History and Contributions of Theoretical Computer Science

John E. Savage
Department of Computer Science
Brown University
Providence, RI 02912
savage@cs.brown.edu

Alan L. Selman
Department of Computer Science and Engineering
University at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
selman@cse.buffalo.edu

Carl Smith
Department of Computer Science
University of Maryland
College Park, MD 20741
smith@cs.umd.edu

Abstract

We briefly review some of the major accomplishements of theoretical computer science. Results from theoretical computer science have had enormous impact on the developement of programming languages and other areas of computer science. The impact of reseach in theoretical computer science is now being felt in the areas of Cryptography,

Communication Networks, Multimedia and Graphical Systems, Parallel Computation, VLSI, Learning as well as Programming Langauges and Software. Theoretical computer science has also had influenced in Biology, Mathematics, Manufacturing and Astronomy.

Contents

1 Introduction

For more than fifty years, as computing systems have diminished in size and grown in complexity, theoretical computer scientists have built the foundations of their discipline by developing models of computation and related methods of analysis. After the early recognition of the relevance of the theory of formal languages to the practice of compiler construction, theoretical computer science became a cornerstone of virtually every computer science undergraduate degree program.

In the early days of computer science a great deal of time and energy was devoted to the development of basic concepts, design of fundamental algorithms, identification and development of major subdisciplines, and the classification of problems by their difficulty, activities that actively engaged many theoretical computer scientists. Today the emphasis is primarily on the design and implementation of very large computer systems. The role of theoretical computer scientists today is to examine fundamental problems of the field through modeling, analysis, and experimentation. General computer science research has also changed very significantly as well. Because modern computer systems are often too large to be studied solely through experimentation, even "practical" computer scientists find themselves using models and analysis, the tools of theoretical computer science, to study these systems.

Below we provide a brief history of theoretical computer science followed by a review of its contributions to the practice of computer science as well as other scientific and engineering disciplines.

2 A Brief History

Theoretical computer science is a cornerstone for computer science. It "underlies many aspects of the construction, explanation, and understanding of computers" [Revolution]. "Many...theoretical concepts from different sources have now become so embedded in computing and communications that they pervade the thinking of all computer scientists." [Revolution]

What is theoretical computer science? "Theoretical computer scientists seek to understand computational phenomena, the expressibility of languages, the design and performance of algorithms, and general limits on computation. Thus, they ask what is computation, what can be computed, how it can be done, and at what cost. In this quest, they use formal models, methods of analysis, and some experimentation. They learn from and contribute to practice. Finally, they seek to expand the core activities of the field to better address hard computational problems." [Loui]

2.1 Historical Highlights

Theoretical concepts can take decades to be assimilated into the mainstream of computing, but when they are assimilated they can have a profound practical impact. The stored-program computer, a concept central to computer science, owes its origins to Alan Turing, who studied the fundamental nature of computation in the 1930s. The practice of programming computers was significantly advanced by the development of the theory of automata and languages by Chomsky and others in the 1950s. Building on the foundations of context free grammars, Knuth and others introduced algorithms and data structures for the efficient and practical parsing of high-level languages, leading to tools such YACC, thereby enabling the software revolution of the 1960's. In the 1970s theoreticians, exploring the intrinsic complexity of computational problems, identified the large class of NP-complete problems, everyday problems that appear to be so difficult to solve that no foreseeable increase in computing power would enable their exact solution. Theoreticians interested in studying computational complexity were led to the discovery of hard problems that serve as the underpinnings for modern computer-security systems, notably the RSA public-key cryptosystem. Also, they have demonstrated the utility of mathematical logic and automata theory to the verification of complex computer systems; for example model-checking technology is now widely used by hardware vendors.

Research innovations in the last ten to fifteen years have resulted in new formulations and results that promise a big impact in the future. We now have fast (polynomial-time) algorithms that provide approximate answers, with fixed performance bounds, to many NP-complete problems. We now use randomized algorithms that provide fast solutions to hard problems with high probability. We also employ interactive proof systems (the goal is to convince one player of the truth of a statement known to a second player) to verify electronic exchanges. These are but a few examples of recent and current successes.

The explanatory value of theoretical computer science is illustrated by the modern Web browser (originally developed at CERN and the National Center for Supercomputing Applications at Illinois scientific computing centers). It embodies the concept of the abstract machine developed in the 1970s. When a user follows a link to data, a browser invokes the appropriate interpreter (an abstract machine) to process the data, for example to view an image or run a Java program.

2.2 Prospects for the Future

In concert with the PITAC committee [PITAC], we believe that future computer systems will be large and complex, exhibiting complex interactions. Understanding such systems will be an enormous intellectual challenge that requires the efforts of both theoretical and experimental computer scientists. We must develop scalable hardware and software system models capable of handling the high emergent complexity of such systems and subject them to analysis before making large investments in their implementation. Emphasizing these challenge, George Strawn, of the office of the Assistant Director for CISE, said "we don't understand at a scientific level many of the things that we are building." [Strawn, G.]

To summarize, "We need more basic research – the kind of groundbreaking, high-risk/high-return research that will provide the ideas and methods for new disciplinary paradigms a decade or more in the future. We must make wise investments that will bear fruit over the next forty years." [PITAC]

3 Theory in the Practice of Computing

We now give some examples that demonstrate the important role that theoretical computer science has played in understanding practical computer science problems.

3.1 Cryptography and Secure Computation

The field of cryptography continues to flourish, with contributions to both theory and practice. New techniques for cryptanalysis (differential and linear cryptanalysis) have greatly enhanced our ability to assess the strength of conventional cryptosystems, while the development of factorization and discrete logarithm techniques based on the number field sieve provides a deeper understanding of the foundations of public-key cryptography. New protocols have been devised for applications such as electronic cash and key escrow that may impact practice. the practice of electronic commerce. Relationships between cryptography and other fields, such as machine learning and approximation, have been greatly expanded. Techniques for proving security of cryptographic primitives and protocols have been significantly improved.

3.2 Communication Networks

On-line algorithms and multicommodity flow algorithms have proven useful for admission control and for routing of virtual circuits in asynchronous transfer mode (ATM) networks. Theoretically optimal algorithms led to the development of a new, practical admission control and routing algorithm. Extensive simulation of this algorithm showed that it significantly out-performs the standard approaches. Variants of this algorithm will be implemented in ATM switches manufactured by AT&T. Research in load balancing and web caching by theoretical computer scientists led to the creation of Akamai, a highly visible new web caching company.

3.3 Computational Geometry

Techniques from computational geometry have been used in a wide variety of other areas of computing. For example, algorithms for generating Delaunay triangulations and triangulations with various local properties have been applied to mesh generation in computational fluid dynamics. Voronoi diagrams and data structures for nearest neighbor searching are used in clustering algorithms, which in turn are central to the speech and image compression required for multimedia systems to run on personal computers. Techniques developed in computational geometry for computing triangulations, line segment intersections, and terrain visibility are used in geographic information systems. Visibility graphs and the visibility complex have been used in systems for computer vision and computer graphics. Graph drawing algorithms are used in advanced graphic user interfaces and visualization systems.

3.4 Parallel Computer Architecture

Algorithm development and the analysis of parallel architectural models such as mesh-connected processors, hypercubes, cube-connected cycles, and butterfly networks has informed and shaped the design of many parallel multiple processor machines in use today. Research on algorithms for routing in networks, including (multi-phase) randomized routing, has also influenced parallel machine design.

3.5 Software Systems

Many software systems have incorporated ideas and algorithms first developed in a theoretical framework. For example, evolving algebras have been used to specify languages (e.g., C, Prolog, and VHDL), to define real and virtual architectures (e.g., APE, PVM, and Transputer), to validate standard language implementations (e.g., Prolog and Occam), and to validate distributed protocols. Versions of epistemic logic are widely used for the design and analysis of existing and new authentication protocols. Interactive theorem provers like ProofPower, a commercial prover based on an academic prototype (HOL), are used to verify properties of critical systems. Coordinating Communicating Systems (CCS) has been found to be an invaluable aid in the modeling, analysis, and design of safety-critical systems. Components of real systems involved in land and air transport, process control, and computer operating systems have been analyzed using CCS. Process calculi and related modal logics have also been used, for example, to formally specify and analyze a cache coherence protocol for a multi-processor architecture currently under development; to prove the correctness of a leader election protocol for a point to point network; and to design and analyze a rendezvous-based scheduler to be used in an embedded software system.

3.6 Programming Languages

The methods of structured operational semantics, which derived input both from the lambda calculus and its model theory, reached a point in the 1980's where full-scale languages could be defined in such a way that properties of the language (e.g., determinacy of evaluation and the lack of dangling pointers) could be rigorously proven. Simultaneously, semantic and syntactic theories of types have been deployed to yield language designs that are provably "type-sound," leading both to a significant increase in the reliability of a language and to greater efficiency in implementation (since type-information need not be present at run-time in a type-sound language). Standard ML [Meta Language] is an industrial-strength language whose formal definition (in 1990) exploited these advances.

Standard ML serves as a vehicle for many research projects, particularly those concerned with mechanized reasoning, program analysis, and compiler construction. Standard ML also serves as the subject of study for many investigations in programming language design.

Monads were introduced in the late 1980's to computing science as a way of structuring denotational semantics. semantics, increasing our understanding of programming languages. Many different language features, including non-termination, state, exceptions, continuations, and interaction, can be viewed as monads. More recently, they have been widely used as a programming technique in pure functional programming languages such as Haskell. Monads are used externally to extend the capabilities provided by the Glasgow Haskell compiler. They are used to provide input-output and interaction with C and updateable arrays and references.

3.7 VLSI design

The theory of VLSI algorithms and architecture has guided the development of VLSI design. Theory has confirmed the quality of designs for special problems such as matrix multiplication and convolution and has influenced the development of layout algorithms for unstructured problems.

Computer-aided design of digital systems (including VLSI design) depends heavily upon the results and techniques of the theoretical algorithms community. One example of this is in the layout of VLSI circuits. Automated layout uses both fundamental results in graph algorithms and specialized application of methods developed in the theoretical community. Many layout algorithms rely on minimum spanning tree and shortest-path algorithms; improvements in these algorithms and related data structures are directly usable by layout applications. The models and algorithms developed for single-layer routing are a successful application of theoretical techniques. Heuristics for NP-complete layout problems draw heavily on foundations for more basic problems such as graph partitioning and graph coloring.

3.8 Learning Theory

Algorithms have been developed within the computational learning theory community that learn subclasses of finite probabilistic automata. They have already been successfully applied in systems that perform handwriting recognition, speech recognition, part-of-speech-tagging, DNA sequence modelling, and text correction. Experimental evidence comparing these new systems with previously existing systems has been favorable. Learning algorithms and techniques for deterministic automata have been applied to reinforcement learning, which is a promising new paradigm for machine learning. Finally, learning algorithms for automata have also been applied to robot motion planning. Finding algorithms with better performance for these and other applications motivates the search for new, more general subclasses of automata that have efficient learning algorithms.

4 Contributions to Other Disciplines

In this section we give examples of contributions that theoretical computer scientists have made to various science and engineering disciplines.

4.1 Biology

A central algorithmic operation in computational biology is the comparison of two very long DNA sequences to establish a relationship between them. Most sequence comparison programs use dynamic programming techniques. As the lengths of the sequences grow, however, such programs become very slow. To speed them up, heuristics were proposed in 1980's. In 1990, those heuristics were cast into an algorithmic framework, called sparse dynamic programming. That framework, its techniques, and other new ideas from the algorithms community are a foundation for new sequence alignment software.

4.2 Mathematics

Theoretical computer science has also found applications in mathematics. In particular, domain theory has developed in the past 25 years as a mathematical foundation for the semantics of programming languages. Recently, domain theory has been applied in several branches of mathematics, including dynamical systems, measure and integration theory, and fractals. Domain theory has been used in finite-state discrete stochastic processes, iterated function systems, fractal image compression, neural nets, and the Ising model in statistical physics. A new, fast algorithm for polynomial decomposition is included in AXIOM, the symbolic computation language developed by IBM.

4.3 Manufacturing

A program *qhull* for computing convex hulls has been used in the implementation of an algorithm to compute support structures for objects produced through layered manufacturing, a process in which material is deposited and later removed by a laser. During the construction of an object in this fashion, it might be necessary to build external supports either to prevent the object from toppling, or to support floating components and overhanging material. The support structures, if necessary, must be built simultaneously with the object, and hence must be accounted for in the path planning of the laser beam or the deposition nozzle. The use of *qhull* reduced run times from minutes to seconds to compute possible bases for the object to rest on, and to test if the center of mass of the object is directly above the convex hull of the object's base.

4.4 Astronomy

Network flow techniques have been used to determine telescope settings as part of the Sloan Digital Sky Survey, an astrophysics Grand Challenge problem. The goal of the survey is to determine the relative locations and velocities of approximately five million of the visible galaxies. The desired information can be computed from spectral (light frequency) data for each galaxy. A rough estimate of the time and money needed for data collection is around five years and five million dollars. The cost of data collection is proportional to the number of times the telescope must be retargeted. A heuristic based on the network flow theory is currently used to target the telescope.

5 Foundational Research

Recent successes strongly indicate that we can expect a continued flow of important results from theoretical work for the foreseeable future—results that can transform the course of computer science research and, ultimately, the way technology is used. In many cases, these results emerge in unpredictable ways from apparently unrelated investigations of fundamental problems.

While many of the deep theoretical problems that are attracting the best minds in our field are rooted in, or inspired by, overarching challenges, it is often difficult for researchers to properly tackle such problems in the context of an application-driven research environment. One reason for this is the long time period needed for work on fundamental problems to come to full fruition. In addition, solutions to such problems draw on diverse mathematical and computational methods, and so the interaction of a broad community of theoretical researchers is essential.

Moreover, the best theoretical results typically influence the course of research in application areas, so it is extremely useful to maintain an identifiable corpus of theoretical knowledge that is accessible to the Computer Science community and to the scientific research community at large.

A third reason for the importance of foundational research that it targets high-risk and speculative problems whose solutions often have surprising or unpredictable consequences. Such research often provides the seed corn for major innovations.

For all of these reasons, unfettered research in foundational theoretical areas is vital; it provides a better understanding of the capabilities and limitations of computers and ensures future innovations in science and technology.

5.1 Computational Complexity

Fundamental questions remain on the relationships among models of computation, information representation and manipulation, and on good ways to express algorithms. The P versus NP question is perhaps the most famous of these, the refinement of which has led to many other important questions in complexity theory. P is the collection of all problems that can be solved in polynomial time and includes all the problems that can be solved efficiently by computers. The class NP (nondetermistic polynomial time) includes literally thousands of problems from operations research that crop up routinely in manufacturing and networking applications. The fastest known algorithms for problems in this class can only handle very small data sets. Unless P equals NP, we will never be able to obtain exact solutions for realistically sized versions of these problems. Progress on complexity-theoretic problems, even when of the "negative" type (such as providing evidence for the intractability of certain problems) can completely change computer scientists' approaches to practical problems in surprising ways. For one thing, researchers no longer waste time seeking efficient solutions to intractable problems. Instead, they invent and learn techniques for coping with intractability. The computational hardness of certain problems has been exploited for cryptography. Currently, computational hardness of certain problems is being harnessed to obtain efficient deterministic (error-free) algorithms for problems where randomness (and thus error-prone) algorithms previously seemed necessary.

The theory of computing community continues to produce wonderful fundamental ideas, and, over time, these influence practice in important ways. The interplay among concepts such as pseudorandom number generation, interactive proofs, and secure cryptographic protocols is beautiful and deep, and has significant potential to impact the practice of cryptography. The introduction of interactive proof systems and probabilistically checkable proofs has broadened and enriched our understanding of the concept of proof. Probabilistically checkable proofs have turned out to be a fundamental tool for studying the limits of polynomial-time approximation algorithms.

Foundational questions will require a concerted effort in the areas of classical Turing machine-like models and variants (such as randomized or quantum models), models of learning, formal methods and program inference, models of nonsymbolic reasoning, logical characterization of complexity classes, lower bounds, models of on-line computation, models for communication of information, models for inferring information from incomplete data, models for data storage and retrieval in a multimedia context, and parallel and distributed models. Study of connections between models and results in more than one of these areas can be particularly fruitful. For example, on-line algorithms, common in key frameworks such as operating systems, financial control, and real-time systems, have generated fundamental concepts that are important for distributed systems design, in particular where information flow is more complex than traditional input/output.

5.2 Design and Analysis of Algorithms

Foundational work on algorithms design has the goal of breaking long-standing barriers in performance. There are many situations where complicated algorithms, such as Strassen and Schönhage's multiplication algorithm, were deemed inferior for years. With steady increases in problem sizes such algorithms now are preferred on appropriate high-performance computing platforms.

In other cases, (such as linear programming) initial breakthroughs in reducing asymptotic running time, while not practical in and of themselves, serve to stimulate new research that eventually leads to practical algorithms. What tends to happen is that once a barrier, such as the existence of a polynomial-time algorithm for a problem, is broken, there is strong justification and real motivation for researchers to revisit a problem that previously appeared impenetrable. Very often, painstaking refinement of the seminal breakthrough technique leads to a truly practical algorithm. Ultimately, this type of research has long-lasting impact on the practice of computing. Tantalizing open questions remain, such as whether there is a polynomial time algorithm for graph isomorphism, or whether one can efficiently learn Boolean formulas that are in disjunctive normal form from random examples.

5.3 New Theories of Algorithms and Heuristics

While theoretical work on models of computation and methods for analyzing algorithms has had enormous payoffs, we are not done. In many situations, simple algorithms do well. Take for example the Simplex algorithm for linear programming, or the success of simulated annealing on certain supposedly "intractable" problems. We don't understand why! It is apparent that worst-case analysis does not provide useful insights on the performance of many algorithms on real data. Our methods for measuring the performance of algorithms and heuristics and our models of computation need to be further developed and refined. Theoreticians are investing increasingly in careful experimental work leading to identification of important new questions in the algorithms area. Developing means for predicting the performance of algorithms and heuristics on real data and on real computers is a grand challenge in algorithms.

On numerous occasions, theory of computing research has provided the insights that explain why popular, important heuristic algorithms work. Significantly, these insights have suggested major improvements to the heuristics. One example of this scenario was the study by Turner in the 1980s that explained why the decades-old Cuthill-McKee heuristic for minimizing the bandwidth of sparse linear systems works well in practice; this understanding allowed Turner to devise an improved heuristic. A second example, also from the 1980s, explained why the Kernighan-Lin graph bisection heuristic, which is important in the field of circuit layout, works well in practice.

6 Summary

Results from theoretical computer science have been regularly applied to the field of computer science and to other disciplines. In this paper we have highlighted recent contributions to communication networks, parallel computer architecture, software systems, VLSI design, learning theory, biology, mathematics, manufacturing, and astronomy. We can expect similar contributions in the future.

7 Acknowledgements

Parts of this paper were prepared by the SIGACT Long Range Planning Committee (Amihood Amir, Manuel Blum, Michael Loui, Christos Papadimitriou, John Savage, and Carl Smith) [SIGACT]. Other portions are drawn from a report funded by the National Science Foundation and co-authored by Anne Condon, Herbert Edelsbrunner, E. Allen Emerson, Lance Fortnow, Stuart Haber, Richard Karp, Daniel Leivant, Richard Lipton, Nancy Lynch, Ian Parberry, Christos Papadimitriou, Michael Rabin, Arnold Rosenberg, James S. Royer, John Savage Alan L. Selman Carl Smith Eva Tardos, and Jeffrey Scott Vitter [Challenges].

References

[Loui] A. Condon, F. Fich, G. N. Frederickson, A. Goldberg, D. S. Johnson, M. C. Loui, S. Mahaney, P. Raghavan, J. E. Savage, A. Selman, and D. B. Shmoys. Strategic directions in research in theory of computing. *ACM Computing Surverys*, Vol. 28, pp. 575–590, 1996.

[PITAC] B. Joy and K. Kennedy. *Information Technology Research: Investing in Our Future*. President's Information Technology Advisory Committee, Report to the President, 1999.

[Strawn, G.] G. Strawn, *Science News*, Vol. 155, 1999.

[Revolution] *Funding a Revolution: Government Support for Computing Research*. Committee on Innovations in Computing and Communications: Lessons from, History, National Research Council, pp. 184, 1999.

[Challenges] A. Condon, H. Edelsbrunner, E. A. Emerson, L. Fortnow, S. Haber, R. Karp, D. Leivant, R. Lipton, N. Lynch, I. Parberry, C. Papadimitriou, M. Rabin, A. Rosenberg, J. Royer, J. Savage, A. Selman, C. Smith, E. Tardos, and J. Vitter. Challenges for Theory of Computing: Report of an NSF-Sponsored Workshop on Research in Theoretical Computer Science. *SIGACT News*, June 1999. (See http://www.cse.buffalo.edu/ selman/report.)

[SIGACT] Amihood Amir, Manuel Blum, Michael Loui, Christos Papadimitriou, John Savage, and Carl Smith. Contributions of Theoretical Computer Science to Practice. *SIGACT*, 1996 (See http://sigact.csci.unt.edu/sigact/longrange/contributions.html.)