TheorieLearn:
Autograded Resources for Theoretical Computer Science

SIIP Implementation and Exploration Project, 2nd Year Renewal

Jeff Erickson, Carl Evans, Yael Gertner, Brad Solomon
Department of Computer Science
https://theorielearn.github.io

2023–24 Highlights

- **In the 2023-24 academic year alone, our resources (*not* including contributions to the main PrairieLearn platform) were used by over 4500 Illinois students in CS 173, CS 225, CS 277, both sections of CS/ECE 374, CS 401, and CS 403.**
- **Our resources are also being used in theoretical CS courses at Utah State and UC Irvine.**
- **We published two research papers, and two ongoing research projects are slated for submission to SIGCSE in August.**
- **Preliminary data analysis shows positive correlations between engagement with our PrairieLearn resources and improved exam performance in CS 374.**

Progress in 2023–24

Thanks to generous support from the SIIP program and the Department of Computer Science, we made significant progress on several different fronts during the 2023–24 academic year, including several new guided problem sets, progress on several new question types and interactive elements (mostly aimed at CS 225), project-supported computer science research, and broader publicity and feedback. We have contributed exercises to other classes at Illinois that are not formal participants in our project; elements, bug fixes, and feature requests to the main PrairieLearn codebase; and significant updates to other open-source projects. In the 2023-24 academic year alone, our course-specific resources have been used by more than 4500 students at Illinois, as well as an unknown number of students at two other universities.

Assessment of theoretical content in CS 225

One of our goals for this year was to develop summative assessment exercises for CS 225, the sophomore-level data structures class. Progress on this front has been slower than expected, primarily because of the difficulty of balancing the open-endedness of most theory content with the limited scope of autograding. Nevertheless, our efforts have had an effect on the *pedagogy* of CS 225. In Spring 2023, instructor Carl Evans included at least one *manually-graded* theoretical question in each of the biweekly CS 225 examlets.

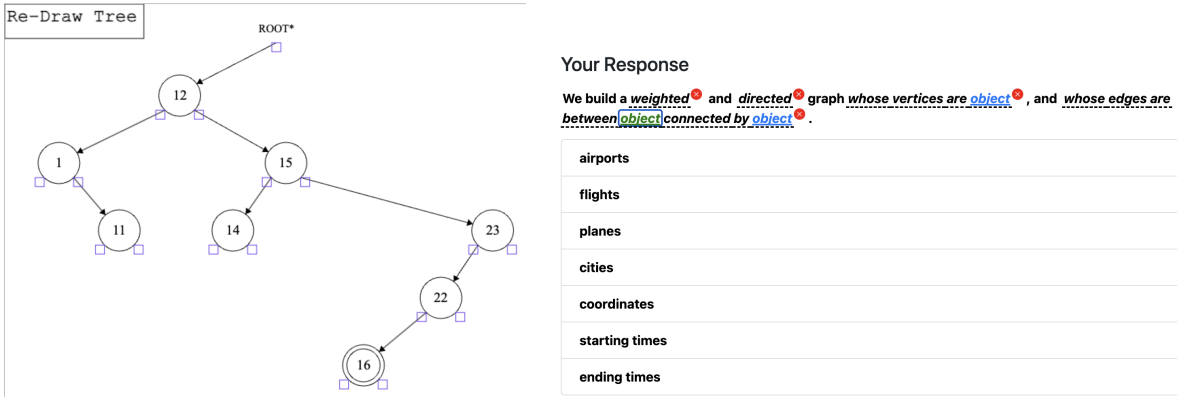
Interactive elements

Two papers describing our finite-state-machine builder element have been accepted for publication. The first paper[1] describes our improvements to the `automata` Python library, which we use for grading and feedback generation; the second paper[2] describes the student and instructor/author interfaces, automatic counterexample generation, partial credit algorithms, and

While we have deployed several new problems for CS 374, most of our development of question types and interactive elements has been aimed at CS 225. For example, we have modified the automata editor into three different **interactive tree-builder elements**; to be used for different types of data structure problems. In all variants, the layout of the tree is updated automatically when users add or delete nodes; we found that allowing students to move nodes around arbitrarily was more confusing than helpful.

- The most basic variant supports only labeling nodes with search values and adding and deleting leaves. These limited operations are sufficient to support exercises about insertions and deletions in binary search trees.
- A more complex variant also supports arbitrary reassignment of left and right child pointers, shown in the figure to the right. This variant is useful for testing understanding of trees that maintain balance through rotations, such as AVL trees and red-black trees.
- Finally, a third variant supports multiple search keys at each node, and splitting/merging nodes, which are crucial for modeling B-trees.

We plan to continue developing this tool to support highlighting nodes and edges (for example, to highlight search paths), performing rotations, and other higher-level operations. We also anticipate using variants of this tool for questions not only about search trees, but also about heaps, Huffman codes, recursion trees, and parse trees.



One of our most novel developments is our scaffolded writing tool, which allows students to construct English sentences from a hidden context-free grammar.[3] Students generate sentences one token at a time, using an interface that closely resembles the auto-complete feature of several mobile messaging apps; as the student

[1] Caleb Evans and Eliot W. Robson. automata: A Python package for simulating and manipulating automata. *J. Open Source Software* 8(90):5759, 2023.

[2] Eliot W. Robson, Samuel Ruggerio, and Jeff Erickson. FSM Builder: A tool for writing autograded finite automata questions. To appear in *Proc. 29th Annual ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE), 2024.

[3] Jason Xia and Craig Zilles. Using context-free grammars to scaffold and automate feedback in precise mathematical writing. *Proc. 54th SIGCSE*, 479–485, 2023.

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 currently use this tool to support guided problems on dynamic programming, graph algorithms, and Np-hardness reductions.

One common complaint about the scaffolded-writing element is that the limited interface can be confusing, especially when students come up with sentences that may be correct but do not match the question's hidden grammar. Another complaint is that fixing mistakes at the beginning of a sentence requires deleting and then manually rechoosing all later tokens. To try to address these issues, we are developing a "**top-down" variant of the scaffolded writing tool**, which allows students to generate sentences "breadth-first" instead of "depth-first"; a snapshot of our new element is shown on the right. Students can expand or contract tokens representing subexpressions in any order. We have deployed a prototype of this element in CS 225 to gather informal student feedback to support further development.

Ongoing research

For the past four semesters, we have surveyed students in CS 374 about their experience working with both the autograded PrairieLearn exercises and the traditional written homeworks. Our ASEE 2023 paper[4] summarized our initial evaluation of Fall 2023 survey results; as we hoped, the survey results revealed that students found the guided problem sets more enjoyable and less stressful than written homeworks, and gave students more confidence in their own mastery of the course material. The Spring 2023 and Fall 2023 results showed nearly identical results; the Spring 2023 survey is ongoing.

Students reported that the guided problem sets helped them understand the process of solving problems, helped them by serving as a link between lecture and the written homework and exams, and helped by serving as an easier onboarding experience to problem solving. We are currently pursuing a **deeper exploratory analysis** of our survey results, together with homework and exam scores and PrairieLearn usage data, to better understand what effect our resources have on student learning. Our analysis has already revealed a few encouraging patterns. For example, students with higher grades reported more that they understood the process of solving problems, and students that engaged more with the guided problem sets (as measured by the number of attempts submitted) had a positive increase from midterm to final in every grade category. We are still investigating whether increased engagement is helpful to struggling students. We are hoping to submit results of our analysis to SIGCSE 2025 in August. RAs Eliot Robson and Hongxuan Chen have been instrumental in this ongoing data analysis.

We are also conducting more basic **theoretical-computer-science education research**, which will inform what types of exercises we should develop in the future. There is remarkably little prior work in this area—historically, almost all CS education research is focused on introductory courses—but there are signs of increasing interest from the CS education community. Specifically, we are investigating barriers that students face when learning to design graph algorithms. RA Hongxuan Chen conducted "think-aloud" interviews with 15 students, where the students solve a small number of graph algorithm design problems; Hongxuan and RA Katherine Braught are currently coding and analyzing those interviews. We hope to submit our results to SIGCSE 2025 in August.

---

[4] Jeff Erickson, Jason Xia, Eliot Wong Robson, Tue Do, Aidan Glickman, Zhuofan Jia, Eric Jin, Jiwon Lee, Patrick Lin, Steven Pan, Samuel Ruggerio, Tomoko Sakurayama, Andrew Yin, Yael Gertner, and Brad Solomon. Auto-graded scaffolding exercises for theoretical computer science. *Proc. ASEE 2023.*

Impact and publicity

During the 2023-24 academic year alone, our resources have been used by over 1100 students in CS 374 (including the ECE-taught section of the course starting in Spring 2024), over 1700 students in CS 225, and almost 1600 students in CS 173.

We maintain a public-face project web site at https://theorielearn.github.io, with pointers to our public practice instance, research papers, SIIP proposals, and (eventually) our main public repository.

We participated in a well-attended "Spiffy PrairieLearn assessments" workshop at SIGCSE 2024, the flagship conference in computer science education. We gave a demo of an exercise where students assemble a pseudocode description of an NP-hardness reduction—transforming one graph into another—using Seth Poulsen's Proof Blocks element.[5,6] If the student's reduction is incorrect, the grading code automatically generates a counterexample graph. The appearance of the counterexample drew audible gasps and applause from the audience.

Resources that we developed for CS 374 are now being used at two other universities: Illinois PhD Seth Poulsen is using them in his algorithms course at Utah State University, and Michael Schindler is using them in his automata and formal language course at UC Irvine.

Goals for 2024–25

We have already deployed guided problem sets that cover all topics in CS 374, but for most of these topics, we only have a few fully developed exercises. We plan to focus on **building more exercises** of the types we already have, both to provide students with additional opportunities for practice ("*working* examples") and to give instructors more choice about which exercises to assign for credit. Before each exam in CS 374, the instructors distribute an "exam fodder" document containing dozens of problems for each topic covered on that exam, of similar scope and difficulty to actual exam questions, *without* solutions. (Indeed, many of the fodder problems are taken directly from old exams; however, instructors generally create new problems for each exam.) Over time, we would like to implement most of these hundreds of fodder problems as guided problem sets.

We will continue to maintain close communication both with the staff for all relevant courses. Carl is teaching CS 173 in Fall 2024 and CS 225 in Spring 2025; Brad is teaching CS 225 in Fall 2024 and CS 277 in Spring 2025. Jeff is next scheduled to teach CS 374 in Fall 2025, but he is in close communication with next year's instructors Sariel Har-Peled (CS), Chandra Chekuri (CS), and Abhishek Umrawal (ECE). In particular, the departure of Ben Cosman, the regular CS 173 instructor for several years, gives us an opportunity to **significantly ramp up development of new elements and exercises for CS 173**. We will also continue working closely with the core PrairieLearn team, both to promote broadly useful components up to the main code base, and to advocate (and serve as guinea pigs) for major feature requests. We will also continue our outreach to instructors at Illinois and elsewhere whose classes might benefit from our resources.

---

[5] 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.

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

One major goal in the service of greater adoption is to **shift our target development platform to a *public* Github repository**, with code explicitly released under an MIT License and question text explicitly released under a Creative Commons Attributions (CC-BY) license. Instructors could choose to either adopt our resources verbatim via PrairieLearn's question-sharing feature, or to copy our resources into their own repositories and modify them for their own purposes. We also plan to develop software tools to help automate copying questions between repositories. In the longer run, we need to move to a development model where "TheorieLearn" provides software infrastructure, but individual courses are responsible for their own assessments.

We plan to **identify and train a replacement for Eliot Robson**, who has been our excellent technical manager for years. Eliot is on track to complete his PhD in the near future, possibly as soon as summer 2025; fortunately for us, he has committed to continuing his management role until he graduates. Eliot is funded by a fellowship next semester; the ideal budget optimistically includes an RAship for his prospective successor.

Finally, we plan to continue to **pursue related computer science education research**. Our current graph-algorithms study focuses on only one step in the design pipeline—correctly modeling the input data as a graph. We are tentatively planning a followup study focusing on another equally important and difficult step, namely determining the right question to ask *about* that graph—Is this a question about reachability, or shortest paths, or cycle detection, or something else? The ideal budget includes an RAship for Hongxuan to continue his work in this direction. Independently from this SIIP project, Jeff and Yael (along with Geoffrey Herman, Seth Poulsen, and Micheal Schinder) plan to join a multi-university NSF proposal, led by Diana Franklin at the University of Chicago, to study theoretical computer science education. The precise scope of that project is still under discussion, but we are optimistic that both projects will be strengthened by our collaboration.

2023–24 Spending

In the 2023-24 academic year, we received **$73,595.00** in SIIP funding, plus a promise of matching funds from the Department of Computer Science. We also still have a balance of **$4,125.80** from the initial SIIP startup grant.

- Undergraduate developers: five in Fall 2023, seven in Spring 2024, and two in Summer 2024
    - We budgeted for 8 undergraduates working 10 hours per week, but we had fewer developers, and most developers worked significantly less than 10 hours per week, especially in the spring..
- Graduate RAs:
    - In the proposed 2022-23 budget, we significantly overestimated the cost of RAs. The actual cost charged to the SIIP account was $2930 per month per RA; our proposed 2023-24 budget uses this estimated rate.
    - **Eliot Robson** (Fall 2023 and Spring 2024) — technical management, CS education research, research data analysis
    - **Hongxuan Chen** (Fall 2023 and Summer 2024) — CS education research, research data analysis
    - **Katherine Braught** (Summer 2024) — research data analysis

Our only other expense this academic year is Eliot Robson's trip to ITiCSE this summer. **All past expenses were drawn from SIIP funding, except for one Fall 2023 RA, which was paid by the CS department.**

The following table outlines our 2023–24 expenses, extrapolated through Summer 2024.

2023–24 SIIP Funding (including startup carryover)	**$77,720**			
Expenses by semester	**Fall**	**Spring***	**Summer***	**Comments**
Graduate Assistants	$11,720	$11,720	$17,580	1 Fall + 1 Spr + 2 Summer (3 months)
Undergrad Hourlies	$5,274	$5,414	$3,000	5 Fall + 7 Spr + 2 Summer (12hr/wk x 13 wk)
Travel			$3,000	One student to ITiCSE
Totals	**$16,994**	**$17,134**	**$23,580**	
Total Expenses	**$57,708**			
Remaining Balance	**$20,012**			

*Table 1. 2023–2024 expenses from SIIP funds (*projected)*

2023–24 Budget Request

We are proposing a smaller budget for 2023–24, in part in anticipation of a smaller overall budget for the SIIP program, in part to reduce the number of undergraduate developers from eight to five. Managing a team of eight developers has proved more difficult than expected; average productivity was significantly lower in the spring than in the fall, perhaps because each individual developer received less attention from Eliot and the faculty. We believe we can accomplish the same goals with a smaller, more focused team. Four of our existing developers have indicated interest in continuing next semester, and we have identified a fifth undergraduate who is highly qualified and eager to join the team.

The Department of Computer Science has agreed to fund up to 11 months of 50% RAship for the 2024–25 academic year, conditioned on departmental support not exceeding SIIP support.

As recommended by our EIF Mariana Silva, we are providing two budget requests for SIIP:

- Our **ideal** budget request for **$50,318** covers two RAships (including one funded by the department), five undergraduate developers, and a small amount of conference travel.
- Our **minimal** budget request for **$20,588** covers one RAship (one semester covered by SIIP and one semester and summer covered by the department), four undergraduate developers, and no travel. Under a minimal budget, we would likely prioritize transitioning to a public repository and limit our recruiting efforts for Eliot's successor to first-year computer science PhD students (who are all funded by departmental fellowships).

Salaries / Wages	**Proposed Budget**	**Comments**
Graduate Assistants	$64,460	2 50% 11-month RAships, at CS 2022-23 post-qual rate ($2930/mo), no overhead or tuition
Undergrad Hourlies	$36,100	5 undergrads × $19/hour (2023–24 CS rate for senior URAs) × 10 hours/week × 38 weeks (8/23/24–5/12/25)
Travel	$2,000	one student to SIGCSE or ASEE
Total Budget	**$102,560**	
CS Dept commitment	$32,230	1 50% 11-month RAship
Carryover balance	$20,012	
Requested SIIP funds	**$50,318**	

*Table 2. **Ideal** 2023–24 budget request*

Salaries / Wages	**Proposed Budget**	**Comments**
Graduate Assistants	$32,230	1 50% 11-month RAship, at CS 2022-23 post-qual rate ($2930/mo), no overhead or tuition
Undergrad Hourlies	$28,880	4 undergrads × $19/hour (2023–24 CS rate for senior URAs) × 10 hours/week × 38 weeks (8/23/24–5/12/25)
Total Budget	**$61,110**	
CS Dept commitment	$20,510	1 50% RAship for spring (4 months) and summer (3 months)
Carryover balance	$20,012	
Request SIIP funds	**$20,588**	

*Table 3. **Minimal** 2023–24 budget request*