Theory of Computing: Goals and Directions

Alfred V. Aho
Columbia University

David S. Johnson
AT & T

Richard M. Karp (Chair)
University of Washington

S. Rao Kosaraju
Johns Hopkins University

Catherine C. McGeoch
Amherst Colloge

Christos H. Papadimitriou
University of California at Berkeley

Pavel Pevzner
University of Southern California

March 15, 1996

Abstract

This report assesses the current goals and directions of the Theory of Computing (TOC) community and suggests actions and initiatives to enhance the community's impact and productivity. The report discusses the field from a global perspective, but its recommendations are targeted specifically for education and research in the United States. Most of the recommendations relate to the following fundamental point: In order for the Theory of Computing to prosper in the coming years, it is essential to strengthen communication with the rest of computer science and with other disciplines, and to increase the impact of theory on key application areas.

Section 1 traces the history of the field and defines the four main categories into which our recommendations fall: building bridges between theory and applications, algorithm engineering, communication and education. Section 2 gives a sampling of the notable past achievements in the Theory of Computing. Section 3 discusses some opportunities for stimulating interactions between TOC and applied areas. Section 4 proposes an applied research initiative, *Information Access in a Globally Distributed Environment*, which identifies an exciting current technological area that we believe presents challenging opportunities for excellent theoretical work. Section 5 proposes a second applied research initiative, *The Algorithmic Tool Kit*, that would exploit and extend the body of theoretical knowledge in the field of generic algorithms and models of computation. Section 6 proposes a redirection of effort in graduate education with two purposes in mind: to better prepare theoreticians to interact creatively with practitioners, and to provide future practitioners with the background they will need to benefit from this exchange. .

1 Introduction

Theory and theoreticians have played a major and brilliant rôle in the history of our field. In fact, two of the pioneers of Theoretical Computer Science are also among the founders of Computer Science (CS): Alan M. Turing and John von Neumann. The two decades after the death of these giants (roughly 1955 to 1975) were the formative period for the field of Theory of Computing. During this period theoreticians developed a set of basic concepts and methodologies that transcend application domains and set our science apart from other applied and natural sciences. These fundamental contributions include *automata- and language-theoretic models*, *data structures and algorithms*, *methodologies for evaluating algorithm performance*, *the theory of NP-completeness*, *logics of programs and correctness proofs*, and methods of *public-key cryptography.* During this period theoreticians also contributed crucially to some of the most celebrated engineering triumphs of CS, especially in the areas of compilers, databases, and multitasking operating systems.

By the end of this formative period the field had discovered a realm of deep problems (especially those relating to the P vs. NP question) of the kind that take decades to solve and bring to a young science the aura of depth and respectability, and a sophisticated and powerful methodology was emerging, proudly borrowing from logic and combinatorics. Also, commonality of methodology and objective brought social cohesion: By 1975 theory conferences had already been running for about ten years (FOCS and STOC in the U.S., ICALP in Europe).

During the next twenty years theoretical CS attracted to its ranks some of the brightest young scientists, and celebrated some major achievements in fundamental research. The field branched out into a number of promising new areas with real potential for practical exploitation. These areas include *interactive proof systems*; *PAC learning*; *probabilistic, approximate and on-line algorithms*; *computational geometry*; *models and algorithms for distributed and asynchronous computation*; *cryptography and secure protocols*; *probabilistically checkable proofs*; *logics of knowledge*; and *quantum computing.*

Despite these remarkable achievements, trouble is looming for the field of theoretical research in computer science. The current professional climate is marked by saturation in the academic job market, shrinking budgets for federal funding agencies, and a sense of disconnection from other areas of computer science. It is time for a reassessment of the directions that theoretical computer science is taking.

This report grew out of an NSF-sponsored workshop, held on June 2, 1995 with the purpose of assessing the current goals and directions of the Theory of Computing (TOC) community and suggesting actions and initiatives to enhance the community's impact and productivity. The workshop was preceded by an open meeting on the same topic at the 1995 ACM Symposium on Theory of Computing, at which a number of position papers were presented. We discuss the history, current state, and future of theoretical research in computer science from a *global* perspective, since the sense of community in theoretical computer science transcends national boundaries. However, we recognize that the circumstances under which theoretical computer scientists are trained and the patterns of activity in theoretical computer science research vary greatly from country to country. For this reason our suggestions about graduate education and employment, and our recommendations about areas of emphasis and funding directions, are targeted specifically for research programs in the United States. The report is not directed to a single funding agency, but rather to the entire computer science community and to all the U.S. agencies that participate in the funding of theoretical computer science. We believe that the recommended changes are critical to the ability of the United States to remain a leader in computer science education, research and technology development.

In undertaking this task the committee found it useful to identify three distinct (although heavily interacting and overlapping) types of research activity within TOC.

1. The field of **core theory** studies computation as an abstract phenomenon, and explores the mathematical techniques that are (presently or potentially) useful in computer science. In recent years research in core theory, supported mainly by the NSF Theory of Computing program, has established solid foundations for cryptography, interactive computation, computational learning, parallel and distributed computation, randomized computation, on-line computation and reasoning about knowledge. These notable and largely unexpected developments contribute greatly to the appeal and distinctive flavor of theoretical computer science. Core theory has achieved a remarkable unity, in which seemingly unrelated topics are found to have surprising connections and to be amenable to investigation by common mathematical techniques. One example is the use of probabilistically checkable proofs to prove that the approximate solution of certain optimization problems is NP-hard. A second is the variety of interrelationships among cryptography, pseudorandom number generation, complexity theory and computational learning theory. A third is the broad applicability of recurrent mathematical themes such as rapidly mixing Markov chains, pairwise independent random variables, expander graphs, and discrete Fourier analysis. Since core theory develops through unexpected innovations it is difficult to establish programmatic goals. Instead, each proposal should be evaluated in terms of its own inner logic and potential for making a real difference. Because of shrinking funds and the need to develop the more applied side of theory, proposals in this area will have to be held to a high standard.

2. The field of **fundamental algorithms** uses sophisticated mathematical techniques to develop solutions to problems that transcend application domains. This area includes such central topics as

combinatorial algorithms, approximation algorithms, on-line algorithms, numerical algorithms, algorithms for computational geometry, and data structures for search, graph operations, and point set manipulation.

3. Finally, **application-specific theory** studies concrete computational phenomena arising in application areas, and adapts models and techniques from the two categories above, or develops new formal models and techniques for solving problems in these areas. The Human Genome Project is an example of an application area that has stimulated many recent algorithmic developments. For another example, there is a large community of theoreticians who develop models and algorithms tuned for specific kinds of parallel machines. In general, however, this area of research has not flourished to the same extent as core theory and fundamental algorithms. In part this is due to a value structure prevalent in the TOC community which places excessive weight on mathematical depth and elegance, and attaches too little importance to genuine links between theory and concrete applications. The neglect of application-specific theory by the TOC community is in large part responsible for the general computer science community's failure to understand or appreciate contemporary theoretical research. We hope that this report will stimulate members of the TOC community to recognize that it is often intellectually as challenging for the theoretician to pursue significant applications as to work on the now classical core problems, and that the body of knowledge that the theory community has developed is ripe for exploitation in application domains.

Most of our recommendations relate to the following fundamental point: **In order for TOC to prosper in the coming years, it is essential to strengthen our communication with the rest of computer science and with other disciplines, and to increase our impact on key application areas**. If we do so then funding will flow to TOC from a variety of sources, young theoretical computer scientists will have good employment opportunities, our field will contribute to the revolutionary developments that are surely coming in the field of computing, and we will draw intellectual stimulation from challenging problems that demand solution. If instead we turn inward then our chances for adequate funding and employment opportunities will be compromised, we will fail to take advantage of the intellectual stimulation that comes from exposure to concrete problems, and the applications that need our help will not get it.

Although our primary emphasis is on outreach to related fields, we also consider it essential to maintain a strong activity in core theory and fundamental algorithms, since these are the areas that give TOC its unique character, and in which the most dramatic new developments occur. There is little that any committee can do to program new breakthroughs in core theory and fundamental algorithms. Such breakthroughs arise from original ideas that could not have been anticipated. The best one can do is to select those proposals that seem to have the best chance of leading to fundamental advances, and for this purpose we know of no system better than the peer review system that NSF has traditionally used. The TOC community is relatively free of factions and cliques, and has a high degree of consensus about the criteria for good research; for these reasons, the peer review system has served this field well.

In the following paragraphs we state some broad recommendations for the future development of TOC. These are followed in later sections by more specific recommendations of research directions at the applied end of the TOC spectrum. These recommendations are intended not to infringe on the funds available to support investigator-initiated theory research, but rather to stimulate additional funding for applied theory from funding agencies and directorates concerned with specific application areas.

Our broad recommendations fall into four main categories: building bridges between theory and applications, algorithm engineering, communication and education.

Building bridges between theory an applications. There are a number of striking recent cases in which the theory of computation has exerted a direct influence on applied fields:

1. Algebraic algorithms for robot motion planning, originally studied from a complexity-theoretic point of view, have influenced applied developments in robotics.

2. Close ties have been established between theoreticians working in computational learning theory and the more applied machine learning community within artificial intelligence.

3. Theoretical computer scientists in industrial research laboratories are developing and implementing protocols for secure communication over computer networks, based on fundamental complexity-theoretic results in cryptography.

4. A new generation of efficient large-scale linear programming codes has been developed, building on theoretical analyses of interior-point methods for linear programming.

5. Theoretical ideas from coding theory have improved the error resilience of the transmission of multimedia data over packet networks.

Many further examples could be drawn from such fields as geometric modeling, relational databases, network optimization and computational biology.

In each of these cases, the exploitation of theoretical results was far from trivial, and required a deep understanding of both the underlying theory and the application domain. Such applied studies require theoreticians to cope with the messiness of the real world, often at the expense of the elegance, mathematical depth and universality that the TOC community prizes so much. Nevertheless, in order to maximize its impact on the rest of computer science and on other fields, the TOC community will have to encourage the strenuous effort that is required to carry a theoretical idea through to the point where it actually begins to have a practical impact. This is particularly important because, in the present atmosphere of shrinking funding for basic research in computer science, most of the theoreticians we train will find that their best opportunities for interesting and productive work lie in applied domains.

Algorithm engineering. Within theoretical computer science algorithms are usually studied within highly simplified models of computation and evaluated by metrics such as their asymptotic worst-case running time or their competitive ratio. These metrics can be indicative of how algorithms are likely to perform in practice, but they are not sufficiently accurate to predict actual performance. The situation can be improved by using models that take into account more details of system architecture and factors such as data movement and interprocessor communication, but even then considerable experimentation and fine-tuning is typically required to get the most out of a theoretical idea. Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice. This can only happen if the TOC community comes to recognize algorithm engineering — the experimental testing and tuning of algorithms — as integral to its mission.

Communication. We must *explain* to the rest of the CS community, to the broader scientific community, and to the society as a whole our past accomplishments, our present endeavors, and our potential for outstanding future contributions. Section 2 of this report lists some of the major accomplishments of the theory community and gives examples of the kind of exposition that is wanted, but there is a need for a continuing flow of expository articles making developments in TOC accessible to a broad audience.

We can reach out to nonspecialists by including broadly accessible expository introductory sections in our research papers. These should include realistic assessments of the applicability of our results, avoiding false claims but making sure that real opportunities for applications are pointed out. Often a theoretical paper will relate to an applied problem in a loose or metaphorical way, but will not contribute directly to solving the applied problem; in such cases, excessive claims of applicability must be avoided.

The unique, universal, and ubiquitous themes of computer science can only be developed by theoreticians who maintain strong communication ties with one another, and for this reason theoreticians often band together in "theory groups" within their academic departments or research organizations. It is important that such organizational structures should not insulate theoreticians from other computer scientists. Theory researchers who commit themselves to a particular applied topic should be willing to step outside their theory groups for a period of time or even establish a permanent dual affiliation.

Education. Graduate students narrowly trained in theoretical computer science will miss out on opportunities for intellectual growth and will have limited opportunities in the job market that is expected to prevail for the foreseeable future. It is important that theoreticians develop an awareness of applications and experimental research. For the health of the applied branches of computer science, it is equally important that applied computer scientists receive a good exposure to the methods and results of the Theory of Computing.

The remainder of the report fleshes out the picture developed in this Introduction and suggests a series of actions consistent with the broad recommendations we have given. Section 2 gives a sampling of the notable past achievements in the Theory of Computing. Section 3 proposes some opportunities for stimulating interactions between TOC and applied areas and thus generating new sources of funding for theory research. Section 4 proposes an applied research initiative, *Information Access in a Globally Distributed Environment*, which identifies an exciting current technological area that we believe presents challenging opportunities for excellent theoretical work. Section 5 proposes a second applied research initiative, *The Algorithmic Tool Kit*, that would exploit and extend the body of theoretical knowledge in the field of generic algorithms and models of computation. Section 6 proposes a redirection of effort in graduate education with two purposes in mind: to better prepare theoreticians to interact creatively with practitioners, and to provide future practitioners with the background they will need to benefit from this exchange.

2 Achievements of Theoretical Computer Science

Theoretical computer science has to its credit a number of achievements that deserve to be understood by everyone concerned with computer science and computer applications, either because they have had a substantial effect on an application area or because they provide important new ways of thinking about computation. These include

- theory of parsing and compilation
- NP-completeness
- basic data structures
- advanced data structures
- algorithmic developments in computational geometry
- interior methods for linear and nonlinear programming
- fast text searching
- complexity theory of parallel computation
- theory of program verification
- relational database theory
- public-key cryptography
- RSA and other cryptosystems
- interactive proofs
- zero-knowledge proofs
- probabilistically checkable proofs
- impossibility results from distributed computing theory

- logics of knowledge
- temporal logic
- computational learning theory

Many of these topics are not well understood outside the theory community, and there is a need for expository articles that explain them to a broader audience, as well as overview articles for theoreticians themselves. It is outside the scope of this report to provide the needed expositions, but we present a few brief illustrative examples.

Interior Point Algorithms. One of the major developments in the last 20 years in the field of optimization has been the emergence of interior point methods as serious practical competitors to the simplex method for linear programming. The success of these methods can in large part be attributed to theoretical concern with the asymptotic worst-case running time of algorithms. Klee and Minty, in 1972, showed that the simplex method, although often very fast in practice, could in the worst case take exponential time under most common pivoting rules. This gave rise to the important theoretical question of whether Linear Programming was solvable in polynomial time, a question that practitioners considered to be *only* a theoretical question, given the many reported successes of the simplex method. In 1979, Khachian discovered the first polynomial time algorithm for Linear Programming, but although the algorithm's worst-case running time was substantially better than that of simplex, its empirical performance was typically much worse. Practitioners took this to be evidence of the irrelevance of worst-case analysis, but theorists took it as motivation to look for better polynomial-time algorithms.

In 1984, Narendra Karmarkar devised such an algorithm, based on the old idea of seeking the optimal solution through the interior of the polytope rather than via its surface. Equally important, Karmarkar took the key step (still not common among theoreticians) of implementing the algorithm. In doing so, he discovered that, like the simplex method, it typically ran much more quickly than its worst-case guarantee indicated. His initial claims proved controversial, as other researchers could not at first duplicate his results. The reason for this was that Karmarkar, coming from a computer science background, had implemented the algorithm using modern data structure techniques (themselves a product of concern with asymptotic performance), something that was not yet common in the mathematical programming community.

The ferment caused by Karmarkar's results and claims has been immensely beneficial for the field of mathematical programming. Not only are there now a variety of commercially available interior point codes currently solving applied problems not amenable to other methods, but, faced with this competition, simplex packages have also improved dramatically (in part by adapting modern data structures and programming techniques themselves).

Relational Databases. Databases have grown to a ten billion dollar industry in which the United States enjoys a commanding lead. Remarkably, this industry is centered around a clean mathematical idea, the *relational model* proposed by Codd in 1970 — before that, databases were based on much more awkward and *ad hoc* models. Although the inventor of the relational model did not come from the traditional theory community, he did use the tools of theory for formulating and championing his model. The most striking and compelling argument for the relational model was a theorem stating that two apparently very different approaches to asking queries in it — the algebraic and the logical query languages — are in fact equivalent. The relational model was intentionally rigorous and open-ended theoretically. It has spawned a very active theoretical research tradition which created a rich and useful theory of database design, as well as its extensions to object-oriented, deductive, and novel media databases. The second technical pillar of the database industry, efficient access methods based on Bayer and McCreight's *B-tree*, draws from a long line of theoretical research on balanced search trees, which culminated, with superb timing, also in 1970. The story of relational databases teaches us that principled, rigorous theoretical research in an applied area is most useful (and leads to more interesting theory) when done early in the development of the field.

NP-completeness. All scientific fields have methodologies for attacking problems; but after several methods have been tried without success, scientists must face the uncomfortable possibility that they might have missed a simple trick. In computer science we formulate and try to solve algorithmically scores of different computational problems and their innumerable special cases and variants; it is hard to visualize the frustration and wasted energy that would prevail in our field today without *NP-completeness*. By establishing that many well-known stubborn problems from different disciplines are all computationally equivalent, and in fact can be solved efficiently if and only if P=NP, Karp, based on previous work by Cook, created an extremely influential new research paradigm. Researchers in all areas of computer science today resort to this evidence of impossibility whenever a computational problem resists their efforts at efficient solution; this way, research effort is redirected to goals such as approximate algorithms, and heuristics are pursued without guilt. What is more, NP-completeness has become a kind of "cultural ambassador" of Computer Science to other fields, as the concept is used as indirect evidence of complexity in extremely diverse contexts (a bibliographic search finds that approximately 6,000 articles in conferences and journals every year, in fields ranging from political science to mathematical biology and power engineering, have "NP-complete" among their keywords).

3 Bridges to Applications: Some Promising Directions

It is essential for the TOC community in the U.S. to maintain a strong presence in the core of theoretical computer science. In addition, it is important to bring the work on fundamental algorithms into closer contact with computational practice, and to develop a stronger presence in selected application-specific areas.

In this section we suggest some possible future connections between TOC research and important application areas. We have selected areas where there is a good possibility of attracting funding from sources which have not traditionally supported theoretical computer science.

Application: Theory of Software Construction. As each generation of computer technology brings radical reductions in the cost of hardware, our inability to achieve a significant reduction in the cost of software development becomes a greater cause for concern. The demand for new software systems far outstrips our ability to construct them. There is a large gap between theory and practice in the area of software construction. At the interface between computer science and mathematical logic there have been impressive advances in our theoretical understanding of programming language semantics, specification languages and formal theories of program correctness, but these developments have not led to comparable advances in the pragmatic task of producing reliable software. It is time for this field to redirect itself towards this task. We recommend continuing support of research on the scientific foundations of software engineering and the verification of software systems, but with a strong preference given to investigations that are directly oriented toward the software development process.

Application: Computational Biology. The field of biology has been undergoing a conceptual revolution, in which cells and organisms are viewed as information processing systems. Biologists have learned that digital information in the form of DNA governs the chemical processes of the cell and constitutes the genetic endowment that is passed from parent to child. A vast effort is underway to sequence the human genome, elucidate the information it contains, and identify the abnormalities within it that account for genetic diseases. This quest has led to the development of bioinformatics as an important new professional specialty, and has introduced a need for new combinatorial algorithms and database techniques to support the acquisition and interpretation of genetic information. A growing number of theoretical computer scientists are rising to this challenge. While much of their work to date has been well motivated, certain topics have become fashionable which sound biologically relevant but omit key aspects, such as the presence of errors in all biological data. To ensure the relevance of future research in this area, proposals should be encouraged in which algorithm researchers work with biologists to formulate and solve real problems. We are hopeful that this effort will draw support from the biology side as well as from the traditional funding sources for theoretical computer science.

Future Application Areas. The choice of application areas for future major funding efforts should address the roadblocks that limit the effectiveness of computing and telecommunications in our society. Proposals for theoretical work motivated by the ubiquity of computers and communications, parallel and distributed systems, and large knowledge repositories should be warmly encouraged. Encouragement should also be given to theoretical research that supports the efficient construction of portable and reusable software. In the following two sections we propose two specific initiatives intended to focus theoretical research on these objectives. The first of these is entitled "Information Access in a Globally Distributed Environment" and the second is called "The Algorithmic Tool Kit." We anticipate that some theoreticians, looking at these two initiatives, will ask "Where's the theory?". Indeed, these initiatives largely represent virgin territory for theory, in which the necessary abstractions and theoretical models have not yet been formulated. We are confident that these areas can be a rich source of interesting and relevant theoretical ideas.

4 Information Access in a Globally Distributed Environment

From the inception of computer science as a discipline, theoretical work motivated by the random-access machine model of computation has had significant impact on the practice of computing; and the practice of computing has helped nourish theoretical development and use of new models of computation. The development of the national information infrastructure offers abundant opportunities for similar contributions from theory to practice, as well as strong motivation for the development of new theoretical models and the identification of new problems. The return on investments in theoretical research in this area should be substantial because much of the new infrastructure is still undeveloped.

Timely, accurate information is vital for our economy, health, and society. As computers and communications become ubiquitous, it should become possible for people wherever they are to find the information they need, in the media they desire, and in a form they can understand. However, we are far from achieving this goal:

- New models and methods are needed for the organization, storage, retrieval, processing, and presentation of multimedia information. Relational database theory had a profound impact on the design of today's database management systems but at present we do not have a similar foundation for the effective design of large, distributed, multimedia knowledge repositories. General models and algorithms for multimedia query languages are at the moment rudimentary and unsatisfactory.

- We need seamless mechanisms for allowing diverse information sources to interoperate with all intended applications. Our ultimate goal is to develop models around which to effectively organize and access all of human knowledge.

- We need methods for ensuring appropriate levels of security, safety, and privacy among users of highly-interconnected resources. Theoretical research in cryptography, communication protocols, system security and network reliability already informs technological developments in this area, but more work is needed to handle the questions and decisions likely to arise in the future.

- To support the rapid development of the new multimedia applications, we need tools and services for discovering, locating, and managing information distributed throughout networks of networks. The structure, form, function, and location of these tools and services pose difficult algorithmic problems in a number of areas including network and protocol design, routing, flow control, scheduling, and bandwidth optimization.

We therefore propose a research initiative on Information Access in a Globally Distributed Environment. The intent is to stimulate theoretical research motivated by the ubiquity of computers and communications, parallel and distributed systems, and large knowledge repositories. The goal is to provide the scientific foundation upon which we can design an infrastructure that permits universal access to the nation's information resources in as cohesive and cost-effective a manner as possible.

5 The Algorithmic Tool Kit

It is a notorious fact that software designers are continually reinventing the wheel. Software components are typically designed to meet a highly specific set of operating requirements and to run on a specific machine or family of machines. Because the operating conditions under which a program will work are so highly constrained, large software systems become rigid as they evolve over time, and eventually must be scrapped when they can no longer adapt to changing environments and functional requirements. Because software is not sufficiently portable or reusable, it is difficult for software designers to make use of the work of others, and a great deal of duplication of effort results.

It is also clear that in many cases a considerable time lag exists between the discovery of a better algorithm for some problem and the implementation of that discovery in application software. Indeed many theoretical advances in algorithm design are never implemented or tested at all. A long-standing communication gap between theoreticians and practitioners has led to a situation where software developers are not aware of leading-edge research in algorithm design, and theoreticians are not informed about the relevance of their results to real problems.

To counteract these trends we propose a research initiative to create the Algorithmic Tool Kit, an integrated suite of portable and reusable software components which we call *algorithmic building blocks*. These building blocks would be specified and implemented in an abstract setting, but would be capable of being ported to a variety of computing platforms and supporting a wide variety of applications. Typical applications might include VLSI design systems, database systems, graphics and multimedia applications, digital libraries, transaction processing systems, spreadsheet applications, and high-speed integrated communication networks.

The Algorithmic Tool Kit is related to a number of current efforts in academia and industry to build software repositories, notably the Leda project at Saarbrucken, Germany. It would exploit the vast body of knowledge concerning the representation, design, analysis and verification of algorithms and data structures that has been built up by the computer science research community. The initial set of building blocks would consist of implementations of fundamental data structures such as heaps, trees, dictionary search structures and graphs. The next layer of building blocks would be built upon this foundation, and would include implementations of well-analyzed algorithms for sorting, searching, parsing and string processing, manipulation of algebraic and geometric objects, discrete and continuous optimization, logical inference, and so forth. At the next layer would be building blocks to support communication and data access in parallel and distributed environments. These would include algorithms for routing, scheduling and resource allocation, protocols for communication, security and cryptographic protection, and implementations of distributed computation primitives such as mutual exclusion and distributed consensus. The choice of further building blocks to be included would be developed through consultation between specific application communities and researchers working on the design and analysis of algorithms.

The Tool Kit would be freely accessible over the Internet, and there would be a mechanism for collecting reports of user experience.

For the sake of reusability and portability, the algorithms would be specified at a level of abstraction that suppresses many machine-dependent details. The choice of a specification language for the algorithms would be a major research challenge. The specification language would provide the "glue" connecting the separate building blocks, ensuring that they used common data formats and had compatible interfaces, so that blocks could be freely composed together without unexpected side effects. This compatibility would permit the rapid prototyping of complex applications.

Many kinds of research efforts would be crucial to the success of the Algorithmic Tool Kit. These include: choosing appropriate specification and implementation languages; developing careful implementations of sophisticated algorithms; developing standards of software quality, including such aspects as functionality, serviceability, reliability and performance; assessing performance of algorithms and data structures under various environmental scenarios; developing methodologies for testing and validating interfaces between building blocks; and building search tools that find appropriate building blocks for a particular application.

We believe that this initiative will present exciting and relevant challenges to theoreticians concerned with the design, analysis and verification of algorithms, build strong bridges between the theory community and application developers, and lead to the creation of a valuable body of reusable and portable software.

6 Education

A critical problem facing theory students is a lack of job opportunities after graduation. Furthermore, there is a perception that theory students are less prepared for jobs in industry than are students in other areas of computer science, in part because mainstream theoretical research has become separated from real issues of development and application. For members of underrepresented groups who already experience some isolation within the discipline, this second layer of separation can be especially difficult to overcome.

To remedy the problem and to change the perception, graduate programs in computer science must be shaped to meet two broad goals. First, all graduate students, theoreticians and non-theoreticians alike, should be aware of both the potential and the limitations that both old and new theoretical results have for helping to solve applied problems. Second, young theoreticians must obtain greater exposure to practical research problems in computer science and other disciplines.

We recommend that academic computer scientists at institutions throughout the United States undertake a critical self-assessment of the relationships between theory and other research areas within their own programs. How much exposure do young theoreticians get to applications? How is theoretical research presented to graduate students in other areas? How much cross-disciplinary education occurs in the classroom, and how much occurs in the general research environment? How much isolation is there between different research groups at the professor level? Each institution has a responsibility to shape its educational progam to meet the needs of its graduate students; perhaps as never before, those needs include familiarity with a broad spectrum of computer science rather than narrow specialization.

We also recommend several measures by which funding agencies can support efforts to broaden the education of theory students and to strengthen the ties between theoreticians and practitioners. We focus on graduate and postdoctoral education, since curriculum development and research at the undergraduate and K-12 levels involve broader constituencies than just the TOC community.

- Develop fellowship programs to support summer or year-long visits by theory graduate students and postdocs to industrial research sites. These visits would allow students to collaborate on projects involving application of theoretical results to real problems.

- Develop programs to support summer or year-long visits by theory *faculty* to industrial research sites. Educators who are not familiar with applications of their research will not be able to pass this knowledge on to their students.

- Support multi-disciplinary, multi-institution, project-oriented research efforts, both large and small. The Human Genome Project is a good example of a large coordinated effort that supports the integration of theory and practice. It would be desirable to fund a range of smaller projects representing joint efforts by researchers with different areas of expertise.

- Support mechanisms by which theoreticians and practitioners (students and faculty) can work together and exchange problems and solutions. One possible medium for this type of exchange is the World Wide Web. Another is a series of topic-oriented workshops, perhaps sponsored by DIMACS.

7 Summary of Recommendations

The field of theoretical computer science has an outstanding record of achievement. It has contributed a rich set of fundamental concepts as well as many concrete results of value to practitioners. Nonetheless, the field is going through a critical period, especially in the United States. The U.S. job market for theoreticians is contracting and research funding is becoming difficult to obtain. Moreover, as theoretical computer science matures, there is a natural tendency towards specialization which makes its achievements less accessible, and therefore less appreciated by applied computer scientists and computer users. To help the theory of computing community in the U.S. prosper in this difficult environment, we offer the following recommendations.

Recommendation 1. It is essential for the TOC community to strengthen its communication with the rest of computer science and with other disciplines, and to increase its impact on key application areas.

Recommendation 2. It is also essential to maintain a strong activity in core theory and fundamental algorithms. Proposals in these areas are best evaluated by peer review.

Recommendation 3. Research which builds bridges between basic theory and applications must be given much greater priority. Funding for such bridging activity should come not at the expense of core theory, but rather by attracting funding from sources that have not traditionally supported theoretical computer science.

Recommendation 4. Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.

Recommendation 5. Theoretical computer scientists should provide accessible expositions of their work, both within their research articles and in expository articles aimed at a broad audience.

Recommendation 6. Theoretical computer scientists should establish close ties with practitioners, rather than isolating themselves within theory groups.

Recommendation 7. Graduate students in theoretical computer science should be exposed to applications and experimental research, and graduate students in other areas of computer science should be exposed to the methods and results of theoretical computer science.

Recommendation 8. We recommend continuing support of research on the scientific foundations of software engineering and the verification of software systems, but with a strong preference given to investigations that are directly oriented toward the software development process.

Recommendation 9. The ongoing revolution in biology, with its emphasis on information processing within organisms, offers great challenges and opportunities for theoretical computer scientists. We recommend augmented funding for collaborations between theoretical computer scientists and biologists. This funding should come from the biology side as well as the computer science side.

Recommendation 10. Research in application-specific theory should address the roadblocks that limit the effectiveness of computing and telecommunications in our society. Proposals for theoretical work motivated by the ubiquity of computers and communications, parallel and distributed systems, and large knowledge repositories should be warmly encouraged.

Recommendation 11. Funding agencies should consider a new initiative concerned with information access in a globally distributed environment. Theoretical computer scientists can play a significant role in developing the foundations of this subject.

Recommendation 12. We recommend a new funding initiative, the Algorithmic Tool Kit, as a vehicle for bringing the knowledge of the TOC community to bear on the problem of producing efficient, portable and reusable software.

Recommendation 13. We recommend that academic computer scientists at institutions throughout the United States undertake a critical self-assessment of the relationships between theory and other research areas within their own programs.

Recommendation 14. Funding agencies should support mechanisms by which theoreticians and practitioners (students and faculty) can work together and exchange problems and solutions. These include summer or year-long visits by theory graduate students, postdocs and faculty to industrial research sites and multi-disciplinary, multi-institution, project-oriented research efforts, both large and small.

Acknowledgements Professors Ajoy Datta and Lawrence Larmore of the University of Nevada at Las Vegas hosted the NSF-sponsored workshop at which this committee began its work. Brian Bourgon and Jerry L. Derby of UNLV prepared a transcript of the discussions at the workshop. Andrew Adamatzky, Eric Allender, Dana Angluin, Ronald V. Book, Anne Condon, Charles Elkan, Lance Fortnow, Peter Gacs, Susan Landau, Michael Luby, Jack Lutz, Leonard Pitt, Panos Pardalos, Serge Plotkin, David Shmoys, Carl Smith, D. E. Stevenson, Eva Tardos and Michael S. Waterman contributed to the preparation of white papers presented at the open meeting that preceded the workshop. David Applegate, Yoav Freund, Mike Kearns, Maria Klawe, Dana Latch, Nancy Lynch, Nick Pippenger, Leslie Valiant and Mihalis Yannakakis are among the many who provided advice and assistance to the committee. We thank all these people for their help.