DOI:10.1145/1839676.1839677

Moshe Y. Vardi

On P, NP, and Computational Complexity

The second week of August was an exciting week. On Friday, August 6, Vinay Deolalikar announced a claimed proof that **P** ≠ **NP**. Slashdotted blogs broke the news on August 7 and 8, and suddenly the whole world was paying attention. Richard Lipton's August 15 blog entry at blog@CACM was viewed by about 10,000 readers within a week. Hundreds of computer scientists and mathematicians, in a massive Web-enabled collaborative effort, dissected the proof in an intense attempt to verify its validity. By the time the *New York Times* published an article on the topic on August 16, major gaps had been identified, and the excitement was starting to subside. The **P** vs. **NP** problem withstood another challenge and remained wide open.

During and following that exciting week many people have asked me to explain the problem and why it is so important to computer science. "If everyone believes that **P** is different than **NP**," I was asked, "why it is so important to prove the claim?'' The answer, of course, is that believing is not the same as knowing. The conventional "wisdom'' can be wrong. While our intuition does tell us that finding solutions ought to be more difficult than checking solutions, which is what the **P** vs. **NP** problem is about, intuition can be a poor guide to the truth. Case in point: modern physics.

While the **P** vs. **NP** quandary is a central problem in computer science, we must remember that a resolution of the problem may have limited practical impact. It is conceivable that **P** = **NP**, but the polynomial-time algorithms yielded by a proof of the equality are completely impractical, due to a very large degree of the polynomial or a very large multiplicative constant; after all, $(10n)^{1000}$ is a polynomial! Similarly, it is conceivable that **P** ≠ **NP**, but **NP** problems can be solved by algorithms with running time bounded by $n^{\log\log\log n}$—a bound that is not polynomial but incredibly well behaved.

Even more significant, I believe, is the fact that computational complexity theory sheds limited light on behavior of algorithms in the real world. Take, for example, the Boolean Satisfiability Problem (SAT), which is the canonical **NP**-complete problem. When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole. Garey and Johnson's classical textbook showed a long sad line of programmers who have failed to solve **NP**-complete problems. Guess what? These programmers have been busy! The August 2009 issue of *Communications* contained an article by Sharad Malik and Lintao Zhang (p. 76) in which they described SAT's journey from theoretical hardness to practical success. Today's SAT solvers, which enjoy wide industrial usage, routinely solve SAT instances with over one *million* variables. How can a scary **NP**-complete problem be so easy? What is going on?

The answer is that one must read complexity-theoretic claims carefully. Classical **NP**-completeness theory is about *worst-case* complexity.

Indeed, SAT does seem hard in the worst case. There are SAT instances with a few hundred variables that cannot be solved by any extant SAT solver. "So what?'' shrugs the practitioner, "these are artificial problems." Somehow, industrial SAT instances are quite amenable to current SAT-solving technology, but we have no good theory to explain this phenomenon. There is a branch of complexity theory that studies average-case complexity, but this study also seems to shed little light on practical SAT solving. How to design good algorithms is one of the most fundamental questions in computer science, but complexity theory offers only very limited guidelines for algorithm design.

An old cliché asks what the difference is between theory and practice, and answers that "in theory, they are not that different, but in practice, they are quite different." This seems to apply to the theory and practice of SAT and similar problems. My point here is not to criticize complexity theory. It is a beautiful theory that has yielded deep insights over the last 50 years, as well as posed fundamental, tantalizing problems, such as the **P** vs. **NP** problem. But an important role of theory is to shed light on practice, and there we have large gaps. We need, I believe, a richer and broader complexity theory, a theory that would explain both the difficulty and the easiness of problems like SAT. More theory, please!

Moshe Y. Vardi, EDITOR-IN-CHIEF