Intersect, Vol 8, No 3 (2015)

Where Computer Science and Education Intersect: An Interview with Mehran Sahami, Ph.D.

Joyce B. Kang
Stanford University

Mehran Sahami is a professor and the Associate Chair for Education in the Computer Science department at Stanford University. He is also the Robert and Ruth Halperin University Fellow in Undergraduate Education. Professor Sahami completed his BS and PhD degrees in Computer Science at Stanford before working as a Senior Engineering Manager at Epiphany and a full-time Senior Research Scientist at Google. He taught as a lecturer at Stanford from 2001 to 2006, and joined the CS faculty in 2007. His research interests include computer science education, artificial intelligence, and web search. In this interview, Professor Sahami discusses computer science education, his personal experiences teaching computer science, and the growing role that computer science plays in our society.

JK: How did you first become interested in computer science?

MS: I first got interested in programming in the fifth grade, actually. My elementary school got these little PET computers that each had 4K of memory, and a group of us were taught some basic programming. I got really interested in it, but there just weren't a lot of outlets to do it. Finally, in junior high, we got a personal computer at home, an old Apple IIe, and so I taught myself some more programming on that. It was kind of fun—making games, things like that—and so, I knew I was really interested in it, but I didn't do anything sort of serious with it until I actually got to Stanford. I took 106A [(Programming Methodology, Stanford's introductory computer science course)] when I got here and really loved it, majored in computer science, and went on to graduate school. This was really the place where the interest flowered into real computer science as opposed to just tinkering.

JK: Before serving on the faculty here at Stanford, you worked as a Senior Research Scientist at Google as well as a Senior Engineering Manager at Epiphany, Inc. What is something that you learned from your experiences in the industry that you would not have gained otherwise?

MS: There are certainly things that I gained there that I wouldn't have necessarily gotten in academia. Part of it was the exposure to scale of data and machines, and problems that you have at a place like Google are just different from those that you have here: you don't get billions of web queries at Stanford. I gained an appreciation for scale and the way you could think about approaching problems when you had that much data or that much computation available. But part of it was beyond just the scale issue. It was an appreciation for what were the actual problems that were being dealt with in industry. I think that academic preparation provided a great foundation to then build on, but some of the problems that you see in industry are different from the problems you see in a classroom setting.

JK: It seems that here at Stanford, there's not really a shortage of students who are interested in CS, but this is not the case across the rest of the country. How do you think we can encourage more students to consider CS, especially those traditionally underrepresented in the field?

MS: Right. Well, I think in general it *is* happening. Many other places are also seeing increases in computer science enrollment. In general, it's not as much as Stanford—if you look at the national numbers, which are actually tracked, they haven't increased as much as Stanford—but there *are* some places where there have been pretty dramatic increases as well. I think one of the things is just the pipeline early on in terms of getting students exposed to the notion of computing: what it is, computational thinking, what computer science can even be about. When they get to college, many students have had many years of math, many years of science, many years of writing and reading. They've even potentially had exposure to things like economics or statistics, depending on the school they went to. Surprisingly, computer science is actually not that broadly taught in high schools. Well-resourced high schools will have classes, but that especially skews the pipeline toward high-income and well-resourced places, and so that means many students who come from under-resourced backgrounds haven't necessarily gotten exposure. I think that's the biggest thing: having more exposure to computing—what's possible to do with it, the kinds of problems you can solve, the kind of impact you can have—earlier on in the educational pipeline. Here, we do try to do some outreach activities during the summer with high school students from underrepresented backgrounds or from more challenged socioeconomic backgrounds. We've done workshops in the past for teachers as well, so we try to do what we can, but our main focus is on higher education.

JK: So going off of that idea, what would you say is the most pressing problem in the current way we teach CS, whether at the collegiate level or not, and how might we be able to fix this or overcome it?

MS: One of the problems that I think is a problem now but that wasn't a problem, say, seven years ago, is dealing with the scale. At a place like Stanford, where you have a very large percentage of the entire student population wanting to major in computer science and take computing courses, the demand creates a lot of strain on resources—having enough teaching assistants, having enough faculty, having enough opportunities for students to get help—and so that can create some frustration in the problem that students don't feel like they are getting help. That can potentially lead to unwanted activity, like someone trying to seek out solutions from other sources that aren't authorized. I think that's one of the things that makes dealing with the scale that tricky. I mean, it would be great if CS 106A were a ten-person seminar and I could work personally with every student, but if we did that, that would mean that seven hundred students every quarter would not get access to that class, and so I think that's one of the tensions we need to deal with. Oftentimes, students will request things like making the courses smaller, with the belief that if we made the courses smaller, that student would be in that small course. The reality is, well, we *could* make the courses smaller, but then there's a good likelihood that the student wouldn't be in that course at all. The tradeoff is that as a department we've made the decision to try as much as possible not to cap our classes and allow for students to enroll who want to enroll, but with that needs to come an understanding and responsibility with the students that to have this kind of uncapped enrollment means that the amount of attention we can give to each student individually gets harder and harder as the size of the classes grow. And I think dealing with that tradeoff appropriately is probably the biggest problem we're facing now.

JK: That makes sense. Many argue that student desires to pursue "practical" majors like CS dissuade them from considering life's major questions and pursuing a liberal arts education. How would you respond to people with these views?

MS: I think the important thing is that there's this false dichotomy between doing computing and considering life's big questions. I think it's entirely possible for someone to think of computing as a means to address life's big questions, and if you look at some of the things that are going on in terms of, say, biotechnology research that is computing-intensive, that's a way of addressing a major human health problem through computation. And so, I think part of it is actually this stereotype casting where there's this the belief that, "Oh, someone's going to major in computer science just because they want to go do startups or they want to make apps or whatever, and they just want to make a lot of money." For some fraction of students that's probably true (that's where the stereotype probably comes from), but that really does a disservice to all the other students who are getting into computing because they want to have a big impact on big problems in the world, and they just get painted with the same brush—that they don't care about big problems. So, I think we need to work to try to dispel that myth because there are a large number of students who go into computing precisely because they see it as a powerful means of attacking life's big problems.

JK: What is one thing you wish all of your students would leave your class having gained, if they could only take one thing away from it?

MS: I think just an appreciation for problem solving. And I think that's one thing that sometimes gets overshadowed in the introductory classes because they're about programming. Students sort of feel like it's about the subtleties of the syntax of the language and figuring out how a loop really works and what happens when you use this particular operator in this particular situation, and sure, we need to explain all that stuff so students can write programs, but the point of writing programs is to solve problems. There's a particular way of thinking with respect to having clarity of thought to solve problems with a computer program because you have to be super precise when you specify that solution to a computer. And I think it's that same notion of being precise and clear in one's thinking with the goal of solving a problem in mind, and computing and programming are just a substrate for doing that, but that's really the underlying theme that I hope students get out of the class.

JK: Many students here see and know you as a fantastic teacher, but what is something that you've learned from your students?

MS: There are a lot of things that I've learned from my students over the years. One of them is just about the craft of teaching—the different ways students learn, the different kinds of scaffolding that you can provide to aid their learning process, the diversity of backgrounds that they come from—and that can mean all kinds of things in terms of what sort of teaching methods resonate best with them, where they actually are in their learning process, how confident they feel about what they're learning and about being in an environment at Stanford. In that respect, there's a lot I've learned from my students about how to do a better job, because that's the main reason why many of us are here: because we really care about teaching. We also care about research, but we're very invested in the teaching enterprise. Part of that is paying real attention to whom you're teaching and how they're learning. There's a broad spectrum of things—there are students who have shown me about new problems that they were solving, and new approaches that they were taking that have been interesting, there have been students who have shown me about new technical solutions to problems, and things like that that I hadn't seen before—and I always appreciate that too. At the end of the day, all those things are great, but the thing I value most is that interaction with students that helps me become a better teacher so that I can try to help more students in the future.

JK: What would you say is your philosophy on teaching, and how has it developed through all of your teaching experiences?

MS: Part of it is trying to teach in a way that really resonates with students and that they retain the material and feel confident and empowered by what they're learning. I don't view teaching as, "There's a set of material that needs to be talked about in class, and if I say the words of that material in class, somehow, magically, students will retain it, and they'll perfectly master it." Some students will, and that's great, but part of the teaching process is being aware of the way information is being transmitted and how that information is actually being assimilated and retained. For example, I like to use analogies a lot when I teach. Part of the reason for that is that it allows students to take some new knowledge that they're supposed to be learning, and be able to fit that into knowledge they already have, so they can understand what the parallels are, and they get a more concrete example for how things work. And so the hope is, through those kinds of examples, they'll retain and better understand the material. But it's an ongoing process: by no means have I stopped learning about teaching. There are always new things to learn, and I appreciate being able to work with great colleagues who also deeply care about teaching. We are constantly sharing ideas about things that we're doing or adopting each other's practices, and so being in this kind of environment is just wonderful for that kind of thing.

JK: You played a key role in revising Stanford's undergraduate computer science major. What did you feel was lacking in the old curriculum, and how do the revised major requirements reflect this?

MS: I think the old curriculum was a good curriculum; the main thing it was lacking was flexibility. All students went through the same set of requirements, and there were a few options at the end—like two or three classes that you could sort of make different choices for as electives—but to first order most students were taking the same fifteen classes. In the meantime, in the last twenty or thirty years, there's been a tremendous growth in computing, where there are a lot of sub-areas now, a lot to learn in each specialization, and we wanted to provide the opportunity for each student to be able to go deep into the area of computer science that they were most interested in, and to be able to do that at the undergraduate level. Therefore, the curriculum revision's main goal was to provide that flexibility by narrowing down into the core of a few classes—the stuff that was required for all CS majors—and then providing lots of track options for students to really go deep in the area they were most interested in. And so it has this sort of dual effect. One effect is that students can now learn a lot about the area they really care about so that they can become experts in that area without having to go to graduate school, and at the same time, it allows students who are CS majors to spend their time really focused on the part of CS they really like the most. That's kind of a win-win all around: they're getting really good at the thing they really like, and so we're pretty happy with how things turned out.

JK: In addition to computer science education, your other research interests span machine learning and information retrieval. Could you speak for a little bit on these interests and any others you may have?

MS: Sure. Some of them come from my original research work in graduate school, where my focus was in machine learning. Then, when I went out into industry, it was looking at applications of machine learning and data analytics in different contexts, and part of that even early on was related to text analysis. And then going to Google, it was pretty clearly about information retrieval and search and a lot of the issues that come up there, especially doing it on the scale of the Web and with a lot of the weirdness that happens on the Web. That was sort of the natural progression of research from what I was doing in graduate school, but I was always interested in teaching. In graduate school, I had the opportunity to teach some classes, and as an undergrad, I was a section leader, and even when I went into industry, I was still part-time teaching classes here as a lecturer. When the opportunity came up to come back here full time, that was the chance to make teaching and education my primary focus. There is still some of the research related to machine learning that goes on, but that's mostly research in support of education, so now it's applying machine learning methods or developing new methods to analyze educational data. So the two work together pretty well.

JK: That's really cool! Could you tell me more about a specific project with that idea of tying machine learning to education that has worked?

MS: Sure. So there are a few different ones. One main one is actually a project that is led by Chris Piech, who's a graduate student that I work with. He's done a lot of different work with many different facets, but one example would involve having traces of people solving programming problems. Based on analyzing those traces by doing some clustering, by doing some path analysis, can you do things like be able to understand where students are having difficulty, understand different kinds of modalities to solve a problem, and be able to potentially generate hints automatically (as one way to approach scaling learning)? And so Chris has actually done that at scale. There's an organization called Code.org that has their "Hour of Code," where they have a bunch of online problems that students can solve. Chris got data on over a million students and their paths in terms of solving problems, and sort of based on that, he created a system that could actually give students hints on how to improve their program as they were programming, and that's all generated from just data and machine learning techniques.

JK: That's really neat! So, to switch gears, how do you see the role of computer science in our society today, and how do you see it evolving into the future?

MS: Well, I think CS has already permeated many aspects of life and will continue to grow in terms of its influence and the number of things that rely on computing to work. I mean, fifty years ago, you bought a car, and it was mostly mechanics. Now, you buy a car, and it has somewhere between twelve and fifteen computers in it, doing all kinds of things in the system. Part of that is, to many of us…just transparent—your car just works, or some other gadget just works. But, the way we think of computing as a substrate for information flow is pretty huge, and as humans are social beings, I think computing is going to continue to play a really big role in creating mechanisms that allow us to exchange information more fluidly. But, it's also going to help solve bigger problems. We talked a little bit before about computing in biology, but you could also imagine things like transportation. Can we use computing to get more efficient transportation? People have talked about self-driving cars for about ten years, and in our lifetime it will likely happen. When exactly is unclear (that's kind of debated), but I think it'll happen. If you can do that, you can also think about ways that you can make much more efficient systems. You'll probably need fewer cars if people are sharing them, because most people's cars are doing nothing most of the time—just sitting there parked somewhere. So, if we were to not have to build that many cars, it would be a huge amount of resource savings. If you can also do things like have the cars drive in formation and draft and drive at optimal speeds, there are potentially a lot of energy savings there as well. There are these problems that are weird to think about as computational problems, which are things like improving transportation and getting better energy efficiency, but ultimately, computing is the substrate that helps some of those things happen. And so I think we're going to see more of that as time goes on.

JK: Since computer science has come to impact so many areas of our daily lives, do you foresee any potential challenges or dangers associated with its growth?

MS: I think part of it, just like with anything else, is overreliance. To what extent do people understand how the technology is working and are able to compensate and adapt when something doesn't work right? That's one of the issues that people have—they worry about autonomous cars, for example, because what if they get into accidents? Well, the truth is that autonomous cars don't have to be flawless, they just have to be better than people, and it turns out people actually have a lot of problems with driving, but those are the kinds of tradeoffs we need to look at. When we see a particular technology that has potential upsides and downsides, what's the right way to find the balance for it? It's easy to have gut reactions about things without really thinking about, "Well, what are the implications, what does that really mean?" It's not like we would get zero people dying in car accidents, but if we got one hundred thousand less people killed in car accidents, I think that would be a pretty significant improvement in the situation. Part of that is just understanding how we make those tradeoffs, what kinds of policies we make around these sort of new technologies to try to keep them as safe as possible, but also having an understanding of the limits of what the technology can do.

JK: What possible future applications of computer science are most exciting to you right now?

MS: Probably computational biology, right now. That's because I think about the potential for improving human health on a pretty massive scale in a lot of different ways. And, to the extent that we can do that, I think there are then a lot of other issues we then need to work through—societal implications and resource implications and all that—but I think that it would be a pretty tremendous step forward if we can understand things like diseases and pathologies better, if we can come up with better treatment, and if we can help increase the quality of life people have and the amount of time they have. That's pretty huge.