PrairieLearn for Theoretical Computer Science

SIIP Startup Proposal — April 29, 2022

Jeff Erickson, Carl Evans, Yael Gertner, Brad Soloman, Tiffani Williams
Department of Computer Science

Abstract

We propose to develop resources on the PrairieLearn platform to support the teaching of algorithms, data structures, and other theoretical aspects of computer science, at several different levels of the computer science curriculum. The proposed project extends an existing effort to develop PrairieLearn resources for CS 374 and expands this effort to include CS 225 in the first year, and to include CS 277, CS 401, and CS 403 in future years.

Motivation and Timing

Pedagogy / Philosophy

Traditionally, algorithms classes are taught by showing students several classical algorithms—usually designed using a common technique—proving those algorithms correct, analyzing their running times, and then asking students to design and analyze new algorithms using similar techniques. Unfortunately, this approach is inconsistent with the learning goals of these classes, because it does not expose the *process* of designing new algorithms. Students in these traditional classes see only a polished *fait accompli*, with no indication of where the algorithm came from; they are expected to develop a skill—algorithm design—*which they are never actually taught*.

A common complaint of students in CS 374 is a lack of worked examples to study from, especially before exams. In fact, in a typical semester, we provide complete solutions and grading rubrics for over 100 problems from labs, homeworks, and previous exams, including at least one solved problem on every homework handout. (We also provide several dozen study problems for each exam, but without solutions.) Nevertheless, the students' complaints have merit, because *solutions are only the finished product*.

Almost all coursework in CS 374 consists of open-ended algorithm design and proof questions, most of which require about half a page (on exams) to a page (in homeworks) of semi-structured English to answer. Typical examples include the following:

- Describe a regular expression for the set of all binary strings with an even number of 0s and an odd number of 1s.
- Prove that the language $\{0^n 1^{2n} \mid n \geq 0\}$ is not regular.
- Describe and analyze an algorithm to determine whether any number appears more than $n/4$ times in a given array of n numbers.
- Describe and analyze an algorithm to find the longest common subsequence of three given strings.
- Describe an algorithm to find the shortest walk in a given graph G with colored edges, from vertex s to vertex t, in which no three consecutive edges have the same color.
- Prove that the following problem is NP-hard. [...]

Most problems have multiple correct solutions; in some cases, using significantly different techniques. The freeform nature of these questions is a significant strength of the course, but it does come at a cost. Despite promising work using natural language processing to grade simpler narrative questions,[1] automatically grading narrative work is impossible; almost all work in CS 374 must be graded by human beings—in practice, graduate TAs and undergraduate CAs. Ensuring that TAs and CAs provide timely and consistent feedback is one of the most significant challenges of teaching 374, and this challenge has grown as enrollments have increased.

While we do not want to give up on the freeform narrative problems that are the heart of the course, we believe that students can significantly benefit from scaffolding activities that focus on components of the problem-solving process, that provide targeted feedback, and that can help students gain confidence in their own problem-solving abilities. To that end, for the last two semesters Jeff has been managing a team of undergraduates to develop PrairieLearn resources to support CS 374. **We propose to continue this long-term PrairieLearn development effort, and to expand it to include other computer science classes that teach theoretical computer science topics, specifically CS 225, CS 277, CS 401, and CS 403.**

A significant goal of the proposed project is to prioritize service to the students over ease of implementation. Whenever possible we avoid simple multiple-choice questions, which (even with versioning and randomization) invite hill-climbing. On the other hand, it is relatively straightforward to deploy open-ended *implementation* exercises on PrairieLearn with external graders, similar to machine problems in other CS classes. But "machine problems" are a bad match for CS 374; turning well-designed algorithms into practically efficient code is an important skill, *but that skill is not the focus of this class*. We want students to focus on the structure, correctness, and efficiency of algorithms without worrying about (more strongly, while staying deliberately agnostic about) low-level implementation details. It's also difficult to auto-grade code on any metric other than correctness, but to support learning the design process, it is crucial to recognize and reward *progress toward* a correct solution.

History

Over the past several years, Jeff has been trying to shift his teaching in CS 374 and other courses to emphasize and reinforce the *process* of designing algorithms. The goal is similar to the "derivation-first" approach proposed by Libeskind-Hadas,[2] but not only for classical algorithms. As part of this long-term effort, since August 2021 Jeff has managed and funded a team of students to develop PrairieLearn resources that guide CS 374 students through the design/solution process for many different types of problems. The 374 PrairieLearn development effort was originally spearheaded by undergraduate Jason Xia in Spring 2021, with the encouragement of instructors Chandra Chekuri and Patrick Lin; Jason has continued to play a significant leadership, management, and creative role on the team.[3] The other team members are Eric Jin, Julie Lee, Patrick Lin, Steven Pan, Sam Roggerio, Eliot Robson, Tomoko Sakurayama, and Andrew Yin. Jeff was one of the instructors for 374 in Fall 2021, but not in Spring 2022; teaching assistant Eliot Robson served as a liaison between the 374 instructors (Timothy Chan and Ruta Mehta) and the PL development team.

Independently, Yael Gertner has been developing PrairieLearn resources for the theory courses CS 401 and 403 that are part of the department's iCAN certificate program. The iCAN program and its component courses are designed with the goal of broadening participation in CS and are aimed at college graduates who wish to enter the computing field but have non-computing backgrounds; the needs and goals of iCAN courses are different

---

[1] Max Fowler, Binglin Chen, Sushmita Azad, Matthew West, and Craig Zilles. Autograding "Explain in Plain English" questions using NLP. *Proc. 52nd SIGCSE*, 1163–1169, 2021.

[2] Ran Libeskind-Hadas. A derivation-first approach to teaching algorithms. *Proc. 44th SIGCSE* 573–578, 2013.

[3] Unfortunately for us, Jason is graduating in May 2022 and joining Duolingo.

from classes taken by our undergraduate majors. Immediate feedback and extra practice with solutions are especially important for these courses.

Timing

Two recent developments make this project especially timely.

The first development is a long-planned revision of CS 225. Historically, CS 225 included a significant amount of theory content—in particular, induction proofs, running-time recurrences, and algorithm analysis—reinforcing material taught in the prerequisite discrete math course CS 173. Over roughly the last decade, as enrollment in 225 has grown from 800 students per year to over 1500, the class eliminated all manually graded assessments in favor of auto-grading, and reinforcement of theoretical content all but disappeared. CS 225 also traditionally includes several weeks of instruction in C++, to help computer science majors transition from the Java (and more recently, Kotlin) used in the prerequisite programming class CS 125.

In Fall 2021, the programming intro sequence was changed from having two paths one for CS majors, CS 125 followed by CS 126, and one for everyone else which only included CS 128 into a single path. This new path is composed of CS 124 followed by CS 128. This path moved some of the material from CS 125 into the later course and combined it with as much of the design material from CS 126 as could be taught at scale and an introduction to C++. (ECE students are expected to be introduced to C++ in ECE 120 and ECE 220 before taking CS 225.) This shift will allow CS 225 to remove about three weeks of instruction in C++ from its syllabus, leaving room to reintroduce more theoretical content.

The second development is a significant increase in the number of incoming computer science students at two different levels of the curriculum, caused in part by a deliberate increase in the number of admission offers and in part by unexpectedly high yields, perhaps as a side-effect of the COVID-19 pandemic. These increases will make it significantly more difficult to adequately staff our courses with qualified TAs and CAs..

The Fall 2021 freshman class included 653 computer science majors (including 374 in Engineering), a *46% increase* over the Fall 2020 freshman class of 446 (including 261 in Engineering). Enrollment in our core undergraduate classes is expected to grow significantly as this wave of new students reaches them. Enrollment in CS 124, our introductory programming class has already grown from Enrollment in CS 225 is expected to grow from 1000 in Fall 2021 to 1250 in Fall 2022.

The computer science department is also dramatically increasing the number of students in our on-campus Master of Computer Science program. The number of new MCS students has grown from 71 in Fall 2020, to 164 in Fall 2021, to 494 in Fall 2022, a *seven-fold increase* in just two years. While this expansion will not affect enrollment in our core undergraduate classes, 400- and 500-level CS courses will require significantly larger staff, leaving fewer highly-qualified graduate TAs and undergraduate CAs for lower level courses.

Examples of Existing Resources

In this section, we describe a few examples of the PrairieLearn resources that the existing team has already deployed in CS 374 and CS 401.

Most of the existing CS 374 PrairieLearn resources are organized into “Guided Problem Sets”. Each guided problem set consists of a small number of multi-stage exercises, implemented as a sequence of questions that guide students through the process of solving a design problem. Each of the resources described below supports a single stage in one of these guided problems sets.

Proof Blocks

One of the most natural PrairieLearn tools for theory classes is the Proof Blocks (`pl-order-block`) element developed by Seth Poulson and others at Illinois.[4,5,6] Most directly, we use Proof Blocks in CS 401, CS403 and CS 374 in scaffolding exercises to help students learn to write proofs, and to ask students to sort functions by their "big-Oh" growth rates. We expect to incorporate the "edit distance" partial credit scheme recently implemented by Poulsen et al.,[7] but we also plan to work with PL developers to develop better visual feedback and to better support multiple correct solutions.

We have also experimented with using Proof Blocks to build scaffolding activities for writing *pseudocode*, specifically for NP-hardness reductions, similar in spirit to Parsons problems for code.[8,9] The autograder translates the pseudocode into executable Python code for evaluation, and reports either that the reduction is correct or provides an explicit counterexample. The autograder also uses the usual Proof Blocks ordering constraints to identify any variables in the pseudocode that are used before they are defined. (See the figures on the next page.) We expect to further refine and make significantly greater use of these pseudo-Parsons problems[10] in the future. The figures on the next page show features of an exercise asking for a reduction from the Hamiltonian Cycle problem to the Hamiltonian Path problem.

---

[4] Seth Poulsen, Mahesh Viswanathan, Geoffrey L. Herman, and Matthew West. Proof blocks: Autogradable scaffolding activities for learning to write proofs. Preprint, August 2021, arXiv:2106.11032.

[5] Seth Poulsen, Mahesh Viswanathan, Geoffrey L. Herman, and Matthew West. Evaluating proof blocks as exam questions. *Proc. ICER 2021*, 157–168, 2021. Reprinted in ACM Inroads 13(1): 41–51, 2022.

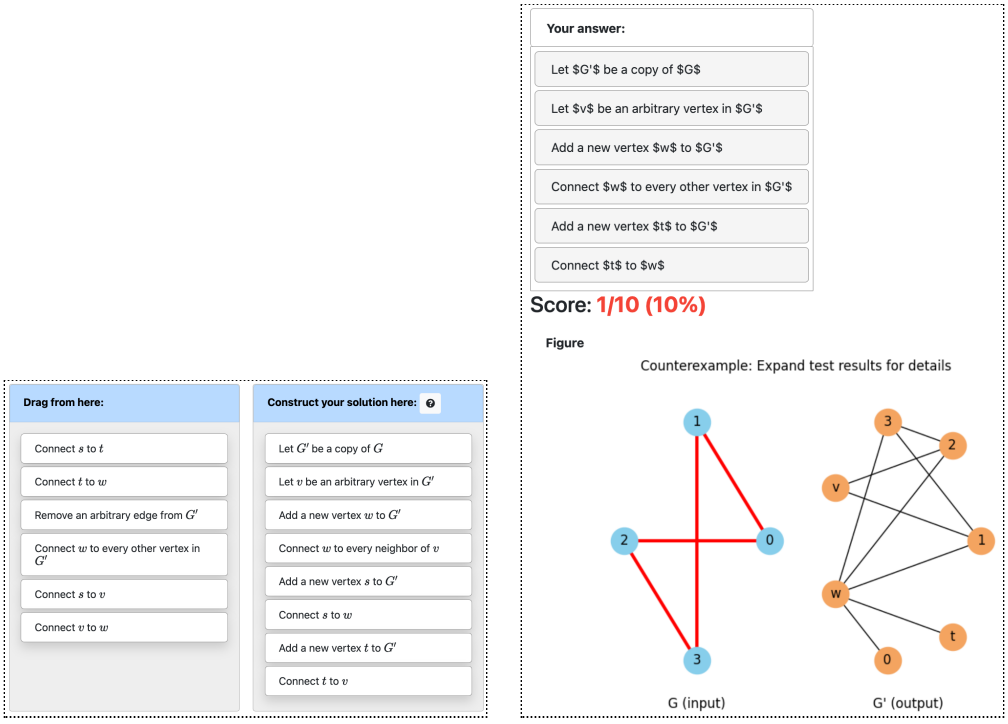
[6] See https://www.proofblocks.org/.

[7] Seth Poulsen, Shubhang Kulkarni, Geoffrey Herman, and Matthew West. Efficient partial credit grading of proof blocks problems. Preprint, April 2022, arXiv:2204.04196.

[8] Dale Parsons and Patricia Haden. Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proc. ACE '06*, 157–163, 2006.

[9] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. A review of research on Parsons problems. Proc. ACE '20, 195–202, 2020.

[10] "Parsons pseudo-problems"?



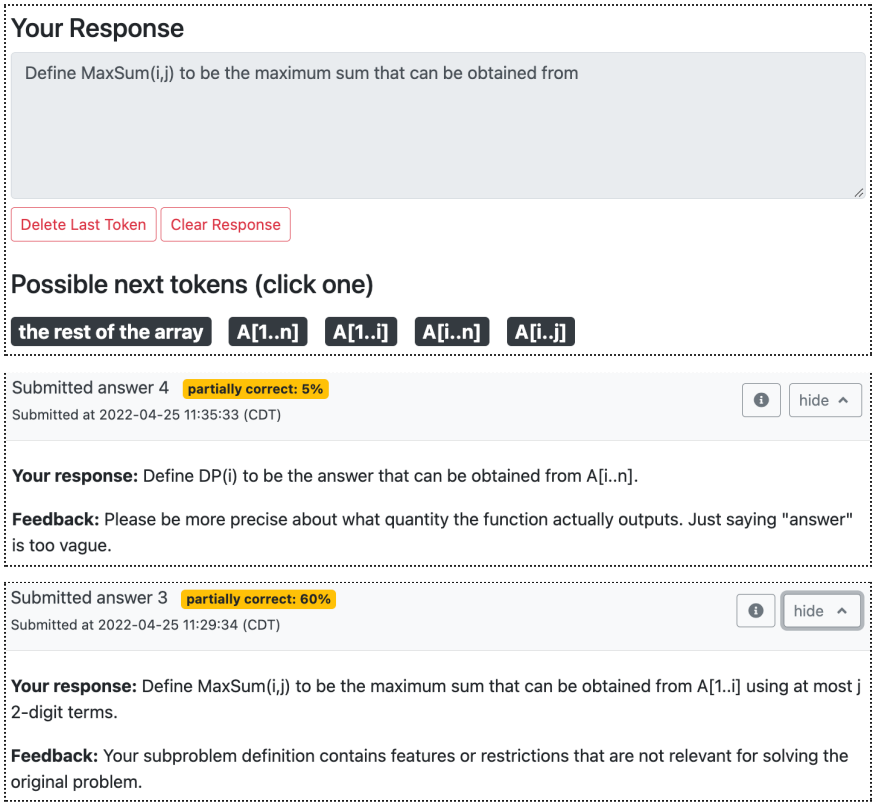
Left: A correct reduction from Hamiltonian Cycle to Hamiltonian Path.
Right: Feedback on an incorrect (but compilable) reduction

Scaffolded Writing

We[11] have also developed a "Scaffolded Writing" element that allows students to generate sentences from a hidden context-free grammar, one token at a time. The interface closely resembles the auto-complete feature of several mobile messaging apps; as the student enters their sentence, they are presented with a list of all possible next tokens. To design a problem for this element, the instructor specifies a grammar that can generate both correct and incorrect answers—ideally *multiple* correct answers and incorrect answers *that cover common student mistakes*—as well as grading code to provide feedback and partial credit that reflects progress toward a correct solution.

We first deployed this tool to support the design of dynamic programming algorithms. An important step in the design process is specifying the underlying recursive function that the algorithm will evaluate. (Although this step is not strictly necessary, in Jeff's experience students who skip this step are significantly more likely to submit incorrect algorithms.) The underlying grammars generate multiple correct solutions (for example, specifying subproblems by either prefixes or suffixes of the input array), specifications that lead to correct but slower algorithms, several common errors (for example, not describing explicitly how the output value depends on input parameters), and a few stylistic errors (for example, naming the recursive function "DP"). Feedback and partial credit follow the same grading rubric used for dynamic programming problems on written homeworks and exams. The following figures show features of an exercise involving the following algorithm design problem. Given an array A[1..n] of decimal digits, break the array into 1-digit and 2-digit terms whose sum is as large as possible. For example, when the input array is [1,9,2,5], the correct answer is 1+92+5 = 98.

[11] mostly Jason Xia



Input interface and examples of partial-credit feedback for scaffolded writing

As part of his senior thesis, Jason Xia performed a user study involving this semester's cohort of CS 374 students; we are hopeful that the results of Jason's study will lead to a publication in SIGCSE or a similar venue. More recently, we have used the same scaffolded writing tool for graph transformation problems, which arise both in the design of efficient graph algorithms and in NP-hardness proofs.

Dynamic Programming Feedback

Another important step at the other end of the dynamic programming pipeline is to find a suitable evaluation order to fill the memoization table. In the last part of our guided problem sets on dynamic programming, we ask students to implement the dynamic programming algorithm in Python. (These algorithms usually consist of a few nested for loops around some casework, so implementation is relatively straightforward.) The team developed a "DP Feedback Array" wrapper class for memoization tables that tracks access patterns to help diagnose coding mistakes. The student must explicitly declare the memoization array as a `MemoArray` object; otherwise, they write standard Python code. The `MemoArray` class overloads the `operator.getitem` and `operator.setitem` functions to track reads and writes, and then reports errors like the following when they occur:

- "On input [3, 1, 4, 5, 9, 2], your code attempted to access `MaxProfit[3,7]`, which is outside the declared bounds of your memoization array."
- "On input [3, 1, 4, 5, 9, 2], your code attempted to read `MaxProfit[2,2]` before it was initialized, most likely during the evaluation of `MaxProfit[1,2]`."
- "On input [3, 1, 4, 5, 9, 2], your code failed to evaluate `MaxProfit[3,4]`."

- "On input [3, 1, 4, 5, 9, 2], your code correctly filled the memoization array `MaxProfit`, but then returned a final answer of 6 instead of the correct answer 4."

A similar error reporter has been independently developed for C++ in CS 225, which provides near-identical feedback on the dimensions and content of the final memoization array.

Targets for 2022–23

The development team will continue to design, implement, refine, and debug PrairieLearn resources for CS 374, as we have for the last two semesters. As we have this semester, we will keep in close contact with the instructors for each semester (Sariel Har-Peled in Fall 2022, Chandra Chekuri in Spring 2023) to make sure that we meet their needs. Sariel and Chandra have already agreed to incorporate PrairieLearn into their offerings of CS 374; as always, they will have the final decisions about which resources to deploy in their classes.

We have several ideas for new resources and infrastructure that we would like to develop for CS 374; many of these will require close collaboration with the core PrairieLearn development team. One simple example is to improve partial credit mechanisms for order-blocks questions—not only adopting "edit-distance" partial credit, but providing clear visual feedback to identify correct portions of students' submissions. (For example, we could indicate blocks that belong to the closest correct solution by shading them green, and/or blocks that should be removed or replaced by shading them dark gray.) We would also like to build an interactive tool for creating and visualizing finite-state automata, by porting Evan Wallace's Finite State Machine Designer[12] into PrairieLearn. Over time, we hope to adapt this tool to visualizing and manipulating similar structures, such as graph algorithms and balanced binary search trees. Finally, we would like to adapt some of our multi-stage problems to support multiple correct solution paths. Currently, because PrairieLearn does not support communication between questions, we give students credit for any correct solution to the first stage of a mutli-stage problem, but then in the second stage impose our own solution; instead, we would like to allow students to stick to their initial choices through the entire problem. We also want to build out several new variants of existing guided problem sets, so that instructors can choose different problems to be submitted for credit, leaving lots of similar examples for student practice, especially before exams.

We also plan to develop new resources for CS 225, at least initially building on top of code infrastructure developed for 374. We cannot expect to deploy most resources developed for CS 374 directly in CS 225; the two courses teach different material to different audiences and with different learning goals. Whereas 374 uses PrairieLearn almost exclusively as a scaffolding or formative assessment tool, CS 225 requires summative assessment tools, because manually grading theoretical work for a class of 1200 students is impractical. We plan to start meeting over the summer to identify appropriate infrastructure to adopt from the existing 374 code base, and to design new assessment resources to be developed over the coming year.

Finally, the team will meet regularly to plan out appropriate PrairieLearn resources for 277, 401, and 403, but we do not expect to deploy significant new resources in those classes during the 2022–23 academic year. The team feels that we need more time to understand how to adapt the materials to the needs of students in these classes.

In the longer term, we are hopeful that this project will foster opportunities for computer science education research, similar to Jason Xia's current scaffolded writing study. Almost all research in CS education is aimed at teaching beginners; remarkably little has been published about teaching more advanced theoretical topics.

[12] Demo available at https://madebyevan.com/fsm/; source available at https://github.com/evanw/fsm.

Indeed, one of the main reasons we are requesting SIIP support is to connect with mentors who have more experience in education research.

Organization and Budget

Mirroring the existing team structure for CS 374 PL development, the team will include 3–5 undergraduate programmers, along with a liaison from the CS 374 TA pool to maintain communication between the team and the current CS 374 instructors. (Jeff is not scheduled to teach CS 374 again until Fall 2023. Next semester's instructor Sariel Har-Peled has already agreed to reassign Eliot Robson to be the liaison TA next semester.) Carl and Brad are teaching CS 225 in both Fall 2022 and Spring 2023, so we do not anticipate needing a second liaison TA.

We are requesting a budget of **$9720**. This is the amount required to support **three** undergraduates working **10 hours/week** for **one 18-week semester** (August 15 through December 23), at the CS-department's standard rate of **$18/hour** for senior undergraduate researchers. In addition to these funds, the computer science department has already committed to funding several undergraduate developers for CS 225, and Jeff will at least match the SIIP budget from his Abassi Professor endowment funds. We also anticipate attracting students to participate for individual-study or senior-thesis credit (as Jeff did for three students in Fall 2021). We will advertise and collect applications through the Computer Science Department's undergraduate-hiring portal.[13] Past experience suggests that we will be able to attract a large pool of qualified applicants.

It will take some time to converge on the most effective schedule of regular meetings, but we will start by following the CS 374 team's schedule. The entire team will meet once a week to report progress, demonstrate new resources, provide high-level feedback, and discuss design issues and long-term strategy. Subteams assigned to each course will meet at least once per week to provide more detailed feedback. The entire team will also communicate asynchronously over Slack, and we will track and discuss pull requests, bug reports, and other issues on Github.

About the Target Classes

Immediate

CS 374 ("Introduction to Algorithms and Models of Computation") is a junior-level theoretical computer science course, which covers a combination of algorithm design and analysis, automata and formal language theory, and complexity theory. Coursework consists almost entirely of open-ended design and analysis problems. CS 374 has a steady-state enrollment of about 550 students per semester, almost all computer science and computer engineering majors, for whom the course is required. Thanks to increased CS undergraduate admissions, the theory group is planning for 750 students in Fall 2023.

CS 225 ("Data Structures") is a sophomore-level course covering elementary data structures and algorithms and their implementations. This course is required for all computer science and computer engineering majors, computer science minor, and transfer applications into computer science and computer engineering. CS 225 typically enrolls about 1000 students every fall and about 500 students every spring; however, thanks to increased CS undergraduate admissions, enrollment in Fall 2022 is expected to be about 1250. CS 225 is one of two prerequisites for CS 374; the other is CS 173 (Discrete Math).

[13] https://course-assistants.cs.illinois.edu/research

Future

CS 277 ("Algorithms and Data Structures for Data Science") is an introduction to elementary concepts in algorithms and classical data structures, with a focus on their data science applications. This is a new course designed for the new "Data Science + X" degree programs; so far there has been only one pilot offering, to less than 10 students. The first Data Science + X students will matriculate in Fall 2023. Based on existing enrollments in the prerequisite class Stat 207, we expect steady-state enrollment in CS 277 to grow to about 100 students per offering by 2024.

CS 401 ("Accelerated Fundamentals of Algorithms I") and **CS 403** ("Accelerated Fundamentals of Algorithms II") are part of our new ICAN (Illinois Computing Accelerator for Non-Specialists) graduate certificate program, which is aimed at students with bachelor's degrees in fields other than computer science. So far each of these classes has been taught twice, each to about 10 students; steady state enrollment is expected to grow to about 40 students per year in each class.

About the Project Team

All five team members are faculty members in the Department of Computer Science.

Jeff Erickson is a full (tenure-track) professor. He is one of the architects and a regular instructor of both CS 374 and the followup algorithms course CS 473. Jeff served as an AE3 Engineering Innovation Fellow from 2017 to 2021; however, he has not previously been a member of any SIIP team. He does research in algorithms.

Carl Evans is a teaching assistant professor and one of the regular instructors of CS 225. Carl has also taught CS 126 (software design studio), CS 173 (discrete structures), and CS 241 (System programming; he also served as a TA for several CS courses as a PhD student at Illinois. Carl has also participated in coordinating the development of CS 128 which was supported by a SIIP grant.

Yael Gertner is a teaching assistant professor. She developed and regularly teaches several courses in the iCAN program, including CS 401 and 403, and she has been actively developing PrairieLearn resources to support these classes. She has also taught CS 173 (discrete structures). Her research interests are in computer science education in the areas of broadening participation in computing and designing interventions to increase students' learning outcomes. Her prior SIIP project was titled Designing Early Interventions to Facilitate Student Study Skills in Introductory Problem-Solving Classes.

Brad Solomon is a teaching assistant professor and one of the regular instructors for CS 225. Brad is the course director and regular instructor for our new course CS 227 (algorithms and data structures for data science); he has also taught CS 173 (discrete structures). His research focuses on developing new algorithms and data structures for the efficient storage, search, and analysis of genomic sequencing data. Brad has not previously been a member of any SIIP team.

Tiffani Williams is a teaching professor; Director of Onramp Programs in the CS department; and a Dean's Fellow in Inclusion, Belonging, and Engagement in the Grainger College of Engineering. She is one of the architects of the department's new Illinois Computing Accelerator for Non-Specialists (iCAN) certificate program, which aims to broaden participation in computing to college graduates with non-computing backgrounds. Tiffani has previously served on a SIIP team with Geoffrey Challen on developing interactive walkthroughs for introductory CS content.