digital seems unlikely in today's world but for future generations this concept might seem as obvious as email is to us today."

"People in the developing world," Hopper and Rice note, "often live in resource-impoverished environments so a physical-to-digital paradigm shift has the potential to enable activities that were hitherto prohibitively expensive, and to support development whilst minimizing its impact. We seek to unlock methods of wealth creation in the virtual world."

Hopper and Rice also suggest the development of a personal energy meter that would measure a person's direct and indirect daily consumption, with individualized breakdowns of "the energy costs of travel, heating, water-usage and transportation of food [that] will help us target areas for reduction in our environmental footprint.... The data collected will not only provide useful information for analyzing consumption patterns but also has the potential to help individuals identify alternatives to their current activities."

"I think we've only just started to address the issue" of green computing, says Hopper. "It's just on the cusp of becoming important, and I think business, not academia, has led the way. They are driven by pragmatic concerns." C

Patrick Kurp is a freelance science writer in Bellevue, WA.

Artificial Intelligence

Supercomputer Defeats Human Go Pro

The new Dutch supercomputer Huygens, armed with the MoGo Titan program, defeated a human professional Go player with a 9-stones handicap. The victory appears to be the first-ever defeat of a high-level human Go player by a supercomputer in an official match.

Until recently, scientists were unable to create a computer program capable of beating even many amateur-level Go players. This state of affairs changed in 2006 when programmers Sylvain Gelly and Yizao Wang devised a revolutionary algorithm that has enabled the MoGo Titan program to attain new heights; since August 2006, MoGo Titan has been ranked number one on the 9x9 Computer Go Server.

Teamed up with the Huygens supercomputer, MoGo Titan achieved a noteworthy victory as its opponent, Kim Myungwan, is an 8 dan pro (the highest level is 9 dan) and a seasoned international competitor. In fact, the day before Myungwan's official match with Huygens and MoGo Titan, he soundly defeated the duo in three blitz games played with varying handicaps.

"The current result forecasts that before 2020 a computer program will defeat the best human Go player on a 19x19 Go board in a regular match under normal tournament conditions," says professor Jaap van den Herik of Maastricht University which, with INRIA France, co-developed MoGo Titan. "This is remarkable, since around 2000 it was generally believed that the game of Go was safe to any attack by a computer program. The 9-stones handicap victory casts severe doubts on this belief."

The Korean-born Myungwan appears to have taken the defeat well. Two days after his loss to MoGo Titan, he won the 2008 U.S. Open.

Technology | DOI:10.1145/1400181.1400187 Alex Wright

Searching the Deep Web

While the Semantic Web may be a long time coming, Deep Web search strategies offer the promise of a semantic Web.

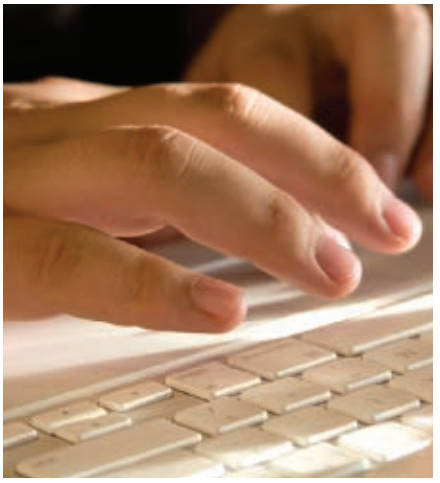
THE WEB IS bigger than it looks. Beyond the billions of pages that populate the major search engines lies an even vaster, hidden Web of data: classified ads, library catalogs, airline reservation systems, phone books, scientific databases, and all kinds of other information that remains largely concealed from view behind a curtain of query forms. Some estimates have pegged the size of the Deep Web at up to 500 times larger than the Surface Web (also known as the Shallow Web) of static HTML pages.

Researchers have been trying to crack the Deep Web for years, but most of those efforts to date have focused on building specialized vertical applications like comparison shopping portals, business intelligence tools, or top-secret national security projects that scour hard-to-crawl overseas data sources. These projects have succeeded largely by targeting narrow domains where a search application can be fine-tuned to query a relatively small number of databases and return highly targeted results.

Bringing Deep Web search techniques to bear on the public Web poses a more difficult challenge. While a few high-profile sites like Amazon or YouTube provide public Web services or custom application programming interfaces that open their databases to search engines, many more sites do not. Multiply that problem by the millions of possible data sources now connected to the Web—all with different form-handling rules, languages, encodings, and an almost infinite array of possible results—and you're have one tough assignment. "This is the most interesting data integration problem imaginable," says Alon Halevy, a former University of Washington computer science professor who is now leading a Google team trying to solve the Deep Web search conundrum.

Deep Web Search 101

There are two basic approaches to searching the Deep Web. To borrow a fishing metaphor, these approaches might be described as trawling and angling. Trawlers cast wide nets and pull them to the surface, dredging up whatever they can find along the way. It's a brute force technique that, while inelegant, often yields plentiful results. Angling, by contrast, requires more skill. Anglers cast their lines with precise techniques in carefully chosen locations. It's a difficult art to master, but



when it works, it can produce more satisfying results.

The trawling strategy—also known as warehousing or surfacing—involves spidering as many Web forms as possible, running queries and stockpiling the results in a searchable index. While this approach allows a search engine to retrieve vast stores of data in advance, it also has its drawbacks. For one thing, this method requires blasting sites with uninvited queries that can tax unsuspecting servers. And the moment data is retrieved, it becomes instantly becomes out of date. "You're force-fitting dynamic data into a static document model," says Anand Rajaraman, a former student of Halevy's and co-founder of search startup Kosmix. As a result, search queries may return incorrect results.

The angling approach—also known as mediating—involves brokering a search query in real time across multiple sites, then federating the results for the end user. While mediating produces more timely results, it also has some drawbacks. Chief among these is determining where to plug a given set of search terms into the range of possible input fields on any given Web form. Traditionally, mediated search engines have relied on developing custom "wrappers" that serve as a kind of Rosetta Stone for each data source. For example, a wrapper might describe how to query an online directory that accepts inputs for first name and last name, and returns a mailing address as a result. At Vertica Systems, engineers create these wrappers by hand, a process that usually takes about 20 minutes per site. The wrappers are then added to a master ontology stored in a database table. When users enter a search query, the engine converts the output into Resource Description Framework (RDF), turning each site into, effectively, a Web service. By looking for subject-verb-object combinations in the data, engineers can create RDF triples out of regular Web search results. Vertica founder Mike Stonebraker freely admits this hands-on method, however, has limitations. "The problem with our approach is that there are millions of Deep Web sites," he says. "It won't scale." Several search engines are now experimenting with approaches for developing automated wrappers that can scale to accommodate the vast number of Web forms available across the public Web.

The other major problem confronting mediated search engines lies in determining which sources to query in the first place. Since it would be impossible to search every possible data source at once, mediated search engines must identify precisely which sites are worth searching for any given query.

"You can't indiscriminately scrub dynamic databases," says former BrightPlanet CEO Mike Bergman. "You would not want to go to a recipe site and ask

PHOTOGRAPH BY AMANCAY MAAHS

about nuclear physics." To determine which sites to target, a mediated search engine has to run some type of textual analysis on the original query, then use that interpretation to select the appropriate sites. "Analyzing the query isn't hard," says Halevy. "The hard part is figuring out which sites to query."

At Kosmix, the team has developed an algorithmic categorization technology that analyzes the contents of users' queries—requiring heavy computation at runtime—and maps it against a taxonomy of millions of topics and the relationships between them, then uses that analysis to determine which sites are best suited to handle a particular query. Similarly, at the University of Utah's School of Computing, assistant professor Juliana Freire is leading a project team working on crawling and indexing the entire universe of Web forms. To determine the subject domain of a particular form, they fire off sample queries to develop a better sense of the content inside. "The naïve way would be to query all the words in the dictionary," says Freire. "Instead we take a heuristic-based approach. We try to reverse-engineer the index, so we can then use that to build up our understanding of the databases and choose which words to search." Freire claims that her team's approach allows the crawler to retrieve better than 90% of the content stored in each targeted site.

Google's Deep Web search strategy has evolved from a mediated search technique that originated in Halevy's work at Transformic (which was acquired by Google in 2005), but has since evolved toward a kind of smart warehousing model that tries to accommodate the sheer scale of the Web as a whole. "The approaches we had taken before [at Transformic] wouldn't work because of all the domain engineering required," says Halevy.

Instead, Google now sends a spider to pull up individual query forms and indexes the contents of the form, analyzing each form for clues about the topic it covers. For example, a page that mentions terms related to fine art would help the algorithm guess a subset of terms to try, such as "Picasso," "Rembrandt," and so on. Once one of those terms returns a hit, the search engine can analyze the results and refine its model of what the database contains.

Rather than relying on Web site owners to mark up their data, couldn't search engines simply do it for them?

"At Google we want to query any form out there," says Halevy, "whether you're interested in buying horses in China, parking tickets in India, or researching museums in France." When Google adds the contents of each data source to its search engine, it effectively publishes them, enabling Google to assign a PageRank to each resource. Adding Deep Web search resources to its index—rather than mediating the results in real time—allows Google to use Deep Web search to augment its existing service. "Our goal is to put as much interesting content as possible into our index," says Halevy. "It's very consistent with Google's core mission."

A Deep Semantic Web?

The first generation of Deep Web search engines were focused on retrieving documents. But as Deep Web search engines continue to penetrate the far reaches of the database-driven Web, they will inevitably begin trafficking in more structured data sets. As they do so, the results may start to yield some of the same benefits of structure and interoperability that are often touted for the Semantic Web. "The manipulation of the Deep Web has historically been at a document level and not at the level of a Web of data," says Bergman. "But the retrieval part is indifferent to whether it's a document or a database."

So far, the Semantic Web community has been slow to embrace the challenges of the Deep Web, focusing primarily on encouraging developers to embrace languages and ontology definitions that can be embedded into documents rather than incorporated at a database level. "The Semantic Web has been focused on the Shallow Web," says Stonebraker, "but I would be thrilled to see the Semantic Web community focus more on the Deep Web."

Some critics have argued that the Semantic Web has been slow to catch on because it hinges on persuading data owners to structure their information manually, often in the absence of a clear economic incentive for doing so. While the Semantic Web approach may work well for targeted vertical applications where there is a built-in economic incentive to support expensive mark-up work (such as biomedical information), such a labor-intensive platform will never scale to the Web as a whole. "I'm not a big believer in ontologies because they require a lot of work," says Freire. "But by clustering the attributes of forms and analyzing them, it's possible to generate something very much like an ontology."

While the Semantic Web may be a long time coming, Deep Web search strategies hold out hope for the possibility of a semantic Web. After all, Deep Web search inherently involves structured data sets. Rather than relying on Web site owners to mark up their data, couldn't search engines simply do it for them?

Google is exploring just this approach, creating a layer of automated metadata based on analysis of the site's contents rather than relying on site owners to take on the cumbersome task of marking up their content. Bergman's startup, Zitgist, is exploring a concept called Linked Data, predicated on the notion that every bit of data available over the Web could potentially be addressed by a Uniform Resource Indicator. If that vision came to fruition, it would effectively turn the entire Web into a giant database. "For more than 30 years, the holy grail of IT has been to eliminate stovepipes and federate data across the enterprise," says Bergman, who thinks the key to joining Deep Web search with the Semantic Web lies in RDF. "Now we have a data model that's universally acceptable," he says. "This will let us convert legacy relational schemas to http."

Will the Deep Web and Semantic Web ever really coalesce in the real world of public-facing Web applications? It's too early to say. But when and if that happens, the Web may just get a whole lot deeper. C

Alex Wright is a writer and information architect who lives and works in New York City.

Science | DOI:10.1145/1400181.1400188 Cyrus Farivar

Clean Elections

With end-to-end auditable voting, a voter can verify whether his or her vote was tallied correctly and whether all of the votes were properly tabulated.

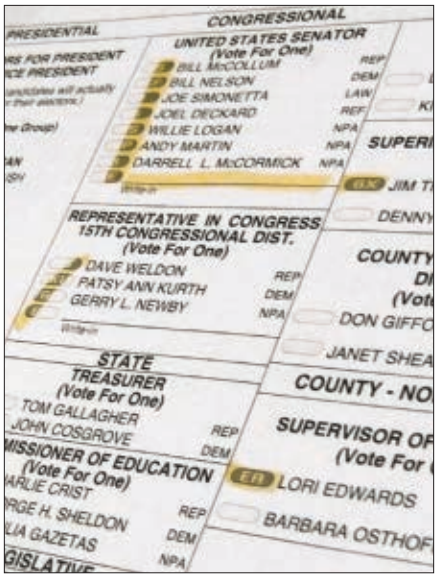
DESPITE THE RENEWED calls for improved voting systems after the debacle in Florida during the 2000 presidential election, little has changed in the way that America votes eight years later. Today, the country still has a veritable mishmash of voting standards and methods. Some counties use lever machines, some use paper ballots, some use electronic voting machines, and still others, after having tried electronic voting, have reverted to the paper ballots that they previously used.

While the debate about the merits of electronic voting versus paper ballots continues in public policy and technology circles, one approach might put the entire controversy to rest: end-to-end (E2E) auditable voting. Also known as E2E verifiable voting, E2E auditable voting ensures the transition from the accurately recorded single ballot to the tally of collected ballots is preserved and maintained in a publicly auditable manner, and enables voters to verify that their individual votes were recorded accurately as well as the ability to show, with a high degree of probability, that all of the ballots were properly tabulated.

(For a debate about electronic voting, see the Point/Counterpoint column on p. 29.)

E2E auditable voting is different from voter-verified paper trail ballots, which addresses the problem of whether each single ballot was recorded correctly, but do not ensure that all of the votes were tallied accurately. The idea behind E2E auditable voting—which uses paper ballots or electronic voting—is that the entire voting system utilizes cryptography to accurately count votes while at the same time preserving the voters' privacy.

"The basic concept, which is almost miraculous, is that a voter can cast a ballot, check that the ballot was counted, and verify that the totals are accurate, without anyone else knowing how they voted, even if the voter wants to prove how they voted to a third party," says David Dill, a professor of computer science at Stanford University.



Sample invisible ink ballot.

Computer scientists, mathematicians, and cryptographers in the United States and abroad have spent years working on E2E auditable voting systems. One early contender was VoteHere, an E2E electronic system developed by Andy Neff, but VoteHere changed its name to Dategrity Corp. in 2005, with the goal of reaching a wider market for its auditing and verification software.

Private Data, Public Verification

After years of research, one approach has captured the most attention in the world of E2E auditable voting and is the farthest along in terms of actually being implemented anytime soon in the United States.

Scantegrity II (the "II" stands for "invisible ink") was developed largely by independent cryptographer David Chaum in collaboration with other scientists in the field, including Ron Rivest, a professor of computer science at MIT; Peter Ryan, a professor of computer science at the University of Newcastle upon Tyne; and Stefan Popoveniuc, a graduate student in the computer science department at George Washington University.

Scantegrity II relies upon a technique of cryptography known as a "cut and choose protocol," which enables zero-knowledge proofs. This technique of cryptography relies on zero-knowledge proofs to show that the information has been encrypted without revealing what the original piece of information is. In this case, it proves that the results were accurately tabulated without revealing how each vote was cast.

A Scantegrity II ballot is similar to a traditional ballot with a list of candidates and an adjacent row of fill-in bubbles. In order to vote, each person uses a special pen to reveal a unique, hidden three-character code that is printed in invisible ink in each bubble. The three-character code serves as a cryptographic marker to indicate the voter's preference. However, without the decryption key, the code is meaningless.

The code effectively encrypts, or locks in, the voter's preference. Then, each possible code for each ballot is randomized and displayed publicly on a Web site.

Once the codes are randomized, a set of tables are used to map each code to a particular candidate. However, the precise path of the trails is concealed under a two-step procedure that connects the location of the coded vote to how that translates to its location on the results board.

After voting, each voter receives a small tear-off receipt containing the serial number of the ballot. The voter can, if he or she chooses to, write down the revealed three-character code, which, when entered in a public Web site, can verify their vote was recorded correctly.

PHOTOGRAPH BY JOE HALL

In order to audit a ballot, an auditor or a voter would use the original ballot key to disclose all possible codes for all candidates, thus revealing the entire computational chain to prove whether the votes were recorded and tabulated accurately. And to verify that a person hasn't cheated during any step of the process, one of the pointers can be revealed to assure that the proper connection between the results and the anonymized encoded ballots is maintained.

Scantegrity II is not a replacement voting system, but works with either precinct-based or central scan systems, a feature that Chaum says makes it attractive to public officials. The printing of ballots, however, requires the capability to print with invisible ink and to print each ballot differently.

Scantegrity II might be used in an upcoming municipal election in Takoma Park, MD, a suburb of Washington, D.C., but the city council has not yet made a final determination.

End-to-end auditable voting systems could put the controversy about the merits of electronic voting versus paper ballots to rest.

Different Approaches, Same Goal

While Scantegrity II appears to be the most public election-ready system, there are several replacement systems—most notably Prêt à Voter (or Ready to Vote), Scratch & Vote, and ThreeBallot—that use different cryptography-based approaches to achieve the same goal of end-to-end voter verifiability.

Developed by Peter Ryan of University of Newcastle upon Tyne, Prêt à Voter does not rely upon the voted values to be encrypted and randomized, but uses a random candidate order that varies from ballot to ballot. Once a vote is cast, the side of the ballot with the list of candidates is destroyed.

The bottom of the non-discarded side of the Prêt à Voter ballot contains a cryptographic string with information on the discarded candidate list order. In order to decrypt the candidate list order and determine the value of a vote, voting officials or party representatives use a series of secret keys to decrypt the ballots.

Prêt à Voter was successfully used in student elections at the University of Surrey last year, and Ryan plans to test Prêt à Voter again in upcoming student elections at the University of Newcastle upon Tyne.

Scratch & Vote was developed by Ben Adida, a research fellow with the Center for Research on Computation and Society at Harvard University, and Ron Rivest, a professor of computer science at MIT, and its format is similar to Prêt à Voter.

A Scratch & Vote ballot is perforated down the middle, and the left side has a list of candidates' names and the right side has a series of corresponding check boxes. Beneath the check boxes is a 2D-barcode. A scratch surface is positioned below the barcode, and a perforation separates the scratch surface from the rest of the right half of the ballot.

If the voter wants to audit the voting process, she selects a second ballot and removes the scratch surface, thereby voiding the ballot, which the voter gives to a trusted party on the premises. The trusted party scans the barcode, reads the randomization data previously hidden under the scratch surface, and can confirm the ballot is correctly formed.

The voter now makes her selection on the first ballot, discards the left side of the ballot (which contains only the randomized candidate order), and gives the ballot to an election official. The election official ensures the scratch surface is intact and detaches the scratch surface for the purpose of discarding it. The voter enters the ballot's remaining checkmark (to indicate her vote) and barcode, which is effectively the encrypted ballot, into a scanner.

The voter can take the remainder of the ballot home and check on a public Web site that her ballot was correctly tabulated, and if it wasn't, she still possesses the remainder of the ballot. In addition, the voter can audit the tally process and the verifiable decryption conducted by the election officials.

With Scratch & Vote, each vote is recorded as an encrypted value by using the box that was checked to determine which encrypted value to use. The values are tallied using homomorphic encryption, which allows for the sum of two encrypted values to be equal to the

Artificial Intelligence

Cooperative Robot Swarms

An enterprising group of undergraduate students at the University of Southampton unveiled a group of inexpensive and identical, matchbox-sized robots at the recent Artificial Life XI conference. The robots communicate with each other via an infrared technology used in mobile phones, and can independently divide up tasks, without instructions from a central control program.

In a demonstration at the conference, the robots, which have green and red lights, autonomously divided themselves into two groups, 80% red and 20% green. When some of the "green" robots were removed from the group, the remaining robots reorganized into an 80/20 split.

Swarms of robots have certain advantages over a single, self-contained robot, according to some roboticists. "You might have some complex robot that is sent to Mars, has a technical problem, and then the mission is basically over," said Claus-Peter Zauner, the leader of the swarm robot project, in an interview with the BBC News. "With swarm robots, even if a percentage of them fails, they'll still be able to achieve their goal."

encrypted sum of that value. In other words, by adding the encrypted ballots together and decrypting them all at once, the candidates for whom the votes were cast can be determined.

"I'm not actively pursuing [Scratch & Vote] for implementation, though I use it regularly to teach the concepts," Adida said via email. "I think it might be a useful system for certain simple elections, but it might simply be more useful as a teaching tool. It's helped a number of folks understand the power of open-audit voting, even if they quickly forget the details."

Invented by Rivest, the ThreeBallot Voting System entails giving a voter three identical ballots. To vote for a candidate, a voter must select that candidate on two of the three ballots. To vote against a candidate, the voter must select that candidate on one ballot. At the polling station, the voter makes a copy of any one of the three ballots, which he or she retains, and the three original ballots are placed in the ballot box.

At the election's conclusion, all of the ballots are published. As each ballot contains a unique seven-digit identifier, a voter can independently verify that his or her vote was counted by searching for the identifier among the published ballots.

Scratch & Vote has helped people "understand the power of open-audit voting, even if they quickly forget the details," says Ben Adida.

ThreeBallot offers some of the advantages of a cryptographic voting system without actually using cryptography. MIT students have conducted a field test with ThreeBallot, however, and discovered problems in terms of usability, privacy, and security.

While many academic experts say that the science behind E2E auditable systems is promising, they also note the need for further research and usability studies.

"We are at the stage where we need to try many different techniques for open-audit voting, and we just don't know what's going to work better in a real-world setting," Adida said via email. "Deciding on a single system now would be putting the cart before the horse."

David Wagner, a professor of computer science at the University of California at Berkeley, suggests that a widespread, multiyear study could be the best way to advance E2E verifiable voting research. "The next step is that you need a system that is very concretely worked out," Wagner says. "One of the things about cryptography is that the devil's in the details. When you're using this fancy mathematics, there's all these details to get right and any one little slip-up can compromise the security of the whole system." C

Based in Oakland, CA, **Cyrus Farivar** freelances as a technology journalist and a radio reporter and producer.

Information Technology

Intel Cuts Electric Cords

Intel researchers have demonstrated a wireless electric power system that could enable notebook computers and other consumer devices to be powered without wall outlets and transformers.

In a recent demonstration at Intel's annual Developer Forum in San Francisco, electricity was wirelessly sent a distance of several feet to a lamp on stage, illuminating its 60 watt bulb, which uses more electricity than the average laptop.

"Something like this technology could be embedded in tables and work surfaces," Intel chief technology officer Justin Rattner told the *New York Times*, "so as soon as you put down an appropriately equipped device it would immediately begin drawing power."

Known as "wireless energy resonant link," the Intel technology could also be embedded in computer components, such as monitors, enabling them to send power to nearby devices.

Bioengineering

Fits on a Fingertip

California Institute of Technology researchers have developed a high-resolution, lens-less microscope that, due to its tiny size, can fit on a fingertip. Called an optofluidic microscope, the device combines traditional computer chip technology with microfluidics, the channeling of fluid flow at incredibly small scales, and uses sunlight for illumination.

Developed by a team of researchers led by Changhuei Yang, an assistant professor of electrical engineering and bioengineering, the microscope has the magnifying power of a high-quality optical microscope and could be used in developing countries to analyze blood samples for malaria or to check water supplies for pathogens.

News | DOI:10.1145/1400181.1401826 Leah Hoffmann

An Inspiring Legacy

Admired and respected by his students and colleagues, Randy Pausch will be remembered as a devoted teacher and innovative researcher.

EXPERIENCE IS WHAT *you get when you don't get what you want...* It was one of several proverbs Randy Pausch managed to infuse with fresh relevance during his "Last Lecture" last fall. The saying was the sort of thing he loved—clichés, the old chestnuts, stories, and snippets of advice he'd collected from colleagues and friends. Pausch had a knack for selecting the right anecdote for an occasion, and for telling it in a way that invested it with new meaning and power. The fact that his stories were entertaining never seemed to diminish their pedagogical value.

We can't change the cards we are dealt, just how we play the hand was another proverb Paush discussed during his "Last Lecture." His own cards, of course, included a fierce battle with the pancreatic cancer that would eventually claim his life on July 25 at the age of 47. Over the course of that battle, he managed to inspire millions of people with a heartfelt talk (and a surprise YouTube video hit) about living well and overcoming obstacles. He also found time to raise awareness and research funding to help others with pancreatic cancer. And he carefully compiled a treasury of pictures, mementos, and opinions for his family.

Those who knew him weren't surprised. "It's vintage Randy," says Gabriel Robins, a professor of computer science and former colleague at the University of Virginia. "He took his own demise and turned it into an educational bonanza."

An innovative researcher and devoted teacher, Pausch is best known in his field for his pioneering work on the Alice Project, a sophisticated computing environment that teaches students how to program through an intuitive graphical interface. His passion for storytelling deeply informed his work on Alice, which enables even middle-school-age children, after just a few hours of online training, to create 3D

PHOTOGRAPH BY DR. GABRIEL ROBINS

animations. As students concentrate on making games and movies, Pausch discovered, they forget they're also learning how to program.

"Alice gets people hooked into the big picture of computer science, instead of the syntactical details," explains Dan Siewiorek, director of Carnegie Mellon University's (CMU's) Human-Computer Interaction Institute. "He had a way of cutting straight to the issue and getting at the kernel."

Alice was the foundation of Pausch's popular course on building virtual worlds, which drew students from numerous departments to collaborate on interactive animations. It also paved the way for CMU's Entertainment Technology Center (ETC), a joint program created by Pausch and Don Marinelli, a professor of drama and arts management. The ETC offers a two-year master's degree so technologists and artists can collaborate on projects in digital entertainment.

"Randy was fearless," says Andries van Dam, a former mentor and professor of computer science at his undergraduate alma mater, Brown University. Though Alice remains Pausch's main legacy, it was far from the only contribution he made to the field. A 1992 project at the University of Virginia called Virtual Reality on Five Dollars a Day, for example, also stands as a testament to his talent and resourcefulness. At the time, virtual reality systems were both bulky and expensive. Pausch managed to put one together using parts from a commodity PC and store-bought toys. The system's total cost: approximately $5,000, or less than $5 for each day he'd spent working on it.

"You could always count on Randy to have an unconventional opinion," says van Dam. "It's exactly the way science should work."

Pausch's accomplishments were celebrated within the scientific community. Among the honors he received are ACM's Karl V. Karlstrom Outstanding Educator Award, the ACM Special Interest Group on Computer Science Education Award for Outstanding Contributions to Computer Science Education, and the National Science Foundation's Presidential Young Investigator Award. His colleagues applauded his aptitude for bringing people from different disciplines together in a spirit of collaboration. And his students remember his ability to make computer science come alive, not just through the showmanship of his lectures, but through thoughtful, well-chosen examples.

Pausch's story will be told for years to come. His work lives on, as well. Van Dam and his colleagues at Brown have raised money to endow an undergraduate research internship in Pausch's name. Carnegie Mellon will honor his legacy with a memorial footbridge that connects its Center for Computer Science to an adjacent arts building. It has also created a memorial fund to enable researchers to continue Pausch's work on Alice. Perhaps the best memorial, however, comes from Pausch himself. His "Last Lecture" continues to inspire and amaze YouTube viewers across the world, and the best-selling book-length version has already been translated into 30 languages. C

Based in Brooklyn, NY, **Leah Hoffmann** is a science and technology writer.

Group Term Life Insurance**

10- or 20-Year Group Term Life Insurance*

Group Disability Income Insurance*

Group Accidental Death & Dismemberment Insurance*

Group Catastrophic Major Medical Insurance*

Group Dental Plan*

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan***

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh Affinity Group Services at **1-800-503-9230** or visit **www.personal-plans.com/acm.**

The plans are subject to the terms, conditions, exclusions and limitations of the group policy. For costs and complete details of coverage, contact the plan administrator. Coverage may vary and may not be available in all states.

*Underwritten by The United States Life Insurance Company in the City of New York, a member company of American International Group, Inc.

**Underwritten by American General Assurance Company, a member company of American International Group, Inc.

***Coverage is available through Assurant Health and underwritten by Time Insurance Company.

AG5217

MARSH
Affinity Group Services
a service of Seabury & Smith

DOI:10.1145/1400181.1400189

Martin Campbell-Kelly

Historical Reflections

Will the Future of Software be Open Source?

Tracing the course of influential computing developments and considering possible paths to new paradigms.

IF ONE WAS forecasting the future of software today, it is likely that open source software (OSS) would figure prominently in most projections. Indeed, open source zealots might expect to see OSS everywhere, with "innovation networks" abounding, Microsoft humbled, and Linux on every desktop. Personally, I wouldn't bet on it.

Historians are cautious about forecasting the future, with good reason. They know that when technical experts gaze into the crystal ball, they usually extrapolate well but fail to spot those discontinuities that can transform a technology. One such attempt at futurology was the book *The Future of Software*, published in 1995.[a] The book included contributions from leading experts in the field. They correctly extrapolated that PCs would become more powerful, numerous, pervasive, and software would proliferate to fill the applications vacuum. That was correct to a point, but their collective

take on new software development methods and technologies was wide of the mark. One contributor forecast that visual programming by ordinary users would herald the "fall of software's aristocracy." Another predicted the maturing of the software factory, by which our "craft industry" would be transformed "toward Ford-style mass production." Another contributor expected to see stunning advances in natural language interfaces. What no contributor foresaw, or even mentioned, was the impact of open source software and development techniques. At the very moment they were making their projections, Linux was under their nose but they could not see it.

The idea of open source software goes back to the very dawn of computing, when the mainframe computer was getting established in the early 1950s. At that time, and for many years after, IBM and the other computer manufacturers gave their software away for free—software was seen largely as a marketing initiative that made their hardware more saleable. Software was supplied in both source and object code form because some people found the source code useful and there was no reason not to let them have it. Where manufacturers' provision fell short, cooperative user groups, such as IBM's SHARE, coordinated the writing and free distribution of programs. When it came to applications, computer users wrote their own or hired a "software contractor," such as the Computer Sciences Corporation or Electronic Data Systems, to write software for them.

There was a radical transformation in the software world in 1964, with the

a Leebeart, D. Ed., *The Future of Software*. MIT Press, Cambridge, MA, 1995.

ILLUSTRATION BY CELIA JOHNSON

launch of IBM's System/360 computer. The 360 created, for the first time, a standard computer platform, and it massively expanded the computer population, particularly in medium-sized businesses. Most of the new computer owners did not have the resources to hire a staff of programmers or to buy the services of a software contractor. There was thus an applications vacuum filled by the first software product firms. These firms wrote programs for specific industries (such as the insurance or construction industries), or for generic, cross-industry functions (such as payroll or stock control). The sales of individual software products were quite modest: if a product had 100 or so customers it was considered quite successful. Software product prices were high, typically $50,000 upward. This was not only because of the low sales volume, but because software writing was very capital intensive. The only way to run a software business was to hire a team of programmers plus a mainframe computer and put them to work. This cost at least $1 million a year (closer to $10 million in today's currency).

The first software products were usually supplied in both source code and object code. This was necessary because customizing software was a little-understood technology and most users configured their application software by modifying the source and recompiling it. Software-product companies were, naturally, concerned about disclosing source code, because if it fell into the hands of a competitor it would make it easy for them to produce a competing product. In a somewhat uneasy compromise, paying customers received a copy of the source code but were bound by the license terms with a trade secrecy clause requiring them not to disclose the source code or documentation to third parties.

The advent of personal computers, which occurred during the late 1970s, gave rise to a new software industry that rewrote the rules for making and selling software. The cost of computer power plummeted, the computer population soared, and the number of software firms increased exponentially. However, although the hardware-cost barrier to software making had been lowered, code development still needed a disciplined environment of salaried programmers who worked office hours in the same physical location. Although computer networks existed in the 1980s, they were slow and impractical—software development remained a same-time, same-place, collaborative activity. PC software products were comparatively inexpensive (usually less than $500), but this was only because the sales volume was high compared with mainframe software. Software writing remained an expensive, highly capitalized activity.

In the new PC environment, with thousands of software companies and millions of users, it was no longer feasible for software companies to supply their source code to users, or their products would be rapidly duplicated. Firms such as Microsoft, Lotus, and WordPerfect had invested hundreds of millions of dollars in software development; disclosing their software would have been akin to giving away the family jewels. Of course, software had some legal protection through copyright laws, but this did not protect the data structures and algorithms that would have been exposed by access to the source code. By the mid-1980s source code disclosure had almost completely ceased—in 1983, IBM was one of the last major companies to stop disclosing source code in its so-called OCO (object-code only) policy. Competitors and users alike objected to the OCO policy, but IBM was resolute and was doing no more or less than the rest of the industry. By the mid-1980s, trade secrecy was endemic in the software products industry.

The ascendancy of the Internet in the early 1990s began another radical transformation of software development. Inexpensive network access removed

The idea of open source software goes back to the very dawn of computing, when the mainframe computer was getting established.

the constraint of having salaried programmers working together in a dedicated facility. It was now possible for programmers to collaborate in software development via the Internet—whether they were salaried personnel or volunteers, and whether they were trained computer professionals or talented amateurs. This was the birth of today's open source community. Linux was the defining product of the community, and the open source principle was also responsible for a large portion of the Internet infrastructure. Besides enabling the new open source development regimen, the Internet also removed the barriers to software distribution. Whereas the existing software products industry had used retail channels, which could carry only a limited range of products, or had used an (expensive) sales force, open source products were freely available for downloading from the Internet. Open source programs soon appeared in many of the established software categories.

In the initial euphoria of open source in the mid-1990s, it looked as though in the future software would be "free" in both senses of the word: free of cost to consumers, and with freely available source code. Ten years on, however, it became clear that nothing was that simple. Fundamentally, open source was a new development method. Traditionally, code development accounted for 10%–15% of the cost of a software product. The rest of the cost was for activities such as marketing, packaging, and after-sales support (for example, telephone help lines). For users, too, software was only a fraction of what came to be called the TCO (total cost of ownership), which included computer and infrastructure costs and technical support. Today, there are numerous firms supplying open source products, and their cost structure turns out to be not very different compared to traditional software firms. They spend 10%–15% of their income on code development, and the rest is taken up with activities such as marketing and after-sales support. Because of the open source development method, it may well be that their products are better and less expensive than their proprietary equivalents, but for most users they do not drastically change their total information processing costs.

So, if a person was attempting to peer into the future of software today, what would he or she predict? Such a forecast has two dimensions: first, predictable extrapolation, and, second, the unknowable paradigm shifts that might take place. Predictably, we can expect the open source paradigm to gain in strength and to be increasingly adopted by the traditional software industry, and that there will be some convergence between the two sides of the industry. But in the next 10 or 15 years there will surely be unanticipated technological discontinuities, comparable with the launch of the IBM System/360 in the 1960s, the personal computer in the late 1970s, and the open source movement in the 1990s.

The first software products were usually supplied in both source code and object code.

History shows us that the preferred software development method of the day has always been the one that seemed to work best within the contemporary technological and economic constraints, particularly the costs of computer ownership, programming personnel, and data communications. The next paradigm shift might well be the currently much-hyped SaaS (software as a service)—software delivered as a service over the Internet rather than as a product installed on a local computer. SaaS seems to offer a technological prospect in which both proprietary and open source software can flourish. But it is at least as likely that some other technological development—perhaps already here and waiting in the wings—will create a software future that is currently unimaginable. That's the fundamental reason historians are so reluctant to attempt to predict the future of software. C

Martin Campbell-Kelly (M.Campbell-Kelly@warwick.ac.uk) is a professor in the Department of Computer Science at the University of Warwick, where he specializes in the history of computing.

Calendar of Events

October 13–15
ASSETS '08: The 10th International ACM SIGACCESS Conference on Computers and Accessibility
Halifax, Nova Scotia,
Contact: Simon Harper,
Phone: 44-0-786-656-8334,
Email: simon.harper@manchester.ac.uk

October 14–17
CBHPC '08: Component-Based High Performance Computing
Karlsruhe, Germany,
Contact: Masha Sosonkina,
Phone: 515-294-6751, Email: masha@scl.ameslab.gov

October 14–17
CBSE '08: 11th International Symposium on Component-Based Software Engineering
Karlsruhe, Germany,
Contact: Ralf H. Reussner,
Phone: 49-721-608-3934, Email: reussner@ipd.uka.de

October 14–17
IIiX08: International Interaction in Context Symposium
London, United Kingdom,
Contact: Mounia Lalmas,
Phone: +44 020 7882 5200,
Contact: mounia@dcs.qmul.ac.uk

October 16–18
SIGITE '08: ACM Special Interest Group for Information Technology Education Conference
Cincinnati, OH,
Sponsored: SIGITE,
Contact: Mark Stockman,
Email: mark.stockman@uc.edu

October 19
PLOP '08: Pattern Languages of Programs
Nashville, TN,
Sponsored: SIGPLAN,
Contact: Ademar Aguiar,
Phone: 351-22-7125947,
Email: ademar.aguiar@fe.up.pt

October 19–22
SIGUCCS Fall '08: ACM SIGUCCS Fall Conference
Portland, OR,
Sponsored: SIGUCCS,
Contact: Terry B Wolff,
Phone: 213-821-2316,
Email: twolff@usc.edu

DOI:10.1145/1400181.1400190

Deborah G. Johnson

Computing Ethics

Computer Experts: Guns-for-Hire or Professionals?

Considering the responsibilities of those who build systems fundamental to significant social functions, institutions, and values.

In the 1980s, when I first began thinking and writing about ethics and computing, there was much speculation about how computing would and should develop as an occupation or a set of occupations. At that time, of course, one had to turn to the histories of other fields to learn about paths to professionalization. With more than 25 years behind us, the picture remains unclear. What is the state of the field of computing now and where should it go?

Professionalization is of interest not for its own sake, but for what it would do to promote a socially responsible deployment of computing expertise. To get quickly to the heart of the matter, we might use a distinction as a foil: although it oversimplifies a complex situation, the distinction between guns-for-hire and professionals frames the issues of professionalization in stark form. A gun-for-hire is someone who puts his or her expertise up for sale to the highest bidder; he or she will do anything anyone wants as long as it is legal. By contrast, and in what is admittedly an idealized paradigm, professionals have standards; they take responsibility, individually and collectively, for setting standards of practice acknowledging that law is limited and will not adequately protect the values that should guide the field. Typically, professions act collectively through an organization that promulgates and enforces a code of ethics and professional conduct, and that articulates the core values of the profession, for example life (in medicine), safety (in engineering), and accuracy (in auditing). Are computer experts guns-for-hire or professionals?

Sociological accounts of professions have suggested that we think of professions as systems or mechanisms for managing expertise. A group convinces society that restrictions should be placed on who engages in a particular occupation. It convinces society there is a body of knowledge that should be mastered before one practices, for example, before one treats the sick or represents another in a court of law or audits a financial statement. The group convinces society that competence can only be determined by those who have already mastered the relevant body of knowledge. Thus, experts, not outsiders, should be in charge of specifying requirements for the field and deciding who has met the requirements.

When a group successfully makes these claims, society grants the group the power of self-regulation. However, this power is granted in exchange for the group's commitment to manage its activities to achieve social good, or at least not in ways that are harmful to society. When doctors professionalized, the intention was to distinguish themselves from "charlatans" and "quacks," those who claimed they could heal patients but who had no scientific understanding of how the human body worked. Once the system of medicine was established, patients could expect that when they went to a "doctor," they would be treated by someone with a certain level of competence. This serves the interests of those who are sick and, in turn, the broader society.

Professionalization often occurs against a backdrop of concerns about the pressures of the marketplace; that is, professionalization is targeted, in part at least, to take certain issues out of the marketplace. When an occupational group has specified standards and articulated its values, then members will (at least, they are expected to) refuse to do anything inconsistent with those standards and values—no matter how much a client or customer is willing to pay. The standards and values become part of the professional culture.

The distinction between guns-for-hire and professionals doesn't map neatly onto computing. Rather than a sharp division, there seems to be a

ILLUSTRATION BY ADAM MCCAULEY

continuum with computer experts falling at different places in terms of their adherence to standards and recognition of professional or social values. Perhaps the most striking characteristic of computing is variability. Clients, customers, and the public encounter computer experts with a wide range of qualifications, experience, and competence. There is enormous variation in the kinds of jobs that computer experts have, in the nature of their expertise, and in the type and amount of education they have. Education is perhaps the easiest way to distinguish one computer professional from another, though it isn't necessarily the most telling. Certification is another means by which skill and competence are established, though certification is used for fairly narrow domains of expertise.

Certification and degrees in higher education are means by which individuals obtain "credentials"; they fulfill one of the functions that are associated with professions. In the paradigm of professions, a single overarching organization such as the American Medical Association is in charge of credentialing. In computing, software engineering is the subfield that has taken the biggest steps toward professionalization. The field has adopted curriculum requirements in higher education, requirements that are targeted to ensure a particular kind of competence. The state of Texas has taken the establishment of requirements one step further by creating a system for software engineering licenses. Still, software engineering aside, variation is the most salient feature of computer experts.

A key feature of any profession—from the perspective of professional ethics—is how it manages the differential in knowledge between its members (experts) and those whom they serve. Computer experts generally work either as employees in organizations (including corporations, government agencies, and nongovernmental organizations) or as consultants hired to perform work for clients. Often their employer or client does not have the expertise to understand or evaluate the work being performed. Moreover, the work of computer experts often has implications for many who are indirectly affected—users of the products produced, recipients of services that are computerized, the public who rely on computerized systems in everything from public transportation to the Internet. The important question here is: how do computer experts understand and manage their relationships with non-experts who rely upon them?

A key feature of any profession—from the perspective of professional ethics—is how it manages the differential in knowledge between its members (experts) and those whom they serve.

These relationships are central to practice in the field and how experts manage these relationships is an important aspect of professional ethics. Consider the following three different ways to conceptualize expert/non-expert relationships. First, computer experts might think of themselves as merely agents. They might presume that their client, employer, or supervisor is in charge and the expert's role is merely to implement the decisions made by those higher up. Essentially the expert sees him- or herself as the means to an employer's or client's ends. This model takes us back to thinking about computer experts as guns-for-hire. There are at least two problems in adopting this model. First, we know that when computer experts implement decisions, they often have a good deal of latitude and their choices can have powerful consequences. "Code is law" as Lawrence Lessig argued in his 1999 book, *Code and Other Laws of Cyberspace*. The work of computer experts may structure an environment, facilitate and constrain behavior, and materialize social values in one form or another. To characterize this work as that of an agent is to deny the real power that computer experts have. Second, if computer experts operate as if they are agents, their clients and employers don't get the full benefit of their expertise. Clients, employers, and the public need computer expertise for higher-order decisions, that is, they need help identifying goals and strategies, not just implementation. When you go to a doctor, you don't tell the doctor what to do, leaving implementation to the doctor's discretion; you want the doctor to determine what needs to be done and to discuss with you the options that are available; you want the doctor to explain the risks and benefits of alternative approaches.

Second, we might think of the proper role for computer experts as paternalistic. Computer experts, it might be argued, are in the best position to understand needs, comprehend potential risks and benefits, as well as foresee the consequences of implementing a system in various ways. Thus, non-experts need computer experts to act on their behalf. According to this model, a client, employer, or the public should transfer all decision-making authority to the computer expert. This provides what is missing in the first model for clients, employers, and the public to get the full benefit of the expert's knowledge. The problem is that computer experts aren't experts with regard to values, interests, and preferences. The model oversteps the expertise of anyone who is competent in computing for no matter how well trained or how much experience a computer expert has, he or she is not an expert on someone else's needs and values.

The third model of the relationship between non-experts and experts combines elements of the first two models and is best suited to the complexities of decision making in computing. It is a model in which experts and those whom they serve share responsibility. Decisions are made through interaction and iteration. Referred to as the fiduciary model ("fiduciary" means trust), this model calls for a relationship of trust between experts and non-experts. The client/employer/public must trust the expert to use his or her knowledge to pursue their interests and values. The professional must trust that the client/employer/public

will give the professional relevant information, will listen to what the professional says, and ultimately share in the decisions that must be made.

The fiduciary model seems the best model to emulate because it recognizes both the multidimensional character of decision making in computing and the differential in knowledge between experts and non-experts. Computer experts aren't just building and manipulating hardware, software, and code, they are building systems that help to achieve important social functions, systems that constitute social arrangements, relationships, institutions, and values.

What is the simple message in all of this? In a word, it is "trust." In two words, it is "public trust." I used the fiduciary model as the model for all expert/non-expert relationships when in reality there are significant differences between a client-professional, employer-employee, and expert-public relationship. However, one of the distinguishing features of professions, as hinted at earlier, is that they are committed to public good even when they are serving clients and employers.

Computer experts have power—in virtue of their expertise, in virtue of their occupational roles, and simply because so many non-experts depend on their work.

Whether the field of computing evolves to come closer to the paradigm of a profession (or not), whether computer experts choose to see themselves as guns-for-hire (or not), computer experts must act so as to be worthy of public trust. There is much to be gained in doing this and much to be lost in failure. If computer experts don't act in a manner that garners and maintains public trust, then the field and its potential to create enormous benefit will not be fully realized. Sure, computing won't go away, but progress will be slowed and diverted as outside regulators jump in and the public has a mixed experience. Computer experts have power—in virtue of their expertise, in virtue of their occupational roles, and simply because so many non-experts depend on their work. While it is rarely acknowledged and even less often stated, this power has been implicitly granted on the basis of a tacit promise that computers and computing would make for a better world. We go forward building computer infrastructures for essential functions and the public—which does not have the expertise to judge—presumes this is for the good. While computing has taken some positive steps to develop public trust, a lot more could be done. C

Deborah G. Johnson (dgj7p@virginia.edu) is the Anne Shirley Carter Olsson Professor of Applied Ethics at the University of Virginia, Charlottesville, VA.

DOI:10.1145/1400181.1400191

Alex E. Bell

From the Front Lines

DOA with SOA

Diagnosing the symptoms of failing to accommodate critical software architecture properties that often result in the demise of projects.

IT LOOKS LIKE today is finally the day that we all knew was coming, it was only a matter of time. An ambulance has just pulled up to haul away Marty the Software Manager after having been pummeled by his boss for failing to deliver on promises of cost savings, improved software reuse, and reduced time to market that had been virtually guaranteed by merely adopting service-oriented architecture (SOA). Everything was supposed to be so different. As opposed to the currently unfolding scenario involving an ambulance, Marty's mental vision of the future had been one of a Brinks truck speeding to the scene to relieve his coffers from buckling under the strain of overflowing cash.

Should anyone really be surprised? After all, Marty is probably still sporting the hook in his mouth from having been reeled in by Victor the Vendor's SOA fishing pole. The hype and propaganda sprinkled onto the bait that Marty swallowed must have caused a mind-numbing sense of euphoria that resulted in business and technical justification for his decisions to be ignored. Marty had been successfully convinced that his project was the only one not cashing in on the booming SOA silver-bullet bonanza.

Despite Marty's headlong charge into the SOA arena, he would have had difficulty describing it the same way three times. In his defense, however, many others have different ideas about what SOA is and is not. Thankfully, I have the benefit of a teenage daughter in the household so there is no shortage of expert opinion on any topic. Out of curiosity, I asked her what she thought was meant by service-oriented architecture. She told me that this was an approach used to construct the shops where she buys her fashionable apparel. There are certainly different opinions about what SOA might be, but this one might be a bit extreme.

Some might say they are doing the SOA tango by merely using XML, WSDL, SOAP, and UDDI technologies. Others may believe they are reverently saluting the SOA flagpole if they are using workflow and their classes are stateless. In actuality, SOA describes an architectural style that is independent of using a particular technology. This architectural style involves advertisement of services in some form of a registry that users can discover, introspect, hook up to, and invoke at their discretion. Sure, SOA is enabled by technologies such as those mentioned here, but others such as CORBA and DCOM have been enabling it for years.

For some software organizations, the primary dilemma is not so much about deciding whether or not adoption of SOA would help realize their development objectives, but instead, determining which technologies and design tactics should be used to do so. Not all SOA users or prospective users

PHOTOGRAPH BY ROY L. ANGELES

even realize they have options when it comes to how they should use SOA to develop their software. Unfortunately, the SOA lemmings who fail to consider these options have the potential for actually causing negative impacts to their software architectures as opposed to capitalizing on some of the benefits that SOA truly does offer.

I wonder which straw finally broke Marty's SOA back. In an effort to be good SOA practitioners, did Marty's software staff generalize some services to such an extent that they were not able to meet even their own product's needs? The idea of developing distributed infix services sounded good, but the performance impacts of remote process invocations to simply add and subtract numbers might have been a bit of an SOA stretch. Even though these infix services were written with the hallmark SOA qualities of being discoverable, stateless, composable, and not dependent on any other services, Marty fell into a trap that many projects seem to be falling into these days. He did not align his architectural and design decisions to the beacons characterizing his project's objectives: Quality Attributes.[a] Instead, he let rampant SOA hype and propaganda blind him. As a result, he couldn't see that the relentless pursuit of flexibility had trumped the importance of performance and usability in his product.

Perhaps Marty's misfortune was an aftereffect of firing all of his systems engineers due to a belief that adoption of SOA transformed the traditional software lifecycle into one where the only relevant activities are ones of development and integration? There is a great deal of money to be saved by assuming that jack-in-the-box architectures will just pop up amidst a haphazard collection of services without having to invest time and effort associated with traditional software engineering approaches.

There might still be yet another possibility at the root of Marty's looming ambulance ride: did he choose the wrong level of abstraction with which to implement SOA? As opposed to service users hooking into them directly at the "stub" level, there are often circumstances where a service layer encapsulating such stubs has the potential of improving important properties that include performance, availability, and survivability. Specifically, performance can be improved by short-circuiting remote method invocations in the event that requested information has been previously fetched. Availability can be improved by the service layer hooking up to alternate service providers in the case of failures or SLA violations. Survivability can be improved by providing service users with some fidelity of response even if connectivity to the actual service provider is temporarily unavailable.

Not all SOA users or prospective users even realize they have options when it comes to how they should use SOA to develop their software.

As just suggested, the benefits of encapsulation should not be ignored when selecting the tactics with which to best implement SOA. In fact, even in the context of my daughter's rather interesting definition of SOA, she recognizes the value of encapsulation, "Dad, I can enjoy my clothes without having to know any of the details of how they were made." Perhaps your SOA usage tactics should heed such wise words and similarly hide applicable implementation details from service users? Why should a user of an extremely simple service be bothered with having to know about UDDI, for example, when a resulting side effect might be to write more code to connect to the service than required to implement the service itself? A preferred implementation from a Quality Attribute realization perspective might be to hide UDDI from clients beneath a thin service layer.

Adoption of SOA is not about throwing a switch where roll-up-the-sleeves engineering can suddenly be avoided or where time and money unexpectedly become available with which to properly identify services using business process analyses. Sure, some development activities might be shortened as the result of reusing certain components or purchasing others, but, how often is one able to actually purchase preexisting services that provide a product's core "business logic"? How many software managers are actually willing to add risk to their development schedules to find out how a particular service imminently required can be generalized to meet the needs of unknown or future users? In order to realize SOA's benefits, engineering and planning are still required as opposed to just hoping that a bunch of random services glued together by a workflow engine will get the job done.

Sadly, Marty did not make it to the hospital. Whichever combination of misguided business decisions and implementation tactics may have finally led to his demise, Marty was DOA—dead on arrival—just like his SOA project. It did not have to be this way. Adoption of SOA did not constitute authorization for Marty to ignore best practices or to show contempt for common sense. Yes, Marty's architecture was flexible, but of what value is flexibility when other critical architectural properties have been ignored or trumped?

How is your SOA health? Do you presume that SOA can only be enabled by Web services? Do you believe that the benefit of properties such as performance and abstraction are only important in "old-fashioned" architectures? Have you aligned your usage of SOA to your Quality Attributes? Pay close attention to how you answer, you will not want to miss the warning signs of a possible DOA with SOA in your future.

a Non-functional properties that drive software architecture (L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice, 2nd edition*, Addison Wesley, 2003).

Alex E. Bell (alex.e.bell@boeing.com) is a software architect with The Boeing Company.

DOI:10.1145/1400181.1400192

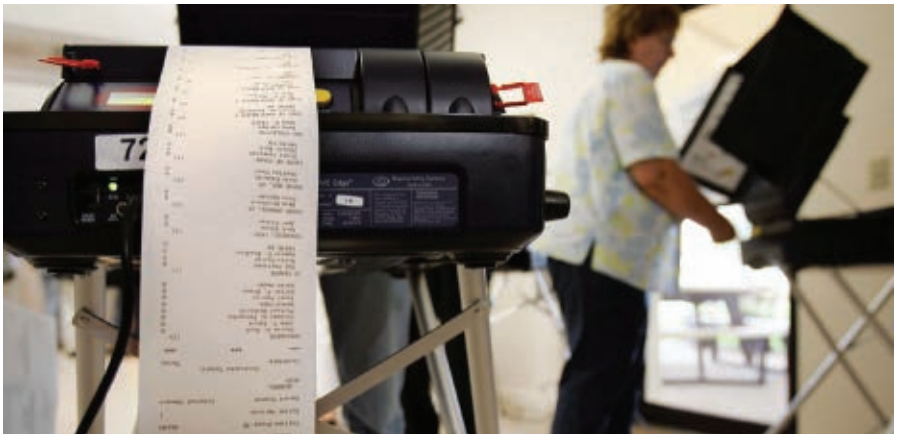
David L. Dill Daniel Castro

Point/Counterpoint

The U.S. Should Ban Paperless Electronic Voting Machines

Debating the public policy issues involved in proposed efforts toward improving voting systems while considering the range of technical and societal challenges.

Point: David L. Dill



WHEN U.S. VOTERS go the polls next month, it will be impossible to determine whether the victorious candidates in many states were elected by a software bug, a virus in the voting system, the voting system programmers, or the voters themselves. Those states have voting machines that rely entirely on electronic ballots (these machines are referred to as direct-recording electronic voting machines, or DREs). There is no way to tell whether the votes recorded by DRE machines match those selected by the voters.

The solution is straightforward: Ban the use of untrustworthy paperless DREs, and demand that readily available systems that are auditable, accurate, reliable, accessible, and cost-effective be used in their place.

Paperless electronic voting is unworkable in principle with current technology. It is based on the mistaken idea that we can build computers that can be trusted to carry out operations whose results cannot be independently verified. But that's a practically impossible problem to solve, even given our best efforts. There is no way to know whether any of the many people involved in the design, implementation, and manufacture of the machines made a mistake or introduced a malicious change. If that were to happen, enough votes could be corrupted to change the outcomes of many elections—invisibly. This fact raises questions about all elections utilizing paperless DREs. Even if the machines are counting votes perfectly, we have no way of confirming that.

Why are paperless DREs more risky than the computers we rely on for banking, medical equipment, and flight software? It's because there is independent verification of the results of operating these other systems. If your plane lands in the wrong city or crashes, or your pacemaker malfunctions, either you or your survivors know about it. If banking software makes an error, you can check your statements to find it. But paperless DREs have no independent verification. If votes are changed in a plausible way, how will anyone ever know?

In reality, current DREs are not even close to "best efforts," as has been shown repeatedly, especially in the last year. Security reviews in California[a] and more recently in Ohio[b] documented breathtaking blunders in the security designs of the most widely used DRE systems in the U.S, which collectively process millions of votes. In each case, a single person with limited access could introduce a virus into the system during one election that could take over all the voting systems in the jurisdiction in the next election.[c]

It is urgently necessary to ban cur-

a M. Bishop, "Overview of Red Team Reports," Top-to-Bottom Review, California Secretary of State's Office; www.sos.ca.gov/elections/voting_systems/ttbr/red_overview.pdf.

b A press release on the EVEREST study of voting equipment security for the Ohio Secretary of State is available at www.sos.state.oh.us/SOS/PressReleases/2007. Detailed reports are available at www.sos.state.oh.us/SOS/elections/voterInformation/equipment/.

c There is a video of team at Princeton showing several ways to hack the Diebold AccuVote-TS DRE at youtube.com/watch?v=aZws98jw67g.

PHOTOGRAPH BY MARIO TAMA

rent paperless DREs. Many states have already done so, but many states have not. All voters who go to the polls in Maryland and Georgia are forced to use paperless DREs, as are many voters in other states. Some other states are using paper ballots now, but could decide to convert to paperless e-voting in the future. Without federal legislation, voters in some states will be stuck with DREs for a long time.

Congress should mandate a specific class of paper trails: every voter should mark and cast a voter-verified paper ballot (VVPB). Each ballot can be counted by hand or scanned in the precincts by a scanner that checks it for overvotes or stray marks (the technical term for this type of system is precinct-count optical scan or PCOS). If there is a problem, the voter has a chance to fix the ballot or fill out a new one. Otherwise, the ballot is counted and deposited in secure ballot box. Or ballots can be counted by hand if desired. These systems can be made accessible to voters with a wide range of disabilities through the use of ballot-marking devices, which allow paper ballots to be read, marked, and verified via an accessible electronic interface.

Most studies have shown that PCOS systems are at least as accurate as any other voting system. They are less costly than touchscreen machines, and, if they fail, marked ballots can be stored in a ballot box and counted later. Most importantly, the hand-marked ballots can be verified and counted without having to trust computerized systems. Optical scan systems are already the dominant technology in the U.S.—they have been used for many years and the technology is steadily improving.

Some have argued that legislation requiring paper ballots would hamper innovation in voting technology. But the main problem in voting technology is not a lack of innovation, but how to prevent and recover from bad innovations.

Why paper and not some other permanent medium such as recordable compact discs? Paper can be read and written by people or machines, and, importantly, by (almost) everyone without machine assistance. Votes on paper cannot be removed or changed without detection. Critical documents on paper have been handled for many centuries and the procedures are easily understood by poll workers and election administrators. For example, it is easily recognized as a problem if a poll worker disappears into a back room for a few hours with a box of ballots.

Of course, simply using paper ballots does not guarantee election integrity. The ballots must be protected, and the processes for storing, transporting, handling, and counting them must transparent. Crucially, paper ballots enable the routine auditing of elections by choosing ballots from randomly selected precincts or machines and manually counting them to see if they match the machine totals.

Some have argued that legislation requiring paper ballots would hamper innovation in voting technology. But the main problem in voting technology is not a lack of innovation, but how to prevent and recover from bad innovations. State and local governments chose to purchase tens of thousands of DREs in spite of the dire warnings of computer technologists and activists—then the true risks of DREs turned out to be even worse than the warnings. The existing requirements and certification process did little to protect the voting system from this and other bad ideas.

A federal VVPB mandate would channel vendor R&D efforts into improving optical scan technology, instead of developing and marketing lucrative but ultimately dubious systems like DREs. If and when a radically new technology is proposed, the law can be changed—after a thorough debate about the true benefits, costs, and risks of that new technology—a debate that would have averted the disastrous experiment with DREs over the last few years. C

David L. Dill (dill@cs.stanford.edu) is a professor of computer science and electrical engineering at Stanford University and has been working actively on policy issues in voting technology since 2003.

Counterpoint: Daniel Castro

ALL VOTERS WANT and deserve secure elections; unfortunately, no voting system currently on the market offers voters verifiable proof that their ballot has been counted. Some activists have been especially concerned about the integrity of votes cast on direct recording electronic (DRE) voting systems, since these devices rely on software that can be difficult to audit. While most individuals agree that voting technology should be improved, many people disagree on the best way to improve it. In particular, a vocal group of activists have popularized the idea of using paper audit trails—basically a paper receipt produced by the DRE—as a countermeasure to fraud and error. Unfortunately, this proposal is an incomplete solution to a much larger problem. Moreover, improving voting systems is not merely a technical challenge but also a public policy challenge.

Computer science is an academic discipline that is based in logic and proof, and we should rely on these valuable methods in our analysis of voting system technology. To understand the scope of the problem one must first understand that the voting process does not end at the ballot box; to have secure elections every step of the voting process must be secure. Specifically, ballots must be cast as intended, collected as cast, and counted as collected. Paper audit trails only provide verification of

the first step—that the ballot was cast as intended. That's good, but not good enough. It does not matter to voters if the voting system correctly cast their ballots, if they cannot verify that election officials correctly counted them.

In fact, narrowly focusing on paper trails ignores the importance of securing all steps in the voting process. Improving election security will involve improving multiple security controls including software testing, physical security, parallel testing, and pre- and post-election auditing. Moreover, paper audit trails are not the only option to verify that ballots are cast as intended. Many types of audit trails will suffice, including those that use audio and video. For example, a research team at Auburn University has developed the Prime III voting system, which produces a private, independent, voter-verified video audit trail of the on-screen interactions between the voter and the voting system.

Additionally, an entirely new class of voting systems has been designed by cryptographers that offer end-to-end (E2E) verifiability of all three steps of the voting process. These E2E systems give voters a paradoxical combination of proof and privacy—proof their ballot is included in the final vote tally and privacy to prevent vote selling and voter coercion. Examples of E2E voting systems include PunchScan (see www.punchscan.org), VoteHere (www.votehere.com/vhti.php), and Scratch & Vote.[1] (In addition, see the news story "Clean Elections" on page 16. —Ed.)

Unfortunately, many of these considerations have been absent from the debate, which has narrowly focused on whether or not to require paper audit trails rather than the larger question of how to improve voting systems. In order to provide a convincing answer to this question security experts and election officials must develop a quantifiable risk analysis framework for evaluating and comparing risk in voting systems. In addition, they must conduct a cost-benefit analysis of the proposed policies for improving voting systems. These two initiatives will provide the evidence and knowledge base on which to base any decisions on proposed design changes to voting systems. Most debate on voting system improvements is premature given that security experts and elections officials have not yet developed a comprehensive risk analysis to compare voting systems. To skip these steps is not only bad science, but bad policy.

The crucial first step to improving voting systems is for the Election Assistance Commission—the federal commission charged with improving elections—to conduct a rigorous and methodical risk assessment of each class of voting system (such as DRE, optical scan, and lever). To date, there has been no comprehensive risk assessment of this type that would allow a meaningful comparison of the relative risks of different voting systems. No voting system is perfect, but as with any system, the key is to find an acceptable level of risk. In addition, a risk assessment would give policymakers a realistic picture of the differences in security between different voting systems. A number of projects have laid the foundation for such a framework, including the NIST's *Developing an Analysis of Threats to Voting Systems*[3] and the Brennan Center report *The Machinery of Democracy: Voting System Security, Accessibility, Usability, and Cost*.[2]

The second step for improving voting systems is to conduct a cost-benefit analysis of proposed voting system improvements. A cost-benefit analysis would reveal the hidden impact of these proposals on security, usability, accessibility, and cost. For example, paper audit trails reduce some risks from software threats but introduce new risks from the chain-of-custody of the paper trails. In addition, paper audit trails decrease accessibility, as blind voters are unable to independently verify the paper audit trail. Paper audit trails are also expensive—in addition to the cost of printers, counties must pay to securely collect, transfer, track, store, and count the paper trails.

Mandating paper audit trails could preclude any chance of implementing these systems in the near future. Rather than turn back the clock on voting technology, we should develop policies that encourage innovation in our voting systems.

While voting system security receives a lot of attention, it is only one of many requirements that voting systems must satisfy. For example, a completely secure voting system is worthless if it is so complex that nobody can use it. Similarly, voters will reject an extremely user-friendly voting system if it is not secure. In voting systems, as with any other type of system, competing values should be balanced against each other. Only with both a risk assessment and a cost-benefit analysis in hand can policymakers implement those design changes that offer the best overall improvements in security, usability, accessibility, and cost.

Finally, security experts and election

PHOTOGRAPH BY JOE RAEDLE

officials must recognize that improving voting systems is not a short-term project. Most of the substantive improvements in voting systems will likely not come from short-term patches, but through long-term technical innovation. In particular, cryptography and E2E voting systems offer potential for revolutionizing voting. Yet mandating paper audit trails could preclude any chance of implementing these systems in the near future. Rather than turn back the clock on voting technology, we should develop policies that encourage innovation in our voting systems. To begin, federal funding needs to be available to sponsor voting system research and development, pilot testing, and risk assessment evaluations.

In addition, voting system guidelines should define functional standards (such as requiring independent, voter-verifiable audit trails), rather than technologically restrictive design standards (such as paper audit trails). Functional standards define the minimum operational requirements to which a system must conform. Since functional standards do not define any specific technology or process, they are flexible enough to allow researchers to develop new approaches to solve existing problems. Just as government should not require that all computers run Windows, neither should it require that all voting machines use paper.

Policymakers cannot disregard voting system technology, and computer scientists cannot ignore the public policy implications of their recommendations. The real challenge is not to design the perfect voting machine, but to design the perfect election. This question is neither exclusively in the domain of computer science nor exclusively in the domain of public policy. Instead, experts from many fields must work together to develop a solution that satisfies all of the characteristics of a good election. While quick-fix ideas may sound good on paper, a deeper analysis shows that many of these proposals suffer serious faults. Moreover, paper trails are not a short-term solution to security, as they only address a small portion of a larger problem. Reinforcing the front door of a house is pointless if the back door is wide open. Instead of trying to apply an unproven and expensive paper patch to existing voting systems, security experts and policymakers should lay out a strategy to advance voting system technology based on a reasoned analysis and solid evidence. C

References

1. Adida, B. and Rivest, R.L. Scratch & vote: Self-contained paper-based cryptographic voting. In Proceedings of the 5th ACM Workshop on Privacy in Electronic Society (WPES'06) (Alexandria, VA, Oct. 30, 2006), ACM, NY, 29–40.
2. Norden, L. et al. *The Machinery of Democracy: Voting System Security, Accessibility, Usability, and Cost.* Technical report, Brennan Center for Justice at NYU School of Law, October 2006.
3. The National Institute of Standards and Technology. *Developing an Analysis of Threats to Voting Systems*; vote.nist.gov/threats/.

Daniel Castro (dcastro@itif.org) is a senior analyst at the Information Technology and Innovation Foundation (www.itif.org) a non-profit, non-partisan public policy organization in Washington, D.C.

Rebuttal: David L. Dill

I HAVE ARGUED that the U.S. voting system is in crisis due to the ill-advised adoption of inherently flawed DRE (direct-recording electronic) voting machines, which are opaque and highly insecure against attacks by both insiders and outsiders. Fortunately, this problem can be easily solved by using voter-marked ballots and precinct-count optical scan technology (PCOS), which is already in widespread use and has proven to be reliable and cost-effective. In particular, I do not argue for adding printers to DREs—PCOS is the best option for voter verification of ballots.

Daniel Castro says a paper trail will not solve all problems in voting. That's true, and no surprise to advocates of paper ballots. Paper ballots are an essential ingredient in a trustworthy election system, which must also include rigorous physical security of ballots, manual counts to audit election results, and other procedural and legal safeguards. But trustworthy elections are impossible with current paperless DREs. The manufacturers and programmers of

the machines, and even external attackers with no special access, can completely control the storage and counting of votes.[a]

Castro says the focus on paper trails ignores other aspects of voting systems. In reality, advocates of PCOS systems have thought through the broader issues, including cost, accuracy, and accessibility. In all these dimensions, PCOS systems are competitive with DRE systems.

Castro argues we cannot act without a "quantifiable risk analysis framework," and a "cost-benefit analysis." Risk analysis and cost-benefit analysis are great ideas; DREs would never have been purchased had these types of analyses been performed and heeded. However, legislation need not wait for further study because DRE systems are clearly much riskier than PCOS systems, a fact that demands prompt action. The most comprehensive study so far (which is the basis for a summary cited by Castro) concludes that a single individual could alter the outcome of a close election on paperless DREs, but that a much larger team of attackers would be required to steal an election using PCOS—assuming appropriate procedures including manual audits.[b] As for cost-benefit analysis, PCOS systems obtain the benefits of DREs and more, at lower cost.[c]

Castro claims there are other ways of solving the problems of electronic voting, including the Prime III system and several end-to-end systems (Punchscan, VoteHere, and Scratch&Vote). Prime III has video and audio (rather than paper trails) that would be very difficult to audit in practice. Punchscan and Scratch&Vote are arguably voter-verified paper ballot systems, albeit cryptographic ones. More importantly, these systems will not be available to replace DREs for years (if ever). VoteHere's system, which also had paper receipts, never caught on, possibly because election officials, technical reviewers, and the public found it difficult to understand.

It is unacceptable in a democracy to have election results that could be undetectably tainted by bugs or malicious software. There is no excuse for further delay in implementing a readily available solution to this serious problem. C

a M. Bishop, "Overview of Red Team Reports," Top-to-Bottom Review, California Secretary of State's Office; www.sos.ca.gov/elections/voting_systems/ttbr/red_overview.pdf.

b Norden, L. et al. *The Machinery of Democracy: Protecting Elections in an Electronic World.* Brennan Center for Justice at NYU School of Law, October 2006 (see p. 50 and p. 83); brennan.3cdn.net/52dbde32526fdc06db_4sm6b3kip.pdf.

c See www.verifiedvotingfoundation.org/article.php?list=type&type+77.

Rebuttal: Daniel Castro

WHILE DAVID DILL makes a passionate case for paper ballots, he omits one stubborn fact: historically, paper ballots are at the root of most voting fraud. This is not surprising since paper ballots can be easily changed, lost, stolen, or invalidated. Yet his solution is to throw more money at precinct-count optical scan (PCOS) systems. While these paper-based voting machines have some initial appeal, they are not a panacea.

First, his claim that PCOS systems are less costly than other forms of voting technology is simply false. This is akin to claiming that apples are more expensive than oranges. The total cost of a voting system for a county depends on many factors: the price and quantity of the voting devices, the number of elections per year, the lifecycle of the equipment, and the cost of recounts, storage, maintenance, and disposal.[2] Moreover, any proposal to change voting technology must also take into account the cost of switching technology, such as retraining election officials.

Second, PCOS systems can be hacked. In fact, the Brennan Center writes in its report on voting systems, "Nothing in our research or analysis has shown that a Trojan horse or other software attack program would be more difficult against PCOS systems than they are against DREs."[1] Manual recounts prevent some attacks, but not all of them. For example, an attacker could disable the over/under-vote alert on the optical scanners in certain counties resulting in many invalid ballots. Since over/under-votes account for up to 4% of total votes, this attack could swing a close election.

Moreover, PCOS systems do not provide voters any proof their ballots were included in the final tally. Neither do PCOS systems offer any kind of guarantee to voters that no illegitimate ballots have been added to the tallies. The only way to achieve that level of confidence is to provide end-to-end (E2E) verifiability, which is why I recommend E2E voting systems as a long-term solution.

As a short-term solution, we should tighten up security requirements to eliminate known vulnerabilities and ensure consistent election procedures. Election officials can use pre- and post-election auditing to make sure the machine does what it is supposed to do, parallel testing to make sure it works correctly during the election, and hash-code testing to make sure the software that is on the machine is the same software that was previously tested and is on file.

States can make their current e-voting systems reasonably secure without a federal requirement for paper audit trails. Switching every county to PCOS or paper ballots would cost over $1.1 billion, and still not solve the security problem. And ultimately, switching to PCOS or paper ballots is a waste of time, money, and effort because it does not move us to where we want to go: end-to-end verifiability. Requiring paper ballots will only move us sideways or even backward—we should move forward. C

References

1. Norden, L. et al. *The Machinery of Democracy: Protecting Elections in an Electronic World.* Brennan Center for Justice at NYU School of Law, October 2006.
2. Norden, L. et al. *The Machinery of Democracy: Voting System Security, Accessibility, Usability, and Cost.* Technical report, Brennan Center for Justice at NYU School of Law, October 2006.

ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students

Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM delivers resources that advance computing as a science and profession.

As a member of ACM, you join nearly 90,000 other computing professionals and students worldwide to define the largest educational, scientific, and professional computing society. Whether you are pursuing scholarly research, building systems and applications, or managing computing projects, ACM offers opportunities to advance your interests.

MEMBER BENEFITS INCLUDE:

- A subscription to the completely redefined ***Communications of the ACM***, ACM's flagship monthly magazine
- The option to subscribe to the full **ACM Digital Library**, with improved search functionalities and **Author Profile Pages** for almost every author in computing
- The **Guide to Computing Literature**, with over one million bibliographic citations
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full and unlimited access to over 3,000 online courses** from SkillSoft
- **Full and unlimited access to 1,100 online books**, featuring 500 from Books24x7®, and 600 from Safari® Books Online, including leading publishers such as O'Reilly (Professional Members only)
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- ***TechNews***, ACM's tri-weekly email digest delivering stories on the latest IT news
- ***CareerNews***, ACM's bi-monthly email digest providing career-related topics
- ***MemberNet***, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and high-quality **Postini spam filtering**
- And much, much more!

ACM's worldwide network ranges from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network, and enjoy the advantages that come from sharing in their collective expertise, all of which serves to keep our members at the forefront of the technology world.

I invite you to share the value of ACM membership with your colleagues and peers who are not yet members, and I hope you will encourage them to join and become a part of our global community.

Thank you for your membership in ACM.

Sincerely,

John R. White
Executive Director and Chief Executive Officer
Association for Computing Machinery

Association for Computing Machinery

Advancing Computing as a Science & Profession

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: ACACM29

You can join ACM in several easy ways:

Online
http://www.acm.org/join

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

Fax
+1-212-944-1318

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

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

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

Please print clearly

Name

Address

City | State/Province | Postal code/Zip

Country | E-mail address

Area code & Daytime phone | Fax | Member number, if applicable

Purposes of ACM

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

I agree with the Purposes of ACM:

Signature

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

choose one membership option:

PROFESSIONAL MEMBERSHIP:

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

STUDENT MEMBERSHIP:

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

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

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

RETURN COMPLETED APPLICATION TO:

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

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

Satisfaction Guaranteed!

payment:

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

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

❑ Professional Member Dues ($99 or $198) $ _____

❑ ACM Digital Library ($99) $ _____

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

Total Amount Due $ _____

Card # Expiration date

Signature

DOI:10.1145/1400181.1400194

Is it getting any easier to understand other people's code?

BY GEORGE V. NEVILLE-NEIL

Code Spelunking Redux

IT HAS BEEN five years since I first wrote about code spelunking[9] and though systems continue to grow in size and scope the tools we use to understand those systems are not growing at the same rate. In fact, I believe we are steadily losing ground. So why should we go over the same ground again? Is this subject important enough to warrant two articles in five years? I believe it is.

The oft-quoted Moore's Law about the increasing power of computers actually works against the code spelunker. The more powerful computers become, the more we demand that they do, which increases the complexity of the software that runs on them. Processor speeds increase and that means more lines of code can now be run in the same amount of time. Available memory gets larger so we can now keep more state or code in memory. Disks get larger and require less power (in the case of flash), and suddenly we're able to carry around what were once considered huge amounts data in our pockets. What was termed the "software crisis" in the 1970s has never really abated, because each time software engineers came up with a new way of working that reduced complexity, the industry moved forward and demanded more.

Complexity increases in many directions, including lines of code, numbers of modules, and numbers of systems and sub-systems. More complex systems require more lines of code to implement. As they grow their systems, software teams often integrate more code from outside resources, which leads to complex interactions between systems that may not have been designed with massive integration in mind.

These numbers should not be surprising to any software engineer, but they are a cause for concern. Although it was unlikely that the numbers would shrink, all but one of them grew by more than 50%, and although the number of lines may have grown linearly, the interactions among the new components that these numbers represent have not grown in a linear fashion. If we assume that all modules in a system can interact freely with all other modules, then we have a system in which the potential number of interactions is expressed as n(n-1)/2, an equation that should be familiar to those who work in networking as it represents a fully connected network. If a system grows from 100 modules to 200 modules, a 100% growth rate, then the number of potential connections grows from 4,950 to 19,900, a 302% growth rate.

One reliable measure of the number of interfaces into a system is the number of system calls provided to user programs by an operating system kernel. Since the publication of my first article on code spelunking the Linux kernel has grown from just shy of 200 system calls to 313, an increase of more than 50% (see Table 1).[7]

Two New Tools

My first article on code spelunking covered several tools, including global,[3] Cscope,[1] gprof,[4] ktrace,[8] and truss.[11] I continue to use these tools on a daily

basis but in the past five years two new tools have come to my attention that, although they may not have been specifically designed with code spelunking in mind, both make significant contributions to the field. The tools are Doxygen[2] and DTrace.[6] Here, I discuss each tool and how it can help us understand large code bases.

Doxygen. Right at the top of the Doxygen Web page[2] we find the following: "Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D." As the blurb says, Doxygen was designed with documenting source code in mind—and it is quite a good system for documenting source code so that the output is usable as man pages and manuals—but it has a few features that make it applicable to code spelunking, too.

What Doxygen does is read in all, or part, of a source tree, looking for documentation tags that it can extract and turn into nicely formatted output suitable for documenting a program. It can produce Unix man pages, LaTeX, HTML, RTF, PostScript, and PDF.

What is most interesting for the code spelunker is Doxygen's ability to extract information from any source code by running pre-processors over the code in question. Doxygen is a static analysis tool in that it analyzes the source code of a program but does not look into the program state while it is running. The great thing about a static analysis tool is that it can be run at any time and does not require that the software be executing. In analyzing something like an operating system, this is extremely helpful.

The features that make Doxygen most relevant to our work are those related to how data is extracted from the source code. When you start out with the intention to document your own code with Doxygen you are already working with the system and very little extra needs to be done. If you're code spelunking an unknown code base then you will need to be more aggres-

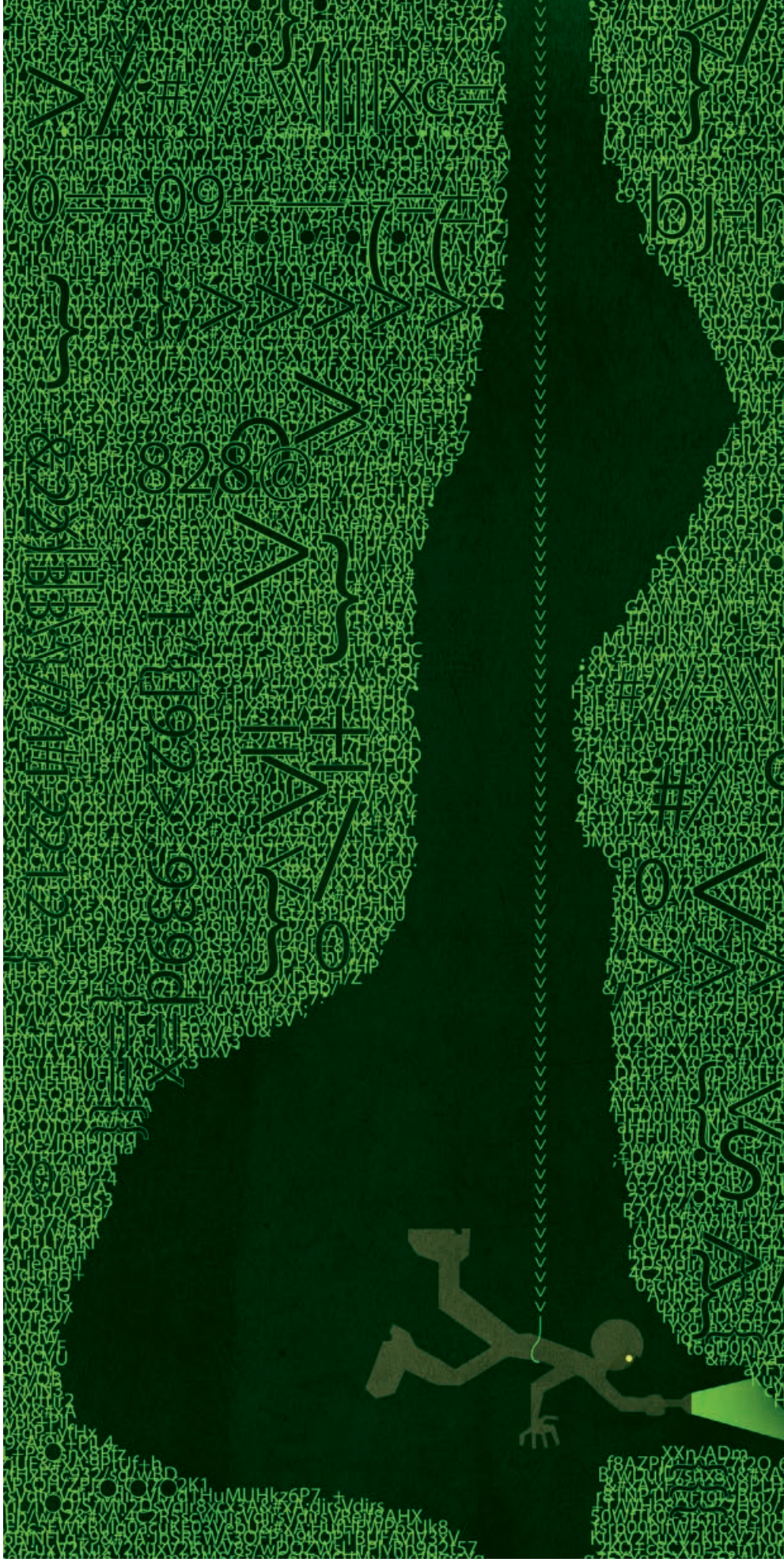


ILLUSTRATION BY JOHN HERSEY

sive and manually turn on certain features in the Doxyfile, which is Doxygen's configuration file. These features are listed in Table 2.

The option "HAVE_DOT" is the most important one because it's what allows Doxygen to generate the most useful output for the code spelunker, including class, collaboration, call, and caller graphs. We'll now take a brief look at two of these types of graphs. The code that we're analyzing in this article is the TCP/IP stack of the FreeBSD Operating System. The BSD TCP/IP stack has been studied in the past[12] and continues to be studied by researchers working on the TCP/IP protocol suite.

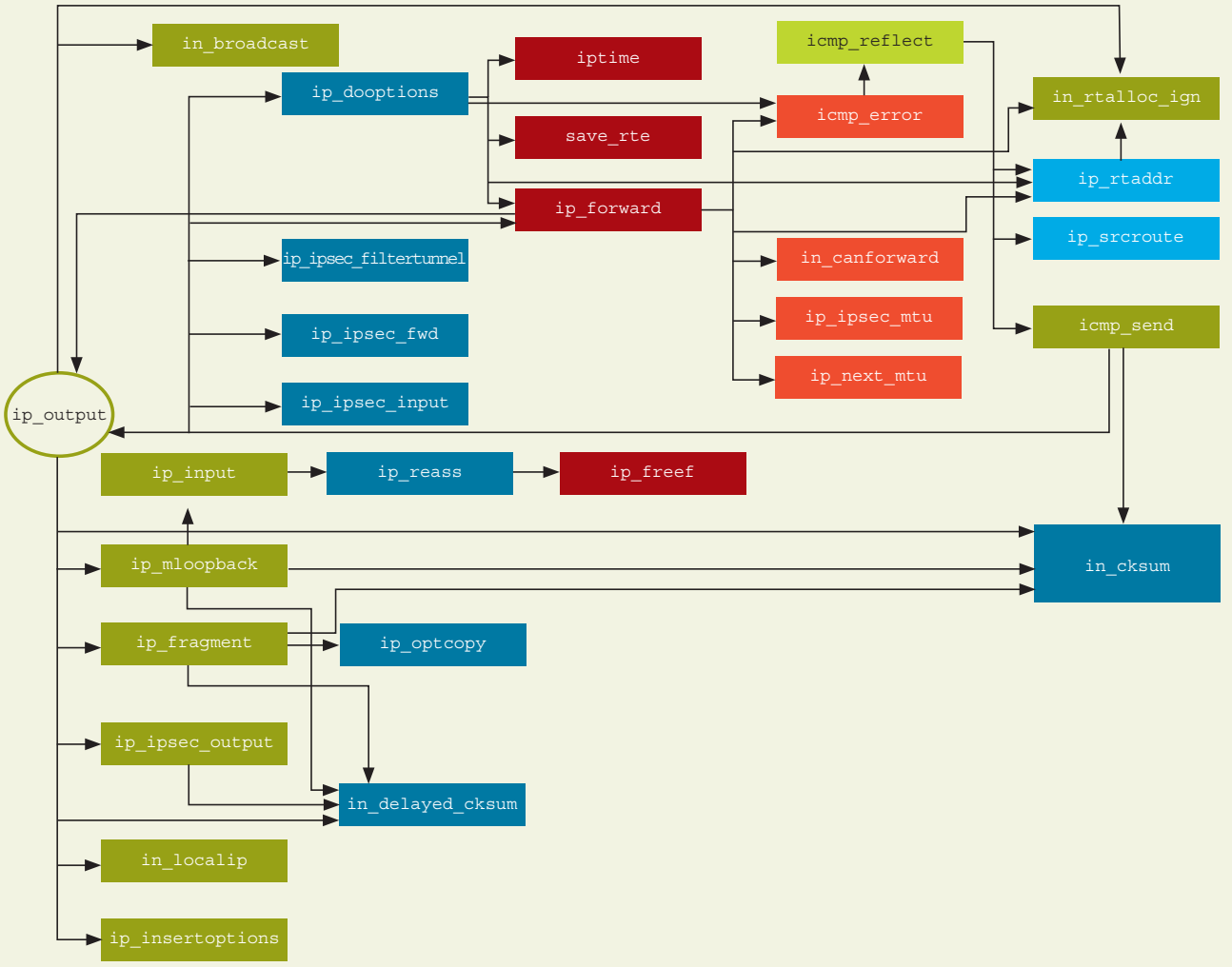
For our examples we will look at a single function, `ip _ output()`, which is called in various parts of the network stack in order to send an IP datagram to the network. The `ip _ output()` function is quite important to the stack because all normal packet transmissions flow through it. If a bug was found in this function, or if the API needed to be changed for some reason, it would be important to trace back all of the current users (callers) of the function. In Figure 1 we see the caller graph produced by Doxygen for `ip _ output()`.

In Figure 1 we see that no fewer than 16 separate routines, in nearly as many modules, depend on the `ip _ output()` function. To effect a fix or update the API all of these routines will need to be investigated. The red-bordered boxes in Figure 1 show nodes that had a greater number of incoming edges than could be shown comfortably in the graph, which is a good indication that the function contained therein is an important component of the system.

The opposite of a caller graph is a call graph. A call graph is more familiar to users of the tools mentioned previously, such as Cscope and global, which allow the user to interactively move through the call graph of a function, jumping into and out of underlying functions while browsing the source code. Doxygen gives us a different way of interacting with the call graph. Figure 2 shows the call graph for the `ip _ output()` function.

The call graph, like the caller graph, gives us a good visual overview of how the function fits into the overall system. Both of these figures function as maps

Figure 1. Caller graph for `ip_output()`.



from which we can derive clues as to how the software is structured. One clue that is relatively easy to see is that there is another hot spot in the packet output code, namely `tcp_output()`, which is called from seven different routines.

The kind of information that Doxygen can show comes at a price. Generating the graphs shown here, which required analyzing 136 files comprising 125,000 lines of code, took 45 minutes on a dual-core 2.5GHz Macbook Pro laptop. Most of the time was taken up by generating the call and caller graphs, which are by far the most useful pieces of information to a code spelunker.[5]

DTrace. One of the most talked about system tools in the last few years is DTrace, a project from Sun Microsystems released under the CDDL that has been ported to the FreeBSD and Mac OS/X operating systems. Regardless of whether the designers of DTrace were specifically targeting code spelunking when they wrote their tool, it is clearly applicable.

DTrace has several components: a command line program, a language, and a set of probes that give information about various events that occur throughout the system. The system was designed such that it could be run against an application for which the user had no source code.

DTrace is the next logical step in the line of program tracing programs that came before it, such as ktrace and truss. What DTrace brings to code spelunking is a much richer set of primitives, both in terms of its set of probes and the D language, which makes it easier for code spelunkers to answer the questions they have. A program like ktrace only shows the system calls that the program executes while it's running, which are all of the application's interactions with the operating system. On a typical OS these number in the low hundreds, and while they can give clues to what a complex piece of software is doing, they are not the whole story. Ktrace cannot trace the operating system itself, which is something that can now be accomplished using DTrace.

When people discuss DTrace they often point out the large number of probes available, which on Mac OS X is more than 23,000. This is somewhat misleading. Not all of the probes are immediately usable, and in reality, having such an embarrassment of riches makes picking the most useful probes for a particular job difficult. A probe is some piece of code in an application, library, or the operating system that can be instrumented to record information on behalf of the user. The probes are broken down into several categories based on what they record. Each probe is delineated by its Provider, Module, Function, and Name. Providers are named after systems such as io, lockstat, proc, profile, syscall, vminfo, and dtrace itself. There are several distinct providers available in Mac OS X, although naively printing them all will show you that several exist on a per-process basis. The per-process probes show information on core data within

Table 1. Comparing the sizes of the systems as discussed in 2003 and today.

Program	Version	Files	Lines	% Chg Lines
Apache Web Server	1.3	471	158,332	
	2.2.8	1108	374,993	136%
Emacs	21	2586	1,317,915	34%
	22	2598	1,771,282	
FreeBSD Kernel	5.1	4758	2,140,517	
	7.0	6723	3,556,087	66%
Linux Kernel	2.4.20-8	12417	5,223,290	
	2.6.25-3	19483	8,098,992	55%
Python	2.2.3	1158	356,314	
	2.5.2	2379	910,573	155%

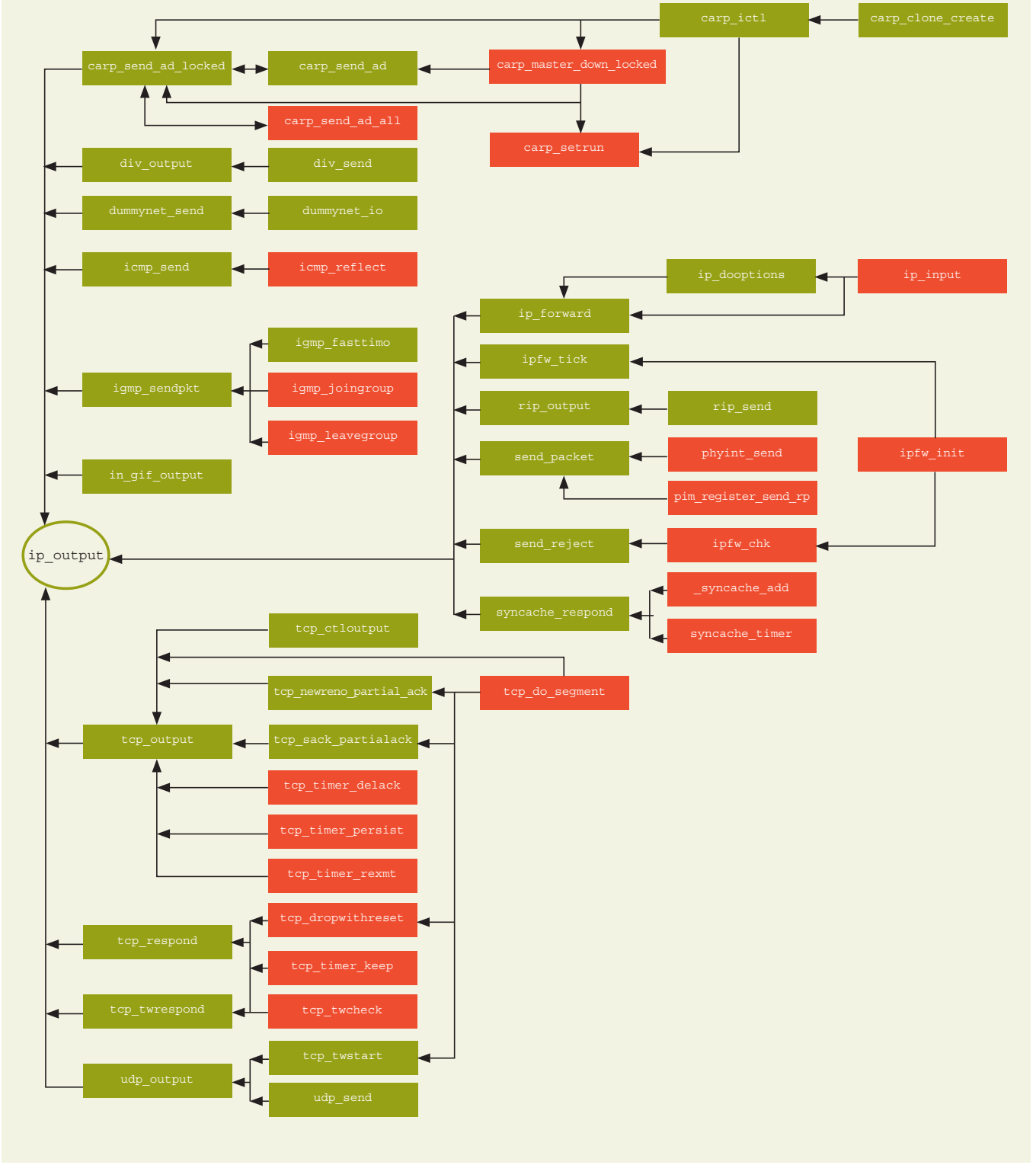
Table 2. Features in the Doxyfile.

Feature	Meaning
`EXTRACT_ALL`	Extract everything you can from the source code.
`SOURCE_BROWSER`	Create a full cross-reference of the source code.
`CLASS_DIAGRAMS`	Create class diagrams and inheretance graphs.
`HAVE_DOT`	Create useful code spelunking graphs.
`CALL_GRAPH`	Makes a call graph following all function calls.
`CALLER_GRAPH`	Outputs a graph of the caller dependencies.

Table 3. Providers available in Mac OS X.

Provider	Purpose
dtrace probes	related to dtrace itself
fbt	entry and exit points for functions
io	I/O probes
lockstat	Probes related to locking
plockstat	pthread lock related probes
proc	Process specific information
profile	Profiling and performance data
syscall	Information on system calls
vminfo	Virtual Memory probes

Figure 2. Call graph for `ip_output()`.



the process as well as on locks. Some of the providers available on Mac OS X are shown in Table 3.

Probes are not only associated with providers but also with modules, which are the modules you want to instrument, as well as with functions, which are also subject to observation. The name of a trace point specifies a single probe. All of these categories are put together in a DTrace script or command line to form a sort of address that specifies what the engineer is trying to observe. The canonical form is provider:module:function:name, with an empty section indicating a wildcard. Although the two manuals from Sun are excellent references on DTrace,[6, 10]

Figure 3. Abbreviated output from the execution of ls under DTrace.

```
> sudo dtrace -s calls.d -c ls
dtrace: script 'calls.d' matched
   5906 probes

[output of ls command removed for brevity]

dtrace: pid 7008 has exited

 strcoll                 148
 strcoll_l               148
 __error                 326
 free                    353
 strcmp                  381
 wcwidth                 614
 __none_mbrtowc          662
 mbrtowc                 662
 putchar                 705
 pthread_getspecific     1424

>
```

a quick example will demonstrate how it can be used for code spelunking.

When presented with a new and unknown system to spelunk one of the first things to find out is which services the program uses. Programs like ktrace and truss can show this type of information but DTrace extends this ability greatly. We will now find out what services the ls program requires to execute as well as which ones are used most often.

```
1: pid$target:::entry
2: {
3:  @[probefunc] = count();
4: }
```

The script here is written in the D language and should be relatively easy to decipher for anyone familiar with C. The script contains a single function, which counts the entry into any call that the ls program makes. Where a C programmer might find a function name and argument list we instead see what is called a predicate. The predicate is a way of selecting the probes that DTrace will record data for. The predicate on line 1 selects the entry into any call for the associated process. When the calls.d script is executed with dtrace in Figure 3, its pid$ variable is replaced with the process ID of the program that is given after the -c command-line argument.

DTrace also allows the tracing of live processes by replacing -c with -p and the program name with a live process ID. Figure 3 gives abbreviated output from the execution of ls under DTrace. Only the last several lines, those with high entry counts, are shown. From this snapshot we can see that ls does a lot of work with the string functions strcoll and strcmp, and if we were trying to optimize the program we might look first at where these functions were called.

With thousands of predefined probe points, and the ability to dynamically create probes for user processes, it's obvious that DTrace is the most powerful code spelunking tool developed in the last decade.

Continuing Challenges

In reviewing the tools mentioned here—as well as those that are not—a few challenges remain apparent. The first challenge is the move by some developers away from a tool-based approach to an all-in-one approach.

A tool-based approach can best be understood by looking at the programs available on any Unix-like system. The use of several programs, mixed and matched, to complete a task has several obvious benefits that are well documented by others. When working with large code bases, the downfalls of an all-in-one approach, such as an IDE, become a bit clearer. A system such as the FreeBSD kernel is already several hundred megabytes of text. Processing that code base with tools like Cscope and global in order to make it more easily navigable generates a further 175MB of data. Although 175MB of data may be small in comparison to the memory of the average desktops or laptops, which routinely come with 2GB to 4GB of RAM, storing all that state in memory while processing leads to lower performance in whatever tool is being used. The pipeline processing of data, which keeps in-memory data small, improves the responsiveness of the tools involved. Loading the FreeBSD kernel into Eclipse took quite a long time and then took up several hundred megabytes of RAM. I have seen similar results with other IDEs on other large code bases.

An even larger challenge looms for those who work on not only large, but heterogeneous, code bases. Most Web sites today are a melange of PHP or Python with C or C++ extensions, using MySQL or PostgreSQL as a database backend, all on top of an OS written in C. It is often the case that tracking down particularly difficult problems requires crossing language barriers several times—from PHP into C++ and then into SQL, then perhaps back to C or C++. Thus far I have seen no evidence of tools that understand how to analyze these cross-language interactions.

The area that deserves the most attention is visualization. Of all the tools reviewed, only Doxygen generates interesting and usable visual output. The other tools have a very narrow, code-based focus in which the user is usually looking at only a small part of the system being investigated.

Working in this way is a bit like trying to understand the United States by staring at a street sign in New York. The ability to look at a high-level representation of the underlying system without the fine details would be perhaps the best tool for the code spelunker. Being able to think of software as a map that can be navigated in different ways, for instance, by class relations and call graphs, would make code spelunkers far more productive.

One last area that has not been covered is the network. Network spelunking, the ability to understand an application based on its network traffic, is still in a very nascent state, with tools like Wireshark being the state of the art. Many applications are already running online and to being able to understand and work with them at the network level is very important. C

References

1. CScope Man Page; http://cscope.sourceforge.net/cscope_man_page.html.
2. Doxygen Web Site; http://www.stack.nl/~dimitri/doxygen/.
3. GNU GLOBAL Source Code Tag System. Tama Communications Corp., Apr. 21, 2008.
4. gprof; http://www.gnu.org/manual/gprof-2.9.1/gprof.html.
5. Graphviz Web Site; http://www.graphviz.org/.
6. How To Use DTrace. Sun Microsystems, 2005. Available on the Web at http://www.sun.com/software/solaris/howtoguides/dtracehowto.jsp.
7. HPC System Call Usage Trends. Terry Jones, Andrew Tauferner, Todd Inglett Linux Clusters Institute 2007.
8. ktrace: standard tool on open source OSes.
9. Neville-Neil, G.V. Code spelunking: Exploring cavernous code basis. *ACM Queue* (Sept. 2003). ACM, NY.
10. Sun Microsystems. Solaris Dynamic Tracing Guide 2005; http://docs.sun.com/app/docs/doc/817-6223
11. Truss is available on Solaris.
12. Wright, G.R. and Stevens, W.R. *TCP/IP Illustrated, Vol. 2: The Implementation.* Addison-Wesley Professional. 1995.

George V. Neville-Neil (gnn@acm.org) is a columnist for *Communications* and *ACM Queue*, as well as a member of the *Queue* Editorial Board. He works on networking and operating system code and teaches courses on various subjects related to programming.

The Best Place to Find the Perfect Job... Is Just a Click Away!

No need to get lost on commercial job boards.
The ACM Career & Job Center is tailored specifically for you.

JOBSEEKERS

- Manage your job search
- Access hundreds of corporate job postings
- Post an anonymous resume
- Advanced Job Alert system

EMPLOYERS

- Quickly post job openings
- Manage your online recruiting efforts
- Advanced resume searching capabilities
- Reach targeted & qualified candidates

NEVER LET A JOB OPPORTUNITY PASS YOU BY!

START YOUR JOB SEARCH TODAY!

http://www.acm.org/careercenter

Association for Computing Machinery

Advancing Computing as a Science & Profession

POWERED BY JOBTARGET

DOI:10.1145/1400181.1400195

How do we apply the concept of resource orientation by designing representations to support interactions?

BY ERIK WILDE AND ROBERT J. GLUSHKO

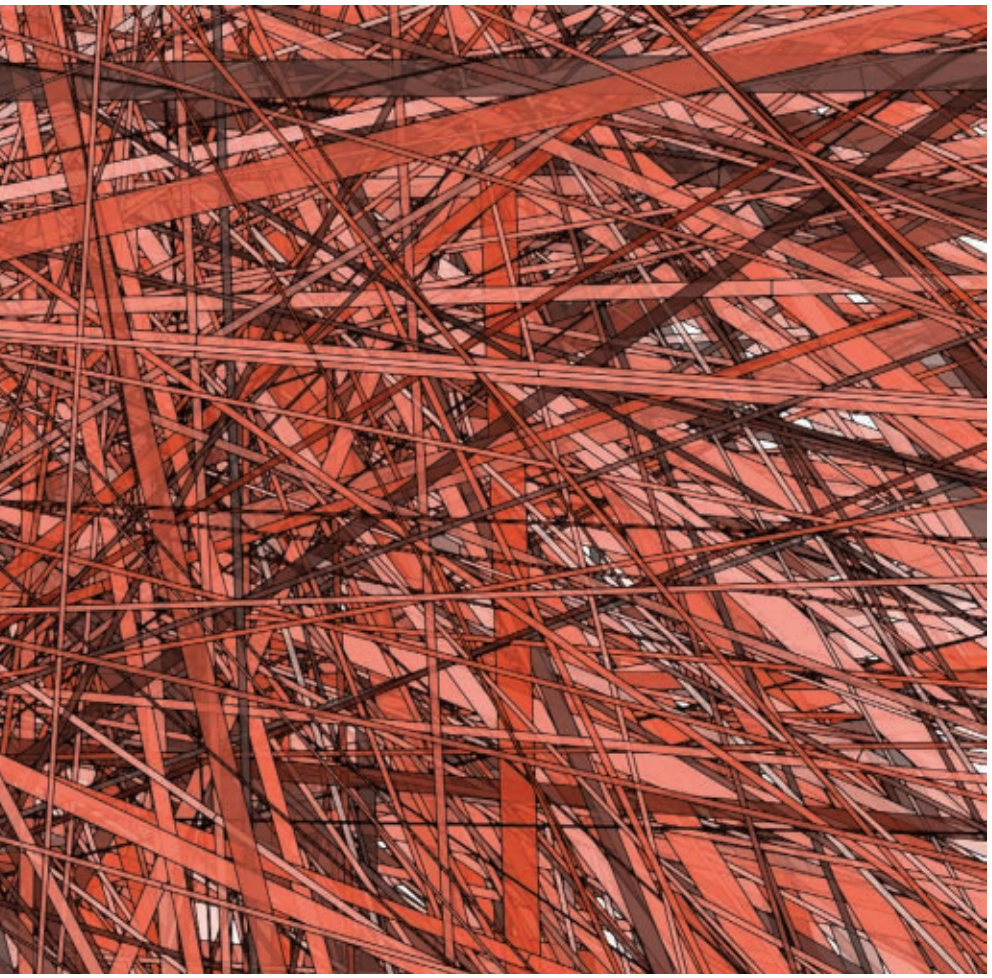
Document Design Matters

THE CLASSICAL APPROACH to the data aspect of system design distinguishes *conceptual*, *logical*, and *physical* models. Models of each type or level are governed by metamodels that specify the kinds of concepts and constraints that can be used by each model; in most cases metamodels are accompanied by languages for

describing models. For example, in database design, conceptual models usually conform to the *Entity-Relationship (ER)* metamodel (or some extension of it), the logical model maps ER models to relational tables and introduces normalization, and the physical model handles implementation issues such as possible denormalizations in the context of a particular database schema language. In this modeling methodology, there is a single hierarchy of models that rests on the assumption that one data model spans all modeling levels and applies to all the applications in some domain.[a]

The "one true model" approach assumes homogeneity, but that does not work very well for the Web. The Web as a constantly growing ecosystem of heterogeneous data and services has challenged a number of practices and theories about the design of IT landscapes. Instead of being governed by "one true model" used by everyone, the underlying assumption of top-down design, Web data and services evolve in an uncoordinated fashion. As a result, a fundamental challenge with Web data and services is matching and

a This simplified view may change if different implementations of the same conceptual model use different lower-level models, perhaps for performance optimization reasons. In this case, the lower-level models are derived from the conceptual model, but they are also based on different assumptions about access patterns and required performance.



mapping local and often partial models that not only are different models of the same application domain, but also differ, implicitly or explicitly, in their associated metamodels.

This challenge is central in the design of interfaces for machine-to-machine communications over the Web because in the general case, no entity can authoritatively and unilaterally define the interfaces and data models or choose the metamodel that describes them.[b] On the Web, when peers use different conceptual models internally, they still can collaborate by using an intermediate model as a representation for their communications. This requires that each peer has a way of mapping its own conceptual model to the intermediary model's conceptual model, which is then serialized according to the representation defined for that model, and then subsequently parsed, instantiated, and mapped to the other peer's conceptual model.

This description may sound as if the intermediary model now is the "one true model" mentioned earlier. The important difference to the assumption that there is an implicit model in all collaborating applications is that the Web approach focuses on exchange-oriented *resource representations* and employs those as the way how data models are represented for communications. However, this article is not about the concept of resource-orientation in the abstract (see Prescod[4] for a good discussion of the fundamental differences of approaches); it is about the question of how to properly apply this concept by designing representations for the specific purpose of supporting interactions.

Somewhat paradoxically, mapping and serializing models is made more difficult by the flexibility and ubiquity of the Extensible Markup Language (XML). Many people assume that merely by providing XML-based representations, universal interoperability and loose coupling can be achieved almost magically[5]—even though different metamodels that simply have some XML representation may make it almost impossible to access the conceptual information in such an XML serialization by using XML tools alone. In many cases, this exaggerated expectation of what XML can do is caused by mixing modeling layers, for example by assuming that *any* data serialized as XML can be conveniently processed using XML tools.

Without further qualification, the argument that "we use XML, which is a widely accepted data format, and thus our interface is easy to use" makes about as much sense as "we use bits, which are a widely accepted data format, and thus our interface is easy to use." On a simplistic technical level, these statements are true, but it is essential to understand the assumptions about application-level models and required tools that are implicitly designed into interfaces, because these assumptions critically shape their usability and accessibility (borrowing these terms from user interface design). *Usability* refers to the ease of use of a given interface and how easy it is for users to accomplish their goals; *accessibility* measures the extent to which it is possible to access all relevant information through a given interface when accessing it using various access methods.

In the context of designing Web service APIs, there is an ongoing debate about function-orientation, often associated with the concept of Remote Procedure Calls (RPC), vs. resource-orientation as the preferred architectural style for designing loosely coupled information systems. Function-orientation overlaps in many ways with the concepts of object-orientation (encapsulation and access through public methods), whereas resource-orientation focuses on the exchange of purposeful and self-contained documents. While the function-oriented style uses encapsulation and remote access to encapsulated data models through application-specific functions, the resource-oriented

b This of course may be different if single entities or coordinated groups like standards organizations define shared data models, but even then this "one true model" only applies to the members of this group, and may not necessarily fit the needs and constraints of other potential users outside of the group.

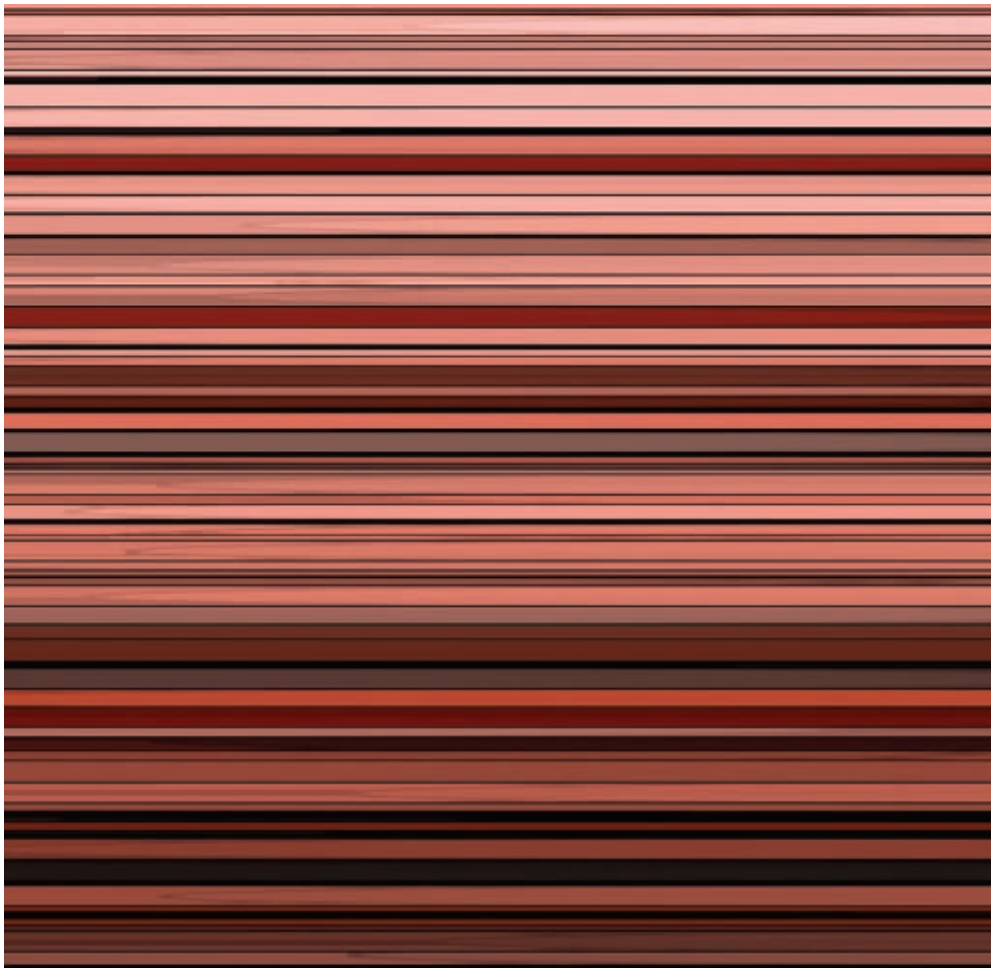
style focuses on the exchange of openly exposed data models, and provides only a small and fixed number of access methods.

Metamodels for RPC and REST

This article focuses on resource-orientation as the underlying architectural style of an IT architecture. For function-orientation, the design of APIs usually follows similar guidelines as API design in system programming;[3] these APIs are function-oriented and provide access to the entities that are locally implemented by the API provider. The "data model" of these APIs usually is rather simple; it is a set of simple types that can be used as types of input and output parameters of a function. The complexity of such an API thus lies in a concrete API design itself; that is, in the data model of the objects providing the functions, not in the metamodel of the API function definitions.

In contrast, with resource orientation, the situation is reversed. In this approach, the main focus is on document exchange, and the structure of these documents is usually much more elaborate than a function signature. On the surface, this difference typically manifests itself by a large number of functions with relatively simple signatures in function-oriented interfaces. On the other hand, resource-oriented interfaces typically define far fewer interactions, but they are coarse-grained ones that have more elaborate signatures in the form of complex resource representations.

Resource-orientation thus often needs more elaborate metamodels to define these resource representations. The selection of the metamodel is an important decision in the context of resource-orientation. Depending on the application, document design can come up with very different requirements for the data model; the spectrum of document models that are appropriate for application scenarios can range from simply structured regular sets of name/value pairs, to highly structured narrative document types mixing natural language with typed data, with a continuum of intermediary and mixed forms in between. The levels of abstraction and granularity, the robustness of the semantic specification, and the extent to which standard conceptual components are reused in this data model are critical document design matters,[2] but they are not the central concerns of this article and will not be discussed further.

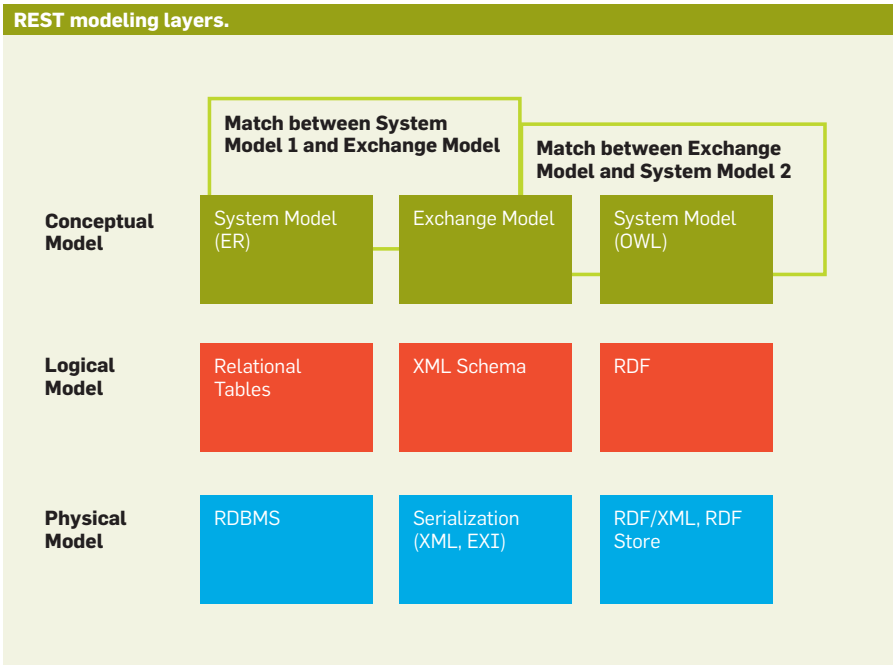


Our first important observation is that the selection of an API metamodel is of greater importance in resource-orientation than it is in function-orientation. Metamodels for resource-orientation typically are more complex, or at least span a larger range of possible requirements for concrete application scenarios, than in the case of function-orientation.

Another important observation is that on the Web, there often is no one true data model for a given application scenario. Instead, peers exchange representations of resources, which is the reason why the architectural style underlying the Web is called *Representational State Transfer (REST).*[1] In REST, when two peers communicate, three models are involved: the internal model of one peer, the representation model that is used for communications, and the internal model of the other peer. It may be the case that both peers use the same internal model, but this is not a requirement of the Web. As long as peers share a common understanding of the intermediary representation model, they can interact.

One of the fundamental differences between function-orientation and REST is that REST's goal is not integration, but collaboration. Function-orientation provides function-based access to a model that is implemented on the provider side; this model often is not exposed, and it is assumed that it is the model shared by all participants. On the other hand, REST provides an explicit representation of a model that is exchanged as a whole; and while participants are free to use that model for their implementation, they are also free to map it to a model they see as a better fit for their processing needs. When a model is

REST modeling layers.



shared, the participants can be more tightly coupled (often necessary for performance reasons) than when it is not, they can for example access the model through fine-grained functions that provide direct access to the model's components; in contrast, the loose coupling embodied in resource exchange enables easier substitution of service providers, which is often desirable for business reasons.

We call the internal models that participants have *system models*, and the representation model used for communication the *exchange model*. One complication in REST is the fact that both models have metamodels, and for successful collaboration it is necessary that the metamodel of the exchange model is made explicit, and is sufficiently described to match it to system models and thus to collaborate.

The Importance of Metamodels

Many real-life problems in Web Service architectures are rooted in implicit assumptions about metamodels. Two examples can be used to illustrate the role of the metamodel for the exchange model, and why it is important in the design of a REST architecture.

The first example starts with a tiny difference between "plain XML" and a data model's metamodel. If a data model is based on DTDs or XML Schema, it may use default values, which are not part of the XML syntax itself, but defined by these schema languages for the metamodel. Applications processing the XML without using the DTD or the XML schema will not recognize these default values, because they are part of a specific schema-based metamodel, and encoded in the schema, not in the instance. Other XML mechanisms—for example XML Namespaces, XML Base, and XInclude—can also affect the interpretation of the model, because a model based on any of these specifications has to be interpreted according to their rules. Thus, even for something as interoperable as XML, it is essential to be explicit about the metamodel that is assumed as governing the data.

The second example is based on another popular metamodel, the Semantic Web. The Semantic Web's metamodel is RDF, and depending on the application, there often is a Schema defining a model for that metamodel (schemas for the Semantic Web are often called ontologies). RDF has several possible serializations; one of them is RDF/XML, where RDF triples are serialized as an XML document. For a peer in such a scenario to work correctly, it must parse the RDF/XML, and then transform the resulting XML document tree into a set of triples, a task that is non-trivial because of the syntax variations defined by RDF/XML. In addition, if there is a schema for the model, it must be taken into account when working with the RDF data, because Semantic Web applications should deal with the deductive closure of the RDF graph they process, not the (syntactic) graph itself. So while XML is being used in this scenario, it only appears on the level of the physical model (as the serialization format), whereas the logical model is RDF.[c]

These two examples illustrate the importance of metamodels for the exchange model. Depending on the choice of the metamodel for the exchange model, peers are required to understand and implement the metamodel and the schema language that encodes the semantics as well as additional constraints for the data model.

There are a number of XML metamodels (after all, XML itself is only a syntax[5]), all of which are tree-based, but they use slight variations of the basic XML structures: The *XML Information Set (Infoset)* is the oldest metamodel of XML; its main contribution is that it includes *XML Namespaces* in the metamodel. The *Post Schema Validation Infoset (PSVI)* is XML Schema's variant of an XML metamodel; its main contribution is its support for typed trees. The *XQuery 1.0 and XPath 2.0 Data Model (XDM)* is becoming the defacto standard for XML metamodels; its main contribution is its support of sequences and thus the ability to represent non-XML values as well as XML trees. In addition to these basic XML metamodels, applications may choose to support additional metamodel features such as XInclude. Extensibility and versioning support often are desirable properties of data models, but for the XML metamodels listed here these are not readily available as metamodel features. Instead, they must be implemented by the data model, and best practices and design patterns can be used for guiding such a design.

APIs use Metamodels

When API designers use popular

c Because RDF/XML is just a syntax for RDF data, it is also possible to serialize RDF as non-XML data; a popular format for this is *Notation 3 (N3)*. In such a scenario, it becomes even more apparent that RDF and XML have very little in common, and that RDF's XML syntax is just an implementation detail of a specific serialization, and does not provide interoperability on a higher level.

metamodels like RDF or UML, it is tempting to assume that users of the API will use the same metamodel as the provider. As a result, these API designers use their system metamodel as the exchange metamodel. We have identified this tendency to use system models as exchange models in Web contexts as *Web blindness*.[5] Instead of creating an architecture based on the smallest possible set of assumptions (which in the case of Web data and services should be XML metamodels), the exchange metamodel then is dictated by the environment of the data or service provider.

We argue it is better practice in API design to think of an API as being designed by its users, rather than its providers, a point that also has been made for function-oriented APIs.[3] Such a consumer-oriented definition of the exchange model focuses on the simplest possible set of assumptions about the consumers, making the API as usable and accessible as possible.

The argument that 'we use XML, which is a widely accepted data format, and thus our interface is easy to use' makes about as much sense as 'we use bits, which are a widely accepted data format, and thus our interface is easy to use.'

Metamodels in Practice

The following example illustrates how these metamodel considerations can inform the design of a resource-oriented API to use the smallest possible set of assumptions.

Let's assume that in a procurement application based on an ER system model, a purchase order is assembled to be sent to a supplier. The exchange model is defined as an XML Schema with additional semantic annotations, so that the application semantics (such as billing address and shipping address) are defined in the model. The order can then be assembled by matching the ER concepts of addresses to the corresponding XML concepts of the order document, that is, by mapping addresses in the system model to addresses in the exchange model. The order is then serialized as an XML document and sent to the supplier through some communications channel.

On the supplier side, when the order is received, the XML document is parsed and the exchange model is reconstructed so that it can be used by the supplier's order management application. Just to illustrate the point that the system models on the two sides can be radically different, let's make the unlikely assumption that the supplier uses Semantic Web technologies. Using a mapping from XML to RDF, the exchange data can be mapped to RDF triples representing the order, because a matching process was used to identify how concepts in the exchange model can be mapped to the receiver's system model.

In such a scenario, the order is represented in three different conceptual models on its way from the sender to the receiver. This process is shown in the accompanying figure and described here in more detail.

Model Matching and Mapping

The existence of system and exchange models makes it necessary to match and map models. *Model matching* is the process of finding identical concepts in two models.[d] In most cases, this process has to be done manually. If both models use the same metamodel, it often is possible to use tools to identify matching concepts, and even to generate code that will map data from one model to the other. For example, a popular functionality in XML editors is schema matching, where candidate conceptual matches between two schemas are made based on element and attribute names, and then refined manually in the editor if these matches are not exact.

It is important to point out that model matching usually is only partial. As there is no "one true model" of the domain, the system models of the participants typically cover larger application areas than the exchange model. It is also possible that the exchange model defines concepts that are not used in one of the system models, in which case this part of the exchange model will be ignored in the respective mapping. In this case, it is important which parts of the exchange model are mandatory and which are optional.

Model mapping is the process that takes data represented in one model, uses the conceptual matching of two models, and maps this data to another model. Mapping can take

d This conveniently ignores the fact that in many cases concepts in different models are never fully identical; they often are partial and/or conditional matches only.

place between two models using the same metamodel (such as mapping one XML document to a semantically equivalent XML document using a different vocabulary), or it can take place between models based on different metamodels (such as mapping an XML document to RDF data). Mapping always takes place between a system model and an exchange model; this decouples all participating system models.

One of the biggest problems in the current stack of XML technologies is there (quite surprisingly) is no appropriate conceptual modeling language for XML. In terms of the stack of conceptual, logical, and physical models, XML Schema is not really a conceptual modeling language; it is primarily focused on markup design. It has some rudimentary conceptual modeling features, but is essentially a language for describing logical models. It is possible to use existing conceptual modeling languages (such as ER or UML) and automatically generate XML logical models (such as XML Schemas) from these, but these generated schemas often are not well-designed from a markup perspective. And critically, these generated schemas will not support any of the XML features that are not covered at all by the conceptual modeling language used as the starting point (most notably, support for narrative structures with mixed content).

This means the ideal process shown in the figure currently lacks support for the conceptual model layer of the exchange model (assuming XML as the exchange model's physical model). In many cases, applications use their own adhoc conceptual models and the logical model (such as an XML Schema) is then augmented with additional information to also serve as the conceptual model, or to refer to it.

We strongly recommend the design of documents be informed by potential consumers and their use cases, not by the producers of documents.

Popular Web Metamodels

XML metamodels (as described earlier) are the most popular metamodels for Web data and services, but they sometimes are thought of as being too expensive in terms of processing, especially if the data model being represented is simple. An alternative physical model for XML metamodels is the *JavaScript Object Notation (JSON)*, which can only encode a subset of possible XML structures, and represents this subset in a way that can be interpreted as a JavaScript object. JSON is useful if the peers (or at least one of them) are JavaScript-based, and if the restrictions of the JSON syntax (no mixed content, restricted names, no difference between elements and attributes, no ordering) are acceptable for the exchange model.

A popular metamodel alternative to XML is RDF, the model of the Semantic Web. It is a metamodel based on triples, which can be used to construct complex graph structures. RDF works well for representing graphs, but has no specific support for the narrative end of the document type spectrum, which is well supported by XML. And while RDF is the universally accepted metamodel in the Semantic Web community, its use as an exchange model still limits the potential set of peers considerably more than an XML metamodel, because RDF tools are not as ubiquitous as XML tools.

Topology Troubles

The idealized matching of models shown in the figure can become complicated when the structures supported by the metamodels do not match very well. A typical example is when the data model is defined in UML and is able to represent arbitrary graphs. XML is tree-structured and thus not a good fit for such a model. In such a case, there are two basic alternatives:

- If the model is defined using some tool, in many cases these tools provide "export" or "generate schema" facilities for exporting the model as an XML Schema. While technically these generated schemas are XML representations of the model, typically they are rather cumbersome to work with on the XML level, because they are generated from a generic mapping of the metamodel to XML, and consequently do not take into account any specifics of the actual model.[e]
- The other possibility is to use the approach we described earlier and as-

e More advanced tools support annotating the model so that the mapping can be guided by annotations. This method yields better results as generated schemas, but the fundamental problem of misaligned metamodels of course cannot be changed.

sume a consumer of the model who does not know the non-XML metamodel or have tools to work with it. What kind of XML model would best serve the needs of that consumer? What are the use cases that should be served well by the XML model? After designing the XML model with the consumer in mind, a mapping between the two models can be defined.

This second alternative is more complex and thus expensive, because it requires a more specific approach, whereas the first one is based on a generic mapping. The second approach is more likely to produce a document design that will be usable and accessible on the XML level. If there is a realistic set of use cases to work with, the second approach can be based on real-world requirements.

The generic mapping approach often only gives lip service to the requirement to provide information in XML: While the generated XML model does represent the original data model, it often does so in ways that are only sufficiently usable and accessible for consumers using the non-XML metamodel. In such a scenario, the implicit assumption is that all consumers will use the non-XML metamodel, which restricts the set of potential users.

Such a restriction may be appropriate if it is a conscious decision. For example, if the metamodel is much more expressive than XML for the specific requirements of an application, it may be almost impossible to come up with a reasonable mapping to an XML model. However, if such a decision is made, there is no specific reason to use XML as the exchange model anymore, because it is treated as an opaque representation of a non-XML conceptual model, and only is used for the physical layer of the exchange model.

Physical XML

Using the figure as the explanation for the potential problems discussed in the previous section, it is easy to explain what happens. If the designer of an exchange format uses a non-XML conceptual metamodel because it seems to be a better fit for the data model, XML is only used as the physical layer for the exchange model. The logical layer in this case defines the mapping between the non-XML conceptual model, and any reconstruction of the exchange model data requires the consumer to be fully aware of this mapping. In such a case, it is good practice to make users of the API aware of the fact that it is using a non-XML metamodel. Otherwise they might be tempted to base their implementation on a too small set of examples, creating implementations that are brittle and will fail at some point in time.

Real-world examples for this case are interfaces using RDF/XML. Naive implementations might assume that the data is encoded in a specific tree structure that can be accessed using XML tools such as XPath or XQuery. They might base their assumption on a set of sample documents, all of which might have been produced by the same RDF software and thus represent the same RDF/XML serialization strategy. However, such an implementation will most likely fail if it encounters documents where other serializations of the same RDF data are used. Thus, the only robust way in such a scenario is to use a full RDF/XML parser, and to first reconstruct the RDF data before doing anything with it.

Document Design Matters

This article has extended API design matters into the world of resource-orientation and REST as its currently most popular exponent. While function-oriented API design matters are mostly about the design of a concrete API, document design matters in the world of resource-orientation have a quite different emphasis. This emphasis is mostly on the models governing the design of an actual API; that is, the metamodels for these APIs. The design of REST APIs has more facets than only the selection of a particular metamodel for the representation of resources, but this is the most fundamental choice, and it directly affects the usability and accessibility of such an API.

We strongly recommend that the design of documents be informed by potential consumers and their use cases (such as, the environments in which these consumers are expected to develop and execute code, and the tasks they are expected to do), not by the producers of documents. By using the complete spectrum of possible consumers as input to the document design process for exchange models, it is possible to escape the implicit assumptions about supported metamodels and available tools that sometimes make data and services on the Web more difficult to work with than necessary.

Usability and accessibility of APIs, in particular of Web service APIs, should become central goals of API design, and a focus on well-designed document models based on widely available metamodels is a good starting point for these goals. While XML provides a proven foundation for defining exchange models, the lack of a conceptual model for XML makes it somewhat harder than it should be to define such a model. Because of this situation, using some non-XML conceptual model is a solution that is frequently chosen by practitioners, often resulting in exchange models embodying implicit assumptions. We hope this critical gap for XML-based resource orientation will soon be filled by some yet-to-be-invented language, capable of representing conceptual models for XML. Such a language would make it possible to better describe exchange models for resource-oriented APIs by supporting an easy way of generating schemas (logical models) for defining the representation (the markup) of exchange models. C

References

1. Fielding, R.T., Taylor, R.N. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology 2*, 2 (May 2002), 115–150.
2. Glushko, R.J., McGrath, T. *Document Engineering*. MIT Press, Cambridge, MA Aug. 2005.
3. Henning, M. API design matters. *ACM Queue 5*, 4 (May 2007), 24–36.
4. Prescod, P. Roots of the REST/SOAP debate. In *Proceedings of Extreme Markup Languages Conference* (Aug. 2002).
5. Wilde, E., Glushko, R.J. XML fever. *Commun. ACM 51*, 7 (July 2008), 26–31.

Erik Wilde (dret@berkeley.edu) is a visiting assistant professor in the School of Information at the University of California at Berkeley, where he is also technical director of the Information and Service Design program.

Robert J. Glushko (Glushko@ischool.berkeley.edu) has 20 years of experience with SGML and XML and is an adjunct professor at the University of California at Berkeley's School of Information. His teaching and research interests in business architecture and information systems and service design are complementary to and "higher in the stack" than Erik Wilde's.

DOI:10.1145/1400181.1400197

As the line between GPUs and CPUs begins to blur, it's important to understand what makes GPUs tick.

BY KAYVON FATAHALIAN AND MIKE HOUSTON

A Closer Look at GPUs

A GAMER WANDERS through a virtual world rendered in near-cinematic detail. Seconds later, the screen fills with a 3D explosion, the result of unseen enemies hiding in physically accurate shadows. Disappointed, the user exits the game and returns to a computer desktop that exhibits the stylish 3D look-and-feel of a modern window manager. Both of these visual experiences require hundreds of gigaflops of computing performance, a demand met by the GPU (graphics processing unit) present in every consumer PC.

The modern GPU is a versatile processor that constitutes an extreme but compelling point in the growing space of multicore parallel computing architectures. These platforms, which include GPUs, the STI Cell Broadband Engine, the Sun UltraSPARC T2, and increasingly multicore x86 systems from Intel and AMD, differentiate themselves from traditional CPU designs by prioritizing high-throughput processing of many parallel operations over the low-latency execution of a single task.

GPUs assemble a large collection of fixed-function and software-programmable processing resources. Impressive statistics, such as ALU (arithmetic logic unit) counts and peak floating-point rates often emerge during discussions of GPU design. Despite the inherently parallel nature of graphics, however, efficiently mapping common rendering algorithms onto GPU resources is extremely challenging.

The key to high performance lies in strategies that hardware components and their corresponding software interfaces use to keep GPU processing resources busy. GPU designs go to great lengths to obtain high efficiency and conveniently reduce the difficulty programmers face when programming graphics applications. As a result, GPUs deliver high performance and expose an expressive but simple programming interface. This interface remains largely devoid of explicit parallelism or asynchronous execution and has proven to be portable across vendor implementations and generations of GPU designs.

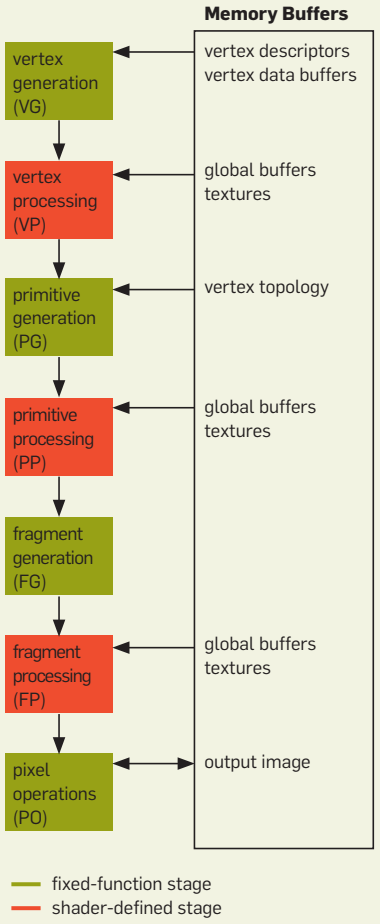
At a time when the shift toward throughput-oriented CPU platforms is prompting alarm about the complexity of parallel programming, understanding key ideas behind the success of GPU computing is valuable not only for developers targeting software for GPU execution, but also for informing the design of new architectures and programming systems for other domains. In this article, we dive under the hood of a modern GPU to look at why interactive rendering is challenging and to explore the solutions GPU architects have devised to meet these challenges.

The Graphics Pipeline

A graphics system generates images that represent views of a virtual scene. This scene is defined by the geometry,

ILLUSTRATION BY DREW FLAHERTY

Figure 1: A simplified graphics pipeline.



orientation, and material properties of object surfaces and the position and characteristics of light sources. A scene view is described by the location of a virtual camera. Graphics systems seek to find the appropriate balance between conflicting goals of enabling maximum performance and maintaining an expressive but simple interface for describing graphics computations.

Real-time graphics APIs such as Direct3D and OpenGL strike this balance by representing the rendering computation as a *graphics processing pipeline* that performs operations on four fundamental entities: vertices, primitives, fragments, and pixels. Figure 1 provides a block diagram of a simplified seven-stage graphics pipeline. Data flows between stages in streams of entities. This pipeline contains fixed-function stages (green) implementing API-specified operations and three programmable stages (red) whose behavior is defined by application code. Figure 2 illustrates the operation of key pipeline stages.

VG (vertex generation). Real-time graphics APIs represent surfaces as collections of simple geometric primitives (points, lines, or triangles). Each primitive is defined by a set of vertices. To initiate rendering, the application provides the pipeline's VG stage with a list of vertex descriptors. From this list, VG prefetches vertex data from memory and constructs a stream of vertex data records for subsequent processing. In practice, each record contains the 3D (x,y,z) scene position of the vertex plus additional application-defined parameters such as surface color and normal vector orientation.

VP (vertex processing). The behavior of VP is application programmable. VP operates on each vertex independently and produces exactly one output vertex record from each input record. One of the most important operations of VP execution is computing the 2D output image (screen) projection of the 3D vertex position.

PG (primitive generation). PG uses vertex topology data provided by the application to group vertices from VP into an ordered stream of primitives (each primitive record is the concatenation of several VP output vertex records). Vertex topology also defines the order of primitives in the output stream.

PP (primitive processing). PP operates independently on each input primitive to produce zero or more output primitives. Thus, the output of PP is a new (potentially longer or shorter) ordered stream of primitives. Like VP, PP operation is application programmable.

FG (fragment generation). FG samples each primitive densely in screen space (this process is called *rasterization*). Each sample is manifest as a fragment record in the FG output stream. Fragment records contain the output image position of the surface sample, its distance from the virtual camera, as well as values computed via interpolation of the source primitive's vertex parameters.

FP (fragment processing). FP simulates the interaction of light with scene surfaces to determine surface color and opacity at each fragment's sample point. To give surfaces realistic appearances, FP computations make heavy use of filtered lookups into large, parameterized 1D, 2D, or 3D arrays called *textures*. FP is an application-programmable stage.

PO (pixel operations). PO uses each fragment's screen position to calculate and apply the fragment's contribution to output image pixel values. PO accounts for a sample's distance from the virtual camera and discards fragments that are blocked from view by surfaces closer to the camera. When fragments from multiple primitives contribute to the value of a single pixel, as is often the case when semi-transparent surfaces overlap, many rendering techniques rely on PO to perform pixel updates in the order defined by the primitives' positions in the PP output stream. All graphics APIs guarantee this behavior, and PO is the only stage where the order of entity processing is specified by the pipeline's definition.

Shader Programming

The behavior of application-programmable pipeline stages (VP, PP, FP) is defined by *shader functions* (or shaders). Graphics programmers express vertex, primitive, and fragment shader functions in high-level *shading languages* such as NVIDIA's Cg, OpenGL's GLSL, or Microsoft's HLSL. Shader source is compiled into bytecode offline, then transformed into a GPU-specific binary by the graphics driver at runtime.

Shading languages support complex data types and a rich set of control-flow constructs, but they do *not* contain primitives related to explicit parallel execution. Thus, a shader definition is a C-like function that serially computes output-entity data records from a single input entity. Each function invocation is abstracted as an independent sequence of control that executes in complete isolation from the processing of other stream entities.

As a convenience, in addition to data records from stage input and output streams, shader functions may access (but not modify) large, globally shared data buffers. Prior to pipeline execution, these buffers are initialized to contain shader-specific parameters and textures by the application.

Characteristics and Challenges

Graphics pipeline execution is characterized by the following key properties.

Opportunities for parallel processing. Graphics presents opportunities for both task- (across pipeline stages) and

data- (stages operate independently on stream entities) parallelism, making parallel processing a viable strategy for increasing throughput. Despite abundant potential parallelism, however, the unpredictable cost of shader execution and constraints on the order of PO stage processing introduce dynamic, fine-grained dependencies that complicate parallel implementation throughout the pipeline. Although output image contributions from most fragments can be applied in parallel, those that contribute to the same pixel cannot.

Extreme variations in pipeline load. Although the number of stages and data flows of the graphics pipeline is fixed, the computational and bandwidth requirements of all stages vary significantly depending on the behavior of shader functions and properties of scene. For example, primitives that cover large regions of the screen generate many more fragments than vertices. In contrast, many small primitives result in high vertex-processing demands. Applications frequently reconfigure the pipeline to use different shader functions that vary from tens of instructions to a few hundred. For these reasons, over the duration of processing for a single frame, different stages will dominate overall execution, often resulting in bandwidth and compute-intensive phases of execution. Dynamic load balancing is required to maintain an efficient mapping of the graphics pipeline to a GPU's resources in the face of this variability and GPUs employ sophisticated heuristics for reallocating execution and on-chip storage resources amongst pipeline stages depending on load.

Fixed-function stages encapsulate difficult-to-parallelize work. Programmable stages are trivially parallelizable by executing shader function logic simultaneously on multiple stream entities. In contrast, the pipeline's nonprogrammable stages involve multiple entity interactions (such as ordering dependencies in PO or vertex grouping in PG) and stateful processing. Isolating this non-data-parallel work into fixed stages allows the GPU's programmable processing components to be highly specialized for data-parallel execution and keeps the shader programming model simple. In addition, the separation enables difficult aspects of the graphics computation to be encapsulated in optimized, fixed-function hardware components.

Mixture of predictable and unpredictable data access. The graphics pipeline rigidly defines inter-stage data flows using streams of entities. This predictability presents opportunities for aggregate prefetching of stream data records and highly specialized hardware management of on-chip storage resources. In contrast, buffer and texture accesses performed by shaders are fine-grained memory operations on dynamically computed addresses, making prefetch difficult. As both forms of data access are critical to maintaining high throughput, shader programming models explicitly differentiate stream from buffer/texture memory accesses, permitting specialized hardware solutions for both types of accesses.

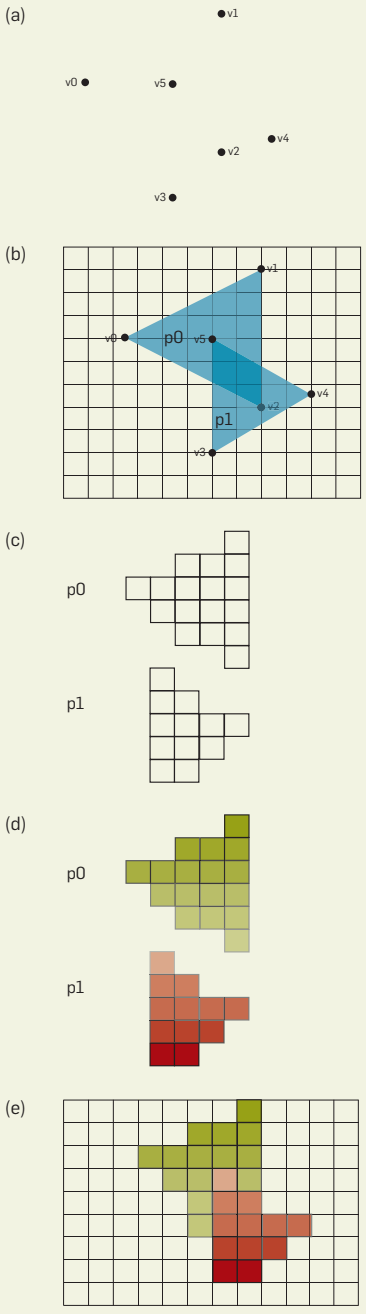
Opportunities for instruction stream sharing. While the shader programming model permits each shader invocation to follow a unique stream of control, in practice, shader execution on nearby stream elements often results in the same dynamic control-flow decisions. As a result, multiple shader invocations can likely share an instruction stream. Although GPUs must accommodate situations where this is not the case, the use of SIMD-style execution to exploit shared control-flow across multiple shader invocations is a key optimization in the design of GPU processing cores and is accounted for in algorithms for pipeline scheduling.

Programmable Processing Resources

A large fraction of a GPU's resources exist within programmable processing cores responsible for executing shader functions. While substantial implementation differences exist across vendors and product lines, all modern GPUs maintain high efficiency through the use of multicore designs that employ both hardware multithreading and SIMD (single instruction, multiple data) processing. As shown in the table here, these throughput-computing techniques are not unique to GPUs (top two rows). In comparison with CPUs, however, GPU designs push these ideas to extreme scales.

Multicore + SIMD Processing = Lots of ALUs. A logical thread of control is realized by a stream of processor in-

Figure 2: Graphics pipeline operations.



(a) Six vertices from the VG output stream define the scene position and orientation of two triangles. (b) Following VP and PG, the vertices have been transformed into their screen-space positions and grouped into two triangle primitives, *p0* and *p1*. (c) FG samples the two primitives, producing a set of fragments corresponding to *p0* and *p1*. (d) FP computes the appearance of the surface at each sample location. (e) PO updates the output image with contributions from the fragments, accounting for surface visibility. In this example, *p1* is nearer to the camera than *p0*. As a result *p0* is occluded by *p1*.

structions that execute within a processor-managed environment, called an execution (or thread) context. This context consists of state such as a program counter, a stack pointer, general-purpose registers, and virtual memory mappings. A single core processor managing a single execution context can run one thread of control at a time. A multicore processor replicates processing resources (ALUs, control logic, and execution contexts) and organizes them into independent cores. When an application features multiple threads of control, multicore architectures provide increased throughput by executing these instruction streams on each core in parallel. For example, an Intel Core 2 Quad contains four cores and can execute four instruction streams simultaneously. As significant parallelism exists across shader invocations in a graphics pipeline, GPU designs easily push core counts higher.

Even higher performance is possible by populating each core with multiple floating-point ALUs. This is done efficiently through SIMD (single instruction, multiple data) processing, where several ALUs perform the same operation on a different piece of data. SIMD processing amortizes the complexity of decoding an instruction stream and the cost of ALU control structures across multiple ALUs, resulting in both power- and area-efficient chip execution.

The most common implementation of SIMD processing is via explicit short-vector instructions, similar to those provided by the x86 SSE or PowerPC AltiVec ISA extensions. These extensions provide instructions that control the operation of four ALUs (SIMD width of 4). Alternatively, most GPUs realize the benefits of SIMD execution by *implicitly* sharing an instruction stream across threads with identical PCs. In this implementation, the SIMD width of the machine is not explicitly made visible to the programmer. CPU designers have chosen a SIMD width of four as a balance between providing increased throughput and retaining high single-threaded performance. Characteristics of the shading workload make it beneficial for GPUs to employ significantly wider SIMD processing (widths ranging from 32 to 64) and to support a rich set of operations. It is common for GPUs to support SIMD implementations of reciprocal square root, trigonometric functions, and memory gather/scatter operations.

The efficiency of wide SIMD processing allows GPUs to pack many cores densely with ALUs. For example, the NVIDIA GeForce GTX 280 GPU contains 480 ALUs operating at 1.3GHz. These ALUs are organized into 30 processing cores and yield a peak rate of 933GFLOPS. In comparison, a high-end 3GHz Intel Core 2 Quad CPU contains four cores, each with eight SIMD floating-point ALUs (two 4-width vector instructions per clock) and is capable of, at most, 96GFLOPS of peak performance.

Recall that a shader function defines processing on a single pipeline entity. GPUs execute multiple invocations of the same shader function in parallel to take advantage of SIMD processing. Dynamic per-entity control flow is implemented by executing all control paths taken by the shader invocations in the group. SIMD operations that do not apply to all invocations, such as those within shader code conditional or loop blocks, are partially nullified using write-masks. In this implementation, when shader control flow diverges, fewer SIMD ALUs do useful work. Thus, on a chip with width-S SIMD processing, worst-case behavior yields performance equaling *1/S* the chip's peak rate. Fortunately, shader workloads exhibit sufficient levels of instruction stream sharing to justify wide SIMD implementations. Additionally, GPU ISAs contain special instructions that make it possible for shader compilers to transform per-entity control flow into efficient sequences of explicit or implicit SIMD operations.

Hardware Multithreading = High ALU Utilization. Thread stalls pose an additional challenge to high-performance shader execution. Threads stall (or block) when the processor cannot dispatch the next instruction in an instruction stream due to a dependency on an outstanding instruction. High-latency off-chip memory accesses, most notably those generated by texture access operations, cause thread stalls lasting hundreds of cycles (recall that while shader input and output records lend themselves to streaming prefetch, texture accesses do not).

Allowing ALUs to remain idle during the period while a thread is stalled is inefficient. Instead, GPUs maintain more execution contexts on chip than they can simultaneously execute, and they perform instructions from runnable threads when others are stalled. Hardware scheduling logic determines which context(s) to execute in each processor cycle. This technique of overprovisioning cores with thread contexts to hide the latency of thread stalls is called *hardware multithreading*. GPUs use multithreading as the primary mechanism to hide both memory access and instruction pipeline latencies.

The amount of stall latency a GPU can tolerate via multithreading is dependent on the ratio of hardware thread contexts to the number of threads that are simultaneously executed in a clock (we refer to this ratio as T). Support for

Table 1. Tale of the tape: Throughput architectures.

Type	Processor	Cores/Chip	ALUs/Core[3]	SIMD width	Max T[4]
GPUs	AMD Radeon HD 4870	10	80	64	25
	NVIDIA GeForce GTX 280	30	8	32	128
CPUs	Intel Core 2 Quad[1]	4	8	4	1
	STI Cell BE[2]	8	4	4	1
	Sun UltraSPARC T2	8	1	1	4

[1] SSE processing only, does not account for traditional FPU
[2] Stream processing (SPE) cores only, does not account for PPU cores.
[3] 32-bit floating point operations
[4] Max T is defined as the maximum ratio of hardware-managed thread execution contexts to simultaneously executable threads (not an absolute count of hardware-managed execution contexts). This ratio is a measure of a processor's ability to automatically hide thread stalls using hardware multithreading.

more thread contexts allows the GPU to hide longer or more frequent stalls. All modern GPUs maintain large numbers of execution contexts on chip to provide maximal memory latency-hiding ability (T reaches 128 in modern GPUs—see the table). This represents a significant departure from CPU designs, which attempt to avoid or minimize stalls primarily using large, low-latency data caches and complicated out-of-order execution logic. Current Intel Core 2 and AMD Phenom processors maintain one thread per core, and even high-end models of Sun's multithreaded UltraSPARC T2 processor manage only four times the number of threads they can simultaneously execute.

Note that in the absence of stalls, the throughput of single- and multithreaded processors is equivalent. Multithreading does not increase the number of processing resources on a chip. Rather, it is a strategy that interleaves execution of multiple threads in order to use existing resources more efficiently (improve throughput). On average, a multithreaded core operating at its peak rate runs each thread *1/T* of the time.

To achieve large-scale multithreading, execution contexts must be compact. The number of thread contexts supported by a GPU core is limited by the size of on-chip execution context storage. GPUs require compiled shader binaries to statically declare input and output entity sizes, as well as bounds on temporary storage and scratch registers required for their execution. At runtime, GPUs use these bounds to dynamically partition on-chip storage (including data registers) to support the maximum possible number of threads. As a result, the latency hiding ability of a GPU is shader dependent. GPUs can manage many thread contexts (and provide maximal latency-hiding ability) when shaders use fewer resources. When shaders require large amounts of storage, the number of execution contexts (and latency-hiding ability) provided by a GPU drops.

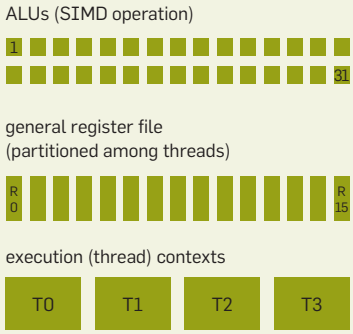
Fixed-Function Processing Resources

A GPU's programmable cores interoperate with a collection of specialized fixed-function processing units that provide high-performance, power-efficient implementations of nonshader stages.

Running a Fragment Shader on a GPU Core

Shader compilation to SIMD (single instruction, multiple data) instruction sequences coupled with dynamic hardware thread scheduling lead to efficient execution of a fragment shader on the simplified single-core GPU shown in Figure A.

Figure A: Example GPU core



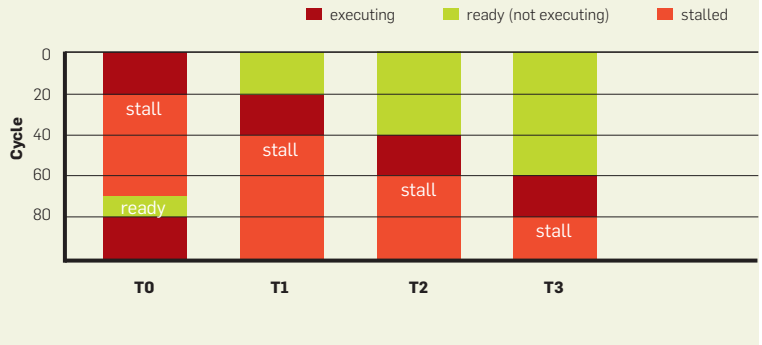
- **The core executes an instruction from at most one thread each processor clock, but maintains state for four threads on-chip simultaneously (T=4).**
- **Core threads issue explicit width-32 SIMD vector instructions; 32 ALUs simultaneously execute a vector instruction in a single clock.**
- **The core contains a pool of 16 general-purpose vector registers (each containing a vector of 32 single-precision floats) partitioned among thread contexts.**
- **The only source of thread stalls is texture access; they have a maximum latency of 50 cycles.**

Shader compilation by the graphics driver produces a GPU binary from high-level fragment shader source. The resulting vector instruction sequence performs 32 invocations of the fragment shader simultaneously by carrying out each invocation in a single lane of the width-32 vectors. The compiled binary requires four vector registers for temporary results and contains 20 arithmetic instructions between each texture access operation.

At runtime, the GPU executes a copy of the shader binary on each of its four thread contexts, as illustrated in Figure B. The core executes T0 (thread 0) until it detects a stall resulting from texture access in cycle 20. While T0 waits for the result of the texturing operation, the core continues to execute its remaining three threads. The result of T0's texture access becomes available in cycle 70. Upon T3's stall in cycle 80, the core immediately resumes T0. Thus, at no point during execution are ALUs left idle.

When executing the shader program for this example, a minimum of four threads is needed to keep core ALUs busy. Each thread operates simultaneously on 32 fragments; thus, 4*32=128 fragments are required for the chip to achieve peak performance. As memory latencies on real GPUs involve hundreds of cycles, modern GPUs must contain support for significantly more threads to sustain high utilization. If we extend our simple GPU to a more realistic size of 16 processing cores and provision each core with storage for 16 execution contexts, then simultaneous processing of 8,192 fragments is needed to approach peak processing rates. Clearly, GPU performance relies heavily on the abundance of parallel shading work.

Figure B: Thread Execution on the example GPU core



These components do not simply augment programmable processing; they perform sophisticated operations and constitute an additional hundreds of gigaflops of processing power. Two of the most important operations performed via fixed-function hardware are texture filtering and rasterization (fragment generation).

Texturing is handled almost entirely by fixed-function logic. A texturing operation samples a contiguous 1D, 2D, or 3D signal (a texture) that is discretely represented by a multidimensional array of color values (2D texture data is simply an image). A GPU texture-filtering unit accepts a point within the texture's parameterization (represented by a floating-point tuple, such as {.5,.75}) and loads array values surrounding the coordinate from memory. The values are then filtered to yield a single result that represents the texture's value at the specified coordinate. This value is returned to the calling shader function. Sophisticated texture filtering is required for generating high-quality images. As graphics APIs provide a finite set of filtering kernels, and because filtering kernels are computationally expensive, texture filtering is well suited for fixed-function processing.

Primitive rasterization in the FG stage is another key pipeline operation currently implemented by fixed-function components. Rasterization involves densely sampling a primitive (at least once per output image pixel) to determine which pixels the primitive overlaps. This process involves computing the location of the surface at each sample point and then generating fragments for all sample points covered by the primitive. Bounding-box computations and hierarchical techniques optimize the rasterization process. Nonetheless, rasterization involves significant computation.

In addition to the components for texturing and rasterization, GPUs contain dedicated hardware components for operations such as surface visibility determination, output pixel compositing, and data compression/decompression.

Understanding key ideas behind the success of GPU computing is valuable not only for developers targeting software for GPU execution, but also for informing the design of new architectures and programming systems for other domains.

The Memory System

Parallel-processing resources place extreme load on a GPU's memory system, which services memory requests from both fixed-function and programmable components. These requests include a mixture of fine-granularity and bulk prefetch operations and may even require real-time guarantees (such as display scan out).

Recall that a GPU's programmable cores tolerate large memory latencies via hardware multithreading and that interstage stream data accesses can be prefetched. As a result, GPU memory systems are architected to deliver high-bandwidth, rather than low-latency, data access. High throughput is obtained through the use of wide memory buses and specialized GDDR (graphics double data rate) memories that operate most efficiently when memory access granularities are large. Thus, GPU memory controllers must buffer, reorder, and then coalesce large numbers of memory requests to synthesize large operations that make efficient use of the memory system. As an example, the ATI Radeon HD 4870 memory controller manipulates thousands of outstanding requests to deliver 115GB per second of bandwidth from GDDR5 memories attached to a 256-bit bus.

GPU data caches meet different needs from CPU caches. GPUs employ relatively small, read-only caches (no cache coherence) that serve to filter requests destined for the memory controller and to reduce bandwidth requirements placed on main memory. Thus, GPU caches typically serve to amplify total bandwidth to processing units rather than decrease latency of memory accesses. Interleaved execution of many threads renders large read-write caches inefficient because of severe cache thrashing. Instead, GPUs benefit from small caches that capture spatial locality across simultaneously executed shader invocations. This situation is common, as texture accesses performed while processing fragments in close screen proximity are likely to have overlapping texture-filter support regions.

Although most GPU caches are small, this does not imply that GPUs contain little on-chip storage. Significant amounts of on-chip storage are used to hold entity streams, execution contexts, and thread scratch data.

Pipeline Scheduling and Control

Mapping the entire graphics pipeline efficiently onto GPU resources is a challenging problem that requires dynamic

and adaptive techniques. A unique aspect of GPU computing is that hardware logic assumes a major role in mapping and scheduling computation onto chip resources. GPU hardware "scheduling" logic extends beyond the thread-scheduling responsibilities discussed in previous sections. GPUs automatically assign computations to threads, clean up after threads complete, size and manage buffers that hold stream data, guarantee ordered processing when needed, and identify and discard unnecessary pipeline work. This logic relies heavily on specific upfront knowledge of graphics workload characteristics.

Conventional thread programming uses operating-system or threading API mechanisms for thread creation, completion, and synchronization on shared structures. Large-scale multithreading coupled with the brevity of shader function execution (at most a few hundred instructions), however, means GPU thread management must be performed entirely by hardware logic.

GPUs minimize thread launch costs by preconfiguring execution contexts to run one of the pipeline's three types of shader functions and reusing the configuration multiple times for shaders of the same type. GPUs prefetch shader input records and launch threads when a shader stage's input stream contains a sufficient number of entities. Similar hardware logic commits records to the output stream buffer upon thread completion. The distribution of execution contexts to shader stages is reprovisioned periodically as pipeline needs change and stream buffers drain or approach capacity.

GPUs leverage upfront knowledge of pipeline entities to identify and skip unnecessary computation. For example, vertices shared by multiple primitives are identified and VP results cached to avoid duplicate vertex processing. GPUs also discard fragments prior to FP when the fragment will not alter the value of any image pixel. Early fragment discard is triggered when a fragment's sample point is occluded by a previously processed surface located closer to the camera.

Another class of hardware optimizations reorganizes fine-grained operations for more efficient processing. For example, rasterization orders fragment generation to maximize screen proximity of samples. This ordering improves texture cache hit rates, as well as instruction stream sharing across shader invocations. The GPU memory controller also performs automatic reorganization when it reorders memory requests to optimize memory bus and DRAM utilization.

GPUs ensure inter-fragment PO ordering dependencies using hardware logic. Implementations use structures such as post-FP reorder buffers or scoreboards that delay fragment thread launch until the processing of overlapping fragments is complete.

GPU hardware can take responsibility for sophisticated scheduling decisions because semantics and invariants of the graphics pipeline are known *a priori*. Hardware implementation enables fine-granularity logic that is informed by precise knowledge of both the graphics pipeline and the underlying GPU implementation. As a result, GPUs are highly efficient at using all available resources. The drawback of this approach is that GPUs execute only those computations for which these invariants and structures are known.

Graphics programming is becoming increasingly versatile. Developers constantly seek to incorporate more sophisticated algorithms and leverage more configurable graphics pipelines. Simultaneously, the growing popularity of GPU-based computing for non-graphics applications has led to new interfaces for accessing GPU resources. Given both of these trends, the extent to which GPU designers can embed a priori knowledge of computations into hardware scheduling logic will inevitably decrease over time.

A major challenge in the evolution of GPU programming involves preserving GPU performance levels and ease of use while increasing the generality and expressiveness of application interfaces. The designs of "GPU-compute" interfaces, such as NVIDIA's CUDA and AMD's CAL, are evidence of how difficult this challenge is. These frameworks abstract computation as large batch operations that involve many invocations of a kernel function operating in parallel. The resulting computations execute on GPUs efficiently only under conditions of massive data parallelism. Programs that attempt to implement non data-parallel algorithms perform poorly.

GPU-compute programming models are simple to use and permit well-written programs to make good use of both GPU programmable cores and (if needed) texturing resources. Programs using these interfaces, however, cannot use powerful fixed-function components of the chip, such as those related to compression, image compositing, or rasterization. Also, when these interfaces are enabled, much of the logic specific to graphics-pipeline scheduling is simply turned off. Thus, current GPU-compute programming frameworks significantly restrict computations so that their structure, as well as their use of chip resources, remains sufficiently simple for GPUs to run these programs in parallel.

GPU and CPU Convergence

The modern graphics processor is a powerful computing platform that resides at the extreme end of the design space of throughput-oriented architectures. A GPU's processing resources and accompanying memory system are heavily optimized to execute large numbers of operations in parallel. In addition, specialization to the graphics domain has enabled the use of fixed-function processing and allowed hardware scheduling of a parallel computation to be practical. With this design, GPUs deliver unsurpassed levels of performance to challenging workloads while maintaining a simple and convenient programming interface for developers.

Today, commodity CPU designs are adopting features common in GPU computing, such as increased core counts and hardware multithreading. At the same time, each generation of GPU evolution adds flexibility to previous high-throughput GPU designs. Given these trends, software developers in many fields are likely to take interest in the extent to which CPU and GPU architectures and, correspondingly, CPU and GPU programming systems, ultimately converge. C

Kayvon Fatahalian (kayvonf@gmail.com) and **Mike Houston** are Ph.D. candidates in computer science in the Computer Graphics Laboratory at Stanford University.

A previous version of this article was published in the March 2008 issue of *ACM Queue*.

DOI:10.1145/1400181.1400198

Knowing the structure of criminal and terrorist networks could provide the technical insight needed to disrupt their activities.

BY JENNIFER XU AND HSINCHUN CHEN

The Topology of Dark Networks

SCIENTISTS FROM A variety of disciplines, including physics, sociology, biology, and computing, all explore the topological properties of complex systems that can be characterized as large-scale networks, including scientific collaborations, the Web, the Internet, electric power grids, and biological and social networks. Despite the differences in their components, functions, and size, they are surprisingly similar in topology, leading to the conjecture that many complex systems are governed by the ubiquitous "self-organizing" principle, or that the internal complexity of a system increases without being guided or managed by external sources.

Still missing from this line of research, however, is an analysis of the topology of "dark" networks hidden from view yet that could have devastating effects on our social order and economy. Terrorist organizations, drug-trafficking rings, arms-smuggling operations, gang enterprises, and many other covert networks are dark networks. Their structures are largely unknown to outsiders due to the difficulty of accessing and collecting reliable data. Do they share the same topological properties as other types of empirical networks? Do they follow the self-organizing principle? How do they achieve efficiency under constant surveillance and threat from the authorities? How robust are they against attack? Here, we explore the topological properties of several covert criminal- and terrorist-related networks, hoping to contribute to the general understanding of the structural properties of complex systems in hostile environments while providing authorities insight regarding disruptive strategies.

Topological analysis focusing on the statistical characteristics of network structure is a relatively new methodology for studying large-scale networks.[1,11] Large complex networks can be categorized into three types: random, small-world, and scale-free.[1] A number of statistics (see Table 1) have been developed to study their topology; three of which—average path length, average clustering coefficient, and degree distribution—are widely used to categorize networks.

In random networks, two arbitrary nodes are connected with a probability p; as a result each node has roughly the same number of links. Random networks are characterized by small l, small C, and bell-shaped Poisson distributions.[1] A small l means an arbitrary node can reach any other node in a few steps. A small C implies that random networks are not likely to contain clusters and groups. Studies by physicists and computer and social scientists have found that most complex systems are not random but present small-world and scale-free properties (see Albert[1] for a comprehensive review of these studies).

A small-world network has a significantly larger C than its random-network counterpart while maintaining a relatively small l.[11] Scale-free networks, on the other hand, are char-

ILLUSTRATION BY LEANDER HERZOG

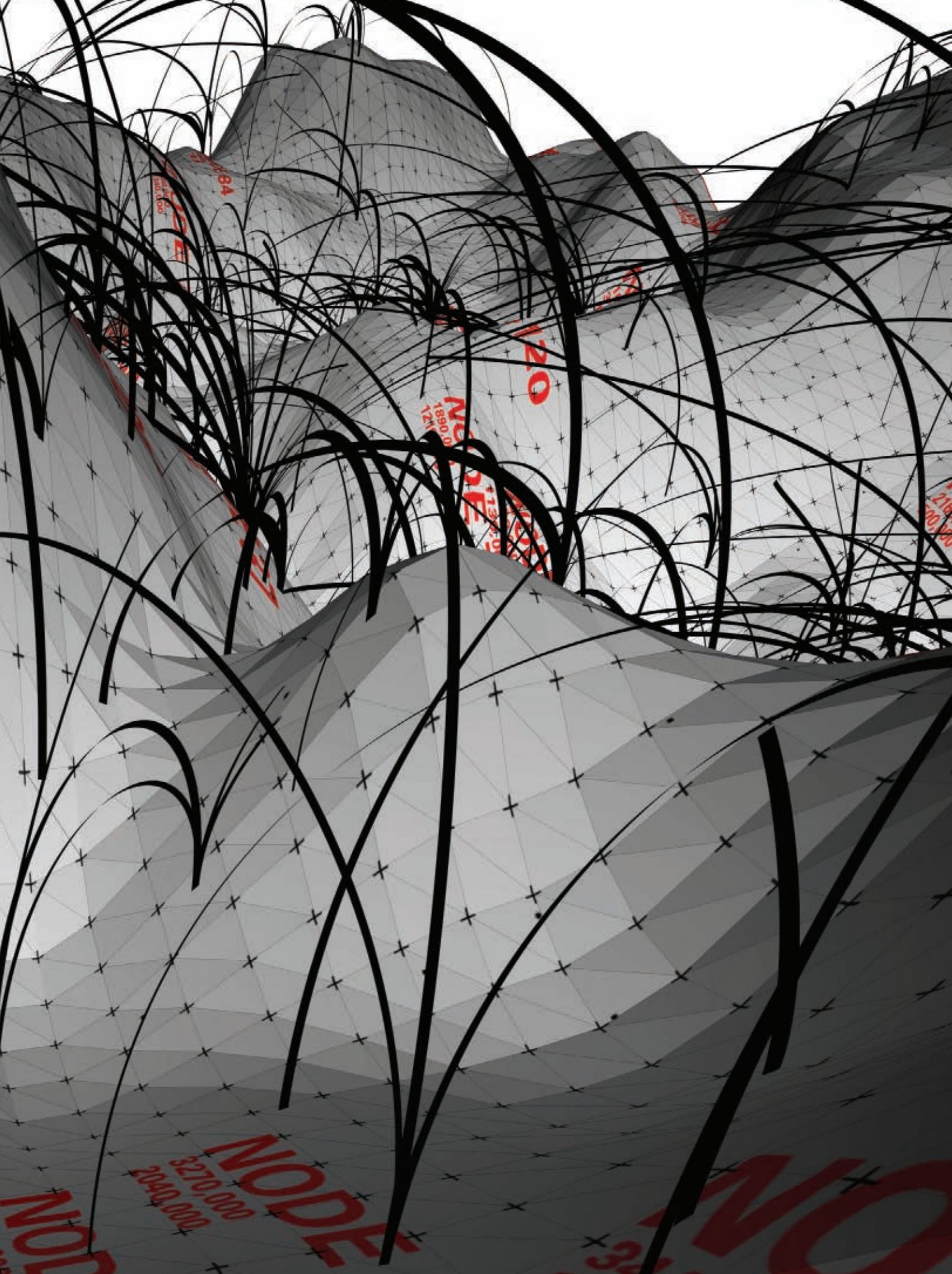

NODE
3270,000
2040,000

Figure 1: The giant component in the GSJ Network (data courtesy of Marc Sageman[9]). The terrorists belong to one of four groups: Al Qaeda or Central Staff (pink), Core Arabs (yellow), Maghreb Arabs (blue), and Southeast Asians (green). Each circle represents one or more terrorist activities (such as the September 11 attacks and the Bali bombing) as noted.

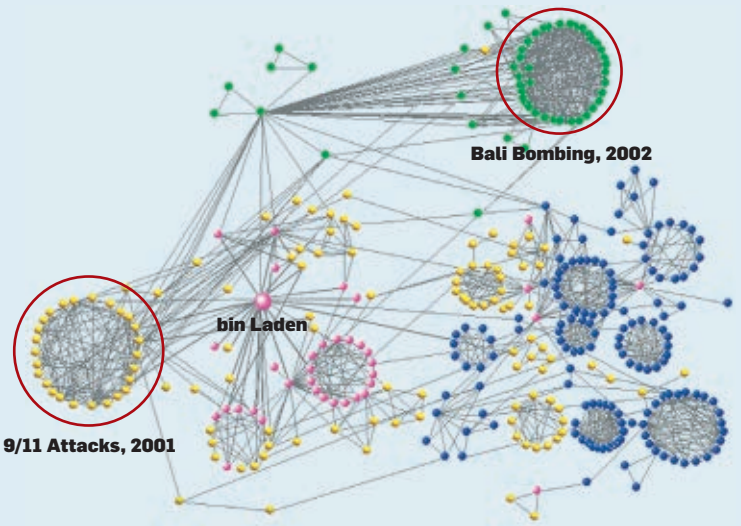


Table 1: Statistics we used for studying network topology.

Statistics	Description
Average Path Length, l[1]	The average of the lengths of the shortest paths between all pairs of nodes in a network.
Average Clustering Coefficient, C[11]	The average of all individual clustering coefficients, C_i, which is the number of links that actually exist among node i's neighbors over the possible number of links among these neighbors.
Average Degree, $<k>$[10]	The average of all individual degrees, k_i, which is the number of links that node i has.
Degree Distribution, $p(k)$[1]	The probability that an arbitrary node has exactly k links.
Link Density, d[10]	The number of links that actually exist over the possible number of links in a network.
Assortativity, r[8]	The Pearson correlation between the degrees of two adjacent nodes.
Global Efficiency, e[4]	The average of the inverses of the lengths of the shortest paths over all pairs of nodes in a network.

acterized by the power-law degree distribution, meaning that while a large percentage of nodes in the network has just a few links, a small percentage of the nodes have a large number of links.[1] Scientists conjecture that scale-free networks evolve following the self-organizing principle, where growth and preferential attachment play a key role in the emergence of the power-law distribution. Preferential attachment implies that the more links a node has, the more new links it is able to attract, manifesting the "rich-get-richer" phenomenon.

Analyzing the topology of complex systems has important implications for our understanding of nature and society. Research has shown that the function of a complex system may be affected to a great extent by its network topology.[1] For instance, the Web's short average path length makes cyberspace a convenient, navigable system in which any two Web pages are (on average) only 19 clicks away from each other. It also has been shown that the greater tendency for clustering in metabolic networks corresponds to the organization of functional modules in cells, contributing to the behavior and survival of organisms. In addition, networks with scale-free properties are highly robust against random failure and errors but notably vulnerable to targeted attacks.[5]

Methods and Data

To understand the topology and function of dark networks we studied four terrorist- and criminal-related networks:

Global Salafi Jihad (GSJ).[9] This terrorist network's 366 members (see Figure 1) include some from Osama bin Laden's Al Qaeda, connected by, perhaps, kinship, friendship, religious ties, and relationships formed after they joined. The GSJ data was provided to us by Marc Sageman, a forensic psychiatrist in private practice in Philadelphia and author of *Understanding Terror Networks*.[9] The network was constructed entirely from open-source data, including publicly available documents and transcripts of court proceedings and press, scholarly, and Web articles. Sageman scrutinized and cross-validated the information about all nodes (terrorists) and links (relationships). However, as he pointed out in his book, the data is also subject to several limitations. First, the members in the network may not be a representative sample of the global Salafi jihad. The data may be biased toward leaders and members captured or identified in attacks. Second, because most of the sources were based on retrospective accounts, the data may be subject to self-reported bias. Despite these limitations, the data provides stunning insight into clandestine terrorist organizations.

Meth World. In trafficking illegal methamphetamines,[12] this network consisted of 1,349 criminals traced and investigated by the Tucson Police Department from 1985 to 2002. Because no information about the social relationships among them is directly available, we were granted access to the police databases and retrieved all the crime incidents in which these people were involved from 1985 to 2002. We created a link between any

two of them if they committed at least one crime together for which they were convicted.

Although the network was carefully validated by the crime analysts in the Tucson Police Department,[12] the co-occurrence links we generated from crime-incident records may not reflect the real relationships among the criminals. Two related criminals would appear to be unconnected if, for example, they never committed a crime together. On the other hand, a coincidental link may connect two criminals if they happened to have participated in the same crime. These two problems—missing link and coincidental link—are also common in other types of networks (such as those involving movie actors[12]) based on the co-occurrence of two nodes in the same events or activities.

Another group of 3,917 criminals involved in gang-related crimes in Tucson from 1985 to 2002.[12] As in Meth World, the links in this network were generated through co-occurrence analysis of the crime-incident records.

A terrorist Web site network ("the Dark Web"). In 2005, based on reliable government sources, we identified 104 Web sites created by four major international terrorist groups—Al-Gama'a al-Islamiyya, Hizballa, Al-Jihad, and Palestinian Islamic Jihad—fetching all of their pages and extracting all of their hyperlinks. We recognized a link between any two Web sites if at least one hyperlink existed between any two Web pages in them.

Results

Table 2 lists the basic statistics of the four elicited networks. Like many other empirical networks, each of them contains many isolated components and a single giant component. The giant component in a graph is defined as the largest connected subgraph.[1] The separation between the 356 terrorists in the GSJ network and the remaining 10 terrorists is because we found no valid evidence to connect the 10 terrorists to the giant component in the network. The giant components in Meth World and the gang network contain only 68.5% and 57.0% of the nodes, respectively. This may be because we collected the data from a single law-enforcement jurisdiction that might lack complete information about all relationships among criminals, causing missing links between the giant component and the smaller components. The isolated components in the Dark Web are possibly the result of the differences in the four terrorist groups' distinctive ideologies.

As in many other network-topology studies (such as Barabási[2]), we performed a topological analysis on only the giant component in the four elicited networks. Table 2 lists the average degrees and maximum degrees of the four networks, showing that some terrorists in the GSJ network and some terrorist Web sites in the Dark Web are extremely popular, connecting to more than 10% of their nodes.

This "assortativity" reflects the tendency for nodes to connect with others that are similarly popular in terms of link degree. The assortativity coefficients of the four networks are all significantly different from 0. The GSJ and the gang networks present positive assortativity, meaning that popular members tend to connect with other popular members. In positively assortative networks, high-degree nodes tend to cluster together as core groups,[8] a phenomenon evident in the GSJ network in which bin Laden and his closest cohorts form the core of the network and issue commands to other parts of the network.[9] In contrast, Meth World and the Dark Web have negative assortativity coefficients, or "disassortativity."

Meth World consists of drug dealers selling illegal methamphetamine to many individual buyers who do not connect with many other buyers or dealers. Moreover, studies have found that street drug-dealing organizations are led by a few high-level individuals who connect with a large number of low-level retail drug dealers.[6] Because high-degree nodes connect to low-degree nodes, Meth World is characterized by disassortative mixing patterns. On the other hand, the disassortativity in the Dark Web is the result of the fact that the popular Dark Web sites routinely receive many inbound hyperlinks from less popular Web sites.

To ascertain if the dark networks are small worlds, we calculated average path lengths, clustering coefficients, and global efficiency (see Table 3). For each network, we generated 30 random counterparts with the same number of nodes and the same number of links as in the corresponding elicited networks. We found that all of them have significantly high clustering coefficients compared to their random counterparts. Moreover, although the

Table 2. Basic statistics and scale-free properties concerning dark networks. The numbers in parentheses in the third row are the percentage of total nodes included in the giant components. The numbers in parentheses in the fifth row are the percentage of total nodes connected to the highest-degree nodes. ** *p*-value < 0.05 * *p*-value < 0.01

	GSJ	Meth World	Gang Network	Dark Web
Number of Nodes, *n*	366	1349	3917	104
Number of Links, *m*	1247	4784	9051	156
Size of Giant Component	356 (97.3%)	924 (68.5%)	2231 (57.0%)	80 (77.9%)
Average Degree, <*k*>	6.97	4.62	5.74	3.88
Maximum Degree	44 (12.4%)	37 (4.0%)	51 (2.3%)	33 (41.3%)
Link Density, *d*	0.02	0.01	0.003	0.05
Assortativity, *r*	0.41**	-0.14**	0.17**	-0.24*
Power-Law Distribution Exponent, γ	1.38	1.86	1.95	1.10
Goodness of Fit, R^2	0.74	0.89	0.81	0.82

differences are statistically significant (greater than three standard deviations), the average path length of the four networks (except for the gang network) is just slightly greater than their random counterparts.

These small-world properties imply that terrorists or criminals are able to connect with any other member in a network through only a few mediators. In addition, the networks are sparse, with very low link density. These properties have important implications for the communication efficiency of the networks. Due to the increased risk of being detected by authorities as more people are involved in a network, short path length and link sparseness help lower the risk of detection and enhance efficiency of communication. As a result, the global efficiency of each network is compatible to their random-network counterparts.

On the other hand, a high clustering coefficient contributes to the local efficiency of all four dark networks. Previous studies have shown evidence of groups and teams in these networks in which members tend to have denser and stronger relationships with one another.[9,12] Communication among group members becomes more efficient, making a crime or an attack easier to plan, organize, and execute.

We also calculated the path length of other nodes to central nodes, finding that members in the three studied criminal and terrorist networks are extremely close to their leaders. For example, the terrorists in the GSJ network are on average only 2.5 links away from bin Laden himself, meaning his command is able to reach an arbitrary member through only two mediators. Similarly, the average path length to the leader of Meth World is only three links.[12] Such a short chain of command also means communication efficiency.

Special attention should be paid to the Dark Web. Despite the small size of its giant component (80 nodes), the average path length is 4.70 links, only slightly larger than the 4.20 links in the GSJ network, which has almost nine times more nodes. Since hyperlinks help visitors navigate Web pages and because terrorist Web sites are often used for soliciting new members and donations, the relatively long path length may be due to the reluctance of terrorist groups to share resources with other terrorist groups.

Moreover, the dark networks present scale-free properties with power-law degree distributions in the form of $p(k) \sim k^{-\gamma}$. Because degree-distribution curves fluctuate, we display the cumulative degree distributions, $P(k)$, in a log-log plot (see Figure 2). $P(k)$ is defined as the probability that an arbitrary node has at least k links. Figure 2 also outlines the fitted power-law distributions. The last two rows of Table 1 report the exponent value, γ, and the goodness-of-fit, R^2, for each network. Figure 2 shows that all these networks are scale-free. The power-law distributions fit especially well at the tails. Note that the three human networks display two-regime scaling behavior, which has also been observed in other empirical networks (such as those involving scientific collaboration).[2]

Two mechanisms have been proposed to account for the emergence of two-regime power-law degree distributions during the evolution of a network.[2] First, new links may emerge between existing network members. This emergence implies that criminals or terrorists who were not related previously could become related over time. This assumption is logical since two unacquainted members could become acquainted through a third member who knows each of them. In the GSJ network, 22.6% of the links were post-joining ties formed among existing members. Second, an existing link may be

Figure 2: Cumulative degree distributions: (a) GSJ network, (b) Meth World, (c) gang network, and (d) Dark Web.

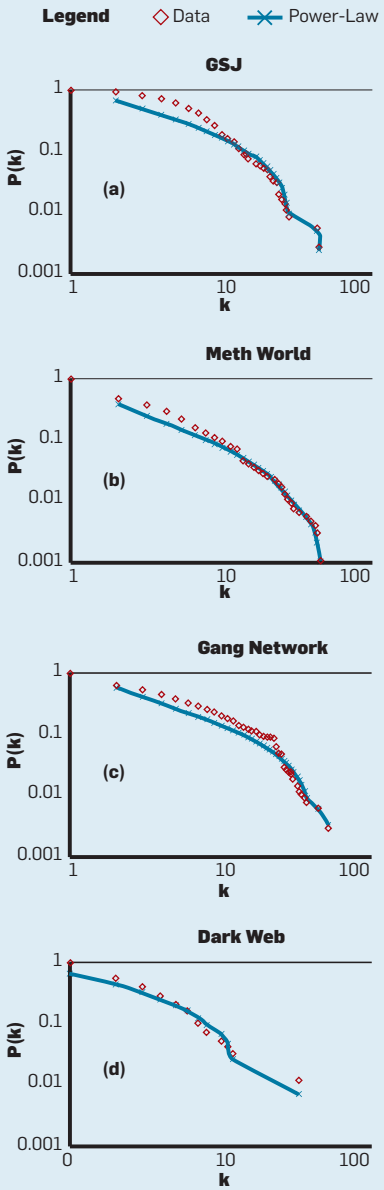


Table 3: Small-world properties of dark networks. Each network includes the metrics in the elicited network (data) and the metrics in the random graph counterpart (random). Numbers in parentheses are standard deviations.

	GSJ		Meth World		Gang Network		Dark Web	
	Data	Random	Data	Random	Data	Random	Data	Random
Average Path Length, l	4.20	3.23 (0.040)	6.49	4.52 (0.056)	9.56	4.59 (0.034)	4.70	3.15 (0.108)
Average Clustering Coefficient, C	0.55	0.020 (0.0029)	0.60	0.005 (0.0014)	0.68	0.002 (0.0005)	0.47	0.049 (0.0155)
Global Efficiency, e	0.28	0.33 (0.004)	0.18	0.23 (0.003)	0.12	0.23 (0.001)	0.30	0.34 (0.019)

rewired—a strong possibility in GSJ and the Dark Web. However, such rewiring would not affect Meth World or the gang network because a co-occurrence link could not be rewired once it was created.

An interesting topology-related question is what mechanisms play a role in producing the properties we observed in dark networks? Short average path length, high clustering coefficient, power-law degree distributions with two-regime scaling behavior in the human networks? That is, can we regenerate the four dark networks based on known mechanisms (such as growth and preferential attachment)? To answer, we conducted a series of simulations in which we generated 30 networks for each elicited human network based on three evolutionary mechanisms:

Growth. Starting with a small number of nodes, at each time step we add a new node to connect with existing nodes in the network;

Preferential attachment. The probability that an existing node will receive a link from the new node depends on the number of links the node already maintains. The more links it has the more likely it will receive a new link; and

New links among existing nodes. At each time step, a random pair of existing nodes may connect, depending on the number of common neighbors they have. The more common neighbors they share the more likely they will also be connected with each other.

We expected that the first two mechanisms would generate a power-law degree distribution[1] and that the third would generate a high clustering coefficient and two-regime scaling behavior.[2] Our simulations showed that the power -law degree distributions are easily regenerated, with R^2 ranging from 0.83 (the gang network) to 0.88 (GSJ). The two-regime scaling behavior was also present in the simulated networks for the human networks. However, the highest clustering coefficient in a simulation was only 0.24 (GSJ), far less than what we obtained from the elicited networks (0.55–0.68). This finding implies that some other mechanisms must have contributed to the substantially high clustering coefficients we observed in the dark networks. We suspect that member recruitment is one such mechanism. Employing active recruitment methods, subgroups of terrorists or criminals are able to attract new members into their groups. The new members quickly become acquainted with many existing members, substantially increasing the clustering coefficients.

The terrorists in the GSJ network are on average only 2.5 links away from bin Laden himself, meaning his command is able to reach an arbitrary member through only two mediators.

Caveats

A notable point is that two problems may have affected the structures of the three elicited human networks—GSJ, Meth World, and the gang network. First, they may have missing links that can cause the networks to appear to be less efficient; there may actually be hidden "shortcuts" connecting distant parts of the networks. Second, the presence of coincidental "fake" links might cause the elicited networks to be more efficient than they would otherwise be since these links are not communication channels.

To test how the results would be affected by missing links, we added various percentages of the existing links to the elicited networks based on three effects used in missing-link-prediction research:[7]

Random effect. A link is added between a randomly selected pair of nodes not originally connected;

Common neighbor effect. A link is added between a pair of unconnected nodes if they share common neighbors; the more common neighbors they share the more likely they will be connected; and

Preferential attachment effect. The probability that a pair of unconnected nodes will be linked together depends on the product of their link degrees.

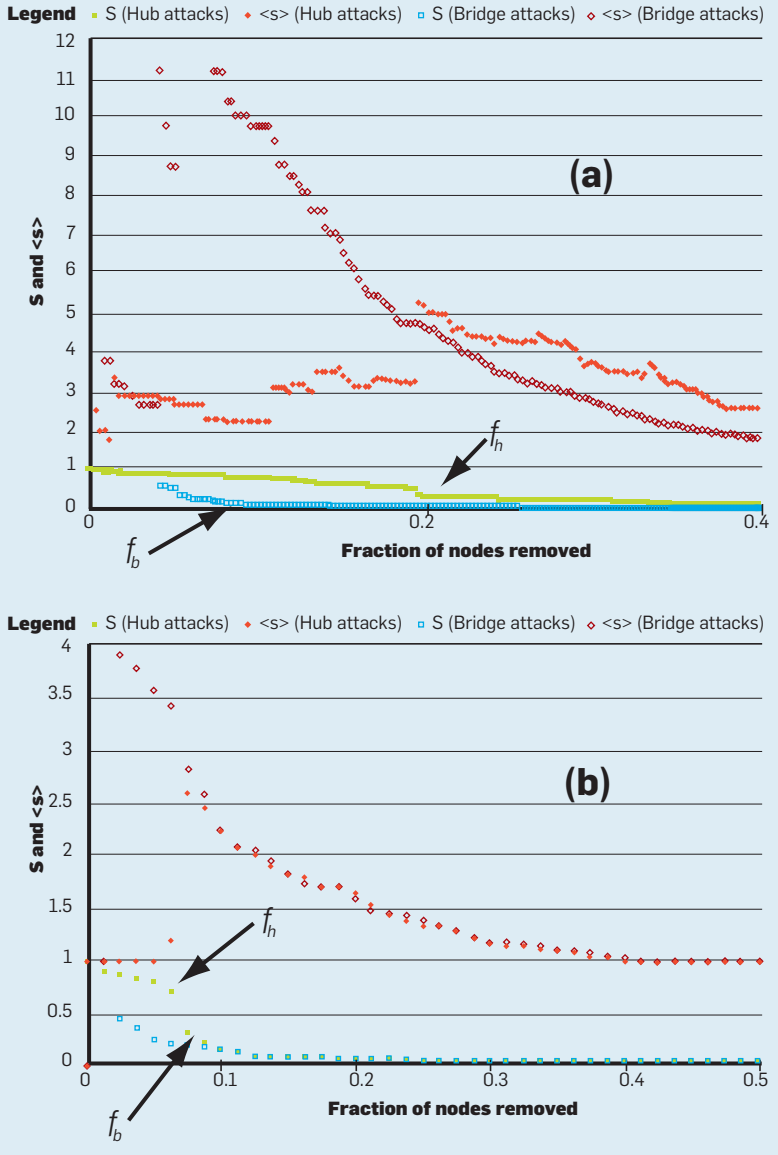
We found that the small-world and scale-free properties of the four networks do not change when missing links are added. For example, when we added up to 10% of the links, the average path lengths ranged from 3.55 links (GSJ, preferential-attachment links added) to 9.45 links (the gang network, common-neighbor links added); the clustering coefficients ranged from 0.45 (GSJ, random links added) to 0.67 (the gang network, common-neighbor links added); and the R2 of power-law degree distributions ranged from 0.61 (GSJ, random links added) to 0.93 (the gang network,

preferential-attachment links added).

We also randomly removed percentages of links to test the effect of "fake" links on the results, finding they were still valid even when we removed 10% of the links.

Prior research found that network topology has a significant effect on a network's robustness against failure and attacks and that scale-free networks are robust against failure (random removal of nodes).[5] Because we found that the four dark networks have scale-free properties, we tested their robustness against only targeted attacks. We simulated two types of attacks in the form of node removal: those targeting hubs and those targeting bridges. While hubs are nodes that have many links (high degree), bridges are nodes through which pass many shortest paths (high "betweenness").[10] When simulating the attacks we distinguished between two attack strategies: simultaneous removal of a fraction of the nodes based on a measure (degree or betweenness) without updating the measure after each removal and progressive removal of nodes with the measure being updated after each removal.

Figure 3: Dark-network robustness against attacks: (a) progressive attacks against the GSJ network and (b) progressive attacks against the Dark Web. Two types of attack are hub (filled markers) and bridge (empty markers).



We plotted the changes in S (the fraction of the nodes in the giant component), $\langle s \rangle$ (the average size of remaining components), and average path length after some nodes are removed. We found that progressive attacks are more devastating than simultaneous attacks. Progressive attacks are similar to "cascading failures" in the Internet where an initial failure might cause a series of failures because high-traffic volume is redirected to the next bridge node.

Figure 3 (a) and (b) shows the difference between the network reactions to bridge attacks and to hub attacks. The critical points, f, at which the network falls into many small components, are marked in the figure. The behavior of Meth World and the gang network is similar to the behavior of the GSJ network, showing that these terrorist and criminal networks are more sensitive to attacks targeting bridges than to those targeting hubs ($f_b < f_h$). However, in Figure 3(b), f_b and f_h are very close, indicating that hub attacks and bridge attacks are equally effective at disrupting a one-regime scale-free network.

These results are consistent with findings from a prior study[5] that pure scale-free networks are vulnerable to both hub and bridge attacks, while small-world networks are more vulnerable to bridge attacks. In small-world networks consisting of communities and groups, many bridges may link different communities together. Intuitively, when they are removed, the network should quickly fall apart. Note that a bridge may not necessarily be a hub since a node connecting two communities can have as few as two links. Small-world networks (such as dark networks) are thus more vulnerable to bridge attacks than to hub attacks.

In the four dark networks we studied, bridges and hubs are usually not the same nodes. The rank order correlations between degree and betweenness in GSJ, Meth World, and the gang network are 0.63, 0.47, and 0.30, respectively. Note that although bridge attacks are more devastating, strategies targeting the hubs are also fairly effective since the networks have scale-free properties. Hub attacks and bridge attacks can be equally effec-

tive in tearing apart a pure scale-free network (such as the Dark Web, with a high degree-betweenness-rank-order correlation, 0.70) in which hubs function simultaneously as bridges connecting different parts of the network.

Conclusion

Dark networks (such as those involving terrorists and criminal narcotics traffickers) are hidden from nonparticipants yet could have a devastating effect on our social order and economy. Understanding their topology yields greater insight into the nature of clandestine organizations and could help develop effective disruptive strategies. However, obtaining reliable data about dark networks is extremely difficult, so our understanding of them remains largely hypothetical. To the best of our knowledge, the data sets we explore here, though subject to limitations, are the first to allow for statistical analysis of the topologies of dark networks.

We found that the covert networks we studied share many common topological properties with other types of networks. Their efficiency in terms of communication and information flow and commands can be tied to their small-world structures, which are characterized by short average path length and a high clustering coefficient. In addition, we found that due to their small-world properties, dark networks are more vulnerable to attack on their bridges that connect different communities within them than to attacks on their hubs. This finding may give authorities insight for intelligence and security purposes.

Another interesting finding about the three elicited human networks we studied is that their substantially high clustering coefficients (not always present in other empirical networks) are difficult to regenerate based on only known network effects (such as preferential attachment and small-world effects). Other mechanisms (such as recruitment) may also play an important role in network evolution. Other research has found that alternative mechanisms (such as highly optimized tolerance) may govern the evolution of many complex systems in environments characterized by high risk and uncertainty.[3] Our future research will focus on the effects of such alternative mechanisms on network topology. In addition, our findings are all based on a static view of the networks we studied; that is, we did not consider a large variety of dynamics that might have taken place in the evolution of the networks, so evolution study is definitely in our plans for future research.

To the best of our knowledge, the data sets we explore here, though subject to limitations, are the first to allow for statistical analysis of the topologies of dark networks.

Please also note that care is needed when interpreting these findings. Because dark networks are covert and largely unknown, hidden links may be missing in the elicited networks. These links may play a critical role in maintaining the function of the covert organizations. As a result, one must be extremely cautious when a decision is to be made to disrupt them. C

References

1. Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics 74*, 1 (Jan. 2002), 47–97.
2. Barabási, A.-L., Jeong, H., Zéda, Z., Ravasz, E., Schubert, A., and Vicsek, T. Evolution of the social network of scientific collaborations. *Physica A, 311*, 3–4 (Aug. 2002), 590–614.
3. Carlson, J.M. and Doyle, J. Highly optimized tolerance: A mechanism for power laws in designed systems. *Physical Review E60*, 2 (Aug. 1999), 1412–1427.
4. Crucitti, P., Latora, V., Marchiori, M., and Rapisarda, A. Efficiency of scale-free networks: Error and attack tolerance. *Physica A 320* (Mar. 2003), 622–642.
5. Holme, P., Kim, B.J., Yoon, C.N., and Han, S.K. Attack vulnerability of complex networks. *Physical Review E 65*, Article No. 056109 (May 2002), 1–14.
6. Levitt, S.D. and Dubner, S.J. *Freakonomics: A Rogue Economist Explores the Hidden Side of Everything.* William Morrow, New York, 2005.
7. Liben-Nowell, D. and Kleinberg, J. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology 58*, 7 (May 2007), 1019–1031.
8. Newman, M.E.J. Mixing patterns in networks. *Physical Review E 67*, 2, Article No. 026126 (Feb. 2003), 1–13.
9. Sageman, M. *Understanding Terror Networks.* University of Pennsylvania Press, Philadelphia, PA, 2004.
10. Wasserman, S. and Faust, K. *Social Network Analysis: Methods and Applications.* Cambridge University Press, Cambridge, U.K., 1994.
11. Watts, D.J. and Strogatz, S.H. Collective dynamics of 'small-world' networks. *Nature 393*, 6684 (June 1998), 440–442.
12. Xu, J. and Chen, H. Untangling criminal networks: A case study. In *Proceedings of the First NSF/NIJ Symposium on Intelligence and Security Informatics* (Tucson, AZ, June 2–3, 2003). Springer, Berlin, Germany, 2003, 232–248.

Jennifer Xu (jxu@bentley.edu) is an assistant professor of computer information systems in Bentley College, Waltham, MA.

Hsinchun Chen (hchen@eller.arizona.edu) is McClelland Endowed Professor in the Department of Management Information Systems and head of the Artificial Intelligence Lab at the University of Arizona, Tucson, AZ.

DOI:10.1145/1400181.1400199

What should be done to reverse falling CS enrollment in the Canadian education system?

BY JACOB SLONIM, SAM SCULLY, AND MICHAEL MCALLISTER

Crossroads for Canadian CS Enrollment

THE DECLINE IN computer science enrollment in practically all Canadian universities has resulted in a shortage of technical graduates that in turn jeopardizes the ability of the Canadian information and communication technology (ICT) sector to remain vibrant, innovative, and competitive in the global economy. The sector's anticipated wave of retirements will exacerbate the situation in the next five to 10 years. With upward of 30% of Canada's GDP directly or indirectly involving ICT, university CS educators must rejuvenate the discipline and rethink the country's long-range strategy to teaching CS to ensure continual renewal of the ICT work force. While some preliminary signs indicate that enrollment levels in 2007–2008 are stabilizing, the public, private, and education sectors must coordinate their approaches to ICT recruiting, training, and development to achieve the required renewal.

We collected CS enrollment data (1999–2007) for most Canadian universities to determine the quantitative extent of the enrollment decrease and personally interviewed many CS department chairs and senior university administrators to learn their views on the causes of the decline. Each university meeting we conducted began with a report on the enrollment decrease for that university, a discussion structured around three basic questions:

- What is the reason for the decline in your university's CS enrollment?;
- How well prepared are the students entering CS in your university?; and
- What actions, if any, is your university taking to mitigate the decrease in enrollment?

We followed with a discussion regarding the personal views of the interview participants about the decline in enrollment.

We interviewed universities until all regions in Canada were represented and the interviews became repetitive as to the information they were able to disclose.

We also interviewed five large industry partners—Business Objects, CIO of Quebec, IBM, Research In Motion, and SG1—for their views on the state of the Canadian ICT sector; the interviews were secured through a list we obtained from Industry Canada as the sponsor of the study. We asked the industry partners whether they are seeing any shortage in the ICT labor market that might be attributed to the enrollment decline and whether they had plans to outsource work as a result. An open discussion again followed the questions.

More important than explanations of past enrollment decreases is the design and articulation of initiatives that might reverse the trend. If the decline is cyclic (see Figure 1), as some of our colleagues in a number of North American universities suggest, then taking immediate steps to reengage students in CS can shorten the duration of the current trough, hasten recovery, and lessen the depth and impact of future troughs. If the decline is not cyclic, as we believe it is, then action (such as inclusion of the societal impact of computing, promotion of the discipline, and reexamina-

tion of CS curricula in light of the current economic and social environment) is necessary to rejuvenate CS enrollment throughout North American universities.

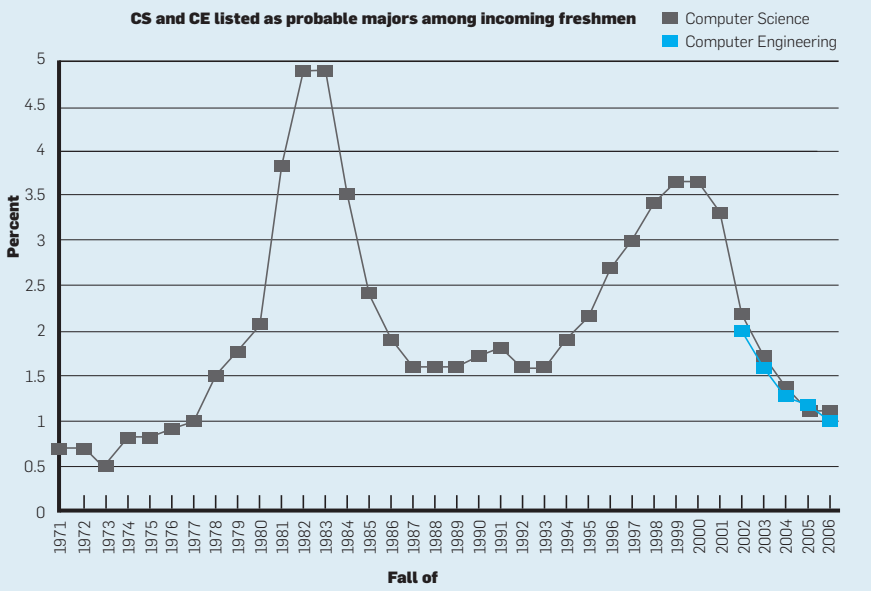
Enrollment Status

While we focus on data from Canadian universities, enrollment data from the Computing Research Association's Taulbee reports suggest common CS enrollment trends in both Canada and the U.S.[1,6,7] We studied enrollment figures available from Statistics Canada[4] that were normalized to report full-time equivalent students; part-time students were counted as a fraction of an enrollment of a full-time student. We augmented the data sets with (non-normalized) enrollment figures from 32 major Canadian universities (1999–2007) provided by the 32 affiliated CS departments (see Figure 2).

We were surprised by the range of enrollment declines across the various regions of Canada. The largest was in Atlantic Canada, with a drop to 37% of its 2001–2002 peak. British Columbia was the exception, at just over 97% of its 2001–2002 peak in 2006–2007. The remaining regions were in between: Ontario and Quebec were between 50% and 60% of their peaks, and the three prairie provinces—Alberta, Manitoba, and Saskatchewan—were near 65% of their peak. These declines are resulting in much smaller graduating classes through at least 2011, thereby adding pressure on hiring in the ICT sector.

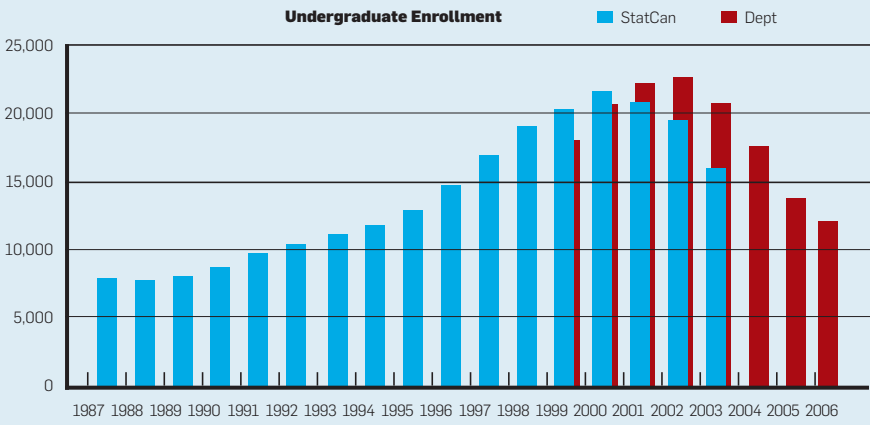
Universities in British Columbia attribute part of their success maintaining CS enrollment levels to several actions: First, the University of British Columbia introduced a new multidisciplinary program within its Faculty of Science that added almost 100 students to CS. Previously, in the 1990s, the same university also began to emphasize recruitment of women students through the Supporting Women in Information Technology program (www.cs.ubc.ca/labs/swift/) and Alternate Routes to Computing program (www.arc.cs.ubc.ca/) that attracted individuals with prior learning to CS. These programs have given the university the highest percentage of women pursuing IT degrees in Canada, ranging from 23% to 26% of 850–1,000 students from 2000 to 2006. Second, the province of British Columbia restructured its university landscape, producing a one-time influx of 200 students from a previously uncounted source in the Statistics Canada data.

Figure 1: Probable CS and CE majors among incoming freshmen at the University of California, Los Angeles (from Vegso[7]).



Source: HERI at UCLA

Figure 2: CS bachelor's enrollment in Canadian universities; source: Statistics Canada via ICT-SITT Industry Canada presentation (blue) and department data (red).



Many institutions have compensated for declining undergraduate enrollment by reducing admission standards and/or increasing the size of their master's and Ph.D. programs (see Figure 3). However, the master's programs are also now beginning to see decreased enrollment as graduating undergraduate classes shrink. Canadian undergraduate graduates declined from 4,900 in 2003–2004 to 3,300 in 2006–2007 and have continued to decline.

The reported explanations for this decrease are not new. The CRA has had a group of CS chairs and deans examining the issue since 2000. The people we interviewed echoed the reasons cited in reports from Australia[2] and Western Europe.[5] The interviews identified common themes across Canada, along with the occasional distinguishing reason specific to a region. We categorize the commonly cited reasons as perception, preparation, and curriculum design

and report here the ideas raised in more than 50% of our university interviews.

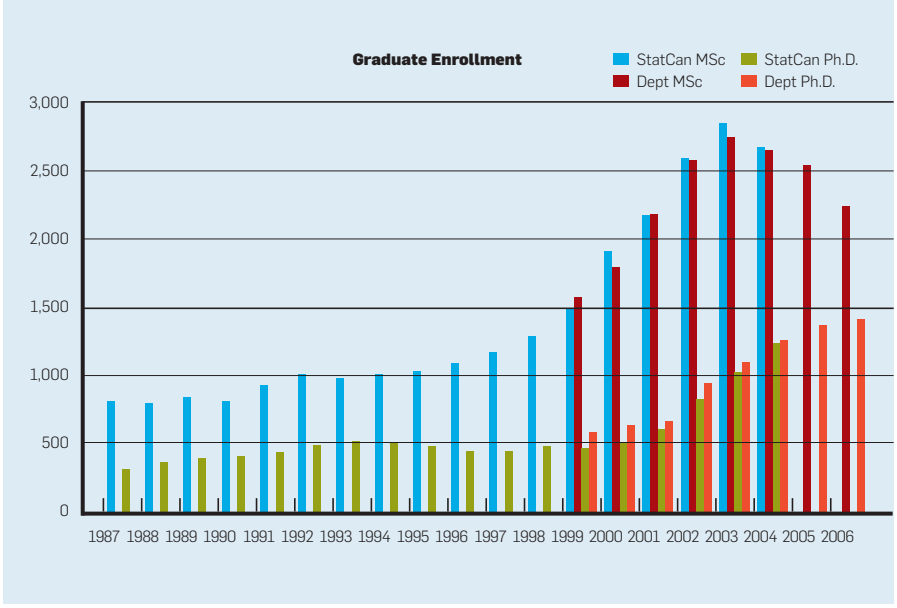
A negative view of CS, typically false, arose in a number of contexts:

- Unhealthy ICT sector in the face of outsourcing;
- Fewer opportunities and lower salaries in the ICT sector;
- Narrow view of the types of problems addressed in the discipline;
- Lack of recognition of the breadth of CS application areas;
- Requirement of, and emphasis on, the discipline's mathematical sophistication;
- Stereotypes of the discipline's students; and
- Earlier CS frontiers people now take for granted.

The relative ease of use and ubiquity of general-purpose computing also mask the challenges that have yet to be solved in CS. As computers have become commonplace, students are increasingly less motivated to study them as a discipline.

Negative perceptions of the discipline and the focus on machines rather than on people have been particularly detrimental in attracting women to the field, so student recruitment is effectively being drawn from significantly less than the full high-school population.

Figure 3: CS master's (blue, green) and Ph.D. (red, orange) enrollment; source: Statistics Canada via ICT-SITT Industry Canada presentation (blue, green) and department data (red, orange).



First-year students are often cited as ill-prepared for the mathematically and science-based approaches common in computing in the majority of CS university programs. Explanations for this lack of preparation include a lack of opportunity to be introduced to the discipline in high school; a shortage of teachers with an adequate understanding of CS; a dearth of computing infrastructure in Canada's high schools; and overwhelmed guidance counselors unable to keep up with the ever-changing discipline or offer comprehensive guidance about IT careers and options.

CS programs in North American universities often exhibit low undergraduate retention rates from the first to the second year. Whatever the reason—incorrect perception of CS, lack of preparation, or loss of interest due to the curriculum—the low retention rate fosters the perception that the discipline is difficult and requires even higher student recruitment numbers.

Finally, CS curricula in North American universities have not adapted to the incoming students or to changes in the industry; CS curricula go stale quickly, and specific topics can be seen by potential students as irrelevant. First-year courses often focus on teaching a computer language to address toy problems like temperature conversion, elevator operations, and simple report generation. The curricula emphasize mathematical accomplishment and typically develop from first principles. Also, Canadian universities often lack mandatory exposure to technology across all their academic programs, leaving fewer

PHOTOGRAPH BY PHILLIP CHEE

students exposed to computing as a discipline on its own or as a first-class contributor to or enabler of multidisciplinary work. We elaborate on the data and the causes of declining Canadian CS enrollment in a separate report.[3]

Looking Ahead

Canadian universities have reacted to declining CS enrollment through recruitment programs, a new area for CS, along with innovative educational programs. These efforts include outreach, variations on first-year courses, joint programs with other disciplines, special adaptation to teaching and learning environments, and more focus on recruiting and retaining students. However, they have not been sufficient. Needed still is a concerted and coordinated Canadian national action plan among the public, private, and education sectors.

An October 2007 IBM Centre for Advanced Studies conference workshop (www-927.ibm.com/ibm/cas/cascon_main/) provided a brainstorming opportunity for 100 industry partners (organized into 10 groups) from throughout Canada, federal and provincial government leaders, CS professors and department chairs, and high school educators. Just as there is no single explanation for declining enrollment, there is no single remedy. The workshop identified the responsibilities all participants needed to accept to restore CS enrollment.

Canada's 10 provinces and three territories have constitutional responsibility for education at all levels, while the federal government focuses on the national health of industry sectors. Thus, federal and provincial responses to boost CS enrollment must differ. The brainstorming workshop (morning session) produced a "top 10" list of directions that might help mitigate the decline:

- Change the high-school curriculum and pedagogy;
- Change the university curriculum and pedagogy;
- Use and promote co-op internships and role models more broadly;
- Improve students' perceptions of CS as a discipline;
- Develop a national advertising and awareness campaign;
- Engage a national society to lobby Canadian federal and provincial governments and function as a repository of teaching resources;
- Win the commitment of the provincial ministers of education to address the specific challenges in CS education at a joint meeting of ministers;
- Change university admission policies to recognize CS as a formal part of the academic high school curriculum;
- Recruit and retain more students in CS ; and
- Leverage the interest of high school and university students already in CS to develop new and engaging programs.

All must focus on changing the public's perception of the industry and its employment opportunities and adapt the CS curricula to remain engaging and relevant for all Canadian students.

The afternoon session developed action plans for each item in each sector:

Private sector. The private sector must promote the demand, availability, and diversity of employment opportunities. It also must set the standards for salaries that, in part, fueled the earlier rush to ICT by some students. These factors make it the most logical and effective sector to spearhead promotion of ICT to the public. The private sector must also help fund the activities that are needed to promote ICT, requiring financial, in-kind, and time commitments from industry partners. The private sector also includes the role models most able to demonstrate success in the ICT sector to which students and their parents can relate and aspire.

The curricular advice from industry partners must also emphasize the interest of the ICT sector as a whole rather than as specific to individual companies or specializations. These partners must help introduce new models of education that leverage and value the hands-on experience available through centers like the IBM Centre for Advanced Studies (www-927.ibm.com/ibm/cas/) and the Business Objects (an SAP company) Advanced Academic Research Centre (labs.businessobjects.com/arc/).

Public sector. As with the private sector, the public sector must also support the activities that promote ICT, especially at the federal level, whether through financial support, in-kind contributions, or information. The federal government, through the Natural Sciences and Engineering Research Council of Canada (www.nserc.gc.ca/index.htm) and other research agencies, must also increase its funding for CS research to create and sustain dynamic projects

that will attract more graduate students to the discipline.

The provincial ministries of education must provide a consistent and relevant curriculum to primary and secondary schools, recognize CS as a teachable academic subject in all 10 provinces, and promote and support the continued development of CS teachers in an area that is always changing.

The federal and provincial governments must also help communicate and promote the number and diversity of employment opportunities in the ICT sector, providing incentives for students to study the discipline as a cornerstone of the national and international knowledge economy.

Universities. Universities must adapt their curricula to engage and retain new students and provide multidisciplinary programs that are attractive to students and industry alike. Moreover, they must create an environment and teaching style that is more appealing to women, introducing new notions of creativity into their programs, and improve retention rates through better engagement of students.

The very structure of university CS programs must also change. Foundation CS courses in the first and second years must capture the breadth of CS; meanwhile, specializations (in both university programs and industry) rely on only a subset of the core, branding the unreferenced material irrelevant. Also, many first-year CS programs assume that students lack a worthwhile computing background on which to build. Universities must rethink what they expect from their core courses and simplify advanced placement in their programs.

Universities are positioned to provide role models (including students, teachers, and practitioners) at the leading edge of technology. They must be excited about the discipline and find ways to convey to their audience the exciting directions in which it is heading. They must also be able to connect their audience with how work in the ICT sector affects the social and economic aspects of daily life.

Universities must also adapt their own regulations to recognize the importance of CS in the competitive global economy by, say, including a beginning course in all university programs, transfer credits or advanced placement into CS programs based on high-school CS courses, and basic CS concepts in all degree requirements, even the humanities. They must therefore revise their silo-based organization to facilitate and encourage multidisciplinary education and programs.

High schools. The high school partners across the Canadian educational system preparing students to consider ICT as a possible career path could now take several new directions: First, they could offer CS courses that introduce students to computing concepts beyond the applications in the students' own studies (some already do this) but must provide consistency and continual updates. Delivery of the material, as well as the resources for teaching it, must engage the students. The high schools, and likely the primary schools, must also find ways to keep women connected with all the sciences as they reach grade 12.

Conclusion

The products and services of the ICT sector represent a cornerstone of the developing global knowledge society. ICT is one factor increasing the productivity of the Canadian work force; ICT must continue to strengthen this productivity if Canadian businesses are to remain competitive. Sustained success in the ICT sector, essential to Canada's national interest and prosperity, requires that all participants—public, private, and educational—continually adapt to changes in technology and its applications. This requires a Canadian national work force that is dedicated, informed, sophisticated, and agile.

For the sake of the overall Canadian economy, it is imperative that the country's public, private, and educational sectors all play a role in increasing the numbers of students studying CS in high school and university in order to sustain the required ICT work force. All must focus on changing the public's perception of the industry and its employment opportunities and adapt the CS curricula to remain engaging and relevant for all Canadian students.

All must invest time and money in solving the current crisis of falling Canadian CS enrollment. Failure to do less than what we propose here would harm future generations. That investment must include stable, long-term financial support for CS education initiatives that recognize the time commitment required of the leaders in all sectors.

Acknowledgments

We would like to thank the people who helped us prepare this article:

- The chairs of Canadian university CS departments for their participation, openness, and feedback in discussing and quantifying the issues facing the Canadian ICT sector;
- The industry partners who met with us for our on-site interviews concerning their perspective on the direction of the discipline;
- Paul Swinwood of the Information and Communications Technology Council (www.ictc-ctic.ca/) for helping sponsor our work;
- Business Objects, an SAP company, IBM, and Research In Motion for providing additional industrial contacts; and
- The Natural Sciences and Engineering Research Council of Canada (www.nserc.gc.ca/index.htm). C

References

1. Computing Research Association. Taulbee Reports, Washington, D.C., 2006; www.cra.org/statistics.
2. Dobson, I.R. *The IT Education Bubble: An Analysis of University Student Statistics 2002–2005. Australian Council of Deans of Science, Australia, 2007*; www.educationalpolicy.org/pdf/DeansOfSci-IT-2_9-07.pdf.
3. Slonim, J., Scully, S., and McAllister, M. *Outlook on Enrollments in Computer Science in Canadian Universities.* Information and Communication Technology Council, 2008; www.ictc-ctic-ca/uploadedFiles/Labour_Market_Intelligence/Outlook_on_enrolments.pdf.
4. Statistics Canada. *Enhanced Student Information System.* 2006; stds.statcan.ca/English/cip/cip_4digit.asp?code=11.
5. Van Leeuwen, J. and Tanca, L. *Student Enrollment and Image of the Informatics Discipline.* Utrecht University, Technical Report No. UU-CS-2007-024. Utrecht, The Netherlands, 2007; www.cs.uu.nl/research/techreps/repo/CS-2007/2007-024.pdf.
6. Vegso, J. *Enrollments and Degree Production at U.S. CS Departments Drop Further in 2006–2007.* Computing Research Association, Washington, D.C., 2008; www.cra.org/CRN/articles/march08/jvegso_enrollments.html.
7. Vegso, J. *Low Interest in CS and CE Among Incoming Freshmen.* Computing Research Association, Washington, D.C., 2007; www.cra.org/wp/index.php?p=104.

Jacob Slonim (slonim@cs.dal.ca) is a professor in the Faculty of Computer Science of Dalhousie University, Halifax, Nova Scotia, Canada.

Sam Scully (sjscully@sympatico.ca) is the former Vice-President Academic and Provost and Professor of Classics in Dalhousie University, Halifax, Nova Scotia, Canada.

Michael McAllister (mcallist@cs.dal.ca) is an associate professor in the Faculty of Computer Science of Dalhousie University, Halifax, Nova Scotia, Canada.

Symposium on Computer Human Interaction for Management of Information Technology

CHIMIT 08

A Turning Point in IT

November 14/15, 2008 | San Diego, CA

CHIMIT is the leading forum for discussing topics on IT management with a focus on people, business, and technology. Join the discussion on issues, solutions, and research drawing upon fields such as human-computer interaction, human factors, computer systems, and management and service sciences.

Workspace Studies
- Ethnographic studies of IT work in context

Processes and Practices
- Development and use of processes in IT

Organizational Knowledge
- Studies of collaboration and coordination

Design
- Design of human-centered IT systems

Tools and Techniques
- Visualizations of complex system behavior

Automation
- Automation/Policy languages
- Human interfaces to automation

Plenary Talks

I Got My Jet Pack and I'm Still Not Happy
David Blank-Edelman, Northeastern

Human-Centered Design: Finding the Sweet Spot Among the Many Stakeholders in the Design of a Complex System
William B. Rouse, Tennenbaum Institute, Georgia Institute of Technology

Panels

Designing for Complexity: New Approaches to System Administration UI's
Leading UI architects and designers from IBM, Microsoft, HP, Salesforce.com, Oracle, and BMC.

October 13
Advance Registration Ends
November 13
Web Registration Ends

In cooperation with acm USENIX

www.chimit08.org

Microsoft hp invent IBM SIGCHI

DOI:10.1145/1400181.1400200

Natural computing builds a bridge between computer science and natural sciences.

BY LILA KARI AND GRZEGORZ ROZENBERG

The Many Facets of Natural Computing

"Biology and computer science—life and computation—are related. I am confident that at their interface great discoveries await those who seek them."

—Leonard Adleman, Scientific American, Aug. 1998

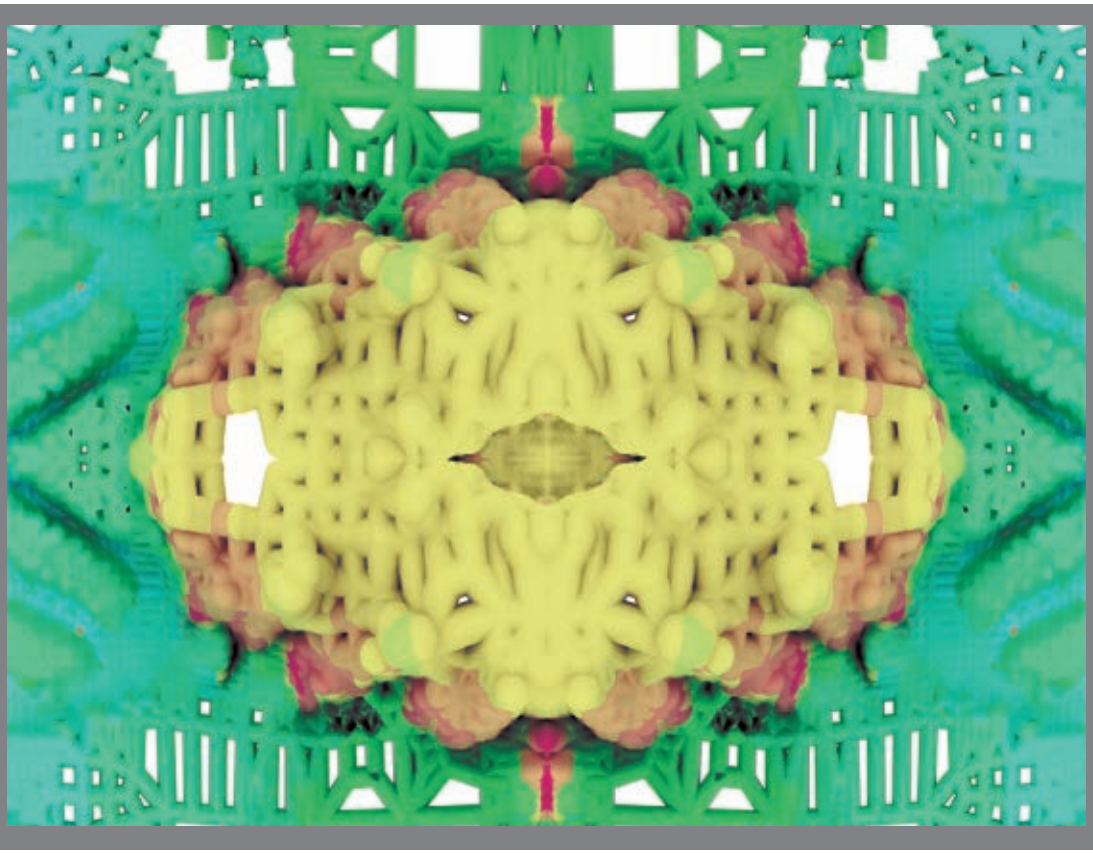
Natural computing is the field of research that investigates models and computational techniques inspired by nature and, dually, attempts to understand the world around us in terms of information processing. It is a highly interdisciplinary field that connects the natural sciences with computing science, both at the level of information technology and at the level of fundamental research.[33] As a matter of fact, natural computing areas and topics come in many flavors, including pure theoretical research, algorithms and software applications, as well as biology, chemistry, and physics experimental laboratory research.

In this review we describe computing paradigms abstracted from natural phenomena as diverse as self-reproduction, the functioning of the brain, Darwinian evolution, group behavior, the immune system, the characteristics of life, cell membranes, and morphogenesis. These paradigms can be implemented either on traditional electronic hardware or on alternative physical media such as biomolecular (DNA, RNA) computing, or trapped-ion quantum computing devices. Dually, we describe several natural processes that can be viewed as information processing, such as gene regulatory networks, protein-protein interaction networks, biological transport networks, and gene assembly in unicellular organisms. In the same vein, we list efforts to understand biological systems by engineering semi-synthetic organisms, and to understand the universe from the point of view of information processing.

This review was written with the expectation that the reader is a computer scientist with limited knowledge of natural sciences, and it avoids dwelling on the minute details of various natural phenomena. Thus, rather than being overwhelmed by particulars, it is our hope that readers see this article as simply a window into the profound relationship that exists between nature and computation.

There *is* information processing in nature, and the natural sciences are already adapting by incorporating tools and concepts from computer science at a rapid pace. Conversely, a closer look at nature from the point of view of information processing *can* and *will*

The vivid images peppered throughout this story offer glimpses of what can happen when nature, art, and computer science join forces. While not directly referenced in this article, these images serve to offer readers some startling perspectives of nature up close as only technology can provide.



Neri Oxman, an architect and researcher currently working for her Ph.D. in design and computation at MIT, formed an interdisciplinary research initiative called Materialecology that undertakes design research in the intersection between architecture, engineering, computation, biology and ecology. Here, she illustrates how plants often grow in fashion to maximize the surface area of their branching geometries while maintaining structural support. This work was done in collaboration with W. Craig Carter, professor of Materials Science and Engineering, at the MIT Media Lab and the MIT Computation Group. For more images, see http://www.materialecology.com/.

change what we mean by computation. Our invitation to you, fellow computer scientists, is to take part in the uncovering of this wondrous connection.[a]

Nature as Inspiration

Among the oldest examples of nature-inspired models of computation are the *cellular automata* conceived by Ulam and von Neumann in the 1940s. John von Neumann, who was trained in both mathematics and chemistry, investigated cellular automata as a framework for the understanding of the behavior of complex systems. In particular, he believed that self-reproduction was a feature essential to both biological organisms and computers.[40]

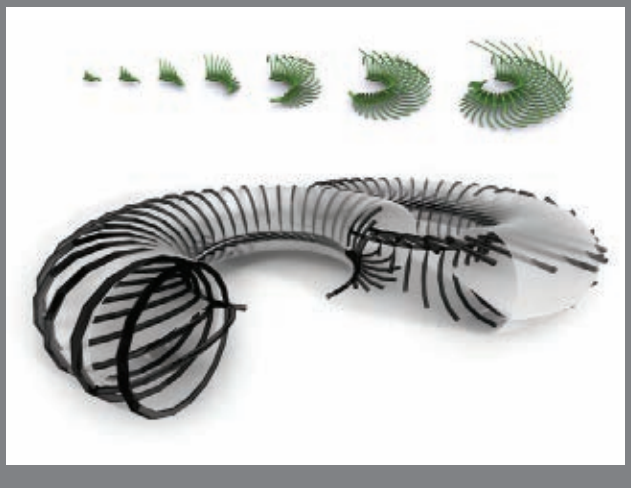
A cellular automaton is a dynamical system consisting of a regular grid of cells, in which space and time are discrete. Each of the cells can be in one of a finite number of states. Each cell changes its state according to a list of given transition rules that determine its future state, based on its current state and the current states of some of its neighbors. The entire grid of cells updates its configuration synchronously according to the *a priori* given transition rules.

Cellular automata have been applied to the study of phenomena as diverse as communication, computation, construction, growth, reproduction, competition, and evolution. One of the best known examples of cellular automata—the "game of life" invented by Conway—was shown to be computationally universal. Cellular automata have been extensively studied as an alternative explanation to the phenomenon of emergence of complexity in the natural world, and used, among others, for modeling in physics and biology.

In parallel to early comparisons[39] between computing machines and the human nervous system, McCulloch and Pitts proposed the first model of artificial neurons. This research eventually gave rise to the field of *neural computation*, and it also had a profound influence on the foundations of automata theory. The goal of neural computation was twofold. On one hand, it was hoped that it would help unravel the structure of computation in nervous systems of living organisms (How does the brain work?). On the other hand, it was predicted that, by using the principles of how the human brain processes information, neural computation would yield significant computational advances (How can we build an intelligent computer?). The first goal has been pursued mainly within the neurosciences under the name of brain theory or computational neuroscience, while the quest for the second goal has become mainly a computer science discipline known as artificial neural networks or simply *neural networks*.[5]

a A few words are in order about the organization of this article. The classifications and labels we use for various fields of research are purely for the purpose of organizing the discourse. In reality, far from being clear-cut, many of the fields of research mentioned here overlap, or fit under more than one category. The general audience for whom this article is intended, our respective fields of expertise, and especially the limited space available for this review affected both the depth and breadth of our exposition. In particular, we did not discuss some fields of research that have large overlaps with natural computing, such as bioinformatics, computational molecular biology, and their roles in, for example, genomics and proteomics. In addition, our explanations of various aspects, themes, and paradigms had to be necessarily oversimplified. As well, the space we devoted to various fields and topics was influenced by several factors and, as such, has no relation to the respective importance of the field or the relative size of the body of research in that field.

An artificial neural network consists of interconnected artificial neurons.[31] Modeled after the natural neurons, each artificial neuron A has n real-valued inputs, $x_1, x_2, \ldots, x_n$, and it computes its own *primitive function* f_A as follows. Usually, the inputs have associated weights, $w_1, w_2, \ldots, w_n$. Upon receiving the n inputs, the artificial neuron A produces the output $f_A(w_1x_1 + w_2x_2 + \ldots + w_nx_n)$. An artificial neural network is a network of such neurons, and thus a network of their respective primitive functions. Some neurons are selected to be the output neurons, and the *network function* is a vectorial function that, for n input values, associates the outputs of the m output neurons. Note that different selections of the weights produce different network functions for the same inputs. Based on given input-output pairs, the network can "learn" the weights $w_1, \ldots, w_n$. Thus, there are three important features of any artificial neural network: the primitive function of each neuron, the topology of the network, and the learning algorithm used to find the weights of the network. One of the many examples of such learning algorithms is the "backwards propagation of errors." Back-propagation is a supervised learning method by which the weights of the connections in the network are repeatedly adjusted so as to minimize the difference between the actual output vector of the net and the desired output vector. Artificial neural networks have proved to be a fruitful paradigm, leading to successful novel applications in both new and established application areas.



From Archimorph, where work is continuing on their L-System and Evolutionary Algorithm, including new images of L-Systems growths as well as diagrams explaining the process of the overall design. For more images, see archimorph.wordpress.com/.

While Turing and von Neumann dreamed of understanding the brain, and possibly designing an intelligent computer that works like the brain, *evolutionary computation*[6] emerged as another computation paradigm that drew its inspiration from a completely different part of biology: Darwinian evolution. Rather than emulating features of a single biological organism, evolutionary computation draws its inspiration from the dynamics of an entire species of organisms. An artificial evolutionary system is a computational system based on the notion of simulated evolution. It features a constant- or variable-size population of individuals, a fitness criterion according to which the individuals of the population are being evaluated, and genetically inspired operators that produce the next generation from the current one. In an evolutionary system, the initial population of individuals is generated at random or heuristically. At each evolutionary step, the individuals are evaluated according to a given fitness function. To form the next generation, offspring are first generated from selected individuals by using operators such as mutation of a parent, or recombination of pairs or larger subsets of parents. The choice of parents for recombination can be guided by a fitness-based selection operator, thus reflecting the biological principle of mate selection. Secondly, individuals of the next generation are selected from the set of newly created offspring, sometimes also including the old parents, according to their fitness—a process reflecting the biological concept of environmental selection.

Evolutionary systems have first been viewed as optimization processes in the 1930s. The basic idea of viewing evolution as a computational process gained momentum in the 1960s, and evolved along three main branches.[13] *Evolution strategies* use evolutionary processes to solve parameter optimization problems, and are today used for real-valued as well as discrete and mixed types of parameters. *Evolutionary programming* originally aimed at achieving the goals of artificial intelligence via evolutionary techniques, namely by evolving populations of intelligent agents modeled, for example, as finite-state machines. Today, these algorithms are also often used for real-valued parameter optimization problems. *Genetic algorithms* originally featured a population of individuals encoded as fixed-length bit strings, wherein mutations consisted of bit-flips according to a typically small, uniform mutation rate, the recombination of two parents consisted of a cut-and-paste of a prefix of one parent with a suffix of the other, and the fitness function was problem-dependent. If the initial individuals were to encode possible solutions to a given problem, and the fitness function were designed to measure the optimality of a candidate solution, then such a system would, in time, evolve to produce a near-optimal solution to the initial problem. Today, genetic algorithms are also modified heavily for applications to real-valued parameter optimization problems as well as many types of combinatorial tasks such as, for example, permutation-based problems. As another application, if the individuals were computer programs, then the genetic algorithm technique would result in "the fittest" computer programs, as is the goal of *genetic programming*.[22]

Cellular automata, neural computation, and evolutionary computation are the most established "classical" areas of natural computing. Several other bio-inspired paradigms emerged more recently, among them swarm intelligence, artificial immune systems, artificial life, membrane computing, and amorphous computing.

A computational paradigm straddling at times evolutionary computation and neural computation is *swarm*

intelligence.[16] A swarm is a group of mobile biological organisms (such as bacteria, ants, termites, bees, spiders, fish, birds) wherein each individual communicates with others either directly or indirectly by acting on its local environment. These interactions contribute to distributive collective problem solving. Swarm intelligence, sometimes referred to as *collective intelligence*, is defined as the problem-solving behavior that emerges from the interaction of such a collection of individual agents. For example, in research simulating flocking behavior, each individual was endowed with three simple possible behaviors: to act as to avoid collision, to match velocity with neighbors, and to stay close to nearby flock mates. The simulations showed that flocking was an emergent behavior that arose from the interaction of these simple rules.

Particle swarm optimization was introduced as a new approach to optimization that had developed from simple models of social interactions, as well as of flocking behavior in birds and other organisms. A particle swarm optimization algorithm starts with a swarm of "particles," each representing a potential solution to a problem, similar to the population of individuals in evolutionary computation.

Particles move through a multidimensional search space and their positions are updated according to their own experience and that of their neighbors, by adding "velocity" to their current positions. The velocity of a particle depends on its previous velocity (the "inertia" component), the tendency towards the past personal best position (the cognitive, "nostalgia" component), and the move toward a global or local neighborhood best (the "social" component). The cumulative effect is that each particle converges towards a point between the global best and its personal best. Particle Swarm Optimization algorithms have been used to solve various optimization problems, and have been applied to unsupervised learning, game learning, scheduling and planning applications, and design applications.

Ant algorithms were introduced to model the foraging behavior of ant colonies. In finding the best path between their nest and a source of food, ants rely on indirect communication by laying a pheromone trail on the way back to the nest if they found food, and following the concentration of pheromones in the environment if they are looking for food. This foraging behavior has inspired a large number of ant algorithms used to solve mainly combinatorial optimization problems defined over discrete search spaces.

A closer look at nature from the point of view of information processing *can* and *will* change what we mean by computation. Our invitation to you, fellow computer scientists, is to take part in the uncovering of this wondrous connection.

Artificial immune systems are computational systems devised starting in the late 1980s and early 1990s as computationally interesting abstractions of the natural immune system of biological organisms. Viewed as an information processing system, the immune system performs many complex computations in a highly parallel and distributed fashion.[11] It uses learning, memory, associative retrieval, and other mechanisms to solve recognition and classification problems such as distinction between self and nonself cells, and neutralization of nonself pathogenic agents. Indeed, the natural immune system has sometimes been called the "second brain" because of its powerful information processing capabilities.

The natural immune system's main function is to protect our bodies against the constant attack of external pathogens (viruses, bacteria, fungi, and parasites). The main role of the immune system is to recognize cells in the body and categorize them as self or nonself.[12] There are two parts of the immune system: innate (non-specific) and adaptive (acquired). The cells of the innate immune system are immediately available to combat against a wide variety of antigens, without requiring previous exposure to them. These cells possess the ability of ingesting and digesting several "known" pathogens. In contrast, the adaptive immune response is the antibody production in response to a specific new infectious agent. Our body maintains a large "combinatorial database" of immune cells that circulate throughout the body. When a foreign antigen invades the body, only a few of these immune cells can detect the invaders and physically bind to them. This detection triggers the primary immune response: the generation of a large population of cells that produce matching antibodies that aid in the destruction or neutralization of the antigen. The immune system also retains some of these specific-anti-

body-producing cells in immunological memory, so that any subsequent exposure to a similar antigen can lead to a rapid, and thus more effective, immune response (secondary response).

The computational aspects of the immune system, such as distinguishing of self from nonself, feature extraction, learning, memory, self-regulation, and fault tolerance, have been exploited in the design of artificial immune systems that have been successfully used in applications. The applications are varied and include computer virus detection, anomaly detection in a time series of data, fault diagnosis, pattern recognition, machine learning, bioinformatics, optimization, robotics, and control. Recent research in immunology departs from the self-nonself discrimination model to develop what is known as the "danger theory," wherein it is believed that the immune system differentiates between dangerous and non-dangerous entities, regardless of whether they belong to self or to nonself. These ideas have started to be exploited in artificial immune systems in the context of computer security.

While artificial immune systems (a.k.a. immunological computation, immunocomputing) constitute an example of a computational paradigm inspired by a very specific subsystem of a biological organism, artificial life takes the opposite approach. *Artificial life* (*ALife*) attempts to understand the very essence of what it means to be alive by building *ab initio*, within *in silico* computers and other "artificial" media, artificial systems that exhibit properties normally associated only with living organisms.[24] *Lindenmayer systems* (*L-systems*), introduced in 1968, can be considered as an early example of artificial life.

L-systems are parallel rewriting systems that, starting with an initial word, proceed by applying rewriting rules in parallel to all the letters of the word, and thus generate new words.[34] They have been most famously used to model plant growth and development,[29] but also for modeling the morphology of other organisms.

Building on the ideas of evolutionary computation, other pioneers of artificial life experimented with evolving populations of "artificial creatures" in simulated environments.[9] One example was the design[36] of evolving virtual block creatures that were selected for their ability to swim (or walk, or jump), and that competed for a common resource (controlling a cube) in a physically simulated world endowed with realistic features such as kinematics, dynamics, gravity, collisions, and friction. The result was that creatures evolved which would extend arms towards the cube, while others would crawl or roll to reach it, and some even developed legs that they used to walk towards the cube. These ideas were taken one step further[25] by combining the computational and experimental approaches, and using rapid manufacturing technology to fabricate physical robots that were materializations of their virtually evolved computational counterparts. In spite of the simplicity of the task at hand (horizontal locomotion), surprisingly different and complex robots evolved: many of them exhibited symmetry, some moved sideways in a crab-like fashion, and others crawled on two evolved limbs. This marked the emergence of mechanical artificial life, while the nascent field of synthetic biology, discussed later, explores a biological implementation of similar ideas. At the same time, the field of Artificial Life continues to explore directions such as artificial chemistry (abstractions of natural molecular processes), as well as traditionally biological phenomena in artificial systems, ranging from computational processes such as co-evolutionary adaptation and development, to physical processes such as growth, self-replication, and self-repair.

While artificial immune systems constitute an example of a computational paradigm inspired by a very specific subsystem of a biological organism, artificial life attempts to understand the very essence of what it means to be alive.

Membrane computing investigates computing models abstracted from the structure and the functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures.[26] More specifically, the feature of the living cells that is abstracted by membrane computing is their compartmentalized internal structure effected by membranes. A generic membrane system is essentially a nested hierarchical structure of cell-like compartments or regions, delimited by "membranes." The entire system is enclosed in an external membrane, called the skin membrane, and everything outside the skin membrane is considered to be the environment.

Each membrane-enveloped region contains objects and transformation rules which modify these objects, as well as specify whether they will be transferred outside or stay inside the region. The transfer thus provides for communication between regions. Various formal mechanisms were developed that reflect the selective manner in which biological membranes allow molecules to pass through them.

Another biologically inspired feature of membrane systems as mathematical constructs is the fact that, instead of dealing with sets of objects, one uses multisets wherein one keeps track of the multiplicity of each object. The computational behavior of a membrane system starts with an initial input configuration and proceeds in a maximally parallel manner by the nondeterministic choice of application of the transformation rules, as well as of the objects to which they are to be applied. The output of the computation is then collected from an a priori determined output membrane. Next to the basic features indicated previously, many alternatives of membrane systems have been considered, among them ones that allow for membranes to be dissolved and created. Typical applications of membrane systems include biology (modeling photosynthesis and certain signaling pathways, quorum sensing in bacteria, modeling cell-mediated immunity), computer science (computer graphics, public-key cryptography, approximation and sorting algorithms, and solving computationally hard problems), and linguistics.

Amorphous computing is a paradigm that draws inspiration from the development of form (morphogenesis) in biological organisms, wherein interactions of cells guided by a genetic program give rise to well-defined shapes and functional structures. Analogously, an amorphous computing medium comprises a multitude of irregularly placed, asynchronous, locally interacting computing elements.[1] These identically programmed "computational particles" communicate only with particles situated within a small given radius, and may give rise to certain shapes and patterns such as, for example, any pre-specified planar graph. The goal of amorphous computing is to engineer specified coherent computational behaviors from the interaction of large quantities of such unreliable computational particles interconnected in unknown, irregular, and time-varying ways. At the same time, the emphasis is on devising new programming abstractions that would work well for amorphous computing environments. Amorphous computing has been used both as a programming paradigm using traditional hardware, and as the basis for "cellular computing," discussed later, under the topics synthetic biology, and computation in living cells.

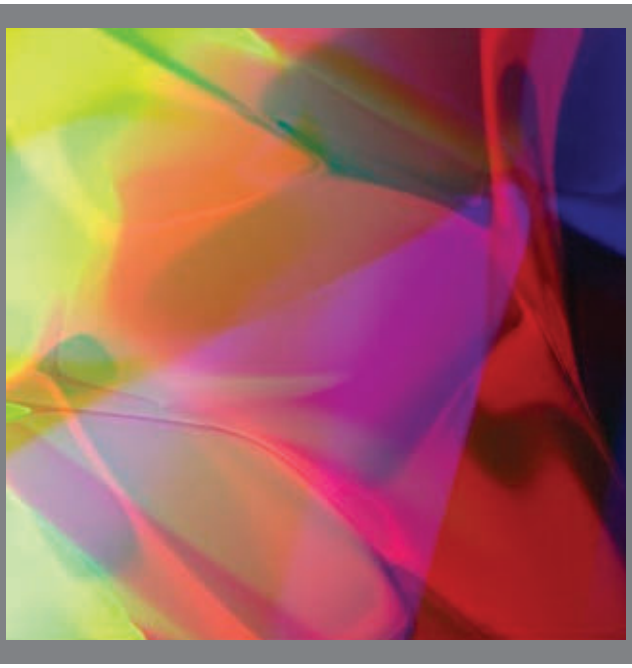
Nature as Implementation Substrate

In the preceding section we saw cellular automata inspired by self-reproduction, neural computation by the functioning of the brain, evolutionary computation by the Darwinian evolution of species, swarm intelligence by the behavior of groups of organisms, artificial immune systems by the natural immune system, artificial life by properties of life in general, membrane computing by the compartmentalized organization of the cells, and amorphous computing by morphogenesis. All these are computational techniques that, while inspired by nature, have been implemented until now mostly on traditional electronic hardware. An entirely distinct category is that of computing paradigms that use a radically different type of "hardware." This category includes molecular computing and quantum computing.[b]

Molecular computing (known also as biomolecular computing, biocomputing, biochemical computing, DNA computing), is based on the idea that data can be encoded as biomolecules — such as DNA strands — and molecular biology tools can be used to transform this data to perform, for example, arithmetic or logic operations. The birth of this field was the 1994 breakthrough experiment by Leonard Adleman who solved a small instance of the Hamiltonian Path Problem solely by manipulating DNA strands in test tubes.[2]

DNA (deoxyribonucleic acid) is a linear chain made up of four different types of nucleotides, each consisting of a base (Adenine, Cytosine, Guanine, or Thymine) and a sugar-phosphate unit. The sugar-phosphate units are linked together by covalent bonds to

b There are several research areas that, because of the limited space, we could not discuss here. Thus, for example, non-classical, unconventional computation[38] focuses on carefully examining and possibly breaking the classical (Turing, von Neumann) computation assumptions, and developing a more general science of computation. A substantial part of this research is concerned with implementing computation on new physical substrates, exploiting in this way computational properties of various physical, chemical, and biological media. A majority of this research is entwined with, and motivated by, natural computing.



McGill University's Laboratory for Natural and Simulated Cognition (LNSC) investigates human cognition through a combination of psychological and computational approaches. Using the Cascade-correlation algorithm, LNSC researchers created a program that outputs a 2D display of random output values of neural networks. The results are sometimes quite phenomenal and artistic. For more, see www.psych.mcgill.ca/labs/lnsc/.

form the backbone of the DNA single strand. Since nucleotides may differ only by their bases, a DNA strand can be viewed as simply a word over the four-letter alphabet $\{A,C,G,T\}$. A DNA single strand has an orientation, with one end known as the 5′ end, and the other as the 3′ end, based on their chemical properties. By convention, a word over the DNA alphabet represents the corresponding DNA single strand in the 5′ to 3′ orientation, that is, the word GGTTTTT stands for the DNA single strand 5′– GGTTTTT –3′. A crucial feature of DNA single strands is their Watson-Crick complementarity: A is complementary to *T*, *G* is complementary to *C*, and two complementary DNA single strands with opposite orientation bind to each other by hydrogen bonds between their individual bases. In so doing, they form a stable DNA double strand resembling a helical ladder, with the backbones at the outside and the bound pairs of bases lying inside. For example, the DNA single strand 5′– AAAAACC – 3′ will bind to the DNA single strand 5′– GGTTTTT – 3′ to form the 7 base-pair-long (7bp) double strand

5′ – AAAAACC – 3′
3′ – TTTTTGG – 5′

Another molecule that can be used for computation is ribonucleic acid, RNA. While similar to DNA, RNA differs in three main aspects: RNA is usually single-stranded while DNA is usually double-stranded, RNA nucleotides contain the sugar ribose, while DNA nucleotides contain the sugar deoxyribose, and in RNA the nucleotide Uracil, *U*, substitutes for Thymine, which is present in DNA.

There are many possible DNA bio-operations that one can use for computations,[21] such as: cut-and-paste operations achievable by enzymes, synthesizing desired DNA strands up to a certain length, making exponentially many copies of a DNA strand, and reading out the sequence of a DNA strand. These bio-operations and the Watson-Crick complementary binding have all been used to control DNA computations and DNA robotic operations. While initial experiments solved simple instances of computational problems, more recent experiments tackled successfully sophisticated computational problems, such as a 20-variable instance of the 3-Satisfiability-Problem. The efforts toward building an autonomous molecular computer include implementations of computational state transitions with biomolecules, and a DNA implementation of a finite automaton with potential applications to the design of smart drugs.

More importantly, since 1994, research in molecular computing has gained several new dimensions. One of the most significant achievements of molecular computing has been its contribution to the massive stream of research in nanosciences, by providing computational insights into a number of fundamental issues. Perhaps the most notable is its contribution to the understanding of self-assembly, which is among the key concepts in nanosciences.[30] Recent experimental research into programmable molecular-scale devices has produced impressive self-assembled DNA nanostructures[35] such as cubes, octahedra, Sierpinski triangles,[32] DNA origami, or intricate nanostructures that achieve computation such as binary counting, or bit-wise cumulative XOR. Other experiments include the construction of DNA-based logic circuits, and ribozymes that can be used to perform logical operations and simple computations. In addition, an array of ingenious DNA nanomachines[8] were built with potential uses to nanofabrication, engineering, and computation: molecular switches that can be driven between two conformations, DNA "tweezers," DNA "walkers" that can be moved along a track, and autonomous molecular motors.

A significant amount of research in molecular computing has been dedicated to the study of theoretical models of DNA computation and their properties. The model of DNA computing in-

Paul W.K. Rothemund, a senior research associate at California Institute of Technology, has developed a method of creating nanoscale shapes and patterns using DNA. The smiley faces are actually giant DNA complexes called "scaffolded DNA origami." Rothemund notes that while the smiley face shape may appear silly, there is serious science behind it. He hopes to use this DNA origami (and other DNA nanotechnologies) to build smaller, faster computers and devices. For more on his work, visit http://www.dna.caltech.edu/~pwkr/.

troduced by Head, based on splicing (a combination of cut-and-paste operations achievable by enzymes), predates the experimental proof-of-principle of DNA computing by almost 10 years. Subsequently, studies on the computational power of such models proved that various subsets of bio-operations can achieve the computational power of a Turing machine, showing thus that molecular computers are in principle possible.[27] Overall, molecular computing has created many novel theoretical questions, and has considerably enriched the theory of computation.

Quantum Computing is another paradigm that uses an alternative "hardware" for performing computations.[19] Already in 1980 Benioff introduced simulations of classical Turing Machines on quantum mechanical systems. However the idea of a *quantum computer* that would run according to the laws of quantum physics and operate exponentially faster than a deterministic electronic computer to simulate physics, was first suggested by Feynman in 1982. Subsequently, Deutsch introduced a formal model of quantum computing using a Turing machine formalism, and described a universal quantum computer.

A quantum computer uses distinctively quantum mechanical phenomena, such as superposition and entanglement, to perform operations on data stored as quantum bits (qubits). A *qubit* can hold a 1, a 0, or a quantum superposition of these. A quantum computer operates by manipulating those qubits with quantum logic gates. The notion of information is different when studied at the quantum level. For instance, quantum information cannot be measured reliably, and any attempt at measuring it entails an unavoidable and irreversible disturbance.

The 1980s saw an abundance of research in quantum information processing, such as applications to quantum cryptography which, unlike its classical counterpart, is not usually based on the complexity of computation but on the special properties of quantum information. Recently an open air experiment was reported in quantum cryptography (not involving optical cable) over a distance of 144km, conducted between two Canary islands.

It is indeed believed that one of the possible contributions of computer science to biology could be the development of a suitable language to accurately and succinctly describe, and reason about, biological concepts and phenomena.

The theoretical results that catapulted quantum computing to the forefront of computing research were Shor's quantum algorithms for factoring integers and extracting discrete logarithms in polynomial time, obtained in 1994—the same year that saw the first DNA computing experiment by Adleman. A problem where quantum computers were shown to have a quadratic time advantage when compared to classical computers is quantum database search that can be solved by Grover's algorithm. Possible applications of Shor's algorithm include breaking RSA exponentially faster than an electronic computer. This joined other exciting applications, such as quantum teleportation (a technique that transfers a quantum state, but not matter or energy, to an arbitrarily distant location), in sustaining the general interest in quantum information processing.

So far, the theory of quantum computing has been far more developed than the practice. Practical quantum computations use a variety of implementation methods such as ion-traps, superconductors, nuclear magnetic resonance techniques, to name just a few. To date, the largest quantum computing experiment uses liquid state nuclear magnetic resonance quantum information processors that can operate on up to 12 qubits.

Nature as Computation

The preceding sections describe research on the theory, applications and experimental implementations of nature-inspired computational models and techniques. A dual direction of research in natural computing is one in which the main goal becomes understanding nature by viewing processes that take place in nature as information processing.

This dual aspect can be seen in *systems biology*, and especially in *computational systems biology*, wherein the adjective "computational" has two meanings. On one hand it means the use of quantitative algorithms for computations, or simulations that complement experiments in hypothesis generation and validation. On the other hand, it means a qualitative approach that investigates processes taking place in cells through the prism of communications and interactions, and thus of

computations. We shall herein address mostly the second aspect, whereby systems biology aims to understand the complex interactions in biological systems by using an integrative as opposed to a reductionist approach. The reductionist approach to biology tries to identify all the individual components of functional processes that take place in an organism, in such a way that the processes and the interactions between the components can be understood. In contrast, systems biology takes a systemic approach in focusing instead on the interaction networks themselves, and on the properties of the biological systems that arise because of these interaction networks. Hence, for example, at the cell level, scientific research on organic components has focused strongly on four different interdependent interaction networks, based on four different "biochemical toolkits:" nucleic acids (DNA and RNA), proteins, lipids, carbohydrates, and their building blocks (see Cardelli,[10] whose categorization we follow here).

The genome consists of DNA sequences, some of which are genes that can be transcribed into messenger RNA (mRNA), and then translated into proteins according to the genetic code that maps 3-letter DNA segments into amino acids. A protein is a sequence over the 20-letter alphabet of amino acids. Each gene is associated with other DNA segments (promoters, enhancers, or silencers) that act as binding sites for proteins that activate or repress the gene's transcription. Genes interact with each other indirectly, either through their gene products (mRNA, proteins), which can act as transcription factors to regulate gene transcription—either as activators or repressors—or through small RNA species that directly regulate genes.

These gene-gene interactions, together with the genes' interactions with other substances in the cell, form the most basic interaction network of an organism, the *gene regulatory network*. Gene regulatory networks perform information processing tasks within the cell, including the assembly and maintenance of the other networks. Research into modeling gene regulatory networks includes qualitative models such as random and probabilistic Boolean networks, asynchronous automata, and network motifs.

As the natural sciences are rapidly absorbing ideas of information processing, and the meaning of computation is changing as it embraces concepts from the natural sciences, we have the rare privilege to take part in several such metamorphoses.

Another point of view,[20] is that the entire genomic regulatory system can be thought of as a computational system, the "genomic computer." Such a perspective has the potential to yield insights into both computation as humans historically designed it, and computation as it occurs in nature. There are both similarities and significant differences between the genomic computer and an electronic computer. Both perform computations, the genomic computer on a much larger scale. However, in a genomic computer, molecular transport and movement of ions through electrochemical gradients replace wires, causal coordination replaces imposed temporal synchrony, changeable architecture replaces rigid structure, and communication channels are formed on an as-needed basis. Both computers have a passive memory, but the genomic computer does not place it in an *a priori* dedicated and rigidly defined place; in addition, the genomic computer has a dynamic memory in which, for example, trancriptional subcircuits maintain given regulatory states. In a genomic computer robustness is achieved by different means, such as by rigorous selection: non (or poorly)-functional processes are rapidly degraded by various feedback mechanisms or, at the cell level, non (or poorly)-functional cells are rapidly killed by apoptosis, and, at the organism level, non (or poorly)-functional organisms are rapidly out-competed by more fit species. Finally, in the case of a genomic computer, the distinction between hardware and software breaks down: the genomic DNA provides both the hardware and the digital regulatory code (software).

Proteins and their interactions form another interaction network in a cell, that of *biochemical networks*, which perform all mechanical and metabolic tasks inside a cell. Proteins are folded-up strings of amino acids that take three-dimensional shapes, with possible characteristic interaction sites accessible to other molecules. If the binding of interaction sites is energetically favorable, two or more proteins may specifically bind to each other to form a dynamic protein complex by a process called *complexation*. A protein complex may act as a catalyst by bringing together other compounds and facilitating

Artist Jonathan McCabe's interests include theories of biological pattern formation and evolution and their application to computer art. He writes computer programs that measure statistical properties of images for use in artificial evolution of computer art. For more, see www.jonathanmccabe.com/.

chemical reactions between them. Proteins may also chemically modify each other by attaching or removing modifying groups, such as phosphate groups, at specific sites. Each such modification may reveal new interaction surfaces. There are tens of thousands of proteins in a cell. At any given moment, each of them has certain available binding sites (which means that they can bind to other proteins, DNA, or membranes), and each of them has modifying groups at specific sites either present or absent. Protein-protein interaction networks are large and complex, and finding a language to describe them is a difficult task. Significant progress in this direction was made by the introduction of Kohn-maps, a graphical notation that resulted in succinct pictures depicting molecular interactions. Other approaches include the textual bio-calculus, or the recent use of existing process calculi (π-calculus), enriched with stochastic features, as the language to describe chemical interactions.

Yet another biological interaction network, and the last that we discuss here, is that of *transport networks* mediated by lipid membranes. Some lipids can self-assemble into membranes and contribute to the separation and transport of substances, forming transport networks. A biological membrane is more than a container: it consists of a lipid bilayer in which proteins and other molecules, such as glycolipids, are embedded. The membrane structural components, as well as the embedded proteins or glycolipids, can travel along this lipid bilayer. Proteins can interact with free-floating molecules, and some of these interactions trigger signal transduction pathways, leading to gene transcription. Basic operations of membranes include fusion of two membranes into one, and fission of a membrane into two. Other operations involve transport, for example transporting an object to an interior compartment where it can be degraded. Formalisms that depict the transport networks are few, and include membrane systems described earlier, and brane calculi.

The gene regulatory networks, the protein-protein interaction networks, and the transport networks are all interlinked and interdependent. Genes code for proteins which, in turn, can regulate the transcription of other genes, membranes are separators but also embed active proteins in their surfaces. Currently there is no single formal general framework and notation able to describe all these networks and their interactions. Process calculus has been proposed for this purpose, but a generally accepted common language to describe these biological phenomena is still to be developed and universally accepted. It is indeed believed that one of the possible contributions of computer science to biology could be the development of a suitable language to accurately and succinctly describe, and reason about, biological concepts and phenomena.[18]

While systems biology studies complex biological organisms as integrated wholes, *synthetic biology* is an effort to engineer artificial biological systems from their constituent parts. The mantra of synthetic biology is that one can understand only what one can construct. Thus, the main focus of synthetic biology is to take parts of natural biological systems and use them to build an artificial biological system for the purpose of understanding natural phenomena, or for a variety of possible applications. In this sense, one can make an analogy between synthetic biology and computer engineering.[3] The history of synthetic biology can be arguably traced back to the discovery in the 1960s, by Jacob and Monod,

of mathematical logic in gene regulation. Early achievements in genetic engineering using recombinant DNA technology (the insertion, deletion, or combination of different segments of DNA strands) can be viewed as the experimental precursors of today's synthetic biology, which now extends these techniques to entire systems of genes and gene products. One goal can be constructing specific synthetic biological modules such as, for example, pulse generator circuits that display a transient response to variations in input stimulus.

Advances in DNA synthesis of longer and longer strands of DNA are paving the way for the construction of synthetic genomes with the purpose of building an entirely artificial organism. Progress includes the generation of a 5,386bp synthetic genome of a virus, by rapid (14-day) assembly of chemically synthesized short DNA strands.[37] Recently an announcement was made of the near completion of the assembly of an entire "minimal genome" of a bacterium, *Mycoplasma Genitalium*.[7] Smith and others indeed found about 100 dispensable genes that can be removed individually from the original genome. They hope to assemble a minimal genome consisting of essential genes only, that would be still viable but shorter than the 528-gene, 580,000bp genome of *M.Genitalium*. This human-made genome could then be inserted into *a Mycoplasma* bacterium using a technique wherein a whole genome can be transplanted from one species into another, such that the resulting progeny is the same species as the donor genome. Counterbalancing objections to assembling a semi-synthetic cell without fully understanding its functioning, the creation of a functionally and structurally understood synthetic genome was proposed,[17] containing 151 genes (113,000bp) that would produce all the basic molecular machinery for protein synthesis and DNA replication. A third approach to create a human-made cell is the one pursued by Szostak and others, who would construct a single type of RNA-like molecule capable of self-replicating, possibly housed in a single lipid membrane. Such molecules can be obtained by guiding the rapid evolution of an initial population of RNA-like molecules, by selecting for desired traits.

Lastly, another effort in synthetic biology is toward engineering multicellular systems by designing, for example, cell-to-cell communication modules that could be used to coordinate living bacterial cell populations.

Research in synthetic biology faces many challenges, some of them of an information processing nature. There arguably is a pressing need for standardization, modularization, and abstraction, to allow focusing on design principles without reference to lower-level details.[15]

European artist Leonel Moura works with AI and robotics. The Swarm Paintings, produced in 2001, were the result of several experiments with an "Ant Algorithm" where he tried to apply virtual emergent pheromone trails to a real space pictorial expression. In this case, a computer running an ant algorithm was connected to a robotic arm that "translated" in pencil or brush strokes the trails generated by the artificial swarm of ants. For more images, see www.leonelmoura.com/.

Besides systems biology that tries to understand biological organisms as networks of interactions, and synthetic biology that seeks to engineer and build artificial biological systems, another approach to understanding nature as computation is the research on *computation in living cells*. This is also sometimes called *cellular computing*, or *in vivo computing*, and one particular study in this area is that of the computational capabilities of gene assembly in unicellular organisms called ciliates.

Ciliates possess two copies of their DNA: one copy encoding functional genes, in the macronucleus, and another "encrypted" copy in the micronucleus. In the process of conjugation, after two ciliates exchange genetic information and form new micronuclei, they use the new micronuclei to assemble in real-time new macronuclei necessary for their survival. This is accomplished by a process that involves re-ordering some fragments of DNA (permutations and possibly inversions), and deleting other fragments from the micronuclear copy. The process of gene assembly is fascinating from both the biological and the computational point of view. From the computational point of view, this study led to many novel and challenging research themes.[14] Among others, it was proved that various models of gene assembly have full Turing machine capabilities.[23] From the biological point of view, the joint effort of computer scientists and biologists led to a plausible hypothesis (supported already by some experimental data) about the "bioware" that implements the process of gene assembly, which is based on the new concept of template-guided recombination.[4, 28]

Other approaches to cellular computing include developing an *in vivo* programmable and autonomous finite-state automaton within *E.Coli*, and designing and constructing *in vivo* cellular logic gates and genetic circuits that harness the cell's existing biochemical processes.

At the end of this spectrum of views of nature as computation, the idea was even advanced by Zuse and Fredkin in the 1960s that information is more fundamental than matter or energy. The Zuse-Fredkin thesis stated that the entire universe is some kind of computational device, namely a huge cellular

automaton continuously updating its rules. Along the same lines, it has been recently suggested that the universe is a quantum computer that computes itself and its own behavior.

Natural Sciences: Ours to Discover

Science advances in ever-widening circles of knowledge. Sometimes it meticulously crawls. Other times it leaps to a new dimension of understanding and, in the process, it reinvents itself. As the natural sciences are rapidly absorbing ideas of information processing, and the meaning of computation is changing as it embraces concepts from the natural sciences, we have the rare privilege to take part in several such metamorphoses.

At this moment we and our natural scientist fellows are awash in wave after gigantic wave of experimental, especially biological, data. Just underneath this tumultuous surface lie ingenious algorithms waiting to be designed, elegant theorems waiting to be proven, natural laws waiting to be discovered that will put order into chaos. For, as Spinoza wrote, "nothing happens in nature that does not follow from her laws."

Conversely, as this review shows, there is an abundance of natural phenomena that can inspire computing paradigms, alternative physical substrates on which to implement computations, while viewing various natural processes as computations has become more and more essential, desirable, and inevitable. All these developments are challenging our assumptions about computation, and indeed, our very definition of it.

In these times brimming with excitement, our task is nothing less than to discover a new, broader, notion of computation, and to understand the world around us in terms of information processing.

Let us step up to this challenge. Let us befriend our fellow the biologist, our fellow the chemist, our fellow the physicist, and let us together explore this new world. Let us, as computers in the future will, embrace uncertainty. Let us dare to ask afresh: "What is computation?", "What is complexity?", "What are the axioms that define life?"

Let us relax our hardened ways of thinking and, with deference to our scientific forebears, let us begin anew.

Literature

The upper-bound placed on the number of references was a real limitation for this review, since the literature on natural computing is vast. For a more complete list of references the reader is referred to the full version of this article at www.csd.uwo.ca/~lila/Natural-Computing-Review.pdf.

Almost each of the areas we mentioned here has an extensive scientific literature as well as a number of specialized journals and book series. There are also journals and book series aimed at the general natural computing community, among them the journals *Natural Computing*, Springer, *Theoretical Computer Science, Series C: Theory of Natural Computing*, Elsevier, the *Natural Computing* book series, Springer, and the upcoming *Handbook of Natural Computing* (G. Rozenberg, T. Bäck, J. Kok, editors, Springer).

Acknowledgments

We gratefully acknowledge comments on early drafts of this paper by T. Bäck, D. Bentley, G. Brassard, D. Corne, M. Hirvensalo, J. Kari, P. Krishna, H. Lipson, R. Mercer, A. Salomaa, K. Sims, H. Spaink, J. Timmis, C. Torras, S. Watt, R. Weiss.

This work was supported by NSERC Discovery Grant and Canada Research Chair Award to L.K., and NSF grant 0622112 to G.R.

References

1. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight Jr., T., Nagpal, R., Rauch, E., Sussman, G., and Weiss, R. Amorphous computing. *Commun. ACM 43*, 5 (May 2000), 74–82.
2. Adleman, L. Molecular computation of solutions to combinatorial problems. *Science 266* (1994), 1021–1024.
3. Andrianantoandro, E., Basu, S., Karig, D., and Weiss, R. Synthetic biology: new engineering rules for an emerging discipline. *Molecular Systems Biology 2* (2006), 1–14.
4. Angeleska, A., Jonoska, N., Saito, M., and Landweber, L. RNA-guided DNA assembly. *J. Theoretical Biology 248* (2007), 706–720.
5. Arbib, M., editor. *The Handbook of Brain Theory and Neural Networks.* MIT Press, 2003.
6. Bäck, T., Fogel, D., and Michalewicz, Z., editors. *Handbook of Evolutionary Computation.* IOP Publishing, U.K., 1997.
7. Barry, P. Life from scratch: learning to make synthetic cells. *Science News, 173*, 2 (2008), 27.
8. Bath, J. and Turberfield, A. DNA nanomachines. *Nature Nanotechnology 2* (May 2007), 275–284
9. Brooks, R. Artificial life: From robot dreams to reality. *Nature 406* (2000), 945–947.
10. Cardelli, L. Machines of systems biology. Bulletin of the EATCS 93 (2007), 176–204.
11. Dasgupta, D. editor. *Artificial Immune Systems and Their Applications.* Springer, 1998.
12. de Castro, L. and Timmis, J. *Artificial Immune Systems: A New Computational Intelligence Approach.* Springer, 2002.
13. De Jong, K. *Evolutionary Computation: A Unified Approach.* MIT Press, 2006.
14. Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D., and Rozenberg, G. *Computation in Living Cells: Gene Assembly in Ciliates.* Springer, 2004.
15. Endy, D. Foundations for engineering biology. *Nature 438* (2005), 449–453.
16. Engelbrecht, A. *Fundamentals of Computational Swarm Intelligence.* Wiley and Sons, 2005.
17. Forster, A. and Church, G. Towards synthesis of a minimal cell. *Molecular Systems Biology 2*, 45 (Aug. 2006).
18. Fox Keller, E. and Harel, D. Beyond the gene. *PLoS ONE 2*, 11 (2007), e1231.
19. Hirvensalo, M. *Quantum Computing, 2nd Ed.* Springer, 2004.
20. Istrail, S., De-Leon, B-T., and Davidson, E. The regulatory genome and the computer. *Developmental Biology* 310 (2007), 187–195.
21. Kari, L. DNA computing—the arrival of biological mathematics. *The Math. Intelligencer 19*, 2 (1997), 9–22.
22. Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
23. Landweber, L. and Kari, L. The evolution of cellular computing: Nature's solution to a computational problem. *Biosystems 52*, 1/3 (1999), 3–13.
24. Langton, C., editor. *Artificial Life.* Addison-Wesley Longman, 1990.
25. Lipson, H. and Pollack, J. Automatic design and manufacture of robotic lifeforms. *Nature 406*, (2000), 974–978.
26. Paun, G. *Membrane Computing: An Introduction.* Springer, 2002.
27. Paun, G., Rozenberg, G., and Salomaa, A. *DNA Computing: New Computing Paradigms.* Springer, 1998.
28. Prescott, D., Ehrenfeucht, A., and Rozenberg, G. Template guided recombination for IES elimination and unscrambling of genes in stichotrichous ciliates. *J. Theoretical Biology 222*, 3 (2003), 323–330.
29. Prusinkiewicz, P. and Lindenmayer, A. *The Algorithmic Beauty of Plants.* Springer, 1990.
30. Reif, J. and LaBean, T. Autonomous programmable biomolecular devices using self-assembled DNA nanostructures. *Commun. ACM 50*, 9 (Sept. 2007), 46–53.
31. Rojas, R. *Neural Networks: A Systematic Introduction.* Springer, 1996.
32. Rothemund, P., Papadakis, N., and Winfree, E. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology 2*, 12 (Dec. 2004).
33. Rozenberg, G. Computer science, informatics and natural computing—personal reflections. In *New Computational Paradigms: Changing Conceptions of What Is Computable.* Springer, 2008, 373–379.
34. Rozenberg, G. and Salomaa, A. *The Mathematical Theory of L Systems.* Academic Press, 1980.
35. Seeman, N. Nanotechnology and the double helix. *Scientific American Reports*, 17, 3 (2007), 30–39.
36. Sims, K. Evolving 3D morphology and behavior by competition. In *Proceedings of Artificial Life IV*. MIT Press, 1994, 28–39.
37. Smith, H., Hutchison III, C., Pfannkoch, C., and Venter, C. Generating a synthetic genome by whole genome assembly: ΦX174 bacteriophage from synthetic oligonucleotides. *PNAS 100*, 26 (2003), 15440–15445.
38. Stepney, S. et al. Journeys in non-classical computation I: A grand challenge for computing research. *Int. J. Parallel, Emergent and Distributed Systems 20*, 1 (2005), 5–19.
39. von Neumann, J. *The Computer and the Brain.* Yale University Press, 1958.
40. von Neumann, J. Theory of Self-Reproducing Automata. U. Illinois Press, 1966. Edited and completed by A.W.Burks.

Lila Kari (lila@csd.uwo.ca) is Professor and Canada Research Chair in Biocomputing in the Department of Computer Science at the University of Western Ontario, London, Canada.

Grzegorz Rozenberg (rozenber@liacs.nl) is Professor at the Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands, and Adjunct Professor in the Department of Computer Science at the University of Colorado at Boulder, USA.

CSCW08

November 8-12, 2008
Hilton San Diego Resort
San Diego, California, USA

CALL FOR REGISTRATION

The 2008 ACM Conference on Computer Supported Cooperative Work

Attend CSCW 2008 for an intellectually stimulating and diverse technical program!

Plenary Speakers

The CSCW 2008 Opening and Closing Keynote speakers cover the spectrum of interests across the CSCW community from the technical and design challenges of creating collaboration systems to the intricacies of their impact on people and practices. The opening Keynote will be by Cory Ondrejka, co-founder of Second Life, and the Closing Keynote will be given by Sara Diamond, president of the Ontario College of Art & Design.

Paper Sessions

The paper sessions will cover emerging areas of collaboration and collaborative systems such as Wikis & Wikipedia, Naughty & Nice issues in Social Networking Systems, Health Informatics, Deployments at Home, and Disrupted Environments. The paper sessions will also include traditional CSCW topics such as Distributed Teams, Media Spaces, Community, Interpersonal Relations in the Group, Work Places and Work Practices.

Workshops

Please consider participating in a CSCW 2008 Workshop! There are more than 10 workshops scheduled during the weekend prior to the main technical program. Some of the scheduled workshops include:

- *Social Networking in Organizations*
- *The Future of Mobile Social Software*
- *Virtual Radical Collocation*
- *Remix rooms: Mashing up media for meetings CSCW and Human Factors – Where are we now and what are the challenges?*
- *Changing work, Changing technology (IFIP 9.1 WG)*

Key Registration Dates

Aug 15, 2008 - Registration Opens | Sept 30, 2008 - Early Registration Deadline | Oct 3, 2008 - Hotel Room Reservation Deadline

CSCW 2008 also contains Panels, Demonstrations and Interactive Poster sessions that provide opportunities to engage in lively discussions with leaders in this field of increasing importance to our global future.

Questions and requests for information should be sent to ***publicity@cscw2008.org***, or visit our website: ***www.cscw2008.org***

research highlights

P. 86
Technical Perspective
Computational Photography on Large Collections of Images
By Marc Levoy

P. 87
Scene Completion Using Millions of Photographs
By James Hays and Alexei A. Efros

P. 95
Technical Perspective
New Developments in Graph Partitioning
By Éva Tardos

P. 96
Geometry, Flows, and Graph-Partitioning Algorithms
By Sanjeev Arora, Satish Rao, and Umesh Vazirani

DOI:10.1145/1400181.1400201

Technical Perspective
Computational Photography on Large Collections of Images

By Marc Levoy

THIS PAPER WILL strike a familiar chord with anyone who has ever taken a picture. The problem is easy to understand—replacing unwanted parts of a photograph. The authors start with an interesting twist, by making a distinction between data that "should have been there" but was obscured by a telephone pole, and data that "could have been there," meaning that it constitutes an incorrect but plausible picture. This is a difficult problem, because faked pictures are relatively easy to spot, as recent scandals in photojournalism have proven. Nevertheless, Hays and Efros obtain impressive results; check out Figure 1 online (http://graphics.cs.cmu.edu/projects/scene-completion/scene-completion.pdf); it looks seamless no matter how closely you zoom into it.

The paper represents the confluence of several noteworthy trends in computing. First, it exemplifies a new application area, *computational photography*, which refers broadly to sensing strategies and algorithms that extend the capabilities of digital photography. Representative techniques include high-dynamic-range imaging, flash-noflash and coded aperture imaging, panoramic stitching, digital photomontage, and light field imaging. ACM SIGGRAPH is at the forefront of this new area. Indeed, of the 108 papers at its 2007 conference, 20 were arguably about computational photography. This paper fits squarely in that group.

Second, the authors exemplify the ongoing convergence of several formerly isolated research communities. To find a suitable replacement for the unwanted part of a photograph, the authors search a collection of images using "gists," an image summarization technique pioneered in the cognitive science community by Aude Oliva and Antonio Torralba. They then find the best seam along which to insert the matching content using graph-cuts, an algorithm first applied to images by Yuri Boykov, Vladimir Kolmogorov, and Ramin Zabih in the computer vision community. Finally, they smooth the seam between new and old imagery using gradient domain blending, a technique introduced into the graphics literature by Raanan Fattal (2003) for tone mapping of high-dynamic-range images.

Third, this paper provides evidence of the notion, gaining credence in many application domains, that simple machine learning algorithms often outperform more sophisticated ones if trained on large enough databases. Natural language translation algorithms work dramatically better if trained on millions of documents than on thousands. Image classification and segmentation algorithms do, too, as the authors argue here. Want to remove a garbage truck from your snapshot of an Italian piazza? Start with a database containing lots of Italian piazzas.

How far can one push this data-centric approach to image matching? While it's impossible to collect all possible images of the world, the authors make the conjecture that one could collect all "semantically differentiable scenes." I'm doubtful. It is well known that features in natural scenes form a heavy-tailed distribution, meaning that while some features in photographs are more common than others, the relative occurrence of less common features drops slowly. In other words, there are many unusual photographs in the world.

A closely related question is: What is meant by data that "could have been there?" The authors define this as all "semantically valid" scenes, but semantic validity is maddeningly difficult to pin down. Indeed, to evaluate their results quantitatively, Hays and Efros resort to a human study: How well can naïve viewers distinguish an algorithmically completed image from a real photograph? In the end this question may prove to be "AI-complete," that is, it's as hard as making computers as intelligent as people.

Regardless of whether we ever answer this question, it is clear that large collections of images are useful. Aside from completing photographs, they can be used to build 3D models of urban monuments (as in Snavely et al.'s Photo Tourism, http://phototour.cs.washington.edu/) or to synthesize textures from image exemplars (as in Kopf et al.'s Solid Texture Synthesis; http://www.johanneskopf.de/publications/solid/index.php). Maybe collections of images can even help me take better pictures! If we sort the collection geographically, as Hays and Efros do in a follow-on paper (in CVPR 2008), then my camera could query a central database to help it decide what color balance to use for the shot I'm now taking of the Grand Canyon. Closer to home, what color balance should the camera use to photograph my wife? It ought to know, since my online albums contain hundreds of photographs of her.

Finally, while the future of computational photography is exciting because of papers like this, the future is also murky because it's not obvious who will use this stuff. When I show this paper to someone outside computing, their first question is: Why? How many people (other than revisionist dictators) would remove a person or large object from a photograph? Tied to this question is the debate about how much effort can we expect consumers to exert after they've taken a photograph. No matter what position you take on these questions, nobody doubts that the coming years will be interesting ones for the computer graphics and vision communities. This paper helps light the way. C

Marc Levoy (levoy@cs.stanford.edu) is a professor of computer science and (jointly) electrical engineering at Stanford University.

DOI:10.1145/1400181.1400202

Scene Completion Using Millions of Photographs

By James Hays and Alexei A. Efros

Abstract

What can you do with a million images? In this paper, we present a new image completion algorithm powered by a huge database of photographs gathered from the Web. The algorithm patches up holes in images by finding similar image regions in the database that are not only seamless, but also semantically valid. Our chief insight is that while the space of images is effectively infinite, the space of semantically differentiable scenes is actually not that large. For many image completion tasks, we are able to find similar scenes which contain image fragments that will convincingly complete the image. Our algorithm is entirely data driven, requiring no annotations or labeling by the user. Unlike existing image completion methods, our algorithm can generate a diverse set of image completions and we allow users to select among them. We demonstrate the superiority of our algorithm over existing image completion approaches.

1. INTRODUCTION

Every once in a while, we all wish we could erase something from our old photographs. A garbage truck right in the middle of a charming Italian piazza, an ex-boyfriend in a family photo, a political ally in a group portrait who has fallen out of favor.[13] Other times, there is simply missing data in some areas of the image: (a) an aged corner of an old photograph (b) a hole in an image-based 3D reconstruction due to occlusion, and (c) a dead bug on the camera lens. Image completion (also called inpainting or hole-filling) is the task of filling in or replacing an image region with new image data such that the modification cannot be detected.

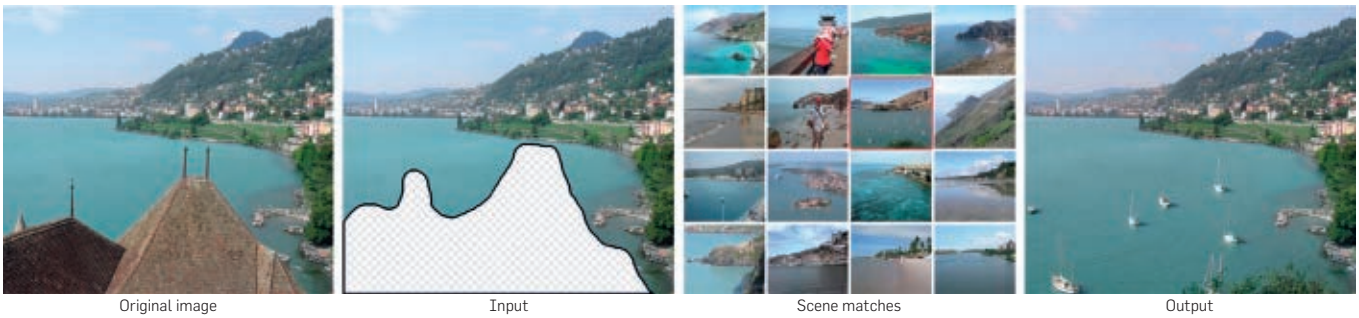
There are two fundamentally different strategies for image completion. The first aims to reconstruct, as accurately as possible, the data that *should have been* there, but somehow got occluded or corrupted. Methods attempting an accurate reconstruction have to use some other source of data in addition to the input image (Figure 1), such as video (using various background stabilization techniques) or multiple photographs of the same scene.[1,19]

The alternative is to try finding a plausible way to fill in the missing pixels, hallucinating data that *could have been* there. This is a much less easily quantifiable endeavor, relying instead on the studies of human visual perception. The most successful existing methods[4,6,24,25] operate by extending adjacent textures and contours into the unknown region. These algorithms are similar to texture synthesis algorithms such as,[8,7,14,15] sometimes with additional constraints to explicitly preserve Gestalt cues such as *good continuation*,[23] either automatically[4] or by hand.[20] Importantly, all of the existing image completion methods operate by filling in the unknown region with content from the known parts of the input source image.

Searching the source image for usable texture makes a lot of sense. The source image often has textures at just the right scale, orientation, and illumination as needed to seamlessly fill in the unknown region. Some methods[6,25] search additional scales and orientations to gain additional source texture samples. However, viewing image completion as constrained texture synthesis limits the type of completion tasks that can be tackled. The assumption present in all of these methods is that all the necessary image data to fill in an unknown region is located somewhere else in *that same image*. We believe this assumption is flawed and that the source image simply does not provide enough data except for trivial image completion tasks.

Typical demonstrations of previously published algorithms are object removal tasks such as removing people, signs, horses, or cars from relatively simple backgrounds. The results tend to be fairly sterile images because the algorithms are only reusing image content that appeared somewhere else in the same image. For situations in which the incomplete region is not bounded by texture regions, or when

Figure 1: Given an input image with a missing region, we use matching scenes from a large collection of photographs to complete the image.



Original image | Input | Scene matches | Output

Figure 2: Results from image completion algorithms including Microsoft Digital Image Pro Smart Erase.

Original image | Input | Criminisi et al. 2003 | Smart erase | Wilczkowiak et al. | Our algorithm

there is too little useful texture, existing algorithms have trouble completing scenes (Figure 2).

2. OVERVIEW

In this paper, we perform image completions by leveraging a massive database of images. There are two compelling reasons to expand the search for image content beyond the source image. (1) In many cases, a region will be impossible to fill plausibly using only image data from the source image. For instance, if the roof of a house is missing or the entire sky is masked out. (2) Even if there is suitable content in the source image, reusing that content would often leave obvious duplications in the final image, e.g. replacing a missing building with another building in the image. By performing hole filling with content from other images, entirely novel objects and textures can be inserted.

However, there are several challenges with drawing content from other images. The first challenge is computational. Even in the single image case some existing methods report running times in the hours[6,8] because of the slow texture search. Texture synthesis-based image completion methods are difficult to accelerate with traditional nearest-neighbor or approximate nearest-neighbor methods because of the high dimensionality of the features being searched and because the known dimensions of the feature being matched on change depending on the shape of the unknown region at each iteration.

The second challenge is that as the search space increases, there is higher chance of a synthesis algorithm finding locally matching but semantically invalid image fragments. Existing image completion methods might produce sterile images but they do not risk putting an elephant in someone's back yard or a submarine in a parking lot.

The third challenge is that content from other images is much less likely to have the right color and illumination to seamlessly fill an unknown region compared to content from the same image. More than other image completion methods, we need a robust seam-finding and blending method to make our image completions plausible.

In this work, we alleviate both the computational and semantic challenges with a two-stage search. We first try to find images depicting semantically similar scenes and then use only the best matching scenes to find patches which match the context surrounding the missing region. Scene matching reduces our search from a database of one million images to a manageable number of best matching scenes (60 in our case), which are used for image completion. We use a low-dimensional scene descriptor[16] so it is relatively fast to find the nearest scenes, even in a large database. Our approach is purely data driven, requiring no labeling or supervision.

In order to seamlessly combine image regions we employ Poisson's blending. To avoid blending artifacts, we first perform a graph cut segmentation to find the boundary for the Poisson blending that has the minimum image gradient magnitude. This is in contrast to minimizing the intensity domain difference along a boundary[25] or other heuristics to encourage a constant intensity offset for the blending boundary.[11] In Section 4, we explain why minimizing the seam gradient gives the most perceptually convincing compositing results.

The image completion work most closely resembling our own, Wilczkowiak et al.[25] also demonstrates the search of multiple images. However, in their case it was only a few images that were hand selected to offer potentially useful image regions. Also related are methods which synthesize semantically valid images either from text or image constraints.[5,12] These methods create semantically valid images through explicit semantic constraints using image databases with semantically labeled regions. The database labeling process must be supervised[5] or semisupervised.[12]

3. SEMANTIC SCENE MATCHING

Since we do not employ user-provided semantic constraints or a labeled database, we need to acquire our semantic knowledge from the data directly. This requires us to sample the set of visual images as broadly as possible. We constructed our image collection by downloading all of the photographs in 30 Flickr.com groups that focus on landscape, travel, or city photography. Typical group names are "lonely planet," "urban fragments," and "rural decay." Photographs in these groups are generally high quality. We also downloaded images based on keyword searches such as "travel," "landscape," and "river." We discarded all duplicate images and all images that did not have at least 800 pixels in their largest dimension and 500 pixels in their smallest dimension. All images were down-sampled, if necessary, such that their maximum dimension was 1024 pixels. Our database downloading, preprocessing, and scene matching are all

distributed among a cluster of 15 machines. In total we acquired about 2.3 million unique images totalling 396 gigabytes of JPEG compressed data.

In order to successfully complete images, we need to find image regions in our database that are not just seamless and properly textured but also *semantically* valid. We do not want to complete hillsides with car roofs or have ducks swimming in city pavement which locally resembles a lake. To help avoid such situations, we first look for *scenes* which are most likely to be semantically equivalent to the image requiring completion. The use of scene matching is the most important element of our image completion method. Without it, our search would not be computationally feasible and our image completion results would rarely be semantically valid. Our scene matching, in combination with our large database, allows us to do image completion without resorting to explicit semantic constraints as in previous photo synthesis work.[5,12]

We use the gist scene descriptor[16,22] which has been shown to perform well at grouping semantically similar scenes (e.g., city, tall buildings, office, fields, forest, and beach) and for place recognition. It must be noted that a low-level scene descriptor in and of itself cannot encode high-level semantic information. Scenes can only be trusted to be semantically similar if the distance between them is very small. The way we address this issue is by collecting a huge dataset of images making it more likely that a very similar scene to the one being searched is available in the dataset. Indeed, our initial experiments with the gist descriptor on a dataset of ten thousand images were very discouraging. However, increasing the image collection to one million yielded a qualitative leap in performance (see Figure 3 for a typical scene matching result). Independently, Torralba et al.[21] have observed a similar effect with a dataset of up to 70 million tiny (32×32) images.

The gist descriptor aggregates oriented edge responses at multiple scales into very coarse spatial bins. We found a gist descriptor built from six oriented edge responses at five scales aggregated to a 4×4 spatial resolution to be the most effective. We augment the scene descriptor with the color information of the query image down-sampled to the spatial resolution of the gist.

Searching for similar scenes first rather than directly looking for similar patches speeds up our search dramatically. Instead of looking for image patches in all one million images at multiple offsets and scales, we can instead eliminate 99.99% of the database quickly by finding the nearest neighbor scenes based on the relatively low-dimensional scene descriptor.

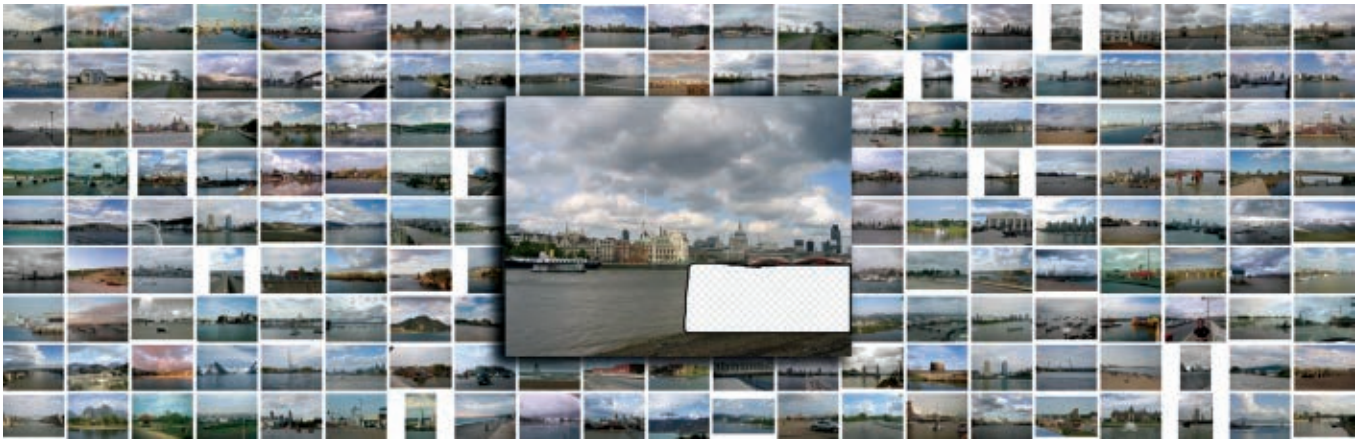
Given an input image to be hole-filled, we first compute its gist descriptor with the missing regions excluded. This is done by creating a mask which weights each spatial bin in the gist in proportion to how many valid pixels are in that bin. We compute the L_1 distance between the gist of the query image and every gist in the database, weighted by the mask. The color distance is computed in the $L^*a^*b^*$ color space. These distances are combined and weighted such that the gist contributes roughly twice as much as the color information to the final distance.

Because we match scenes using arbitrary dimensions of the descriptor (depending on which regions of a query image are missing), we cannot use principal components analysis (PCA) dimensionality reduction as suggested in[16] For the same reason, we do not use any approximate nearest-neighbor scheme since they tend to incur large relative errors when matching on arbitrary dimensions of descriptors with hundreds of dimensions. However, the descriptors are small enough to exhaustively search in a few minutes using a cluster of 15 machines.

4. LOCAL CONTEXT MATCHING

Having constrained our search to semantically similar scenes, we can use traditional template matching to more precisely align those matching scenes to the local image context around the missing region. The local context is all pixels within an 80 pixel radius of the hole's boundary. This context is compared against the 200 best matching scenes across all valid translations and three scales (0.81, 0.90, 1) using pixel-wise SSD error in $L^*a^*b^*$ color space. Only placements (translations and scales) for which the context is fully

Figure 3: The first 164 nearest neighbor scenes for the incomplete image in the center. Most of the scenes are semantically and structurally similar; many are even from the same city (London).



contained in the matching scene are considered. Since we expect our matching scenes to already be roughly aligned with the incomplete image, we discourage spurious, distant matches by multiplying the error at each placement by the magnitude of the offset.

We composite each matching scene into the incomplete image at its best scoring placement using a form of graph cut seam finding[15] and standard Poisson's blending.[17] Using a seam finding operation to composite the images is arguably undesirable for hole filling since a user might want to preserve all of the image data in the input image. Past image completion algorithms[4] have treated the remaining valid pixels in an image as hard constraints which are not changed. We relax this, as in[25] and allow the seam-finding operation to remove valid pixels from the query image but we discourage the cutting of too many pixels by adding a small cost for removing each pixel in the query image which increases with distance from the hole.

When performing a seam-finding operation and gradient domain fusion in sequence, it makes sense to optimize the seam such that it will minimize the artifacts left behind by the gradient domain fusion. Jia et al.[11] uses iterative dynamic programming optimizations to find a seam which leaves minimum intensity difference between the two images after allowing some constant intensity offset. The intuition is that humans are not sensitive to relatively large shifts in color and intensity as long as the shifts are seamless and low frequency. Inspired by this, as well as the fact that our scene matches tend to have colors similar to our query image, we propose a seam-finding heuristic which ignores intensity differences entirely and instead minimizes the difference in image gradients. This heuristic finds seams that avoid cutting high-frequency image content for which Poisson's blending would cause obvious artifacts.

An advantage of our heuristic is that we can quickly find the optimal seam using a graph cut procedure.[3] The details of our graph cut setup can be found in our original SIGGRAPH publication.[9] A very similar metric was mentioned but not demonstrated in[1] As in,[1] we allow the Poisson blending to operate on the entire image domain instead of correcting one of the images. We use the Poisson solver of.[2]

Finally, we assign each composite a score which is the sum of the scene matching distance, the local context matching distance, the local texture energy distance, and the cost of the graph cut. All four components of the score are scaled to contribute roughly equally. We present the user with the 20 composites with the lowest scores.

5. RESULTS AND COMPARISON

We test our algorithm on a set of 51 images with missing regions. All test images come from either the LabelMe database[18] or our own personal photo collections, none of which are contained in our downloaded database. Images in the test set, about a half megapixel each, are higher resolution than the images used in most previous works[4,6] and likewise the missing regions are quite large (56,000 pixels on average). The regions we removed from the photos all had semantic meaning such as unwanted objects, store-fronts, walls with graffiti, roads, etc. The test set is made freely available on the authors' Web page.

Image completion is an inherently underconstrained problem. There are many viable ways to fill a hole in an image. Previous approaches, which operate by reusing texture from the input image, can offer relatively few viable, alternative completions (perhaps by changing parameters such as the patch size). While some such results might look slightly better than others, the semantic interpretation of the image is unlikely to change. On the other hand, our algorithm can offer a variety of semantically valid image completions for a given query image (Figure 4). After compositing, the best-matching patches we present a user with the 20 top image completions (roughly equivalent to one page of image search results). In some cases, many of these completions are of acceptable quality and the user can select the completion(s) which they find most compelling. In other cases, only a few or none of the results were acceptable. The quality of our results benefits from this very simple user interaction and it

Figure 4: The algorithm presents to the user a set of alternative image completions for each input. Here, we show three such alternatives.

is difficult for conventional image completion methods to offer an analogous selection of results.

Some of our image completions are shown in Figure 5. The bottom result is interesting because the scaffolding on the cathedral that was masked out has been replaced with another image patch of the same cathedral. The database happened to contain an image of the same cathedral from a similar view. It is not our goal to complete scenes and objects with their *true* selves in the database, but with an increasingly large database such fortuitous events do occur.

In all of the successful cases, the completion is semantically valid but there might be slight low-level artifacts such as resolution mismatch between the image and patch, blurring from Poisson's blending, or fine-scale texture differences between the image and patch. For failure cases these low-level artifacts are often much more pronounced (Figure 6, top). Another source of failure is a lack of good scene matches which happens more often for atypical scenes (Figure 6, middle). Semantic violations (e.g., half-objects) account for another set of failures. The latter is not surprising since the algorithm has no object recognition capabilities and thus no notion of object boundaries.

For uniformly textured backgrounds (Figure 7, top), existing image completion algorithms perform well. However, our algorithm struggles since our scene matching is unlikely to find the exact same texture in another photograph. Furthermore, image completion algorithms such as Criminisi et al.[4] have explicit structure propagation which helps in some scenes (Figure 7, bottom).

Our hole filling algorithm requires about 5 min to process an input image. The scene matching, local context matching, and compositing would take about 50, 20, and 4 min respectively on a single central processing unit (CPU) but we parallelize all of these across 15 CPUs. Our algorithm is implemented in MATLAB and all of the timings are for Pentium 4 processors.

5.1. Quantitative evaluation

It is difficult to rigorously define success or failure for an image completion algorithm because so much of it depends on human perception. While previous approaches demonstrate performance qualitatively by displaying a few results, we believe that it is very important to also provide a quantitative measure of the algorithm's success. Therefore, to evaluate our method, we performed a perceptual study to see how well naive viewers could distinguish our results, as well as those of a previous approach,[4] from real photographs. The study was performed on a set of 51 test images that were defined a priori and spanning different types of completions. We were careful not to include any potentially recognizable scenes or introduce bias that would favor a particular algorithm. We generated three versions of each image—the real photograph from which the image completion test cases were constructed, the result from Criminisi et al., and the result from our algorithm.

Each of our 20 participants viewed a sequence of images and classified them as real or manipulated. Of the 51 images each participant examined, 17 were randomly chosen from each source, but such that they do not see multiple versions of the same image. The order of presentation was also randomized. The participants were told that some of the images would be real, but they were not told the ratio of real versus manipulated images. We also told the participants that we were timing their responses for each image but that they should try to be accurate rather than fast. Overall the participants classified 80% of the images correctly. No effort was made to normalize for the differences in individual aptitude (which were small).

With unlimited viewing the participants classified our algorithm's outputs as real 37% of the time compared with 10% for Criminisi et al.[4] Note that participants identified real images as such only 87% of the time. Participants scrutinized the images so carefully that they frequently convinced themselves real images were fake.

It is interesting to examine the responses of participants over time. In Figure 8 we measure the proportion of images from each algorithm that have been marked as fake with an increasing limit on the amount of time allowed. We claim that if a participant who has been specifically tasked with finding fake images cannot be sure that an image is fake within 10 s, it is unlikely that an unsuspecting, casual observer would notice anything wrong with the image. After 10 s of examination, participants have marked our algorithm's results as fake only 34% of the time (the other 66% are either undecided or have marked the image as real already). For Criminisi et al. participants have marked 69% of the images as fake by 10 s. For real photographs, only 3% have been marked as fake. All pairwise differences are statistically significant ($p < 0.001$).

6. DISCUSSION

This paper approaches image completion from an entirely new direction—orthogonal and complementary to the existing work. While previous algorithms[4,6,8,25] suggest clever ways to reuse visual data within the source image, we demonstrate the benefits of utilizing semantically valid data from a large collection of unlabeled images. Our approach successfully fills in missing regions where prior methods, or even expert users with the Clone brush, would have no chance of succeeding because there is simply no appropriate image data in the source image to fill the hole. Likewise, expert users would have trouble leveraging such a large image collection—it would take 10 days just to view it with one second spent on each image. Additionally, this is the first paper in the field of image completion to undertake a full perceptual user study and report success rates on a large test set. While the results suggest substantial improvement over previous work, image completion is extremely difficult and is far from solved. Given the complementary strengths of our method and single-image techniques, a hybrid approach is likely to be rewarding.

It takes a large amount of data for our method to succeed. We saw dramatic improvement when moving from ten thousand to one million images. But one million images is still a tiny fraction of the high-quality photographs available on sites like Picasa or Flickr (which has approximately 2 billion images). The number of photos on the entire Internet is surely orders of magnitude larger still. Therefore, our approach would be an attractive Web-based application. A user would

Figure 5: Results. The input and the matching scenes are composited together to create the outputs. The matching scene used in each output is highlighted in red. Note that the algorithm can handle a large range of scenes and missing regions. On rare occasions, the algorithm is lucky enough to find another image from the same physical location as seen in the bottom example.

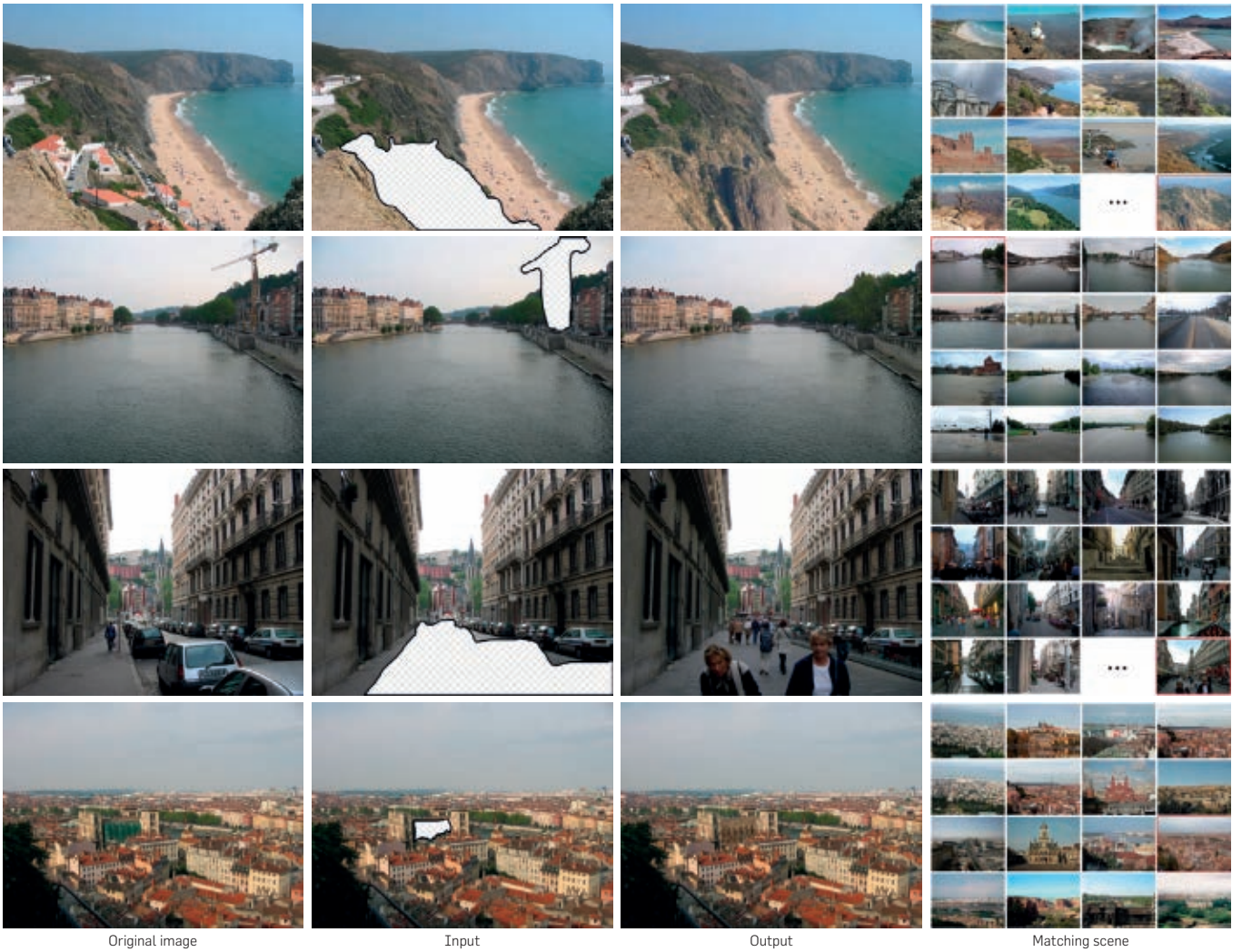


Figure 6: Typical failure cases. Some results exhibit pronounced texture seams (top). Others are failures of scene matching (middle). The last failure mode (bottom), shared with traditional image completion algorithms, is a failure to adhere to high-level semantics (e.g., entire people).

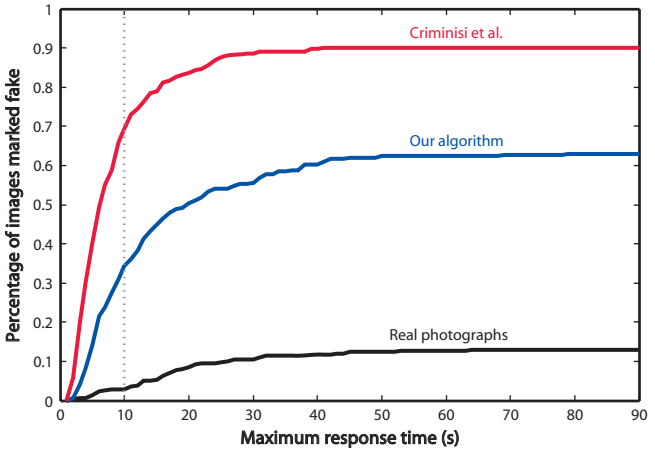
Figure 7: Situations where existing image completion algorithms perform better than our algorithm.

submit an incomplete photo and a remote service would search a massive database, in parallel, and return results.

7. TOWARD BRUTE-FORCE IMAGE UNDERSTANDING

Beyond the particular graphics application, the deeper question for all appearance-based data-driven methods is this: would it be possible to ever have enough data to represent the entire visual world? Clearly, attempting to gather all possible images of our dynamic world is a futile task, but what about collecting the set of all semantically differentiable scenes? That is, given any input image can we find a scene that is "similar enough" under some metric? The truly exciting (and surprising!) result of our work is that not only does it seem possible, but the number of required images might not be astronomically large. This paper, along with work by

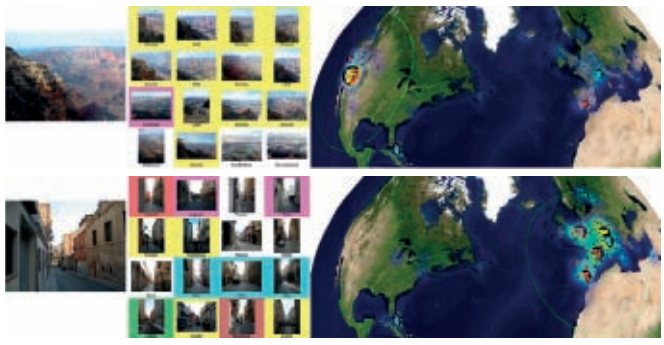
Figure 8: The percentage of images marked as fake within any amount of viewing time by participants in our perceptual study.



Torralba et al.,[21] suggest the feasibility of sampling from the entire space of scenes as a way of exhaustively modeling our visual world. This, in turn, might allow us to "brute force" many currently unsolvable vision and graphics problems!

Further supporting this possibility, we recently used scene matching methods similar to those presented here to estimate the GPS location of an arbitrary image. In a project called IM2GPS,[10] we collect a database of 6 million geotagged photographs from Flickr and show that image matches for a query photo are often "similar enough" to be geographically informative even if we do not match to the exact, real-world location. We represent the estimated image location as a probability distribution over the Earth's surface (see Figure 9). We quantitatively evaluate our approach in several geolocation tasks and demonstrate encouraging performance (up to 30 times better than chance). We show that geolocation estimates can provide the basis for numerous other image understanding tasks such as population density estimation, land cover estimation or urban/rural classification (see[10] for details).

Figure 9: Geolocation estimates for photos of the Grand Canyon and a generic European alley. From left to right are the query photographs, the first 16 nearest scene matches, and the distribution of the top 120 nearest-neighbors across the Earth. Geographic clusters are marked by X's with size proportional to cluster cardinality. The ground truth locations of the queries are surrounded by concentric green circles.



Acknowledgments

We would like to thank Antonio Torralba for the gist code and useful comments and Gabriel Brostow for sharing their hole-filling software. We also thank Tamara Berg, Alex Berg, and Rob Fergus for sharing their image downloading scripts. Finally we are grateful to Flickr for allowing us to download so many images. This work has been partially supported by NSF Graduate Research Fellowship to James Hays and by NSF grants CCF-0541230 and IIS-0546547.

References

1. Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., and Cohen, M. Interactive digital photomontage. *ACM Trans. Graph.*, 23(3):294–302, 2004.
2. Agrawal, A., Raskar, R., and Chellappa, R. What is the range of surface reconstructions from a gradient field? *European Conference on Computer Vision (ECCV)*, May 2006.
3. Boykov, Y., Veksler, O., and Zabih, R. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.
4. Criminisi, A., Perez, P., and Toyama, K. Object removal by exemplar-based inpainting. *CVPR*, 02:721, 2003.
5. Diakopoulos, N., Essa, I., and Jain, R. Content based image synthesis. *Conference on Image and Video Retrieval (CIVR)*, 2004.
6. Drori, I., Cohen-Or, D., and Yeshurun, H. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, 2003.
7. Efros, A. A. and Freeman, W. T. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.
8. Efros, A. A. and Leung, T. K. Texture synthesis by non-parametric sampling. *ICCV*, pages 1033–1038, Corfu, Greece, September 1999.
9. Hays, J. and Efros, A. A. Scene completion using millions of photographs. *ACM Trans. Graph. (SIGGRAPH 2007)*, 26(3): 4, 2007.
10. Hays, J. and Efros, A. A. im2gps: estimating geographic information from a single image. *CVPR*, 2008.
11. Jia, J., Sun, J., Tang, C.-K., and Shum, H.-Y. Drag-and-drop pasting. *ACM Trans. Graph.*, 25(3): 631–637, 2006.
12. Johnson, M., Brostow, G. J., Shotton, J., Arandjelović, O., Kwatra, V., and Cipolla, R. Semantic photo synthesis. *Comput. Graph. Forum (Proc. Eurographics)*, 25(3):407–413, 2006.
13. King, D. *The Commissar Vanishes*. Henry Holt and Company, London, 1997.
14. Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24: 795–802, 2005.
15. Kwatra, V., Schodl, A., Essa, I., Turk, G., and Bobick, A. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.
16. Oliva, A. and Torralba, A. Building the gist of a scene: The role of global image features in recognition. Visual perception. *Progr. Brain Res.*, 155:23–36, 2006.
17. Perez, P., Gangnet, M., and Blake, A. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.
18. Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. LabelMe: A database and web-based tool for image annotation. Technical Report, MIT, 2005, 2005.
19. Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006.
20. Sun, J., Yuan, L., Jia, J., and Shum, H.-Y. Image completion with structure propagation. *ACM Trans. Graph.*, 24(3):861–868, 2005.
21. Torralba, A., Fergus, R., and Freeman, W. T. Tiny images. Technical Report MIT-CSAIL-TR-2007-024, 2007.
22. Torralba, A., Murphy, K. P., Freeman, W. T., and Rubin, M. A. Context-based vision system for place and object recognition. *ICCV*, 2003.
23. Wertheimer, M. Laws of organization in perceptual forms (partial translation). In: W. Ellis, editor, *A Sourcebook of Gestalt Psychology*, pages 71–88. Harcourt Brace and Company, London, 1938.
24. Wexler, Y., Shechtman, E., and Irani, M. Space-time video completion. *CVPR*, 01:120–127, 2004.
25. Wilczkowiak, M., Brostow, G. J., Tordoff, B., and Cipolla, R. Hole filling through photomontage. *BMVC*, 492–501, July 2005.

James Hays and Alexei A. Efros (jhhays@cs.cmu.edu, efros@cs.cmu.edu)
Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA, USA

A previous version of this paper was published in *ACM Transactions on Graphics*, Vol. 26, Issue 3 (July 2007).

DOI:10.1145/1400181.1400203

Technical Perspective
New Developments in Graph Partitioning

By Éva Tardos

ARORA, RAO, AND VAZIRANI discuss the most important developments in approximation algorithms over the last two decades. Graph partitioning has played a central role in algorithms research, both as an important problem with many applications and as a fertile breeding ground for interesting and deep algorithmic techniques. The paper reviews several related papers, starting with the breakthrough of an improved $O(\sqrt{\log n})$ approximation algorithm based on semidefinite programming and also including a faster combinatorial version using expander flows.

Graph partitioning is a natural algorithmic primitive in many settings, including data clustering, divide-and-conquer approaches, parallel computing architectures and algorithms, and packet routing in distributed networks. A well-understood graph cut problem is to partition the vertices into two sets such that there are few edges crossing between the two sets. A smallest cut can be found in polynomial time via a number of different methods, including traditional network flows. While minimal cuts have many applications, they are not useful primitives for the divide-and-conquer type applications; such cuts often result in extremely unbalanced partitions. Graph partitioning refers to a broad class of partitioning problems that seek cuts with few crossing edges, where the two sides of the partition are more balanced.

Finding an optimal balanced partition is intractable (NP-hard), so for the last 40 years attention has been focused on finding nearly optimal partitions; this has resulted in the development of deep techniques in approximation algorithms. The first major breakthrough result appeared in a 1988 paper by Leighton and Rao: an efficient algorithm that finds almost-balanced partitions is an $O(\log n)$ approximation algorithm; that is, where the number of edges that cross the cut is no more than $O(\log n)$ times more than the optimal, where n is the number of vertices of the graph. The authors used a linear programming relaxation of the balanced partitioning problems. Solving the linear program results in a *fractional cut*, where each edge is assigned a fractional extent to which it belongs to the cut. The algorithm provides an interesting technique that "rounds" the fractional values to 0 or 1 and hence turns the fractional cut into a real cut. Since that paper, the linear programming relaxations and rounding paradigm proved a powerful tool in approximation algorithms with a wide range of applications.

The $O(\sqrt{\log n})$ approximation algorithm presented here is a long-awaited improvement in the graph partitioning guarantee. The paper builds on the improved understanding of the Leighton and Rao technique gained over the years. In 1994, Linial, London, and Rabinovich brought to light an intriguing connection between graph partitioning problems and embeddings into high-dimensional metric spaces. A cut can be thought of as a simple embedding into a line: vertices on one side of the cut are mapped to 0, and those on the other side to 1, and any mapping of the points to the one-dimensional metric can be thought of as combinations of cuts (where the two sides of the cut are the nodes mapped to points $\leq\alpha$ and the nodes mapped to points $\geq\alpha$ for some value α). Leighton and Rao's linear program is a relaxation of this one-dimensional metric space to an arbitrary metric. While finding the best cut and the best embedding into the one-dimensional metric are NP-complete, finding the best embedding to an arbitrary metric space can be done in polynomial time. One can derive the $O(\log n)$ approximation by mapping the metric space obtained by the linear program into the L_1 metric space so that the distances are approximately preserved, and then decomposing the L_1 metric space as a product of one-dimensional metric spaces. The new paper uses Euclidian metrics and build on a semidefinite programming technique introduced by Goemans and Williamson that allows them to optimally embed the nodes of a graph into a high-dimensional Euclidian metric.

A well-known disadvantage of using general-purpose linear programming solvers and semidefinite programs in approximation algorithms is that they are slow. The authors here review the faster $O(\sqrt{\log n})$ approximation algorithm of Arora, Hazan, and Kale based on flows and the multiplicative update method. The connection of cut problems to certain multicommodity flow problems pioneered by Shahrokhi and Matula, who note that the edges congested by the flow are the ones good to cut. There have since been great developments in solving this class of linear programs efficiently by using variants of the multiplicative update method. The flow problem naturally related to balanced cut is that of sending one unit of flow between every pair of nodes in the graph. Edges in a small balanced cut will have to be congested as $O(n^2)$ flow wants to cross the cut. Unfortunately, the best routing of flow can have $O(\log n)$ more congestion in some graphs than the congestion predicted by small cuts, thus limiting the quality of the approximation bound achieved.

To speed the computation, Leighton and Rao replace this all-pairs flow with a sparse random pairing, only sending flow between $O(n)$ randomly selected source-sink pairs. If the graph of the source-sink pairs is an expander, then this smaller flow problem works just as well in the algorithm. The authors achieve their improved approximation by showing an expander exists whose flow can be routed with $O(\sqrt{\log n})$ less congestion than random expanders.

Arora, Rao, and Vazirani review an impressive collection of results, bringing to bear a wide array of tools on an important problem. Combining tools from high-dimensional metric spaces with ideas from expander flows, sparsification, and multiplicative update methods results in a simple and fast method that achieves the improved $O(\sqrt{\log n})$ approximation guarantee. C

Éva Tardos (eva@cs.cornell.edu) is chair of and a professor in the Department of Computer Science at Cornell University, Ithaca, NY.

DOI:10.1145/1400181.1400204

Geometry, Flows, and Graph-Partitioning Algorithms

By Sanjeev Arora, Satish Rao, and Umesh Vazirani

1. INTRODUCTION

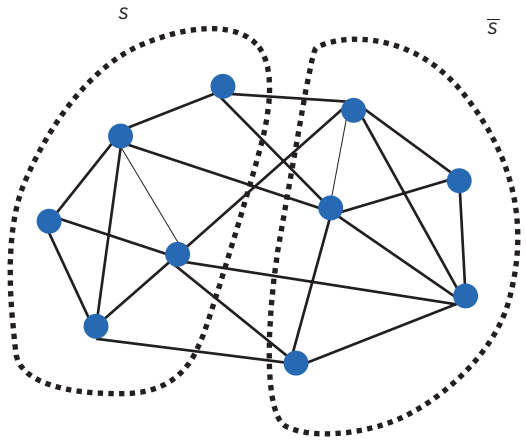
"Graph partitioning" refers to a family of computational problems in which the vertices of a graph have to be partitioned into two (or more) large pieces while minimizing the number of the edges that cross the cut (see Figure 1). The ability to do so is a useful primitive in "divide and conquer" algorithms for a variety of tasks such as laying out very large circuits on laying out very large circuits on silicon chips and distributing computation among processors. Increasingly, it is also used in applications of clustering ranging from computer vision, to data analysis, to learning. These include finding groups of similar objects (customers, products, cells, words, and documents) in large data sets, and image segmentation, which is the first step in image analysis. Unfortunately, most graph-partitioning problems are NP-hard, which implies that we should not expect efficient algorithms that find optimal solutions. Therefore researchers have resorted to *heuristic* approaches, which have been implemented in several popular freeware codes and commercial packages.

The goal of this paper is to survey an interesting combination of techniques that have recently led to progress on this problem. The original motivation for this work was theoretical, to design algorithms with the best provable *approximation guarantees*.* Surprisingly, these ideas have led also to a new framework for designing very fast and practically viable algorithms for this problem. On the theoretical end, the ideas have also led to a breakthrough in a long-standing open question on metric space embeddings from the field of function analysis, and new algorithms for semidefinite programming. These are all surveyed in this paper.

We will actually describe two disparate-seeming approaches to graph partitioning which turn out, surprisingly, to be related. The first approach is geometric, and holds the key to actually analyzing the quality of the cut found by the algorithm. The second approach involves routing *flows* in the graph, which we will illustrate using *traffic flows* in a road network. This approach holds the key to designing algorithms that run fast. The relationship between these two approaches derives from the fact that they are (roughly) dual views of each other, thus resulting in algorithms which are both fast and also produce high-quality cuts.

Below, we first sketch the two approaches, and give more details in Sections 2 through 4.

Figure 1: A graph and a partition into two subsets S, $\bar{S}$. In this case, the two subsets have equal number of vertices; such a partition is called a *bisection*. The number of edges crossing the cut is 7. If the number of vertices on the two sides is within a constant factor of each other (say, factor 2), then we call the partition *balanced*. Balanced partitions are useful in many applications.



1.1. Sketch of the geometric approach

Let us start by describing the geometric approach. We draw the graph in some geometric space (such as the unit disk in two-dimensional Euclidean space $\Re^2$), such that the average length of an edge is short—i.e., the distance between its endpoints is small—while the points are spread out. More precisely, we require that the distance between the average pair of vertices is a fixed constant, say 1, while the distance between the average adjacent pair of vertices is as small as possible. We will refer to this as an *embedding* of the graph in the geometric space.

The motivation behind the embedding is that proximity in the geometric space roughly reflects connectivity in the graph, and a good partition of the graph should correspond to separating a large area by cutting along a geometric curve. Indeed, given the properties of the embedding, even a "random partition" of the space by a simple line or curve should work well! The typical edge is unlikely to be cut by a random partition since it is short while a typical pair of vertices is likely to be separated since the distance between them is large. This means that the expected number of vertices on each side of the cut is large while the expected number of edges crossing the cut is small.†

The actual space that our algorithm will "draw" the graph in is not two-dimensional Euclidean space $\Re^2$, but

* We say that the *approximation guarantee* of the algorithm is C if given any graph in which the best cut has k edges, the algorithm is guaranteed to find a cut which has no more than $C \cdot k$ edges. Sometimes, we also say the algorithm is a *C-approximation*.

the surface of the unit sphere in an n-dimensional Euclidean space, where n is the number of graph vertices. Moreover, the "distance" between points in this space is defined to be the square of the Euclidean distance. This means that we draw the graph so that the sum of squares of the lengths of the edges is as small as possible, while requiring that the square of the distance between the average pair of points is a fixed constant, say 1.

There are some important additional constraints that the geometric embedding must satisfy, which we have suppressed in our simplified outline above. These are described in Section 2, together with details about how a good cut is recovered from the embedding (also see Figure 3 in Section 2). In that Section, we also explain basic facts about the geometry of n-dimensional Euclidean space that are necessary to understand the choice of parameters in the algorithm. The proof of the main geometric theorem, which yields the $O\left(\sqrt{\log n}\right)$ bound on the approximation factor achieved by the algorithm, makes essential use of a phenomenon called *measure concentration*, a cornerstone of modern convex geometry. We sketch this proof in Section 6.

The main geometric theorem has led to progress on a long-standing open question about how much distortion is necessary to map the metric space l_1 into a Euclidean metric space. This result and its relation to the main theorem are described in Section 7.

1.2. Sketch of the flow-based approach

We now describe the flow-based approach that holds the key to designing fast algorithms. In Section 3, we mention why it is actually dual to the geometric approach. To visualize this approach imagine that one sunny day in the San Francisco Bay area, each person decides to visit a friend. The most congested roads in the resulting traffic nightmare will provide a sparse cut of the city: most likely cutting the bridges that separate the East bay from San Francisco.*

More formally, in the 1988 approach of Leighton and Rao[14] (which gives an $O(\log n)$-approximation to graph-partitioning problems), the flow routing problem one solves is this: route a unit of flow between every pair of vertices in the graph so that the congestion on the most-congested edge is as small as possible; i.e., route the traffic so that the worst traffic jam is not too bad. A very efficient algorithm for solving this problem can be derived by viewing the problem as a contest between two players, which we can specify by two (dueling) subroutines. Imagine that a traffic planner manages congestion by assigning high tolls to congested edges (streets), while the flow player finds the cheapest route along which to ship each unit of flow (thereby avoiding congested streets). In successive rounds, each player adjusts the tolls and routes respectively to best respond to the opponent's choices in the previous round. Our goal is to achieve an equilibrium solution where the players' choices are best responses to each other. Such an equilibrium corresponds to a solution that minimizes the maximum congestion for the flow player. The fact that an equilibrium exists is a consequence of linear programming duality, and this kind of two-player setting was the form in which von Neumann originally formulated this theory which lies at the foundation of operations research, economics, and game theory.

Indeed, there is a simple strategy for the toll players so that their solutions quickly converge to a nearly optimal solution: assign tolls to edges that are exponential in the congestion on that edge. The procedure in Figure 3 is guaranteed to converge to solution where the maximum congestion is at most $(1 + \varepsilon)$ times optimal, provided $\mu \geq 1$ is a sufficiently close to 1. The number of iterations (i.e., flow reroutings) can also be shown to be proportional to the flow crossing the congested cut.

But how do we convert the solution to the flow routing problem into a sparse cut in the graph? The procedure is very simple! Define the distance between a pair of vertices by the minimum toll, over all paths, that must be paid to travel between them. Now consider all the vertices within distance R of an arbitrary node v. This defines a cut in the graph. It was shown by Leighton and Rao,[14] that if the distance R is chosen at random, then with high probability the cut is guaranteed to be within $O(\log n)$ times optimal. The entire algorithm is illustrated in the figure below.

Here is another way to think about this procedure: we may think of the tolls as defining a kind of abstract "geometry" on the graph; a node is close to nodes connected by low-toll edges, and far from nodes connected by large toll edges. A good cut is found by randomly cutting this abstract space. This connection between flows and geometry will become even stronger in Section 3.

In our 2004 paper,[6] we modified the above approach by focusing upon the choice of the traffic pattern. Instead of routing a unit of flow between every pair of vertices—i.e., a traffic pattern that corresponds to a complete graph—one can obtain much better cuts by carefully choosing the traffic

Figure 2: Parameter μ depends upon ε, which specifies the accuracy with which we wish to approximate the congestion.

ROUTE PATHS AND CUT Input: $G = (V, E)$ Maintain: d(·) on the edges of G. Initially $d(e) = 1$.

1. Do until the maximum $d(e)$ is n,
   (a) Choose a random (i, j) pair.
   (b) Find a shortest path, p, from i to j.
   (c) Multiply $d(e)$ on each edge of p by μ.
2. Choose a value δ randomly and an arbitrary node u and return the set of nodes within distance δ of u.

† It may appear strange to pick a random partition of this geometric space instead of optimizing this choice. Though some optimization is a good idea in practice, one can come up with worst-case graphs where this fails to provide substantial improvements over random partitions. A similar statement holds for other algorithms in this paper that use random choices.

* Of course, unlike San Francisco's road system, a general graph cannot be drawn in the Euclidean plane without having lots of edge crossings. So, a more appropriate way of picturing flows in a general graph is to think of it as a communications network in which certain vertex pairs (thought of as edges of the "traffic graph") are exchanging a steady stream of packets.

pattern. We showed that for every graph there is a traffic pattern that reveals a cut that is guaranteed to be within $O\left(\sqrt{\log n}\right)$ times optimal. This is proved through a geometric argument and is outlined in Section 3. An efficient algorithm to actually find such a traffic pattern, the routing of the flow, and the resulting cut was discovered by Arora, Hazan, and Kale.[2] The resulting $\tilde{O}(n^2)$ implementation of an $O\left(\sqrt{\log n}\right)$ approximation algorithm for sparse cuts is described in Section 4. Thus, one gets a better approximation factor relative to the Leighton–Rao approach without a running time penalty. Surprisingly, even faster algorithms have been discovered.[4,12,16] The running time of these algorithms is dominated by very few calls to a single commodity max-flow procedure which are significantly faster in practice than the multicommodity flows used in Section 4. These algorithms run in something like $O(n^{1.5})$ time, which is the best running time for graph partitioning among algorithms with proven approximation guarantees. We describe these algorithms and compare them to heuristics such as METIS in Section 5.

2. THE GEOMETRIC APPROACH AND THE ARV ALGORITHM

In this Section, we describe in more detail the geometric approach for graph partitioning from our paper.[6] Henceforth refered to as the ARV algorithm. Before doing so, let us try to gain some more intuition about the geometric approach to graph partitioning. We will then realize that the well-known spectral approach to graph partitioning fits quite naturally in this setting.

2.1. The geometric approach

In Section 1, we introduced the geometric approach by saying "We draw the graph in some geometric space (such as the two-dimensional Euclidean space $\Re^2$). . . ." Well, let us consider what happens if we draw the graph in an even simpler space, the real line (i.e., $\Re$). This calls for mapping the vertices to points on the real line so that the sum of the (Euclidean) distances between endpoints of edges is as small as possible, while maintaining an average unit distance between random pairs of points. If we could find such a mapping, then cutting the line at a random point would give an excellent partition. Unfortunately, finding such a mapping into the line is NP-hard and hence unlikely to have efficient algorithms.

The popular *spectral method* does the next best thing. Instead of Euclidean distance, it works with the square of the Euclidean distance—mapping the vertices to points on the real line so the sum of the squares of edge lengths is minimized, while maintaining average unit squared distance between a random pair of points. As before, we can partition the graph by cutting the line at a random point. Under the squared distance, the connection between the mapping and the quality of the resulting cut is not as straightforward, and indeed, this was understood in a sequence of papers starting with work in differential geometry in the 1970s, and continuing through more refined bounds through the 1980s and 1990s.[1,8,18] The actual algorithm for mapping the points to the line is simple enough that it might be worth describing here: start by assigning each vertex i a random point x_i in the unit interval. Each point x_i in parallel tries to reduce the sum of the squares of its distance to points corresponding to its adjacent vertices in the graph, by moving to their center of mass. Rescale all the x_i's so that the average squared distance between random pairs is 1, and repeat. (To understand the process intuitively, think of the edges as springs, so the squared length is proportional to the energy. Now the algorithm is just relaxing the system into a low-energy state.) We note that this is called the *spectral method* (as in *eigenvalue spectrum*) because it really computes the second largest eigenvector of the adjacency matrix of the graph.

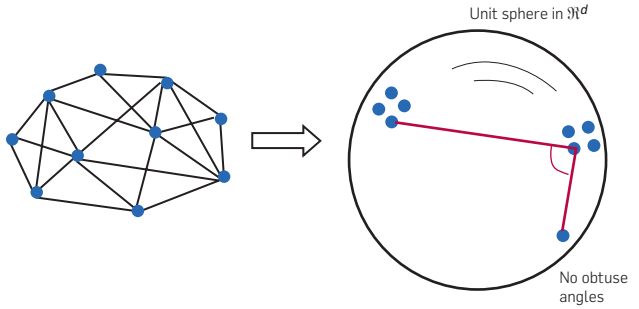
2.2. The ARV algorithm

The algorithm described in Section 1.1 may be viewed as a high-dimensional analog of the spectral approach outlined above. Instead of mapping vertices to the line, we map them to points on the surface of the unit sphere in n-dimensional Euclidean space. As in the spectral method, we work with the square of the distance between points and minimize the sum of the squares of the edge lengths while requiring that the square distance between the average pair of points is a fixed constant, say 1. The sum of squares of lengths of all edges is called the *value* of the embedding.

How closely does such an embedding correspond to a cut? The ideal embedding would map each graph vertex to one of two antipodal points on the sphere, thus corresponding to a cut in a natural way. In fact, the value of such an embedding is proportional to the number of edges crossing the cut. It follows that the minimum-value two-point embedding corresponds to an optimal partitioning of the graph.

Unfortunately, the actual optimal embedding is unlikely to look anything like a two-point embedding. This is because squaring is a convex function, and typically we should expect to minimize the sum of the squared lengths of the edges by just sprinkling the vertices more or less uniformly on the sphere. To reduce this kind of sprinkling, we require the embedding to satisfy an additional constraint (that we alluded to in Section 1.1): the angle subtended by any two points on the third is not obtuse. The equivalent Pythogorean way of saying this is that the squared distances must obey the *triangle inequality*: for any triple of points u,

Figure 3: A graph and its embedding on the *d*-dimensional Euclidean sphere. The triangle inequality condition requires that every two points subtend a non-obtuse angle on every other point.



$v, w, |u-v|^2 + |v-w|^2 \geq |u-w|^2$. The two-point embedding remains viable since it satisfies this constraint. By contrast, the spectral method described above fails to meet this condition, since any three distinct points on a line form an obtuse triangle. (See the accompanying box for a discussion of the hypercube, which is a graph that provides an intuitive picture about the nature of this constraint.)

The conditions satisfied by the embedding are formally described in Figure 4. An embedding satisfying these conditions in a sufficiently high-dimensional space can be computed in polynomial time by a technique called SDP. A semidefinite program is simply a linear program whose variables are allowed to be certain quadratic functions. In our case, the variables are squared distances between the points in $\Re^d$. Indeed, researchers have known about this approach to graph partitioning in principle for many years (more precisely since[10,15]), but it was not known how to analyze the quality of cuts obtained. For a survey of uses of SDP in computing approximate solutions to NP-hard problems, see.[9]

The specific conditions in Figure 4 were chosen to express the semidefinite relaxation of the graph *min-bisection* problem, where the two sides of the cut have to be of equal size. In this case, a feasible solution is to map each partition of an optimal bisection to two antipodal points. Therefore, the optimal solution to the SDP provides a lower bound on the cost of the optimal bisection. How do we recover a good bisection from this embedding? This is a little more involved than the simple "random partition" technique described in the previous subsection.

The key to finding a good cut is a result from[6] that any embedding satisfying all the above conditions is similar to a two-point embedding in the following sense: given such an embedding $\{v_1, \ldots, v_n\}$, we can efficiently find two disjoint "almost antipodal" sets, S and T, each with $\Omega(n)$ points that are at least $\Delta = \Omega\left(1/\sqrt{\log n}\right)$ apart. That is, every point in S has distance at least Δ from every node in T. Once the sets S and T are identified, there is a very simple procedure

Figure 4: Criteria for geometric embedding.

Let the ith vertex get mapped to the point v_i. The mapping must satisfy the following conditions:

- The points lie on the unit sphere:

$$\forall i \quad |v_i|^2 = 1$$

- The points are well spread:

$$\sum_{i<j} |v_i - v_j|^2 \geq 2\binom{n}{2}$$

- Satisfy the triangle inequality:

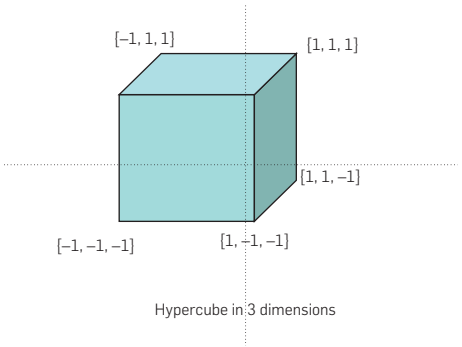
$$\forall i, j, k \; |v_i - v_j|^2 + |v_j - v_k|^2 \geq |v_i - v_k|^2$$

- Edges are short:

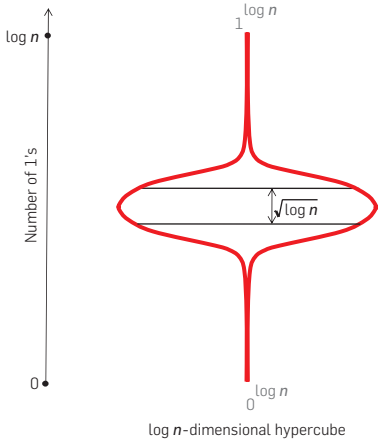
$$\min \sum_{\{ij\} \in E} |v_i - v_j|^2$$

EXAMPLE: THE HYPERCUBE

Insisting that the embedding of the graph into $\Re^d$ satisfies the triangle inequality is quite a severe constraint. Indeed, on the surface of a unit sphere in d dimensions, the maximum number of distinct points that can satisfy this condition is $2d$. An extreme example is the d-dimensional hypercube, which has exactly $n = 2^d$ points. The points are identified with vectors in $\{-1, +1\}^d$, and these vectors obey the triangle inequality since they do so coordinate wise.



Hypercube in 3 dimensions



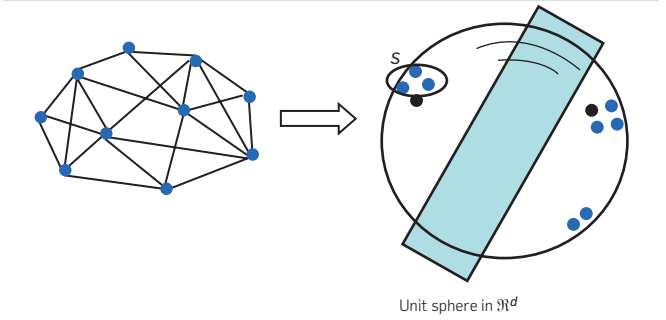
log n-dimensional hypercube

Each point of the d-dimensional hypercube is connected by an edge to exactly d other points—those that differ from it in exactly one coordinate. The optimal bisection cut corresponds to any one of the d dimensions: separate the points according to whether that particular coordinate is −1 or +1. The number of edges crossing this cut is exactly $n/2 = 2^{d-1}$, one for each 1-pair.

Another natural way of bisecting the d-dimensional hypercube is by a level cut. Imagine that we arrange the points on levels according to the sum of the coordinates. There are exactly $d + 1$ levels where the jth level has exactly $\binom{d}{j}$ points. The bisection cut separating the lowest d/2 levels from the remaining $d/2$ levels has $\Theta(\sqrt{d}2^d)$ edges (which is $\Theta\left(\sqrt{d}\right) = \Theta\left(\sqrt{\log n}\right)$ factor worse than optimal bisection cut). This is because the middle level has roughly $2^d/\sqrt{d}$ points, and each point has $d/2$ edges that cross the cut. The interesting thing is to note that there was nothing special about the sum of coordinates, and indeed we could start with any point v at the lowest level, and then assign the rest of the points of the hypercube to levels according to their distance from v. The number of distinct level cuts is 2^{d-1} (choosing v or its "antipodal" point leads to the same level sets).

Our algorithm uses a random projection to define a cut. When run on the hypercube, this corresponds to finding a level cut. Thus, the algorithm fails to discover anything close to one of the d optimal dimension cuts. In this case, the algorithm's answer is indeed $\sqrt{\log n}$ factor away from optimal.

Figure 5: Procedure to produce a cut from the embedding of the graph on the sphere. The points in the thin slice in the middle are thrown out.



for finding a balanced partitioning of the vertices with few crossing edges.† This is done as follows: simply pick a random distance $0 \le r \le \Delta$, and select every vertex i such that v_i is within r of some point in S, i.e. for some $x \in S$, $|v_i - x|^2 \le r$. To see that this works, note that the probability that any edge e with length $l(e)$ is in the cut is at most $l(e)/\Delta$. Thus, the typical number of edges crossing such a cut is at most $1/\Delta$ times the total length of the edges (which, recall, is at most the size of the optimal cut). This yields an $O\left(\sqrt{\log n}\right)$ -approximation algorithm since $\Delta = \Omega\left(1/\sqrt{\log n}\right)$.

We now turn our attention to actually finding the almost antipodal sets S and T. Before we do so, it is instructive to understand an important geometric property of a high-dimensional Euclidean space $\Re^d$. Consider projecting the surface of the unit sphere on to a line u through its center. Clearly, the points on the sphere will project to the interval $[-1, 1]$. If we start from a uniformly random point v on the sphere, what is the distribution of the projected point (v, u) in $[-1, 1]$? It turns out that the projected point has a Gaussian distribution, with expectation 0 and standard deviation $1/\sqrt{d}$. This means that the expected squared distance of a projected point from the center is $1/d$, and a constant fraction of the surface of the sphere is projected at least $1/\sqrt{d}$ away from the origin. Another way to say all this is that if we consider slices of the sphere by parallel hyperplanes the surface area of the slices vary like a Gaussian according to distance of the slice from the center of the sphere.

Now returning to our procedure for identifying sets S and T from the embedding of the graph, we start by projecting the n points $v_1, \ldots, v_n$ on a randomly chosen line $\vec{u}$ through the center. We discard all points whose projection has absolute value less than $1/\sqrt{d}$. The remaining points form two sets P and N, according to whether their projection is positive or negative. By the Gaussian property above, the sets P and N have expected size $\Omega(n)$. Ideally, we would like to stop here and assert that every point in $x \in P$ is far from every point in $y \in N$.

Indeed, it is true (and easily checked) that with high probability over the choice of the line on which we are doing the projection, $|x - y|^2 \ge 1/\log n$. However, we actually want a greater separation of $\Delta = 1/\sqrt{\log n}$, so we enforce this condition by sequentially deleting pairs $\{x, y\}$ that violate it. The remaining points define two sets $S \subseteq P$ and $T \subseteq N$ which are Δ-separated. The difficult part is to show that S and T have cardinality $\Omega(n)$. For the interested reader, we have sketched the main ingredients of the proof of this fact in Section 6. Figure 6 below summarizes the resulting algorithm for finding a good cut.

Figure 6: Finding a good cut.

Given the embedding $v_1, \ldots, v_n$.

- Pick a random line $\vec{u}$ through the origin.
- Let $P = \left\{v_i : (v_i, u) \ge 1/\sqrt{d}\right\}$ and $N = \left\{v_i : (v_i, u) \le 1/\sqrt{d}\right\}$
- Discard pairs of points from $x \in P$ *and* $y \in N$ such that $|x - y|^2 \le \Delta = 1/\sqrt{\log n}$.

  Call the resulting sets S and T.
- Choose random $0 \le r \le \Delta$, and let $X = \left\{i : |v_i - x|^2 \le r\right\}$ for some $x \in S$.
- Output partition $(X, V - X)$.

3. EXPANDER FLOWS

In Section 1.2, we described a dual flow-based approach to partitioning a graph. In this section, we describe how to extend that to the expander flow framework, which leads to an efficient implementation of the $O\left(\sqrt{\log n}\right)$ approximation algorithm. Before we can describe this new framework, we must introduce a number of new concepts. Let us start by recalling the San Francisco Bay Area traffic nightmare scenario of Section 1.2, where we chose to route a traffic pattern described by a complete graph. We described an efficient algorithm for computing good routes for the traffic to minimize the worst traffic jams. We also showed how to use the resulting information to actually find a good partition for the county's road network.

In this section, we will focus on the choice of traffic pattern. We will see that choosing the traffic pattern in a clever way will reveal a much better partition of the road network. To begin with, notice that if the traffic pattern is very local—for example, if each person visits a friend who lives a few blocks away—then the precise location of the congested streets has little information about the actual bottlenecks or sparse cuts in the underlying road network. So we start by formally understanding what kinds of traffic graphs will reveal the kind of information we seek. To be more concrete, by the traffic graph, we mean the weighted graph in which nodes i and j have an edge of weight c_{ij} if they have a flow rate c_{ij} between them.

We require the traffic graph to be "expanding," that is, the total traffic flow out of every subset of nodes is proportional to the number of nodes in the subset. In more

† Note that we started off trying to find a partition into sets of *equal* size, but this method will only yield a partition in which the two sets have sizes within a factor 2 of each other.

mathematical language, we say that a cut $(S, V - S)$ is β-expanding, if the edges crossing this cut have total weight at least β times $\min\{|S|, |V - S|\}$. A graph $H(V, F)$ is said to be a *β-expander*, for some constant β, if for every for subset $S \subseteq V$, the cut $(S, V - S)$ is β-expanding. Expander graphs have no small graph separators. Here, we will be interested in traffic graphs that are β-expanders for some constant β, and where the maximum degree of a node is at most d for some constant d. (The degree of a vertex is the sum of weights of all edges incident to it. We assume that the degree of each vertex is at least 1.) An important property of this class of graphs is that there is an efficient test to distinguish constant degree expander graphs from graphs with small separators. Indeed, this test is based on the spectral method, which works very well for constant degree expanders.

We said earlier that a clever choice of traffic graph will reveal a better way to partition the road network graph. How do we make this clever choice? The idea behind the new algorithm is to search for the best "expanding" traffic graph: which means among all expanding traffic graphs pick one that leads to the smallest traffic jams. This might seem counterintuitive. Shouldn't locating the worst cut correspond to finding a traffic pattern that leads to the worst traffic jams?

To understand this point more clearly, let us introduce some notation. Let $G(V, E)$ be the graph that we wish to partition, and $H(V, F)$ be an expander graph on the same vertex set that we use as the traffic graph to route a traffic flow in G. Recall that this means that one unit of flow must be shipped between each pair of nodes connected by an edge in F. (In fact, the algorithm must allow fractional edges in F, which our description will ignore.) Suppose α is such that the sparsest cut in $G(V, E)$ separates the set of vertices $S \subseteq V$ from $V - S$, by cutting only $\alpha|S|$ edges. Thus α is a measure of how sparse the optimum cut is. Now, regardless of how the flow corresponding to $H(V, F)$ is routed in $G(V, E)$, at least $|S|$ units of flow (traffic) must be routed between S and $V - S$. Since this must be routed through a bottleneck containing only $\alpha|S|$ edges, it follows that the worst edge-congestion must be at least $1/\alpha$. The real issue in designing an approximation algorithm for sparsest cuts is this: how much larger can the edge-congestion be than this lower bound of $1/\alpha$? If it is no larger than C/α, then it will follow that we can approximate the sparsest cut (at least its numerical value) to within a factor of C in the worst case. To obtain the best results, we must try to minimize C. In other words, we wish to pick an expanding traffic graph which leads to the smallest traffic jams.

The main result in ARV[6] on expander flows states that for any graph $G(V, E)$ there is an expander graph $H(V, F)$ that can be routed in $G(V, E)$ with congestion at most $O\left(\sqrt{\log n}/\alpha\right)$. Moreover, using the powerful computational hammer of the ellipsoid algorithm for linear programming, the expander graph $H(V, F)$ as well as its routing in $G(V, E)$ minimizing edge-congestion can be computed in polynomial time. This gives an algorithm for sparsest cuts, different from the geometric one presented earlier, yet achieves the same $O\left(\sqrt{\log n}\right)$ approximation factor in the worst case. It also serves as a starting point for a new framework for designing very efficient algorithms for finding sparse cuts.

In the rest of this section, we will sketch the proof of the main expander flows result of ARV.[6] To do so, we shall generalize the router/toll-collector game from Section 1.2 to incorporate the selection of an expanding traffic graph. The existence of an expanding traffic graph that achieves the desired $O\left(\sqrt{\log n}\right)$ bound will follow from understanding the equilibrium of the game. Since we are interested in the equilibrium solution, rather than detailing an efficient procedure for getting to equilibrium, it is easier to formulate the game as lasting for a single round.

In the generalized game, the toll-collector not only specifies a toll for each edge (the sum of all edge tolls is n, the number of vertices in G), but also a set of prizes. Each prize is associated with a cut in the graph, and is collected when a path crosses the cut. The number of possible cuts is 2^n of course, and the toll collector selects some subset of cuts to place prizes on, such that the sum of all cut prizes is 1. We associate with each pair of vertices the total prize for moving from one vertex to another—the sum of cut prizes for all cuts that separate the two vertices. The task of the routing player is to select a pair of vertices, separated by total cut prize of at least β (where β is a given constant), such that the total toll paid in moving between these vertices is as small as possible. The goal of the toll player is to assign edge tolls and cut prizes to maximize the toll paid by the routing player. It can be shown using linear programming duality that this payoff is essentially the congestion of the best expander flow. The main idea is that the flow player's optimal strategy is specified by a probability distribution over pairs of vertices, which we may view as defining the desired expander graph (for any balanced cut, a random edge from this graph must cross the cut with probability at least β).

Let us consider how the toll player can maximize his take. Assume that $G(V, E)$ has a balanced separator with αn edges crossing the cut. Then by assigning a prize of 1 to this cut and a toll of $1/\alpha$ on each edge of this cut, the toll player can force the routing player to pay at least $1/\alpha$, simply because the routing player is forced to route flow between two vertices on opposite sides of this optimal cut. Can the toll player force the route player to pay a lot more? We show that he cannot force him to pay more than $O\left(\sqrt{\log n}/\alpha\right)$: no matter how the toll player determines tolls and picks a set of cuts, the routing player can always pick a pair of vertices which are separated by a β fraction of the cuts, and such that the cheapest path between them costs $O\left(\sqrt{\log n}/\alpha\right)$. The proof of this fact relies upon the geometric view from the previous section.

To make this connection, we start by viewing the set of cuts as defining a hypercube—each cut corresponds to a dimension, and each vertex gets a ± 1 label depending upon which side of the cut it lies. It is natural to associate each vertex in the graph with a vertex of the d-dimensional hypercube, where d is the number of cuts. We observed in the previous section that the d-dimensional hypercube can be embedded on the surface of a unit sphere in $\Re^d$ and satisfy the triangle inequality. By the main geometric theorem of ARV, it follows that there are two large almost antipodal

sets S, T, each with $O(n)$ vertices, such that every pair of vertices $x \in S$ and $y \in T$ is at least $1/\sqrt{\log n}$ apart. In our example, this means that there are large sets of vertices S, T such that every pair of vertices $x \in S$ and $y \in T$ lies on opposite sides of at least $1/\sqrt{\log n}$ fraction of the cuts. By a simple averaging argument, since every cut separating S from T has at least αn edges, there must be a pair of vertices x and y which are connected by a path of cost at most $1/\alpha$.

We are not quite done yet, since we must actually exhibit a pair of vertices that lie on opposite sides of β fraction of cuts. So far, we have only exhibited a pair of closely vertices that lie on opposite sides of $1/\sqrt{\log n}$ fraction of the cuts. The last step in the proof is to piece together a sequence of $O\left(\sqrt{\log n}\right)$ pairs to obtain a pair of vertices that lie on opposite sides of β fraction of the cuts, and which are connected by a path of cost $O\left(\sqrt{\log n}/\alpha\right)$. To carry out this last step, we actually have to appeal to the internal structure of the proof establishing the existence of the antipodal sets S and T. This "chaining" argument has found other uses in other research in the last few years.

3.1. The duality connection

There is a notion of duality for SDP and an expander flow turns out to be a type of dual solution to the semidefinite program outlined in Figure 1. Thus as in all settings involving duality, a dual solution can be seen as "certifying" a lower bound on the optimal value of the primal. Thus, the expander flow is a way to certify a lower bound on the size of the optimal cut.

4. AHK ALGORITHM FOR FINDING EXPANDER FLOWS

Arora, Hazan, and Kale (AHK)[2] managed to turn the 1-round game described in the previous section into an efficient algorithm, specifically, by designing efficient strategies for the toll player and the routing player.

During the AHK algorithm, the routing player maintains a traffic graph and the toll player maintains edge tolls and cut prizes as above. At each round, the toll player does spectral partitioning on the current traffic graph, finds a sparse cut in it, and places some nonzero prize on it. (This requires adjusting down the prizes on existing cuts because the net sum has to stay 1.) The routing player then finds pairs of nodes as mentioned in Section 3—namely, a pair whose prize money is at least β and whose edge-toll-distance is smallest—and adds the resulting paths to the traffic graph (again, this requires adjusting the values of the existing edges in the traffic graph because the total degree of each node in the traffic graph is constant). The game finishes when the traffic graph is a β-expander, a condition that can be checked by the spectral methods mentioned earlier.

With some work, each round can be implemented to run in $O(n^2)$ time. The key to the performance of the algorithm lies in the fact that the readjustment of tolls can be done in a way so that the game finishes in $O(\log n)$ rounds, which results in a running time of $O(n^2 \log n)$. This uses a feasible version of von Neumann's min–max theorem that Arora, Hazan, and Kale (in a separate survey paper[3]) call the *multiplicative weight update rule*: the toll player updates tolls on the edges and cuts accordingly by always multiplying or dividing by a fixed power of the quantity $(1 + \varepsilon)$ where ε is small. (This updated rule has been around for five decades and been rediscovered in many settings including convex optimization and machine learning; see[3] for a survey.) They set up the game with some care so that the toll on an edge at any time is an exponential function of the flow routed through it, which is where the logarithmic bound on the number of rounds comes from.

4.1. Faster algorithms for semidefinite programs

Motivated by the above discoveries, Arora and Kale[4] have designed a new combinatorial approach for computing approximate solutions to SDPs. This is important because solving SDPs usually is quite slow in practice (though polynomial time in theory). Their idea is to define a multiplicative weight update rule for matrices, which is used in a primal-dual fashion to get a quick approximation. The algorithm has some formal similarities to the "random walk" idea described in Section 5. They obtain the best running times known for a host of SDP-based approximation algorithms. For finding cuts, one may prefer the algorithms of the next section on account of simplicity, though the Arora–Kale approach is comparable in running time and approximation factor. Again, we refer the reader to the survey.[3]

5. TOWARD PRACTICAL ALGORITHMS

So far, we have described a framework for improving the approximation factor or the quality of cut found by graph-partitioning algorithms. In this section, we describe an algorithm of Khandekar, Rao, and Vazirani[12] that modifies the framework to design a much faster algorithm for graph partitioning—its running time is bounded by a small number of calls to a single commodity max-flow subroutine. As in the AHK algorithm from the previous section, the new algorithm may also be viewed as a contest between two players, though the underlying game, called the cut-matching game, is slightly different.

The object of the cut-matching game, played over a sequence of rounds, is to construct an expanding traffic graph. Initially, the traffic graph is the empty graph on n vertices. In each round, the cut player chooses a bisection of the vertex set, and the matching player specifies a perfect matching that pairs up vertices across the bisection. The edges of the perfect matching are added to the traffic graph, and the game continues until the traffic graph has expansion greater than 1. The goal of the cut player is to minimize the number of rounds before the game ends. The matching player, of course, tries to prolong the game.

How does the cut-matching game relate to partitioning a given graph $G(V, E)$? Given a bisection, the matching player tries to find a perfect matching that can be routed through $G(V, E)$ with small congestion. This is accomplished by performing a single max-flow computation, and the corresponding min-cut is a candidate partition of $G(V, E)$ (see Figure 5).

The cut player uses a novel spectral type algorithm,

described in the box below, to find a bisection in almost linear time. It can be shown that by following this strategy the cut player can limit the number of rounds of the game to $O(\log^2 n)$. Now the best candidate partitioning among the $O(\log^2 n)$ rounds is the output of the algorithm, and can be shown to be within a $O(\log^2 n)$ factor of the optimal partition.

PROCEDURE: MATCHING PLAYER

Input: Bisection (S and $V - S$), of graph G.

- Run a concurrent flow computation between S and $V - S$.
  Formulate a flow problem where each node in S is a source and each node in $V - S$ is the sink of one unit of flow.
  The goal is to satisfy each source and sink while minimizing the maximum integral flow across any edge of G. This is a standard single commodity flow problem.
- Decompose the flow into $n/2$ source-sink paths to obtain the matching:
  Greedily follow a path of positive flow arcs from a source node u to some sink. Remove this path from the flow, and repeat.
- Output the matching.

PROCEDURE: CUT PLAYER

Input: Set $\{M_1, \ldots, M_T\}$ of perfect matchings.

1. Let v^0 be a random n-dimensional unit vector.
2. For $t = 1$ to T
   For each edge $\{i, j\} \in M_t$
   $-v^{t+1}(i) = (v^t(i) + v^t(j))/2$
3. Output cut $(S, V - S)$, where
   $S = \{i : v_T(i) \leq m\}$,
   with m being the median value of the $v_T(i)$'s.

The total running time of the matching player is bounded by $O(\log^2 n)$ invocations of single commodity max-flow computations. This is the dominant term in the running time of the algorithm, since the cut player runs in nearly linear time.

The overall algorithm starts by invoking the cut player with the empty set of matchings and invokes the cut player and matching player for $O(\log^2 n)$ rounds. Each invocation of the matching player results in a candidate partition of the graph given by the min-cut in the invocation of the max-flow procedure. The algorithm outputs the best candidate partition over all the rounds. It was shown in[12] that the approximation factor achieved by this partition is $O(\log^2 n)$. As mentioned in Section 4, a different primal-dual algorithm (with some formal similarity to the above algorithm) was used[4] to improve the algorithm to yield an approximation factor of $O(\log n)$. This result was recently matched in[16] by an algorithm that stays within the framework of the cut-matching game. They also showed that the best approximation factor that can be obtained within the cut-matching framework is $\Omega(\sqrt{\log n})$. Designing a $O(\sqrt{\log n})$ approximation algorithm in the cut-matching framework remains an intriguing open question.

It is instructive to compare the flow-based algorithm described here to a clustering-based heuristic embodied in the widely used program METIS.[11] That heuristic proceeds by collapsing random edges until the resulting graph is quite small. It finds a good partition in this collapsed graph, and successively induces it up to the original graph, using local search. The flow-based algorithm may be viewed as a continuous version of this heuristic since each matching may be thought of as "virtually collapsing" each matched pair. In particular, early iterations of the flow-based algorithm will select matchings that mostly consist of random edges. Moreover, the cuts provided by the cut player will rarely partition the resulting connected components. Thus, it will proceed very much like METIS. Indeed, this algorithm might have an advantage over METIS since the "virtual collapsing" of edges in the actual sparse cut need not preclude finding this cut.

We point the reader interested in empirical results to Chris Walshaw's graph-partitioning benchmark (http://staffweb.cms.gre.ac.uk/ c.walshaw/partition/), which provides results of various heuristics on an interesting set of benchmarks. The impressive results for METIS can be compared to SDP (by Lang and Rao), which uses a simplified version of SDP with a max-flow cleanup. Though this resembles the max-flow-based algorithm described in this section, to our knowledge, the specific algorithms discussed in this paper have yet to be rigorously compared against METIS.

6. CORRECTNESS PROOF FOR THE GEOMETRIC APPROACH

In this section, we sketch a proof of the main geometric theorem of,[6] for any well-separated set of points on the surface of the unit sphere in $\Re^d$ that satisfy the triangle inequality, there are two linear-sized almost antipodal sets, S,T, that are $\Delta = 1/\sqrt{\log n}$ separated. The sketch here also incorporates a simplification due to Lee.[13]

Recall that our procedure for identifying these two sets S, T was to project the points on a random line $\vec{u}$ through the origin, discard all points whose projection has absolute value less than $1/\sqrt{d}$, and label the remaining points as being in P and N, according to whether their projection is positive or negative. We then discarded pairs of points from the two sets that were less than Δ separated, to finally obtain sets S and T.

For the theorem to fail, for most choices of directions $\vec{u}$, most points must be paired and discarded. Recall that for a pair to be discarded, the points must be Δ-close to each other while their projection on $\vec{u}$ must be at least $\varepsilon = 2\frac{\sigma}{\sqrt{d}}$ apart. These discarded pairs form a matching for that direction. Let us make a simplifying assumption that all n points are paired and discarded in each direction. (In the actual proof, we use an averaging argument to show that there are $\Omega(n)$ points such that in most directions a constant fraction of them is matched.)

The overall plan is, given a random direction $\vec{u}$, to piece

together a sequence of matched edges, each with a large projection onto $\vec{u}$, into a path., i.e., we find a sequence of points $x_1, \ldots, x_l$ with the property that on average $(x_i - x_{i+i}, u) \geq \epsilon/2$ and therefore $(x_1 - x_l, u) \geq l\,\epsilon/2$. On the other hand, since $|x_i - x_{i+1}|^2 \leq \Delta$, triangle inequality implies that $|x_1 - x_l|^2 \leq l\Delta$, and therefore $|x_1 - x_l| \leq \sqrt{l\Delta}$. Note that ϵ was chosen to be (roughly) the standard deviation of the projection length of a unit vector. Therefore, the projection length of $x_1 - x_l$ is $t = \sqrt{l/\Delta}$ many standard deviations from its mean. The probability that the projection is t standard deviations from its mean is at most $e^{-t2/2}$. So if the number of hops l in the path is $\Omega\left(\sqrt{\log n}\right)$, then the probability of this event is polynomially small. We get a contradiction by applying the union bound to the n^2 pairs of points.

The main challenge then is to piece together such a path. To find a path for a direction u, we clearly cannot limit ourselves to the matching in that direction. To understand how matchings for different directions relate to each other, we need two new ideas. Let us first introduce some notation.

Recall that each point v has a matched pair for half the directions (i.e., either for $\vec{u}$ or $-\vec{u}$). We say that v is ε-covered in these directions. And we say that v is (ε, δ)-covered if, v is ε-covered for atleast δ measure of directions.

The first idea is simple: if v is ε covered in direction $\vec{u}$, then it is almost ε-covered in every neighboring direction $\vec{u}'$. Quantitatively, let $|\vec{u} - \vec{u}'| \leq \gamma$ then v is $(\varepsilon - 2\gamma)$-covered in direction $\vec{u}'$. This fact follows from elementary geometry.

The second idea is to use measure concentration. Consider a set of points A on the surface of the unit ball in R^d. If the measure of A is at least δ for some constant δ, then the γ-neighborhood of A (i.e., all points within distance at most γ from some point in A) has measure almost 1. Quantitatively, the measure is approximately $1 - e^{-t^2/2}$ if $\gamma = t/\sqrt{d}$ (i.e., t standard deviations).

Returning to our proof, we have a point v that is ε-covered for a set of directions A of measure δ. Moreover, the covering points are within Δ of v, and so we are working with a ball of radius $\sqrt{\Delta}$ rather than 1. So, a $\gamma = t\sqrt{\Delta/d}$ neighborhood of A has measure $1 - e^{-t^2/2}$. So, v is also $(\varepsilon/2, \delta')$-covered for δ' very close to 1.

It might seem that we are almost done by the following induction argument: v is covered in almost all directions. Pick a direction $\vec{u}$ at random and let v' cover v in this direction. Now v' is also almost covered in almost all directions. Surely, v' has a good chance of being covered in direction $\vec{u}$ since it was chosen at random. Of course this argument, as it stands, is nonsense because the choice of v' is conditioned by the choice of $\vec{u}$. Here is a potential fix: for random $\vec{u}$ with high-probability v is covered both in direction $\vec{u}$ and $-\vec{u}$. If x and y are the covering points in these two directions, then $x - y$ has a projection on $\vec{u}$ that is twice as long. This time the problem is that most nodes v might yield the same pair of points x and y thus making it impossible to continue with an induction. For example, in a k-dimensional hypercube with $n = 2^k$, every point is $\Theta\left(\sqrt{\log n}\right)$ covered in almost all directions. However, for a particular direction, most points are covered by only a few in the tails of the resulting Gaussian distribution.

To actually piece together the path, we perform a more delicate induction. For random direction $\vec{u}$, node v is covered with high probability by some node x. Also in direction $-\vec{u}$, with probability 1/2, v is matched to some node y (i.e., $\{v, y\}$ formed a discarded pair). So with probability almost 1/2, y is now covered in direction u by x. Note that this time y takes this role only once for this direction, since $\{v, y\}$ is a matched pair.

To actually carry out, the induction requires some work to ensure that boosting using measure concentration can be carried out. Moreover, the size of the covered set does indeed drop. Indeed, other ideas need to be included to allow the induction to continue for $\sqrt{\log n}$ steps.

7. IMPLICATIONS FOR METRIC SPACE EMBEDDINGS

It is well known that there are different ways of measuring distances; for instance, l_1, l_2, and l_p, as well as more exotic methods such as Kullback–Leibler divergence. It is an important question in functional analysis to understand the interrelationship among these different measures. One frequent method of quantifying this is to look at the *minimum distortion* required to realize one distance function with the other distance function.

Let (X_1, d_1) and (X_2, d_2) be two metric spaces, by which we mean that d_i is a *distance function* on X_i that is nonnegative, and satisfies triangle inequality. An *embedding* of X_1 into X_2 is a mapping $f: X_1 \to X_2$. Its distortion is the minimum C such that

$$d_1(x_1, x_2) \leq d_2(f(x_1), f(x_2)) \leq C \cdot d_1(x_1, x_2) \qquad \forall x_1, x_2 \in X_1 \qquad (1)$$

The minimum distortion of X_1 into X_2 is the lowest distortion among all embeddings of X_1 into X_2. Though this notion had been studied extensively in functional analysis, its importance in algorithm design was realized only after the seminal 1994 paper of Linial, London, and Rabinovich.[15] Many algorithmic applications of this idea have been subsequently discovered.

It was realized before our work that the accuracy of the approach to cut problems involving SDPs (see Section 2) is closely related to analyzing the minimum distortion parameter linking different metric spaces (see the survey[17]). Indeed, the ARV analysis can be viewed as an embedding with low "average" distortion, and subsequent work of Chawla, Gupta, Raecke[7] and Arora, Lee, and Naor[5] has been built upon this observation. The final result is a near-resolution of an old problem in analysis: what is the minimum distortion required to embed an n-point l_1 space (i.e., where the points are vectors and distance is defined using the l_1 metric) into l_2? It was known for a long time that this distortion is at least $\sqrt{\log n}$ and at most $O(\log n)$. The Arora–Lee–Naor paper shows that the lower bound is close to truth: they give a new embedding whose distortion is $O\left(\sqrt{\log n} \log\log n\right)$.

We note here a connection to a general demand version of the sparsest cut problem: given a set of pairs of vertices $(s_1, t_1)(s_k, t_k)$, find the cut that minimizes the ratio of number of cut edges and the number pairs that are separated. An illustrative application is how to place few listening devices in a network to listen to many pairs of communicating agents; minimizing the ratio of listening devices to compromised pairs of agents.

The approximation ratio for the natural SDP relaxation

turns out to be equal to the distortion required to embed l_2^2 metrics into l_1. The embeddings of[5,7] actually embed l_2^2 into l_2 (which in turn embeds with no distortion into l_1), thus implying an $O\left(\sqrt{\log n}\log\log n\right)$ approximation algorithm for this general form of graph partitioning.

Acknowledgments

This work was supported by NSF grants MSPA-MCS 0528414, CCF 0514993, ITR 0205594, CCF-0515304, CCF-0635357, CCF-0635401 and ITR CCR-0121555.

References

1. Alon, N. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.
2. Arora, S., Hazan, E., and Kale, S. $O\sqrt{\log n}$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 238–247, Washington, DC, USA, 2004. IEEE Computer Society.
3. Arora, S., Hazan, E., and Kale, S. Multiplicative weights method: a meta-algorithm and its applications—a survey, 2005.
4. Arora, S. and Kale, S. A combinatorial, primal-dual approach to semidefinite programs. In *STOC '07: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 227–236, New York, NY, USA, 2007. ACM.
5. Arora, S., Lee, J. R., and Naor, A. Euclidean distortion and the sparsest cut. J. *Amer. Math. Soc.*, 21(1):1–21, 2008 (Electronic).
6. Arora, S., Rao, S., and Vazirani, U. Expander flows, geometric embeddings and graph partitioning. In *STOC '04: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 222–231, New York, NY, USA, 2004. ACM.
7. Chawla, S., Gupta, A., and Raecke, H. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *SODA '05: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 102–111, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
8. Cheeger, J. A lower bound for the smallest eigenvalue of the Laplacian. In *Problem in Analysis*, pages 195–199, 1970.
9. Goemans, M. X. Semidefinite programming and combinatorial optimization. In *Proceedings of the International Congress of Mathematicians, Vol. III (Berlin, 1998)*, pages 657–666, 1998 (electronic).
10. Goemans, M. X. and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.
11. Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Comput.*, 20(1):359–392, 1998.
12. Khandekar, R., Rao, S., and Vazirani, U. Graph partitioning using single commodity flows. In *STOC '06: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 385–390, New York, NY, USA, 2006. ACM Press.
13. Lee, J. R. On distance scales, embeddings, and efficient relaxations of the cut cone. In *SODA '05: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 92–101, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
14. Leighton, T. and Rao, S. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. *ACM (JACM)*, 46(6):787–832, 1999.
15. Linial, N., London, E., and Rabinovich, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.
16. Orecchia, L., Schulman, L. Vazirani, U., and Vishnoin, N. On partitioning graphs via single commodity flows. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17–20, 2008*, pages 461–470, 2008.
17. Shmoys, D. S. Cut problems and their application to divide and conquer. In D. S. Hochbaum, ed., *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1995.
18. Sinclair, A. and Jerrum, M. Approximate counting, uniform generation and rapidly mixing Markov chains (extended abstract). In *Graph-Theoretic Concepts in Computer Science (Staffelstein, 1987)*, volume 314 of *Lecture Notes in Comput. Sci.*, pages 134–148, Berlin, 1988. Springer.

Sanjeev Arora (arora@cs.princeton.edu) Computer Science Department, Princeton University, Princeton, NJ 08544, USA

Satish Rao (satishr@cs.berkeley.edu) Computer Science Department, UC, Berkeley, CA 94720, USA

Umesh Vazirani (vazirani@cs.berkeley.edu) Computer Science Department, UC, Berkeley, CA 94720, USA

A previous version of this paper, entitled "Expander Flows, Geometric Embeddings, and Graph Partitioning," was published in *Proceedings of the 36th Annual Symposium on the Theory of Computing* (Chicago, June 13–16, 2004).

CAREERS

Florida Institute of Technology
Department of Computer Sciences
Software Engineering Faculty Search
2009-2010

Florida Institute of Technology invites applications for faculty positions at all levels related to software engineering in the Department of Computer Sciences for the 2009-2010 academic year. Applicants with expertise that would reinforce the Department's reputation in areas such as software testing, maintenance & evolution, and system security are strongly encouraged to apply.

Applicants must have a Ph.D. in Software Engineering, Computer Science or in a closely related field. Junior candidates must show outstanding research and teaching potential. Senior candidates must have an exceptional research and teaching record. Salary is competitive and commensurate with appointment rank and qualifications.

The Department currently has 16 faculty members. New faculty joining in 2009 will be expected to assist in improving undergraduate and graduate education, developing quality research programs, attracting new funding, and strengthening our collaborations with industry, government, and other academic institutions. There are approximately 175 undergraduate students, 125 Master's students, and 25 Ph.D. students in the Computer Sciences programs. The Department is housed in the Olin Engineering Complex, with modern laboratories and multimedia classrooms.

The Department has significant research funding from multiple government agencies and corporations, both locally & nationally. For more information about the Department, please visit www.cs.fit.edu. Florida Tech is a selective, research-oriented, private university that attracts high-quality students. The University is located in Melbourne on Florida's Space Coast, one of the nation's most prosperous and growing high-tech areas that offers an exceptional quality of life. The campus occupies 130 tropical acres, including a picturesque 30-acre botanical garden. The campus is 5 minutes from the Indian River estuary, 10 minutes from the Atlantic Ocean, and 50 minutes from both Orlando and the Kennedy Space Center.

Applicants should send a letter of intent, curriculum vitae, research and teaching statements, and full contact information for at least three references, by email to faculty-search@cs.fit.edu, or by regular mail to: Faculty Search Committee, Department of Computer Sciences, Florida Institute of Technology, 150 W. University Blvd., Melbourne, FL 32901.

Review of applications will begin immediately and continue until the positions are filled. Florida Tech is an Equal Opportunity Employer.

Gigcamp Group
Executive/Manager

Gigcamp Group Offer You Work From Home With No Stress And For Some Selected Days At Each Week, This Could Be Your Opportunity, For More Information, Call 206-339-6055 Or Email Direct To gigcampgroup@yahoo.com For More Details.

Regards
Curtis Mills
Recruiting Manager

Intelligent Automation, Inc.
Research Scientist

Research scientist: communication networks. Ph.D. in EE or CS req. Expertise in transport, routing and link layer protocols in multi-hop wireless networks required. Algorithm design, protocol design/ development, discrete event simulation, and network security expected. Good JAVA skills. US/ permanent residency req. http://www.i-a-i.com/.

Iowa State University
Assistant or Associate or Full Professor

The Electrical and Computer Engineering Department at Iowa State University has immediate openings for faculty positions at all levels. Applications will be accepted from highly qualified individuals for regular faculty positions in the department in all core areas of expertise in Electrical or Computer Engineering, especially in:

- COMPUTER ENGINEERING with emphasis on embedded systems;
- SOFTWARE ENGINEERING;
- ASSURANCE AND SECURITY; and
- POWER and energy/power electronics.

Exceptional senior candidates in any area may be considered for endowed research chair/ professorship positions. Faculty positions also are available in interdisciplinary research areas as part of Iowa State University College of Engineering's aggressive mission to fill 50 college-wide positions with faculty who possess the talent to address the challenges that define worldwide quality of life and have global impact. The positions are targeted in the following interdisciplinary research and education cluster areas:

- Biosciences and Engineering
- Energy Sciences and Technology
- Engineering for Extreme Events
- Information and Decision Sciences
- Engineering for Sustainability

Duties for all positions will include undergraduate and graduate education, developing and sustaining externally-funded research, graduate student supervision and mentoring, and professional/institutional service.

REQUIREMENTS: All candidates must have an earned Ph.D. degree in Electrical Engineering, Computer Engineering, Computer Science, or related field, and they must have potential to excel in the classroom and to establish and maintain a productive externally funded research program. Associate and Full Professor candidates must, in addition, have an excellent record of externally funded research and internationally recognized scholarship.

Rank and salary are commensurate with qualifications. Screening will begin on November 1, 2008, and will continue until positions are filled. To guarantee consideration, complete applications must be received by January 19, 2009.

For regular faculty positions, apply online at http://www.iastatejobs.com/, Vacancy #080579.

More information on this position can be viewed at: http://www.ece.iastate.edu/jobs.

For information on positions in the cluster areas and application process, visit http://www.engineering.iastate.edu/clusters.

Candidates may be subject to a background check.

ISU is an EO/AA employee

Metron
Software Develope

Seeking talented programmers in Reston, VA for R&D projects with strong probabilistic and statistical foundations. At Metron, we value innovative research experience, inquisitiveness and insight. Software Developers are responsible for implementing software prototypes incorporating the models. To learn more about this position and apply online, please visit http://www.metsci.com/. Metron, Inc. is an EOE.

Desired Qualifications:

- BS or MS in Computer Science, Math, or other engineering field
- Significant programming experience in JAVA and/or C++
- Familiarity with Python, Perl, Lisp, Ruby, etc.
- Familiarity with Linux and Windows preferred
- Excellent communication skills, both written and oral
- US CITIZENSHIP REQUIRED

Özyegin University
Computer Science Faculty Member

Özyegin University invites applications for faculty positions at all ranks in Computer Science including but not limited to areas such as:

Software Engineering, Networking, Security, Artificial Intelligence, Graphics, Databases, Human-Computer Interaction, Computer Vision and Perception, Robotics, Bio-Informatics, High-Performance Scientific Computing, and Computer Systems.

High priority will be given to the originality and promise of the candidate's work.

Applicant's entrepreneurial interests and experience will be a big plus.

A Ph.D. in Computer Science or a closely related discipline, evidence of the ability for establishing and maintaining a research program and a strong commitment to teaching are required. A successful candidate will be expected to create a

research program, supervise Master's and Ph.D. students, obtain competitive grants, and teach courses at undergraduate and graduate levels. The compensation package will be commensurate to skills and rank, and will be highly competitive.

Applications should include a resume and brief statements of research and teaching interests.

Saint Michael's College
Assistant Professor of Information Technology & Computer Science

The American University of Afghanistan is seeking interested candidates for faculty positions in the new department of Information Technology & Computer Science. We offer a unique opportunity to teach and work in a newly formed university located in Kabul. Since our first students enrolled in 2006 the University is growing steadily, enrolling a new group of entering undergraduate students each semester. The ITCS department has expanded the curriculum and is seeking a wide range of new faculty members for the fall semester of 2009. The successful applicant will demonstrate a commitment to undergraduate teaching and the ability to engage undergraduates in research and class projects. Teaching responsibilities include participation in our introductory courses as well as upper level courses in the applicant areas of interest and expertise, including but not limited to computer networks, databases, eCommerce & web development, programming languages, or operating systems. The teaching load per semester is 3 to 4 courses with no more than three course preparations. A Ph.D. or M.Sc. in a computer related area is required for this position. Salary and level of appointment commensurate with experience. Please see our website for descriptions of the faculty position and how to apply: www.auaf.edu.af

The Hong Kong Polytechnic University
Department Of Computing

The Department invites applications for Professors/Associate Professors/Assistant Professors in Database and Information Systems / Biometrics, Computer Graphics and Multimedia / Software Engineering and Systems / Networking, Parallel and Distributed Systems. Applicants should have a PhD degree in Computing or closely related fields, a strong commitment to excellence in teaching and research as well as a good research publication record. Applicants with extensive experience and a high level of achievement may be considered for the post of Professor/Associate Professor.

Please visit the website at http://www.comp.polyu.ed.hk/ for more information about the Department. Salary offered will be commensurate with qualifications and experience. Initial appointments will be made on a fixed-term gratuity-bearing contract. Re-engagement thereafter is subject to mutual agreement. Remuneration package will be highly competitive.

Applicants should state their current and expected salary in the application. Please submit your application via email to hrstaff@polyu.edu.hk. Application forms can be downloaded from http://www.polyu.edu.hk/hro/job.htm.

Recruitment will continue until the positions are filled. Details of the University's Personal Information Collection Statement for recruitment can be found at http://www.polyu.edu.hk/hro/jobpics.htm.

University of Montana
Tenure-track faculty position, Computer Science

Position Description: The Department of Computer Science seeks applications for a tenure-track assistant professor faculty position to start August 2009.

Duties: The Department and the University require a commitment to a balanced mix of undergraduate and graduate teaching, research productivity, and service in the discipline, Department, and University. The Department and the University also strongly encourage collaborative teaching and research efforts. Applicants in any area of computer science are encouraged to apply. We are also interested in candidates who can collaborate with researchers in a science or applied science area. The successful candidate will have the ability to effectively teach a variety of courses within the Computer Science Department, work independently and collaboratively on research with faculty and students, and contribute to his/her specialty, the Department, and the University.

The University of Montana is one of the nation's outstanding public universities, committed

to liberal arts education, research, and strong professional programs. UM is located in Missoula, a cosmopolitan Rocky Mountain community, often singled out in national publications for its quality of life. Strongly collaborative and application focused, the Department continues to grow and gain both national and international attention for both research and education. The successful applicants will be supported, encouraged, and expected to contribute to both research and teaching efforts.

Qualifications: A Ph.D. in Computer Science or a closely related field is required as is evidence of teaching excellence and research potential. All areas of specialization will be considered within Computer Science.

Submission Deadline: Applications will be reviewed starting on November 1, 2008 with interviews to be done in early December. Applications will be accepted until the position is filled.

Application Procedure: Applicants should submit a resume, graduate transcripts, a letter addressing both research and teaching interests, and three letters of recommendation. Submission of the letter of application and vita by e-mail is encouraged. The University of Montana is an EEO/AA employer.

Union College

Assistant Professor of Computer Science

We invite applications for two tenure-track assistant professor positions beginning September, 2009. A Ph.D. in computer science or a closely related field is required. Candidates must be strongly committed to undergraduate education and have a sustainable research program. We are interested in candidates who work in systems areas such as parallel and distributed computing, networking, and security, or in computer graphics, robotics, or computer gaming, but all areas of computer science will be considered.

The department offers a B.S. in computer science, a B.S. in computer engineering with the Electrical and Computer Engineering department, and a digital art program with the Visual Arts department. We are developing programs in cognitive science and computational science. We offer introductory courses that encourage students with a variety of interests to study CS. Faculty research areas include HCI, NLP, databases, software testing, software design, and computing history. Union College is a highly selective liberal arts and engineering college in New York State's Capital Region, three hours from NYC and Boston. It emphasizes close collaborations between faculty and students and has a campus-wide initiative promoting interdisciplinary activities.

Further information: http://cs.union.edu/ . Applicants should submit an application letter, CV, statements of teaching and research goals, and have three reference letters sent separately. Send applications and recommendations to: Search Committee, Computer Science Department, Union College, Schenectady, NY 12308. Emailed recommendation letters only may be sent to cs-refletters@union.edu.

Applications received by December 15, 2008, will receive full consideration. Union College is an equal opportunity employer and strongly committed to student and workforce diversity.

Ursinus College

Assistant Professor of Mathematics and Computer Science

Ursinus College seeks tenure-track Asst Prof in Math and Comp Sci. starting Fall 2009. PhD + teaching experience req'd. Commitment to teaching excellence in liberal arts setting. See full description at www.ursinus.edu/NetCommunity/CompSci.

University of Chicago

The Department of Computer Science at the University of Chicago

Invites applications from exceptionally qualified candidates in all areas of Computer Science for faculty positions at the ranks of Professor, Associate Professor, Assistant Professor, and Instructor. The University of Chicago has the highest standards for scholarship and faculty quality, and encourages collaboration across disciplines.

The Chicago metropolitan area provides a diverse and exciting environment. The local economy is vigorous, with international stature in banking, trade, commerce, manufacturing, and transportation, while the cultural scene includes diverse cultures, vibrant theater, world-renowned symphony, opera, jazz, and blues. The University is located in Hyde Park, a pleasant Chicago neighborhood on the Lake Michigan shore.

Please send applications or nominations to:
Professor Stuart A. Kurtz, Chairman

Hawai'i Pacific University

Assistant Professor of Computer Science and Information Systems

Hawai'i Pacific University's College of Professional Studies invites applications for an Assistant Professor of Computer Science and Information Systems position starting in Spring 2009. Applicants are expected to have an earned doctorate in Computer Science, Information Systems, or a closely related field; three years of teaching experience at the university level; and three years of experience as an I.S. professional. The successful candidate will be expected to teach various courses; provide guidance to students completing research; and provide professional development for students pursuing a career in Computer Science and Information Systems.

HPU is the largest private university or college in Hawai'i, with almost 8000 students from over 100 countries, all 50 states, and all five major Hawaiian islands. Information about the College of Professional Studies can be found at http://www.hpu.edu/index.cfm?contentID=3823.

To apply, please send letter of application, official transcripts, curriculum vitae, and three letters of recommendation that include an assessment of the candidate's teaching to: Human Resources, Assistant Professor of Computer Science and Information Systems, 1136 Union Mall, Suite 208, Honolulu, HI 96813. Fax: (808) 544-1192. Email: hr@hpu.edu. HPU is an equal opportunity employer.

Department of Computer Science
The University of Chicago
1100 E. 58th Street, Ryerson Hall
Chicago, IL. 60637-1581

or to: apply-080140@mailman.cs.uchicago.edu (attachments can be in pdf, postscript, or Word).

Complete applications consist of (a) a curriculum vitae, including a list of publications, (b) forward-looking research and teaching statements. Complete applications for Assistant Professor and Instructor positions also require (c) three letters of recommendation, sent to recommend-080140@mailman.cs.uchicago.edu or to the above postal address, including one that addresses teaching ability. Applicants must have completed, or will soon complete, a doctorate degree. We will begin screening applications on December 15, 2008. Screening will continue until all available positions are filled. The University of Chicago is an equal opportunity/affirmative action employer.

University of Notre Dame

The Duda Family Chair in Engineering

Department of Computer Science and Engineering

The Department of Computer Science and Engineering at the University of Notre Dame seeks to fill the newly-established Duda Family Chair in Engineering. The inaugural chair holder will have a distinguished record of achievement at the full professor level, and will be expected to work with the other chaired professors and faculty in the Department to further extend the Department's strong research programs, including cross-departmental and crosscollege multi-disciplinary activities.

The Department offers a PhD degree as well as accredited undergraduate programs in Computer Science and Computer Engineering. There are approximately seventy-five students in the PhD program and over one hundred majors in the undergraduate programs. Currently there are seventeen tenure-track and six non-tenure track faculty. Active areas of research include algorithms, bioinformatics and computational biology, computer architecture and nanotechnology, data mining / machine learning, computer vision / image analysis, and networks / systems. The Gates Foundation recently awarded a $20 million grant to Biology and CSE faculty in the bioinformatics area, and the Semiconductor Research Corporation (SRC) together with the state of Indiana and the city of South Bend recently announced that the Midwest Institute for Nanoelectronics Discovery, a nanoelectronics research consortium led by Notre Dame, has received $25 million in new funding.

The University of Notre Dame is a private, Catholic university with a doctoral research university (extensive) Carnegie classification. Notre Dame has an enrollment of over 11,000 students, and it is consistently ranked in USN&WR as a top-twenty national research university. Notre Dame is located in South Bend, Indiana. South Bend is part of a metropolitan area of more than 300,000 residents. It has a vibrant and diverse economy with affordable housing and excellent school systems. Recreational opportunities in the South Bend area include professional and collegiate sports, a thriving arts culture, close proximity to Lake Michigan and Chicago, and a variety of outdoor activities.

Preferred areas of expertise for the inaugural holder of the Duda Chair are data mining /machine learning or systems / networks; however, outstanding applicants and nominees in all areas will be considered. Screening of applications will begin immediately. Applicants should send a statement of interest, CV, and list of references in PDF format to DudaChairSearch@cse.nd.edu.

The University of Notre Dame is an Equal Opportunity, Affirmative Action Employer.

Universidad Javeriana Cali

Tenure track position in Computer Science & Software Engineering

The School of Engineering of Universidad Javeriana Cali is offering tenure track positions in Computer Science and Software Engineering. Annual Salaries plus benefits are US$40K to US$50K.

For information: Dr. Jorge F Estela, Dean of the School, at jfe@javerianacali.edu.co

University of Oregon

Department of Computer and Information Science

Faculty Position

The CIS department seeks applicants for one or more full-time tenure-track faculty positions be-

Dean, College of Computing and Informatics

The University of North Carolina at Charlotte

The University of North Carolina at Charlotte invites applications and nominations for the position of Dean of the College of Computing and Informatics. Reporting to the Provost and Vice Chancellor for Academic Affairs, the Dean provides leadership for the Departments of Computer Science and Software & Information Systems, a growing interdisciplinary Bioinformatics program, and six affiliated research and community engagement units.

The **College of Computing and Informatics** continues to experience tremendous growth since its establishment in 2000. The College offers an interdisciplinary Ph.D. program in Information Technology with multiple tracks, including Computer Science, Software and Information Systems, and Bioinformatics; M.S. degrees in Computer Science and Information Technology; a Professional Science Master's degree in Bioinformatics; B.S. and B.A. degrees; and certificate programs.

The College has gained national recognition through research institutes, including the Charlotte Visualization Center, the Diversity in Information Technology Institute, and the Center for Digital Identity and Cyber Defense Research. The College is designated a National Center of Academic Excellence in Information Assurance Research by the National Security Agency.

The College has taken a leadership role in developing bioinformatics programs in collaboration with the developers of the North Carolina Research Campus. A billion-dollar, 350-acre research park, less than 20 miles from UNC Charlotte, the N.C. Research Campus was founded to serve as home to research programs in metabolomics and plant genomics and a large number of biotech companies.

The College is housed in Woodward Hall, which opened in 2005. The Bioinformatics Research Center is scheduled to move in August 2009 to a new 75,000 sq. ft. building.

UNC Charlotte is the only doctoral-granting institution in a dynamic urban region of 1.8 million people and offers unparalleled educational opportunities to a culturally diverse student body of more than 23,000. (University homepage: www.uncc.edu.)

The Dean is responsible for academic leadership, strategic planning, faculty appointments, budgetary and administrative oversight, development and alumni affairs, and external relations. Academic and community leadership; effective communication with faculty, staff, and students; consensus building; and fundraising are all part of the day-to-day activities of the Dean.

The Dean should possess an earned doctorate and an outstanding record of scholarly achievement that is appropriate for appointment as a tenured, full professor at a research university; an established national reputation; a proven ability to attract external funding; demonstrated results at leading and developing an academic unit and commitment to student success; experience engaging a diverse university community; and a successful record of establishing relationships within university, business and civic communities, and government agencies.

Charlotte is home to the world headquarters of nine Fortune 500 companies, with Bank of America and Wachovia making it the second largest banking center in the United States. The Charlotte region is consistently ranked as one of the fastest growing and most affordable regions in the United States and is home to over 750 foreign-owned firms representing more than 46 countries.

Application Process: Information about the application process and further details on the position and the College of Computing and Informatics may be found atwww.provost.uncc.edu/Searches/DeanCCIU.

Confidential nominations may be submitted by mail to the Chair of the Search Committee, Dr. Owen Furuseth, Dean Search, c/o Office of Academic Affairs, UNC Charlotte, 9201 University City Blvd., Charlotte, N.C. 28223, or by email todeansearch@uncc.edu.

UNC Charlotte is committed to equality of educational opportunity and is an affirmative action employer. Minorities, women and individuals with disabilities are encouraged to apply. Finalists are subject to educational and criminal background checks.

ginning fall, 2009. We anticipate appointments at the rank of Assistant Professor; however, in the case of exceptionally qualified candidates appointments at any rank may be considered. The University of Oregon is an AAU research university located in Eugene, two hours south of Portland, and within one hour's drive of both the Pacific Ocean and the snow-capped Cascade Mountains.

The CIS department is housed within the College of Arts and Sciences and part of the recently dedicated Lorry Lokey Science Complex. The College appreciates the increasing role that computer science plays in other disciplines and supports our goals of strengthening our ties with the other sciences. Applicants interested in interdisciplinary research are encouraged to apply. We offer a stimulating and friendly environment for collaborative research both within the department and with other departments on campus. The CIS department is associated with the Cognitive and Decision Sciences Institute, the Computational Science Institute, the Neuro-Informatics Center, and the Computational Intelligence Research Laboratory.

This department recognizes that computer science is undergoing rapid change as an academic discipline, and accordingly seeks to hire faculty in emerging areas of computer science as well as more established areas including distributed computing, data mining, networking, computational science (visualization, high performance computing), and HCI (usability, accessibility, interfaces).

The CIS department offers B.S., M.S. and Ph.D. degrees. More information about the department, its programs and faculty can be found at http://www.cs.uoregon.edu, or by contacting the search committee at faculty.search@cs.uoregon.edu.

Applicants must have a Ph.D. in computer science or a closely related field, a demonstrated record of excellence in research and a strong commitment to teaching. The successful candidates are expected to conduct vigorous research programs, and to teach at both the undergraduate and graduate levels. Applicants should send their curriculum vitae, names of at least four references, a statement of research and teaching interests, and selected publications to: Faculty Search Committee, Dept. of Computer and Information Science, University of Oregon, Eugene, OR 97403-1202, email: faculty.search@cs.uoregon.edu.

Review of applications will begin January 5, 2009, and continue until the position is filled.

The University of Oregon is an equal opportunity/affirmative action institution committed to cultural diversity and compliant with the Americans with Disabilities Act. We are committed to creating a more inclusive and diverse institution and seek candidates with demonstrated potential to contribute positively to its diverse community.

University Of Pennsylvania

Department Of Computer And Information Science

Faculty Positions

The University of Pennsylvania invites applicants for tenure-track appointments in both experimental and theoretical computer science to start July 1, 2009. Tenured appointments will also be considered. Faculty duties include teaching undergraduate and graduate students and conducting high-quality research.

The Department of Computer and Information Science has undergone a major expansion, including new faculty positions and a new building, Levine Hall, which was opened in April 2003. Over the last few years, we have successfully recruited faculty in artificial intelligence, architecture, databases, machine vision, programming languages, security and graphics. We are now especially interested in candidates in architecture and systems, although outstanding candidates in other areas might also be considered. Successful applicants will find Penn to be a stimulating environment conducive to professional growth

The University of Pennsylvania is an Ivy League University located near the center of Philadelphia, the 5th largest city in the US. Within walking distance of each other are its Schools of Arts and Sciences, Engineering, Medicine, the Wharton School, the Annenberg School of Communication, Nursing, Law, and Fine Arts. The University campus and the Philadelphia area support a rich diversity of scientific, educational, and cultural opportunities, major technology-driven industries such as pharmaceuticals, finance, and aerospace, as well as attractive urban and suburban residential neighborhoods. Princeton and New York City are within commuting distance.

To apply, please complete the form located on the Faculty Recruitment Web Site at:

http://www.cis.upenn.edu/departmental/facultyRecruiting.shtml

Electronic applications are strongly preferred, but hard-copy applications (including the names of at least four references) may alternatively be sent to:

Chair, Faculty Search Committee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Applications should be received by January 15, 2009 to be assured full consideration.

Applications will be accepted until positions are filled.

Questions can be addressed to faculty-search@central.cis.upenn.edu.

The University of Pennsylvania values diversity and seeks talented students, faculty and staff from diverse backgrounds. The University of Pennsylvania does not discriminate on the basis of race, sex, sexual orientation, gender identity, religion, color, national or ethnic origin, age, disability, or status as a Vietnam Era Veteran or disabled veteran in the administration of educational policies, programs or activities; admissions policies; scholarship and loan awards; athletic, or other University administered programs or employment.

University of Rochester

Tenure Track Faculty Positions

The Department of Computer Science at the University of Rochester invites applications for tenure track faculty positions. We seek PhD level candidates in networking, HCI, graphics, and/or machine learning. In addition, we invite applications for a joint Computer Science/Electrical and Computer Engineering position in computer systems and circuits. For full job descriptions and application procedures, see http://www.cs.rochester.edu/recruit.

University of Western Ontario

Postdoctoral Fellow

Applications are invited for a full time Postdoctoral Fellow position (up to 4 years) in the Biocomputing Laboratory in the Department of Computer Science, the University of Western Ontario, London, Ontario, Canada. The successful candidate will have a Ph.D. in computer science or mathematics, and experience in research in formal language theory, complexity theory, information, coding theory or a related discipline. Salary will be competitive and commensurate with the applicant's qualifications and experience. Please send your resume to Prof. Lila Kari, lila@csd.uwo.ca

Vanderbilt University

Assist. Professor, Dept. EECS

THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (EECS) AT VANDERBILT UNIVERSITY is seeking candidates for one or more potential faculty appointments each in CS and EE. Appointments at all ranks will be considered, with a preference for appointment at the assistant professor level. Search areas of emphasis in CS are software engineering, graphics/human-computer interactions, artificial intelligence, and/or web technologies. Search areas of emphasis in electrical engineering are nanoelectronics/photonics, and/or signal/image processing. In both CS and EE, we seek opportunities to add to department capabilities in high performance computing/computational science. A Ph.D. in Computer Science, Computer Engineering, Electrical Engineering, or a closely related field is required, as is experience commensurate with the level of appointment sought.

The EECS has 32 full-time faculty members, 225 undergraduate students, and 170 graduate students. Research awards to the Department average $600k per tenure/tenure-track faculty member. For more information, please visit our web site: http://eecs.vuse.vanderbilt.edu.

Applications consisting of a cover letter specifying the areas of particular interest in EE or CS, a statement of planned research activity and teaching interests, a complete curriculum vitae, and the addresses of four references should be sent to Professor Daniel M. Fleetwood, Chair, EECS Department, Vanderbilt University, PO Box 92, Station B, Nashville, TN 37235-0092. Founded in 1873, Vanderbilt is a private, coeducational university with approximately 6,000 undergraduates and 5,000 graduate and professional students.

Vanderbilt University is an equal-opportunity, affirmative-action employer.

Yeshiva University

Adjunct

Yeshiva College has the following adjunct positions available for the Fall 2008 semester:

Theory of Computation, Networking & Communication, Computer Organization and Assembly.

Contact Michael Breban at breban@yu.edu for further information.

York University

Assistant Professor

The Department of Computer Science and Engineering invites faculty applications in computer graphics with emphasis on digital media, animation and/or interactive/3D systems at the Assistant Professor level in the tenure track stream.

The deadline for applications is November 15, 2008. The full advertisement can be found at http://yorku.ca/acadjobs.

York University is an Affirmative Action Employer.

LAST BYTE

[CONTINUED FROM P.112]

The synthesis of logic and probability allows you to learn this type of holistic representation [of complex systems] from real-world data. It gives you the ability to learn higher-level patterns that talk about the relationships between different individuals in a reusable way.

You've begun applying your techniques to the field of biology.

Originally, it was a method in search of a problem. I had this technology that integrated logic and probability, and we had done a lot of work on understanding the patterns that underlay complex data sets. Initially, we were looking for rich data sets to motivate our work. But I quickly became interested in the problem in and of itself.

What problem is that?

Biology is undergoing a transition from a purely experimental science—where one studies small pieces of the system in a very hypothesis-driven way—to a field where enormous amounts of data about an entire cellular system can be collected in a matter of weeks. So we've got millions of data points that are telling us very important insights, and we have no idea how to get at them.

What have you learned about interdisciplinary collaboration from your work with biologists?

The important thing is to set up a collaborative effort where each side respects the skills, insights, and evaluation criteria of the other. For biologists to care about what you build, you need to convince them that it actually produces good biology. You have to train yourself to understand what things they care about, and at the same time you can train them in the methods of your community.

So it's not just learning a new scientific language, but training yourself to respect a different research process.

It's a question of finding people who are capable of learning enough of the other side's language to make the collaboration productive. C

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

DOI:10.1145/1400181.1400205

Leah Hoffmann

Q&A
A Complex Thinker

Daphne Koller discusses probabilistic relational modeling, artificial intelligence, and her new work with biologists.

IN APRIL, DAPHNE Koller, a professor of computer science at Stanford University, received the first-ever ACM-Infosys Foundation Award in Computing Sciences for her groundbreaking approach to artificial intelligence.

How did your interest in relational logic and probability first develop?

What really sparked my interest was when I went to UC-Berkeley for my post-doc, and I realized how useful probabilistic modeling methods are—techniques like Bayesian networks—but, on the other hand, how brittle they are.

Brittle?

[Probability is] a language that's very restricted in its expressive power because it can only refer to specific, concrete entities.

At Berkeley, I remember speaking with a graduate student who was building a model for vehicle traffic on a freeway. He had this wonderful model that was getting great results, but it only applied to a three-lane freeway. If he wanted to add another lane, it would take him two weeks to do the model.

And that's where logic entered the picture.

The language of logic allows you to come up with much more general rules for describing the properties of objects and their interactions with each other. It's a good way of extending the power of probability to something that's more expressive and forceful.

That would be the synthesis known as probabilistic relational modeling.

What we did is take this language of objects and relations and add the ability to make probabilistic statements, which enables you to represent probability distributions over networks of interrelated individuals.

It also enables you to create models of considerably more complex systems.

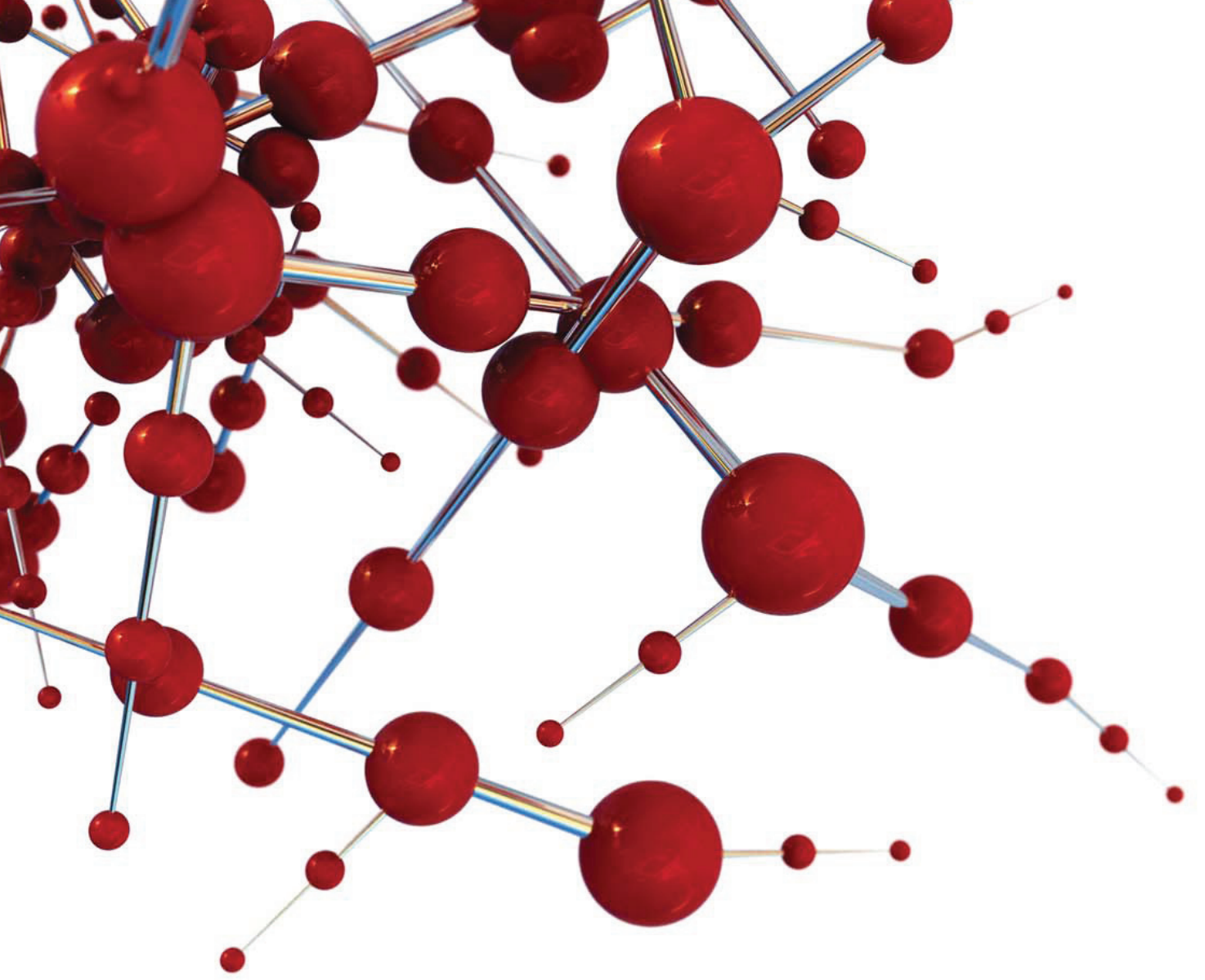
Yes, and that's been a focus of my work ever since. The world is very complex: people interact with other people as well as with objects and places. If you want to describe what's going on, you have to think about networks of things that interact with one another. We've found that by opening the lens a little wider and thinking not just about a single object but about everything to which it's tied, you can reach much more informed conclusions.

Which was an insight you brought to the field of artificial intelligence...

Well, I wasn't the only one involved. There had been two almost opposing threads of work in artificial intelligence: there were the traditional AI folks, who grew up on the idea of logic as the most expressive language for representing the complexities of our world. On the other side were people who came in from the cognitive reasoning and machine learning side, who said, "Look, the world is noisy and messy, and we need to somehow deal with the fact that we don't know things with certainty." And they were both right, and they both had important points to make, and that's why they kept arguing with each other.

How did probabilistic relational modeling help settle the dispute?

[CONTINUED ON P.111]



essays, {craft, art, science} of software, python, eclipse, agile development, onward!, {generative, functional} programming, .net, open source, concurrency, smalltalk, aspects, second life, ruby, service-orientation, objects, embedded, ultra large scale {model, test}-driven passion, fun!, agents, domain-specific use cases, movies, lightning talks, systems, objective-c, development, c#, design patterns, languages, wiki, product-lines, java, refactoring, plop

OOPSLA

2008 NASHVILLE

DEFINE THE FUTURE OF SOFTWARE

www.oopsla.org/submit

CONFERENCE CHAIR
GAIL E. HARRIS
Instantiated Software Inc.
chair@oopsla.org

PROGRAM CHAIR
GREGOR KICZALES
University of British Columbia
papers@oopsla.org

ONWARD! CHAIR
DIRK RIEHLE
SAP Research
onward@oopsla.org

CALL FOR PAPERS
March 19, 2008
Due date for Research Program, Onward!, Development Program, Educators' Symposium, Essays and proposals for Tutorials, Panels, Workshops and DesignFest

July 2, 2008
Due date for Development Program Briefs, Doctoral Symposium and Student Volunteers

Association for Computing Machinery

NASHVILLE CONVENTION CENTER, NASHVILLE, TN
October 19 - 23, 2008