September 2004

Robert Tarjan

The art of the algorithm

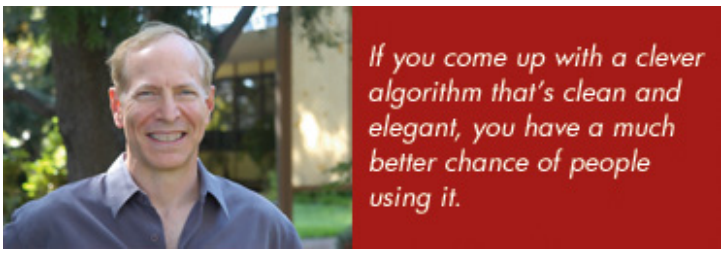
» HP Labs

» Research

» News and events
» Technical reports

» About HP Labs
» Careers @ HP Labs
» People
» Worldwide sites

» Downloads



by Jamie Beckett

Many of the most important innovations in computer science have come from people who are little known outside the field. HP Senior Fellow Robert Tarjan is one of these.

Tarjan has developed algorithms useful in everything from improving chip design to routing telephone calls, from optimizing deliveries in transportation networks to improving searches of large data sets. He's received such major national and international awards as the ACM Turing Award, known as the "Nobel Prize" of computing, the Nevanlinna Prize and most recently, the Blaise Pascal Medal in Mathematics and Computer Science.

A principal developer of sophisticated techniques for attaining very high efficiency in the manipulation of computer data structures, Tarjan has contributed techniques to many fields including computational geometry, graph theory and scientific computing.

"I was lucky to be in the right place at the right time," he says modestly.

At HP, Tarjan is responsible for identifying key technological trends and determining how the company can best respond, as well as for helping to coordinate and strengthen research and development efforts. In addition, he is working with a small group of HP Labs researchers in Princeton, New Jersey on problems in security, and with HP Labs researchers in Palo Alto on applications of algorithmic techniques.

At the same time, he continues his own research in data structures, algorithms and security. He also teaches at Princeton University, where he is the James S. McDonnell Distinguished University Professor of Computer Science.

In the interview that follows, Tarjan talks about creating an elegant algorithm, how he solves difficult problems, his passion for teaching and his views on the future of technology. The master mathematician also describes what he believes differentiates an impractical algorithm from a truly useful one.

What's the current focus of your research?

I have been working with several HP Labs colleagues, including two interns, on a provisioning problem that arises in the Adaptive Enterprise (HP's flexible, dynamically scalable IT infrastructure that allows companies to respond quickly and easily to changing business needs). The goal is to design a layered server farm to minimize the

Related links
» Robert Tarjan's biography
» Tarjan's Web page at Princeton University

News and events
» Recent news stories
» Archived news stories

Artist's conception of a self-adjusting search tree
(Drawn by Jorge Stolfi)
» click here for larger image

hardware cost while maintaining a required quality of service. We have devised both exact and approximate algorithms for the problem.

We are also working on a job scheduling problem, in which the goal is to complete as many jobs as possible on a parallel computer system, given a hard deadline. An added complication is that each job is broken up into a number of tasks, some of which must be completed before others are begun. Finding an exact solution to this problem seems to be computationally intractable, but we are exploring approximate methods that work well in practice.

The Adaptive Enterprise is a rich source of algorithmic problems, but there are many other applications in HP for algorithmic methods. For example, digital printing on the HP Indigo commercial printing presses (which provide on-demand, short-run personalized printing) gives rise to several interesting problems. How do you schedule print jobs most efficiently? How do you best meet customer needs for personalized document layouts?

You're also revisiting some of your earlier work on graph and data structure algorithms.

Yes, I'm working with two PhD students on what is called the dominators problem. This problem has to do with intersecting paths in directed graphs. Such a graph can model, for example, the flow of control in a computer program.

One way to improve the efficiency of such a program is to move blocks of code to places where they will be executed less frequently. For example, if an inner loop is executed many times but contains some initialization code that needs to be executed only once, then this code can be moved outside the loop. The new location of this code must be such that it will be executed no matter by what path the flow of control enters the loop.

This is where our work comes in: A computation can be used to find the possible safe places to move various code blocks. An algorithm I developed with a grad student in the 1970s is used for this purpose in many existing optimizing compilers. Our current work revisits this algorithm to improve it in various ways.

What excites you about this kind of complex work?

Solving puzzles and problems is intellectually stimulating. Finding a path to the right solution, and coming up with an elegant solution, as opposed to something messy, is very satisfying.

Why is simplicity so important? Is an elegant solution better than a messy one?

It is one thing to have an algorithm that is theoretically good. It is another to have an algorithm that is simple enough that someone would want to program it and use it in practice. Spending more time on a problem sometimes allows one to drastically simplify an initial solution, to boil it down to its essence. There are a few problems that I have thought about off and on for years, even decades, looking for simpler solutions.

Elegant algorithms are easy to program correctly, as well as being efficient. A clever algorithm that is clean and elegant is much more likely to be used than a messy one. When people understand how an algorithm works, which is much more likely with an elegant algorithm, they are more likely to have confidence in the results it produces.

Also, elegant solutions are much easier to generalize, to extend to other problems. My goal is to find general approaches and solutions, not ad hoc tricks.

When did your passion for solving problems emerge?

When I was a kid I used to like to solve puzzles and play games. I have carried my childhood enjoyment into my professional life. I played chess, checkers and mathematical games. I stopped playing chess when my younger brother started beating me. He eventually became an international grandmaster.

How do you go about solving a difficult problem?

I immerse yourself in it. I try various ideas and keep thinking about it until I'm sick of it. Then I stop thinking about it and come back to it later. To do mathematics you have to get used to pounding your head against the wall for a long time -- when you stop, it feels good. Ninety percent of the time your ideas don't work, but then the right idea emerges. This can be very exciting.

I'm sure a lot of creativity is unconscious. You just have to get the ideas into your head and sift them around. You have to give it some time. Often I think about problems in my sleep or when I'm out running.

How has your field changed over the years?

When I started doing algorithm design and analysis in the late 1960s in graduate school, the field was much less developed than it is now. The way people developed an algorithm then was to write a computer program and try it out. They rarely tried to do any kind of mathematical analysis.

Since it was a new field, it was easy to find problems to solve. Now the field is getting more and more technical and more and more esoteric.

On other hand, there are more and more opportunities for efficient algorithms to be useful because we have much more computational power. The Web is the biggest graph anyone has had to deal with. It is a very large and very important data set on which to try out algorithms. Developing the theory of algorithms is getting harder, but the work is getting more relevant to practice than it used to be.

How would you describe the unifying theme of your work?

It's all about trying to come up with the fastest, most efficient algorithms for a variety of problems and to understand the connections between different kinds of problems. It's also about understanding data structures and how data structures can be used to solve various sorts of problems.

This work -- data structures and algorithms -- is what I've been doing since I was a graduate student. Also, for almost 10 years I have had a sideline in security-related topics. Right now I'm thinking about what I'd like to work on in the security area with the group in Princeton and others in HP. The overall goal is to build trustworthy systems, with well-understood security and privacy properties.

You won the Turing Award for achievements in the "design and analysis of algorithms and data structures." What's the criteria for a "good" algorithm?

It must be fast. It must use a small amount of storage. And it must got be possible for someone to implement it, to write a computer program to make the thing work, which means it must be as simple as possible.

In the theoretical community, some people produce algorithms that work but aren't practical because no one can program them. I try to design algorithms that work well, both in theory and in practice.

Also a good algorithm can be generalized. It's good to come up with ideas that can be used for other problems -- which is to say, take a specific problem, try to solve it and then try to generalize the results to other situations.

You've been teaching at Princeton since 1985. What's satisfying about teaching? How does it relate to your research?

I see teaching as contributing to my research in a very positive way. The best way to understand ideas most profoundly is to try to present them to someone else. You find out fast where the holes are when you try to explain things to somebody. Teaching requires explaining things to students – people with eager young minds, at least some of whom are really motivated to learn. It's exciting. Even in undergraduate courses, I try to talk about frontiers of research.

There was a brief time when I wasn't teaching. I was at Stanford University and went to Bell Labs on sabbatical, where I spent a year

not teaching. I found I was losing some of my enthusiasm for research.

Working with graduate students also has the advantage of giving you leverage – you don't have to do everything by yourself. You come up with idea, and it's the grad students' job to go off and see if it works.

I like working with other people because you can exchange ideas. In addition, I've had a number of grad students with whom I've had a very good working relationship who are now distributed all over the planet. So it's rewarding in a personal way.

Were you always interested in science? How did you gravitate toward computer science?

When I was a kid I read a lot of science fiction, and I wanted to be an astronomer -- in particular, I wanted to be the first person on Mars. Then I started reading Martin Gardner's mathematical games column in Scientific American. In the eighth grade, I had a very stimulating math teacher, and then I got very seriously interested in math.

When I was going to graduate school, it was a question of mathematics or computer science. I ended up choosing computer science because I thought it was a way to do mathematics that could have a practical impact.

What technologies do you see ahead?

The three great technological waves these days are information technology, nanotechnology and biotechnology. Biotechnology is becoming more and more about processing information, and it's a rich space in which information technology can play. There are huge data sets that we need to understand; to understand them, we'll need to do complicated computations on them. If I was going to start my career over again, I think I would try to work on the boundary between info and bio.

One thing I see in the future of information technology is a critical need for trustworthiness. The security and privacy situation now is a real nightmare. Spam is a problem, viruses are a problem, and these problems are only going to get worse. Our systems are also extremely vulnerable to large-and-small-scale attacks of many sorts.

We need systemic fixes. Patching won't solve these problems. Building trustworthy systems is a very hard problem, made more difficult by the large installed base of faulty software. The developers of existing systems did not anticipate the problems that have emerged as these systems have become widely deployed.

Trustworthiness has to be ubiquitous and it has to be built in from the get-go. I think we should start with a clean slate and develop systems with the trust built in.

Your work has had widespread applications, from robotics to improving chip design to determining the best routing for goods in a transportation network. How does it feel to see your work have so much influence?

It's wonderful. It's very rewarding. I was lucky to be in the right place at the right time. I am glad that a lot of the hard work that I did has had value.

What work are you proudest of?

Proudest? It's hard to choose. I like the self-adjusting search tree data structure that Danny Sleator and I developed. That's a nice one.

How can we inspire our kids to become inventors?

Encourage kids to read. That's probably the number one starting place. We need to stimulate them. We need to present situations or problems in which they can participate actively, instead of passively absorbing entertainment media.

With TV, kids become completely passive. We have to give kids toys or games to which they can apply skill and imagination. I don't think computer games are the answer, at least not most computer games. They don't provide much opportunity for creative imagination and

they present pretty bleak messages -- the whole “shoot them up” kind of thing is scary.

Human beings have made it thus far, but there are plenty of things going on that have their downside as well their upside. All the new developments in rich media are wonderful in some ways, but some of these developments are depriving kids of the opportunity to create their own rich, imaginary environments.

Printable version