COMPLEXITY AND PARALLEL PROCESSING: AN INTERVIEW WITH RICHARD KARP

KAREN A. FRENKEL

In the following interview, which took place at ACM 85 in Denver, Karp discusses the relation of his work to leading-edge computing topics like parallel processing and artificial intelligence. Tracing his experience as a pioneer in highly theoretical computer science, Karp describes how the decision to go against established wisdom led to the work for which he is best known and how a colleague's findings led him to see links between two previously unrelated areas. Throughout, he stresses the exchange of ideas with colleagues that helped yield fundamental insights.

KF **You decided fairly early on in your career to move from mathematics into computer science. Do you see yourself as a theoretical mathematician working in the realm of computer science, or as a computer scientist working on theoretical issues?**
RK I guess I'm somewhere in between an applied mathematician and a computer scientist. A priori I think the work I do could go either in a mathematics or computer science department, but the historical trend has been for computer science departments to take the major initiatives in developing theoretical computer science.

Most math departments have dropped the ball. There are a few exceptions, but in general, they didn't realize the potential of this field quite early enough to begin building it up. So it tended to fall within the purview of computer science departments. Nowadays, mathematics departments are finally becoming much more cognizant of theoretical computer science.

KF **Do mathematicians think about computation differently than computer scientists do?**
RK When mathematicians use computers, they tend to operate in a very nontheoretical manner. If a number theorist wants to factor a number, he'll throw everything at it but the kitchen sink. He wants that answer, and he usually isn't worried about the broader computational complexity issues. It's the same with group theorists or algebraic geometry. They're interested in this particular group, or that particular surface, and they want that answer—they become just like engineers. When I program, I'm the same way. For the first five minutes, I'm very conscious of theoretical issues, but then I just want to make the program work, and I forget that I'm a theoretician.

KF **Why has the traveling salesman problem received so much attention?**
RK The traveling salesman problem epitomizes and is a simplified version of the rather more complicated problems that occur in practice. Everyone knows that the traveling salesman problem is a metaphor or a myth—it's obvious that no salesman is going to worry about absolutely minimizing his mileage—but it is an interesting and an easily defined problem. It probably gets more attention than it deserves because of its catchy name. There are other important prototypical problems with less catchy names, like coloring, packing, matching, scheduling, and so forth. This is the way theory advances—you can't do clean theoretical work by taking on all the complications of real-world problems. So you take cleaner formulations, study them as closely as possible, go deeply into their structure, and hope that the results will transfer over to the real problems.

KF **It seems that you investigate metatheory—classes of problems—rather than real problems.**
RK Yes, that's right. There are three levels of problems. There's the level of solving a very specific instance: You want the shortest tour through the 48 con-

tinental state capitals plus Washington, D.C. That's the level closest to the practitioner. Then there's the level of studying the problem in general, with emphasis on methodology for solving it: You want to know what the complexity of the traveling salesman problem, or the marriage problem is, using the worst-case paradigm. That's one level up because you're not interested just in a specific instance. Then there's a metatheoretic level where you study the whole structure of a class of problems. This is the point of view that we've inherited from logic and computability theory. The question becomes, "What is the whole classification scheme? Let's look at the hierarchy of problem complexities as we increase the bound on the allowable computation time."

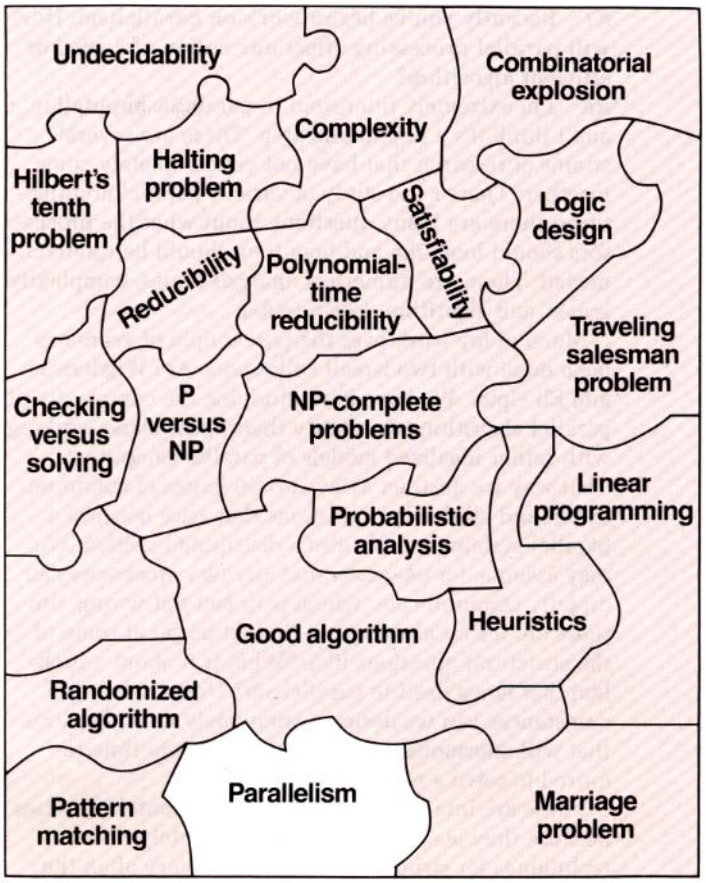
Every now and then, two levels have an interface. Such interfaces are usually very important. A lot of important work in science emerges when two fields meet that had not previously been perceived to be related. The concept of NP-completeness links the abstract study of complexity classes to the properties of particular problems like the traveling salesman problem or the satisfiability problem.

KF The step that you took toward probabilistic analysis was a departure from the worst-case analysis paradigm. And you pursued it despite its detractors. What pushed your decision?

RK I don't mean to give the impression that probabilistic analysis had never been heard of before I thought of it. It certainly had been applied, but mainly to problems of sorting, searching, and data structures, rather than to combinatorial optimization problems.

The decision was particularly difficult because, to a certain extent, I agreed with the detractors. There is a really fundamental methodological problem: How do you choose the probability distributions? How can you possibly know what the population of problem instances is going to be? Nobody had ever taken careful measurements of the characteristics of real-world problems, and even if they had, they would be measuring just one computing environment. But I didn't see any way out, because, if we didn't go the probabilistic route, NP-completeness would just be devastating.

Now there was also a line of research on approximation algorithms that do give guarantees. If you have an NP-complete combinatorial optimization problem, you can relax the requirement of getting an optimal solution and try to construct a fast algorithm that's guaranteed in its worst case to be not more than, say, 20 percent off. This is another very interesting paradigm that was explored, and it gave mixed results. In some problems, it really cleared up the difficulty—you could get a solution as close to optimum as you liked. For some other problems, you could guarantee being off by 22, 33, or 50 percent. Those results were very nice, but I didn't think they were descriptive of what happens when you use practical heuristics. Practical heuristics do very well most of the time, but not in the worst case.



So it wasn't that I relished the idea of working along a new direction whose foundations could be called into question. And it was also a personal risk, in that I could have been seen as flaky. You know, "He can't do the real thing, so he assumes some probability distribution and makes life easier for himself." But again, I just didn't see any other way to proceed. The phenomenon of NP-completeness persuaded me.

KF If you don't have the optimal solution to a problem, how can you know that your heuristic is producing something close to optimal?

RK That's a methodological difficulty. When you run a heuristic algorithm and it appears to be giving very good solutions, you can't be sure, since you don't know where the optimum lies. You may run your program from many different starting points and keep replicating the same solution. If nobody ever finds a better one, you have some circumstantial evidence that your solution is best. You can also invest a very large amount of computer time in a branch-and-bound computation and finally get a solution that you can prove is optimal to within half a percent. Then you run your quick heuristic on the same problem for three minutes or so and see how close it can come. Sometimes you can artificially construct the problem so that you'll know what the optimal solution is. But you've pointed out a severe methodological problem.

KF Recently you've begun work on parallelism. How will parallel processing affect our notion of a good or efficient algorithm?

RK I'm extremely interested in parallel computation, and I think it's a fascinating area. There are several strains of research that have not yet completely come together: There's the study of various parallel architectures; there are many questions about what the processors should look like and how they should be interconnected. There are numerical analysis issues, complexity issues, and algorithm design issues.

Much of my work over the past couple of years has been done with two Israeli colleagues, Avi Wigderson and Eli Upfal. We have been studying the complexity of parallel algorithms on a fairly theoretical plane working with rather idealized models of parallel computers. That way we abstract away certain issues of communication and all the complications that arise because a parallel system is really also a distributed system. We may assume, for example, that any two processors can directly communicate, which is in fact flat wrong. But these are useful abstractions that let us get at some of the structural questions like, "What is it about a problem that lends itself to parallelism? Under what circumstances can we design a completely new algorithm that will enormously reduce the amount of time required to solve a problem?"

These are interesting and important mental exercises because they lead us to discover completely different techniques for structuring algorithms. Very often the parallel algorithms that we come up with are very different from the sequential algorithms that we may use for the same problems.

Most of the work that the theoretical computer science community has been doing on parallel computation has been concerned with making polynomial-time algorithms even faster. We are asking ourselves, "Which problems in that class can be tremendously parallelized? What are the conditions under which computations can be compressed enormously?" In my future work, I will focus more attention on applying parallelism to NP-complete problems. Somebody with a very severe theoretical point of view could say, "That's hopeless, you can never reduce the run time from exponential to polynomial by throwing processors at a problem, unless you have an exponential number of processors." On the other hand, even though you may never be able to go from exponential to polynomial, it's also clear that there is tremendous scope for parallelism on those problems, and parallelism may really help us curb combinatorial explosions.

I intend to look at branch-and-bound, game trees, goal-subgoal structures, Prolog-like structures, backtrack search, and all of the various kinds of combinatorial searches, because I think that such problems are really well suited for parallel computation. The form that the theory will take is not yet clear.

KF What is the relationship between your interest now in parallelism and your earlier work with Raymond E. Miller?

RK There have been two main periods when I have been involved in studying parallel computation. The first was in the early to mid sixties, when I worked with Miller on several descriptive formalisms for parallel computation. In the more recent period, I've worked with Upfal and Wigderson on the design and analysis of parallel algorithms. Miller and I were originally motivated by considerations of whether it was feasible and desirable to design special-purpose hardware to enable computers to perform commonly executed iterative computations in parallel. We came up with several formalisms for describing parallel computations. One was very specific and concrete, and another was on a very highly theoretical plane. The models and methods that we came up with were very similar in spirit to the systolic designs later pioneered by H. T. Kung, Charles Leiserson, and others, although I don't mean to say that we anticipated all their ideas. There were certainly many insights that we did not have. But, in a sense, we were doing it too early—the world wasn't quite ready for it.

We were also interested in certain more qualitative questions like, "What happens if you're running asynchronously, and you don't have a master clock so there is no way of telling whether A happens before B and B happens before A? Can you still have a determinate result for the whole computation even though you can't control the order in which these various events are happening in parallel?"

The recent work is in a different direction. We have been concerned with complexity—with a given number of processors, how fast can you solve a problem? The two developments were quite distinct.

KF Do you think that perhaps your work will also contribute to parallel-processor design and help to determine the best ways to link different processors within a machine?

RK At the hardware level, the kind of thing I do is highly relevant. Laying out an integrated circuit chip is a bit like designing a city of 50,000 people. There are all sorts of combinatorial problems having to do with placing and interconnecting the various circuit modules. At an architectural level, the work I've been doing on parallel computation isn't directly relevant, because I've been using idealized models that fail to address the issues of communication between processors. I hope that my work will begin to move closer to the architectural issues.

KF Can we learn anything exciting about distributed communications and distributed protocols from theoretical studies?

RK Yes. There are some very beautiful theoretical developments having to do with how much you lose when you have to depend on message passing in a sparse network of processors rather than direct point-to-point communication between processors. In a realistic distributed system, the processors have to not only compute but cooperate like a post office where messages will flow between processors. There has been some very nice theoretical work on various kinds of protocols, where—as in the so-called Byzantine Generals problem, another one of those jazzy names—a number of processors have to reach agreement through message passing even when some are faulty and functioning as adversaries, trying to mess things up. It has become apparent that randomization is very powerful there. The kinds of protocols needed for these problems of cooperation and communication in a distributed system can be simplified if coin flipping is permitted. It's a fundamental insight that randomized algorithms can be applied in that setting. So there are many links between theoretical studies and protocols for real-life distributed systems.

KF **People use algorithms that seem to work perfectly well in practice even though they lack the theoretical pedigree. And they might say, "This work is fascinating, but if we can, by trial and error, come across algorithms that work fine, why concern ourselves with theory?" How will your work be applied in the most practical sense in the future?**
RK Some of the most important combinatorial algorithms could never have been invented by a trial-and-error process; having the right theoretical framework was absolutely necessary. Once the general shape of an algorithm has been determined, it is often possible to tune it empirically, but if you proceed in a purely empirical way, your knowledge is limited to the very specific circumstances in which you conduct experiments. The results of analysis, on the other hand, tend to be more susceptible to generalization. The justification for theory, apart from its apparent aesthetic attractions, is that, when you get a theoretical result, it usually applies to a range of situations. It's a bit like simulation versus analysis. They both certainly have their place, but most simulations only tell you about one very limited situation, whereas sometimes analysis can tell you about a whole range of situations. But the solution of combinatorial optimization problems is certainly as much an art as it is a science, and there are people who have wonderfully honed intuitions about constructing heuristic algorithms that do the job.

KF **What will be the focus of research at the Mathematical Sciences Research Institute (MSRI)?**
RK Well, I'm glad that you asked me about MSRI because that project is very close to my heart. It's a research institute up in the hills behind the Berkeley campus, but it's not officially connected with the university, and it supports year-long research programs in the mathematical sciences. In the past, these were mostly in pure mathematics. The primary support comes from the National Science Foundation (NSF).

About two years ago, Steven Smale, of the mathematics department at Berkeley, and I proposed a year-long project in computational complexity, and we were very pleased that it was accepted. I think it's an indication that the mathematics community, which was really slow to involve itself in computational complexity, has now become very receptive to the field. About 70 scientists will participate in this complexity theory research. They are evenly divided between mathematicians and computer scientists.

I'm very proud of the group we have assembled. People are pursuing a wide spectrum of topics. Some are doing metatheory, focusing on complexity classes like P and NP, rather than on concrete problems. Some are working on computational number theory where the central problem is factoring very large numbers. Others are concentrating on combinatorial problems. We're exploring the interface between numerical computation and complexity theory. And parallelism is a major theme. I'm absolutely delighted with the way it's going—the place is really the Camelot of complexity theory. There have already been a number of developments in parallel algorithms, just in the couple of months we've been operational.

KF **Could you be more specific?**
RK It would be premature to mention specific results, except to say that some of them make my earlier work obsolete.

KF **How much money is available for the MSRI project?**
RK The budget of the complexity project at MSRI in round numbers is $500,000 from NSF and $140,000 from the military services.

This program has been something of a windfall for complexity theory, but I very distinctly have the feeling that the general funding picture for computer science is worse than it has been in years. The NSF is undertaking some very worthy new initiatives, but it's doing so without a corresponding expansion in its funding base, so that these initiatives are being funded at the expense of existing programs. Although I must say that the MSRI program is an exception, on the whole, people in theoretical computer science are being squeezed by the reductions in funding as a consequence of changed emphases at NSF, mostly in an engineering direction.

KF **What might be the more practical interests on the part of the Department of Defense and the three services?**
RK The support that's coming from them is principally in the area of parallel and distributed computations, and we're planning to run a workshop in the spring that would bring together mathematicians, numerical analysts, and computer architects. Of course, there are all kinds of meetings on supercomputers and parallel computation these days, but this particular one will specifically explore the interface between complexity theory and the more realistic concerns of computer users and designers.

KF **There has been much debate over the merits of the Strategic Defense Initiative (SDI). Would you like to comment on it?**
RK I don't intend to make a speech about the Star Wars initiative nor do I pretend to be an expert on software engineering. But I have studied some of the

evidence that presses the point that it is very dangerous to build a distributed system of unprecedented proportions that cannot be operational or tested until the critical moment. I am persuaded by those arguments to the extent that I am resolved personally not to involve myself in it.

KF **Researchers in many fields are studying complexity. Can you comment on the relationship, if any, between the study of complexity in computer science and in other disciplines?**
RK Complexity means many different things—there's descriptive complexity and computational complexity. An algorithm may be quite complex in terms of the way its pieces are put together, and yet execute very fast, so that its computational complexity is low. So you have all of these different notions of complexity. It's not clear to me that electrical engineers, economists, mathematicians, computer scientists, and physicists are all talking about the same beast when they use the word complexity. However, I do think there are some very worthwhile and interesting analogies between complexity issues in computer science and in economics. For example, economics traditionally assumes that the agents within an economy have universal computing power and instantaneous knowledge of what's going on throughout the rest of the economy. Computer scientists deny that an algorithm can have infinite computing power. They're in fact studying the limitations that have arisen because of computational complexity. So there's a clear link there with economics.

Furthermore, one could say that traditional economics—here I'm really going outside my specialization—has disregarded information lags and the fact that to some extent we operate without full information about the economic alternatives available to us, much in the same way that a node in a distributed computer network can only see its immediate environment and whatever messages are sent to it. So the analogies are cogent, but one has to be careful because we're not always talking about the same thing when we speak of complexity.

KF **Do you use the term "heuristics" differently than do AI researchers?**
RK People in AI distinguish between algorithms and heuristics. I think that they're all algorithms. To me an algorithm is just any procedure that can be expressed within a programming language. Heuristics are merely algorithms that we don't understand very well. I tend to live in an artificially precise world where I know exactly what my algorithm is supposed to do. Now, when you talk about a program that's going to play good chess, translate Russian into English, or decide what to order in a restaurant—to mention a few tasks with an AI flavor—it's clear that the specifications are much looser. That is a characteristic of programs that those in AI consider heuristic.

KF **David Parnas points out in a recent article that systems developed under the rubric of heuristic programming, that is, programming by trial and error in the absence of a precise specification, are inherently less reliable than programming by more formal methods.**
RK Yes, and that brings us back to SDI. That's one of the reasons for being concerned about it. I think that we have much more apparatus for debugging a program when we can at least define what the program is supposed to do.

KF **Some members of the AI research community respond to that criticism by saying that they're trying to simulate humans and that humans have no precise specifications. The best they can hope for is a simulation of an unreliable system.**
RK I really believe in trying for crisp hypotheses and crisp conclusions. I realize that certain areas in computer science have to be dominated by empirical investigations, but that doesn't relieve us of the responsibility of thinking very hard about what it is we're measuring, what it is we're trying to achieve, and when we can say that our design is a success. And I believe that a certain measure of scientific method is called for. I don't buy the idea that simply because you're simulating the somewhat unknowable cognitive processes of humans you are relieved of the obligation to have precise formulations of what you're doing.

KF **Overall, how do you think computer science is doing as a discipline?**
RK Computer science has enormous advantages because of the tremendous importance and appeal of the field now. In some measures, we have been quite successful. A good portion of the young talent in the country is attracted to our field, especially in the areas of artificial intelligence and theoretical computer science.

In terms of our progression as a science, I think that to some extent we are victims of our own success. There are so many ways to get money, so many things to try, so many exciting directions, that we sometimes forget to think about the foundations of our discipline. We need to have a continuing interplay between giving free rein to our urge to tinker and try all kinds of neat things, and yet at the same time designing our experiments using the scientific method, making sure that the foundations develop well. Our tools are so powerful, the vistas are so great, the range for applications is so enormous, that there's a great temptation to plow ahead. And we should plow ahead. But we also have to remember that we're a scientific discipline and not just a branch of high technology.

KF **You have noted the importance of a mixture of art and science, insight and intuition, as well as the more rigorous methods of investigation. Have there been times when something just came to you and you experienced the so-called eureka phenomenon that inventors describe?**
RK I think we all have experienced it—waking up in the morning and having the solution to a problem. We have to remember that those eureka experiences are

usually preceded by a large amount of hard work that may sometimes seem unproductive. For example, when I read Cook's 1971 paper, it didn't take me very long to see that he had found a key to something phenomenally important, and to press on and try to demonstrate the scope and significance of his results. In a sense, it was almost instantaneous, but it was prepared for by well over a decade of work. I think it's characteristic that these moments when one makes connections come after a long period of preparation.

KF **Have you ever been talking to somebody, and an offhand remark they made caused something to click?**
RK Oh, sure. I find it very helpful to explain what I'm doing because my mistakes usually become obvious to me much quicker. And I listen to others because I'm really a believer in building up one's knowledge base. It greatly increases the probability that one will find unexpected connections later.

CR Categories and Subject Descriptors: A.0 [**General Literature**]: *biographies/autobiographies*; F.0 [**Theory of Computation**]: General; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*computability theory*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism, probabilistic computation*; F.1.3 [**Computation by Abstract Devices**]: Complexity Classes—*reducibility and completeness, relations among complexity classes*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; G.2.1 [**Discrete Mathematics**]: Combinatorics; G.2.2 [**Discrete Mathematics**]: Graph Theory; K.2 [**History of Computing**]: *people*
General Terms: Performace, Theory
Additional Key Words and Phrases: Richard Karp, Turing Award