The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/2911969

<http://cacm.acm.org/blogs/blog-cacm>

The Solution to AI, What Real Researchers Do, and Expectations for CS Classrooms

John Langford on AlphaGo, Bertrand Meyer on Research as Research, and Mark Guzdial on correlating CS classes with laboratory results.

John Langford
AlphaGo Is Not the Solution to AI
<http://bit.ly/1QSqgHW>
March 14, 2016

Congratulations are in order for the folks at Google Deepmind (<https://deepmind.com>) who have mastered Go (<https://deepmind.com/alpha-go.html>).

However, some of the discussion around this seems like giddy overstatement. *Wired* says, “machines have conquered the last games” (<http://bit.ly/200O5zG>) and *Slashdot* says, “we know now that we don’t need any big new breakthroughs to get to true AI” (<http://bit.ly/1q0Pcmg>). The truth is nowhere close.

For Go itself, it has been well known for a decade that Monte Carlo tree search (MCTS, <http://bit.ly/1YbLm4M>; that is, valuation by assuming randomized playout) is unusually effective in Go. Given this, it is unclear the AlphaGo algorithm extends to other board games

where MCTS does not work so well. Maybe? It will be interesting to see.

Delving into existing computer games, the Atari results (<http://bit.ly/1YbLBgl>, Figure 3) are very fun but obviously unimpressive on about a quarter of the games. My hypothesis for why their solution does only local (epsilon-greedy style) exploration rather than global exploration so they can only learn policies addressing either very short credit assignment problems or with greedily accessible policies. Global exploration strategies are known to result in exponentially more efficient strategies in general for deterministic decision process (1993, <http://bit.ly/1YbLKjQ>), Markov Decision Processes (1998, <http://bit.ly/1RXTRCk>), and for MDPs without modeling (2006, <http://bit.ly/226J1tc>).

The reason these strategies are not used is because they are based on tabular learning rather than function fitting. That is why I shifted to Contextual Bandit research (<http://bit.ly/1S4iiHT>) after the 2006 paper. We have learned quite a

bit there, enough to start tackling a Contextual Deterministic Decision Process (<http://arxiv.org/abs/1602.02722>), but that solution is still far from practical. Addressing global exploration effectively is only one of the significant challenges between what is well known now and what needs to be addressed for what I would consider a real AI.

This is generally understood by people working on these techniques but seems to be getting lost in translation to public news reports. That is dangerous because it leads to disappointment (<http://bit.ly/1q11dDW>). The field will be better off without an overpromise/bust cycle, so I would encourage people to keep and inform a balanced view of successes and their extent. Mastering Go is a great accomplishment, but it is quite far from everything.

See further discussion at <http://bit.ly/20106Ff>.

Bertrand Meyer
What's Your Research?
<http://bit.ly/1QRo9Q9>
March 3, 2016

One of the pleasures of having a research activity is that you get to visit research institutions and ask people what they do. Typically, the answer is “I work in X” or “I work in the application of X to Y,” as in (made-up example among countless ones, there are many Xs and many Ys): *I work in model checking for distributed systems*. Notice the “in.”

This is, in my experience, the dominant style of answers to such a question. I find it disturbing. It is about research as a job, not research as research.

Research is indeed, for most researchers, a job. It was not always like that: up to the time when research took on its modern form, in the 18th and early 19th centuries, researchers were people employed at something else, or fortunate enough not to need employment, who spent some of their time looking into open problems of science. Now research is something that almost all its practitioners do for a living.

But a real researcher does not just follow the flow, working “in” a certain fashionable area or at the confluence of two fashionable areas. A real researcher attempts to solve open problems.

This is the kind of answer I would expect: I am trying to find a way to do A, which no one has been able to do yet; or to find a better way to do B, because the current ways are deficient; or to solve the C conjecture as posed by M; or to find out why phenomenon D is happening; or to build a tool that will address need E.

A researcher does not work “in” an area but “on” a question.

This observation also defines what it means for research to be successful. If you are just working “in” an area, the only criteria are bureaucratic: paper accepted, grant obtained. They cover the means, not the end. If you view research as problem solving, success is clearly and objectively testable: you solved the problem you set out to solve, or not. Maybe that is the reason we are uneasy with this view: it prevents us from taking cover behind artificial and deceptive proxies for success.

Research is about solving problems; at least about trying to solve a problem, or—more realistically and modestly—bringing your own little incremental contribution to the ongoing quest for a solution. We know our limits, but if you are a researcher and do not naturally describe your work in terms of the open problems you are trying to close, you might wonder whether you are tough enough on yourself.

Mark Guzdial
CS Classes Have Different Results than Laboratory Experiments—
Not in a Good Way

<http://bit.ly/1UUrOUu>
March 29, 2016

I have collaborated with Lauren Margulieux on a series of experiments and

papers around using subgoal labeling to improve programming education. She has just successfully defended her dissertation. I describe her dissertation work, and summarize some of her earlier findings, in the blog post at <http://bit.ly/23bxRWd>.

She had a paragraph in her dissertation’s methods section that I just flew by when I first read it:

Demographic information was collected for participants’ age, gender, academic field of study, high school GPA, college GPA, year in school, computer science experience, comfort with computers, and expected difficulty of learning App Inventor because they are possible predictors of performance (Rountree, Rountree, Robins, & Hannah, 2004; see Table 1). These demographic characteristics were not found to correlate with problem solving performance (see Table 1).

Then I realized her *lack* of result was a pretty significant result.

I asked her about it at the defense. She collected all these potential predictors of programming performance in all the experiments. Were they *ever* a predictor of the experiment outcome? She said she once, out of eight experiments, found a weak correlation between high school GPA and performance. In all other cases, “these demographic characteristics were not found to correlate with problem solving performance” (to quote her dissertation).

There has been a lot of research into what predicts success in programming classes. One of the more controversial claims is that a mathematics background is a prerequisite for learning programming. Nathan Ensmenger suggests the studies show a correlation between mathematics background and success in programming *classes*, but not in programming *performance*. He suggests overemphasizing mathematics has been a factor in the decline in diversity in computing (see <http://bit.ly/1ql27jD> about this point).

These predictors are particularly important today. With our burgeoning undergraduate enrollments, programs are looking to cap enrollment using factors like GPA to decide who gets to stay in CS (see Eric Roberts’ history of enrollment caps in CS at <http://bit.ly/2368RmV>). Margulieux’s results suggest choosing who gets into CS based on GPA might

be a bad idea. GPA may not be an important predictor of success.

I asked Margulieux how she might explain the difference between her experimental results and the classroom-based results. One possibility is that there *are* effects of these demographic variables, but they are too small to be seen in short-term experimental settings. A class experience is the sum of many experiment-size learning situations.

There is another possibility Margulieux agrees could explain the difference between classrooms and laboratory experiments: we may teach better in experimental settings than we do in classes. Lauren has almost no one dropping out of her experiments, and she has measurable learning. *Everybody* learns in her experiments, but some learn more than others. The differences cannot be explained by any of these demographic variables.

Maybe characteristics like “participants’ age, gender, academic field of study, high school GPA, college GPA, year in school, computer science experience, comfort with computers, and expected difficulty of learning” programming are predictors of success in programming *classes* because of *how* we teach programming *classes*. Maybe if we taught differently, more of these students would succeed. The predictor variables may say more about our teaching of programming than about the challenge of learning programming.

Reader’s comment:

Back in the 1970s when I was looking for my first software development job, companies were using all sorts of tests and “metrics” to determine who would be a good programmer. I’m not sure any of them had any validity. I don’t know that we have any better predictors today. In my classes these days, I see lots of lower-GPA students who do very well in computer science classes. Maybe it is how I teach. Maybe it is something else (interest?), but all I really know is that I want to learn better how to teach.

—Alfred Thompson

John Langford is a Principal Researcher at Microsoft Research New York. **Bertrand Meyer** is a professor at ETH Zurich. **Mark Guzdial** is a professor at the Georgia Institute of Technology.