Teaching Statement

Matthew P. Johnson

mpjohnson@gmail.com ~ nesl.ee.ucla.edu/fw/matt/

What first set me on the path to an academic career, back before I had any idea what “research” entailed, was a desire to teach. One of the most satisfying parts of my work—indeed, of life generally—is sharing knowledge with others, of getting them to understand what I understand, to see what I see. Learning some cool new thing without explaining it to anyone else would seem...not pointless, exactly, but greatly diminished. So I’ve always sought out teaching opportunities. TA-ing in college led to a rare and sought-after “MS-TA-ship” funding my master’s degree; later, while working in industry, I would race from Wall Street up to 116th Street to teach over my (long) lunch hours. Then during grad school there were lots of teaching opportunities, though at some point I reluctantly came to see the wisdom in focusing solely on research for awhile. This was a couple of years ago now, and I can’t wait to return to the classroom.

I’ve had the experience of teaching a variety of different sorts of students over the years, in a variety of settings—at Columbia, NYU Stern, CCNY, and the CUNY Institute for Software Development and Design (CISDD), with class sizes ranging from 8 to ≈80—and so have had occasion to present different facets of CS, in different styles. The intensive SQL classes I taught to adult CISDD students were substantially different, even if there was significant overlap in the core material, from the DBMS classes I taught to undergrad information science majors at NYU, for example, and I taught data structures somewhat differently at Columbia and CCNY. In all my teaching, I try to design the course informed by the backgrounds, needs, and goals of the students involved. So, for example, if the CISDD classes seemed optimized for short-term career payoff, and hence less “academic” than the NYU classes, that’s because they were: their purpose was to provide the students with skills to help them advance in their immediate careers. (It was heartening afterward to hear from some of the students, previously out-of-work, who’d subsequently obtained IT jobs.)

On the other hand, in academic teaching, I convey high (if realistic) expectations to my students. (“Pretend this is the honors section.”) It’s enormously gratifying when students, thus motivated, achieve more than they or their peers had thought possible. Occasionally, pep talks are involved, in which I emphasize—sincerely!—what a profoundly strange, unnatural human activity computing programming is, lacking as it does the forgiveness that obtains, to varying degrees, in all other earlier forms of communication. (This sometimes helps.) And whenever possible, I aim for students to understand things all the way down. I recall a student asking once why additive and multiplicative constants are both dropped in $O(\cdot)$ notation—but not constants in the exponent, although these too are indicated with numerals. This led to a ten-minute discussion (this was in a summer session with 3 1/2-hour meetings (!), so occasional breaks from official “material” were welcome), justifying the distinction as non-arbitrary.¹

Various people, as diverse as Steve Jobs and Scott Aaronson, have urged that we treat CS as one of the liberal arts. In a related spirit, Papadimitriou has advocated for recognition of the “computational lens” as the prism through which scientific thought generally is increasingly refracted. As someone who switched majors from philosophy to CS after concluding that that was where all the philosophical action was at (at this moment in history), this attitude comes naturally to me. Indeed, one of the most memorable parts of my undergraduate education was when, after the guest lecture given by Gregory Chaitin to my graduate information theory class, a bitter argument broke out, with voices raised, between Chaitin and an eminent philosopher of math in the audience, regarding the history of mathematical logic. (Chaitin had a slogan, as I recall, summing up twentieth century technological history as the “glorious failure” of Hilbert’s Program.) All of which is to say, I try to give my students, even in introductory and core CS classes, some sense of

¹Not just by the easy non-explanation that exponents have a bigger effect, of course, but by talking about what we mean by processor speed.
CS as not merely a vocational program with better pay than toaster science but as a fundamental academic discipline, bound up with the deepest philosophical questions and involving real intellectual stakes. I’ve been heartened to find that students are receptive to this, sometimes surprisingly so. When teaching Intro to Computing to CCNY non-majors, for example, I managed to get the students thinking and talking about the Turing Test, Chinese Rooms, and even diagonalization—as well as learning a bit of JavaScript. When pinged awhile back about potentially doing the class again, I was flattered (or “humbled”, as people say) to be told, “[The department chair] said you had done the best job anyone can remember with the CS class for non-majors.” I had to reply, however, that “unfortunately” I wasn’t available, because I’d graduated and was no longer in New York.

As to the particular courses I anticipate focusing on in the future: besides core courses in the major (e.g., discrete math, data structures, computability, programming) and introductory courses for non-majors—both of which I’m genuinely enthusiastic about—the classes I can most clearly envision myself teaching are (naturally) those in and around the areas of my research: advanced undergrad and graduate courses in algorithms, networking, theory, game theory. More speculatively, I can imagine developing new courses on topics such as applied algorithms (proposed course title: “Applied Theoretical Computer Science”), mechanism design and security, and (in a less technical mode) the philosophical implications (and antecedents) of computer science. I’m quite open to teaching other sorts of courses, however. One of the exercises in the CS Education course I took as an MS-TA was “pop-teaching”—preparing and giving a short lecture on a surprise topic (splay trees, say, or trebuchets in medieval warfare) revealed to me late the night before. This training has proved useful: by the second or third time teaching databases, I’d become reasonably competent at SQL.

And now for something completely different. As I detail elsewhere, my work mostly concerns the application of algorithmic analysis to various real-world optimization problems which are often otherwise dealt with heuristically by practitioners. Motivated by various past collaborative experiences of this kind, an educational project I plan to pursue is the design and implementation of a system to help practitioners and non-algorithmic researchers learn what may already be known about their own problems. When studying (what seems to be) a new problem, one often begins by searching the literature for it, but the same abstract problem can appear, in disguise, in other settings. Optimization problems are typically defined formally by some kind of integer program (IP), written in mathematical symbols in \LaTeX documents, which are not easily web-searchable. Although there has for many years been the Complexity Zoo\(^2\) collecting complexity classes, no comparable resource exists for the individual (notable) problems themselves. What I would like to see is a database users could query by entering a (schematized) IP formulation, which would then be normalized and used as a hashcode for the problem. If found, references the problem’s prior work would be returned, and links to generalizations and special cases. The results would be a) more people gaining access to what’s already known about the problems they wish to solve, and b) fewer new papers reinventing old problems. (Of course, that might not sound like a good trade to everyone.)

Beyond resuming classroom teaching, I’m eager both to advise undergraduates and to take on graduate students. Within the large, collaborative projects I’ve worked on as a PhD student and postdoc (primarily the ITA and the NS-CTA), I’ve had the opportunity to informally mentor young graduate students just starting out in research, including those from both applied and theoretical backgrounds. In many cases, these collaborations have led to papers at top conferences. I expect that the long-term advising of students of my own will be deeply rewarding. I became a node in the Mathematical Genealogy tree relatively recently; I imagine that the accomplishment of becoming a non-leaf node will be comparably satisfying.

---

\(^2\)https://complexityzoo.uwaterloo.ca/Complexity_Zoo

mpjohnson@gmail.com  
nesl.ee.ucla.edu/fx/matt/