Teaching Statement

Wen Zhang

Before college, most of my computer science knowledge was self-taught. The process was both rewarding and challenging: I felt a strong sense of accomplishment every time I solved a problem on my own, but I struggled to make consistent progress without proper guidance. When I teach, I aim to create an environment that encourages students to discover knowledge on their own, while providing the support they need to reach their goals—the kind of guidance that I wish I had received as a newcomer to computer science.

Teaching experience

During my undergraduate and graduate studies, I have served as a teaching assistant for several courses at different levels and with different learning goals.

As an undergraduate at Stanford, I worked as a Section Leader for five consecutive quarters in the introductory CS courses: Programming Methodology (CS 106A) and Programming Abstractions (CS 106B/X). In this role, I led weekly sections, held office hours, and conducted "interactive grading sessions", where I provided one-on-one feedback to students on their programming assignments. This experience taught me to identify misunderstandings by posing questions that help reveal a student's thought process, and to adjust my explanations based on students' responses. From my teaching feedback, I was happy to learn that I had had a positive impact on the students' learning:

> Wen is by far the best SL I've had in a CS class by far. He's immensely helpful, responsive, and has helped me understand the material far better [...] I can learn from mistakes and feel more comfortable with the material *(CS 106B, Fall 2014)*

and that my enthusiasm had made section a more enjoyable experience:

> He's super excited about CS which makes everything fun *(CS 106X, Fall 2015)*

Later, I served as a Course Assistant for Stanford's undergraduate compilers course (CS 143). I held office hours, graded programming assignments, and crafted exam questions. I learned to how to create exam problems that required students to integrate concepts from multiple course topics. For example, I designed a midterm question on syntax-directed translation over regular expressions, which one student noted was challenging yet enjoyable:

> This is way too meta. Haha jk it's great

As a graduate student at Berkeley, I served as a Graduate Student Instructor for the undergraduate networking course (CS 168). I took on the task of revamping the programming assignment on distance-vector routing. With my advisor's help, I broke the assignment into nine structured stages with clearly defined goals to guide students through the entire implementation. I also set up infrastructure for auto-grading and held office hours. Through this experience, I learned to design assignments that both reinforced conceptual understanding and provided hands-on experience.

Teaching methodology

Through my teaching experience, I've come to realize that students learn best when they are encouraged to actively *invent* solutions. This process allows students to generalize their solutions to new contexts and, most importantly, builds confidence in tackling unfamiliar problems. These benefits go beyond what rote memorization can offer.

To encourage invention, I strive to solve problems *together with* my students. In the classroom, I would present a programming problem and begin guiding the process through questions: "What strategy do we want to use?" "Where do we start?" "That's a great start—what's the next step?" "Are we done? Have we considered all the edge cases?" All the while, I transcribe the students' suggestions onto the whiteboard, serving as a facilitator who uses the students' own ideas to solve the problem. I would make sure we stay on track, and if everything goes right, we would arrive at a complete solution and declare victory.[1]

As I practiced this teaching method with students in my introductory CS sections, I found that its effectiveness depended on a few important factors:

[1] Incidentally, I learned at a talk on math education last year [1] that this teaching method was also effective in teaching mathematics.

Ensuring broad participation. Collaborative problem-solving only works when everyone feels encouraged and comfortable to participate. To foster this, I take a few simple yet effective steps: inviting quieter students to share their thoughts (I was a quiet student once, so I empathize); making a conscious effort to not interrupt students when they speak (people tend to prematurely cut women off [2]); and making frequent, intentional pauses to give students time to ask questions.

Following the students' lead. In student-driven problem solving, the trajectory can be unpredictable, often deviating from the reference solution. When this happens, I do my best to follow the students' train of thought, adapting to their approach as much as possible. This requires extra preparation, so that I can anticipate unconventional ideas that may come up; and a commitment to respecting each student's contribution and guiding potentially misguided ideas towards viable solutions.

Tying up loose ends. After we solve a problem, I emphasize key points that may not have come up during the exploration—e.g., space and time complexity of the algorithm we have just come up with, and interesting alternative solutions. Doing this ensures that all the teaching goals are covered.

When I become a faculty member, I will likely be lecturing to much larger audiences, which could make collaborative problem-solving more challenging. But it will not be impossible. For example, I have witnessed my advisor, in his populous undergraduate-networking lecture, orchestrate a collective effort to invent a routing algorithm. He did so by selecting students at random to act out each step, then asking another student summarize the findings. I am committed to exploring new ways to keep students engaged, both in small groups and in larger, lecture-style settings.

Teaching interest

I am excited to teach courses in operating systems, distributed computing, and programming languages, and I plan to develop a graduate seminar on database security and privacy. Additionally, I am prepared to teach undergraduate materials in compilers, networking, security, and data structures and algorithms, based on the department's needs.

Mentoring

I have had the pleasure of working with several undergraduate students in research. Shaped by my own positive experiences with undergraduate research, my goals in mentoring students have been (1) to show them that research can be fun, and (2) to build their confidence in problem solving.

To achieve these goals, once a student gets familiarized with the research topic, I strive to identify a nontrivial piece of a research problem for them to tackle. Ideally, this would be a self-contained challenge that, while feasible to solve, requires some conceptual or algorithmic thinking, not merely "grunt work". I would ask the student to come up with a strategy, and then to teach me the strategy along with concrete plans for addressing any key difficulties. After we iterate on the plan, the student would implement their solution in code, which I would review and provide feedback on. Ultimately, their contribution would become a part of the overall system that they "own".

So far, this approach has worked out well. Preliminary versions of Blockaid's compliance caching [3, § 6] (which involved a recursive-backtracking algorithm) and Ote's SQL generation [4, § 5.3] (which involved some tricky relational-algebra transformations) were both implemented by undergraduate researchers. One of these students has since joined a top CS PhD program; the other has taken a relevant job in industry.

I have found that giving students problems that are "challenging yet solvable" keeps them intellectually engaged. And through our problem-solving discussions, students learn that research is not a solitary endeavor and that collaboration is key to making research rewarding and fun.

References

[1] P.-S. Loh. Building human intelligence at scale, to save the next generation from ChatGPT. https://www.youtube.com/watch?v=oO-akX66i24, 2023. [Online; accessed November 2, 2024].

[2] L. Shore. Gal interrupted, why men interrupt women and how to avert this in the workplace. https://www.forbes.com/sites/womensmedia/2017/01/03/gal-interrupted-why-men-interrupt-women-and-how-to-avert-this-in-the-workplace/, 2017. [Online; accessed November 2, 2024].

[3] W. Zhang, E. Sheng, M. A. Chang, A. Panda, M. Sagiv, and S. Shenker. Blockaid: Data access policy enforcement for web applications. In *Operating Systems Design and Implementation, OSDI 2022*. USENIX Association, 2022.

[4] W. Zhang, D. Bali, J. Kerney, A. Panda, and S. Shenker. Extracting database access-control policies from web applications. *CoRR*, abs/2411.11380, 2024. URL https://arxiv.org/abs/2411.11380.