Connect the Dots to Prove It

A Novel Way to Learn Proof Construction

Mark McCartin-Lim
University of Massachusetts Amherst
Amherst, Massachusetts
markml@cs.umass.edu

Beverly Woolf
University of Massachusetts Amherst
Amherst, Massachusetts
bev@cs.umass.edu

Andrew McGregor
University of Massachusetts Amherst
Amherst, Massachusetts
mcgregor@cs.umass.edu

ABSTRACT

This paper describes a new method for helping students improve their ability to develop proofs, a skill necessary for comprehending and appreciating the foundational topics of computer science. Our method transforms ordinary pen-and-paper homework problems into a puzzle-like game, where students connect dots to justify assertions, in a quest to reach a desired goal. We have implemented a software tutoring system using this method, for students to use at home as an optional study aid. Potentially, our system could one day become a full replacement for traditional hand-written homework, which has the additional benefit for course instructors of automating the grading of student work. Our system is also easy to adapt to any class that requires students to write proofs, and it is easy for instructors to create new problems to use with this system. This stands in contrast to many other educational tools for teaching proofs, which are limited to specific topic domains. We have demonstrated the versatility of our system by testing it in two computer science classes at a large public university. One was a Sophomore-level discrete mathematics course where the students were learning first-order prepositional logic, and the other was a Junior-level algorithms course where students were being first exposed to the concept of NP-completeness. Students from our experiments reported that they would like our system to be used in more of their classes.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; *Student assessment*; • **Theory of computation** → *Logic*; • **Applied computing → Computer-assisted instruction**; *Interactive learning environments*; E-learning; • **Mathematics of computing** → *Discrete mathematics*;

KEYWORDS

proofs, intelligent tutoring systems, automated grading, theoretical computer science, discrete math, logic, complexity theory, algorithms, active learning, e-learning

SIGCSE '18, February 21–24, 2018, Baltimore, MD, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159609

ACM Reference format:
Mark McCartin-Lim, Beverly Woolf, and Andrew McGregor. 2018. Connect the Dots to Prove It. In *Proceedings of The 49th ACM Technical Symposium on Computing Science Education, Baltimore, MD, USA, February 21–24, 2018 (SIGCSE '18),* 6 pages.
https://doi.org/10.1145/3159450.3159609

1 INTRODUCTION

In 2014, when Maryam Mirzakhani became the first female mathematician to win the prestigious Fields Medal, she explained how she first developed a passion for mathematical proofs [4]:

> *"It is fun — it's like solving a puzzle or connecting the dots in a detective case."*

Our goal is to inspire all students with that same passion, and we aim to accomplish that goal by introducing a puzzle-based platform for proof construction, where the pedagogical method is to have students construct proofs by literally *connecting the dots*.

As computer science researchers, we specifically developed this platform as part of a tutoring system for helping computer science students in their theoretical courses [12].

Computer science educators have lamented the difficulty of teaching students how to write rigorous proofs in discrete mathematics courses [8, 10]. From our own experience grading student papers, we have found that many students also struggle to write rigorous proofs in more advanced theoretical computer science courses as well, such as those covering algorithms and the theory of computation.

Our proof construction platform provides the following benefits:

- **The puzzle-like game structure should increase engagement.** Puzzles have a rich history of being used to teach mathematics [15], and more recently Puzzle-Based Learning has been adopted in computer science and engineering pedagogy [6]. Digital game-based learning has also been shown to affectively motivate learners in numerous fields [18].
- **Students are encouraged to be precise and meticulous.** Our platform constrains students to only construct rigorous proofs. That way, students are guided away from the common pitfalls that frequently plague informal student proofs such as semantic ambiguity, insufficient detail, gaps in justification, and logical inconsistency.
- **Immediate feedback is provided to the student.** Traditional homework practice is non-ideal because it does not provide timely feedback for students to rectify learning deficiencies. It may be weeks before students get their corrected homework back, and some students may find themselves pushed to the next topic in the course before they have mastered the previous topic.

- **Homework can be graded automatically.** When our platform is used to replace some or all of the traditional written homework, it reduces the amount of resources needed for manually grading student work. This could be a major benefit for teaching very large courses such as MOOCs.
- **It is easy to adopt the platform in a wide variety of contexts.** Instructors can easily adapt it to any topic that involves proof instruction. The simplicity of the interface makes it equally suitable for introducing logic problems to K-12 students or teaching highly abstract mathematics to college students.

In this paper, we describe the platform in detail and we report our findings from a pilot study where we used the platform to provide supplemental aid to students in a discrete mathematics course and to students in an algorithms course.

2 BACKGROUND

2.1 Existing Proof Tools and Their Limitations

There are many Computer Assisted Instruction (CAI) systems designed to help students construct proofs in formal logic courses, including Wilfried Sieg's AProS [20], Peter Andrew's ETPS [1], and the EPGY theorem proving environment [13]. An overview of these and many more can be found in [19]. Many have a lengthy history – for instance, Wilfried Sieg has been developing AProS for decades. These tools automatically verify the correctness of each step of a student's proof. Some have been bundled with popular textbooks [2, 21], while others are Intelligent Tutoring System [7, 11, 16] and provide students with additional hints and feedback.

These tools have not been adapted much for use in computer science courses other than to familiarize students with formal logic. The one exception we are aware of is the equational logic system that Gries and Schneider used to teach a whole course in discrete mathematics [8].

There are technical challenges that make it difficult to adapt most logic-based proof tools to new domains. These tools require proofs to be constructed from a small set of primitive inference rules and axioms. The tedious steps involved in using these rules to construct even the most elementary of proofs taught in undergraduate math and computer science classes is vaguely reminiscent of developing a large software package in assembler code. Therefore, these tools cannot be pedagogically useful outside of teaching pure logic, unless they permit students to skip steps while still supporting the correct verification of the student's proof. This can be accomplished with automated theorem proving as demonstrated in [13] but there is still the problem of determining how many steps to let the student skip. Marvin Schiller's PhD Dissertation [19] looked at resolving this problem, and he has made some progress, as demonstrated in ActiveMath [14].

However, even when those challenges can be resolved, none of these tools permit students to construct proofs in natural language. Outside of formal logic courses, most instructors and textbooks use natural language to describe proofs and expect their students to do the same. Furthermore, a student's ability to construct proofs with a tool that only uses symbolic logic may not directly transfer to their ability to reason correctly using natural language, without explicitly training them to use the same kind of reasoning for natural language. The "Mental Models" theory of cognitive science predicts that humans make systematic logical errors when reasoning, because our natural method of reasoning is inconsistent with formal logic [9].

2.2 The Difference in Our Approach

Our proof construction platform allows students to construct proofs in natural language as well as symbolic logic and is agnostic to the many different flavors of formal systems. Informal proofs explained by an instructor in class can easily be adapted to our platform.

The reason this is possible is because our platform gives the student all the written assertions they need to construct a valid proof of whatever problem they are working on. In other words, they are given all the puzzle pieces they need to construct a proof (as well as some pieces not needed) and are left with the task of figuring out how those pieces connect together.

This is analogous to Parson's Problems [17], a similar kind of puzzle used to teach programming. With Parson's Problems, students are given blocks of code that they need to rearrange to create a program with a desired functionality. A study found a strong correlation between the ability to solve Parson's Problems and the ability to correctly write code from scratch, and concluded that these tasks likely require the same skills [5]. Similarly, constructing proofs with our platform may require much of the same deductive skills needed to construct proofs without the platform.

3 HOW THE PROOF CONSTRUCTION PLATFORM WORKS

We will illustrate how our platform works with the following logic problem: "Emily and Catherine each have their own pizza. Using the given assumptions, prove that Emily is not lactose intolerant." Figure 1 is a screenshot of the completed proof construction using our platform. We are given three assumptions:

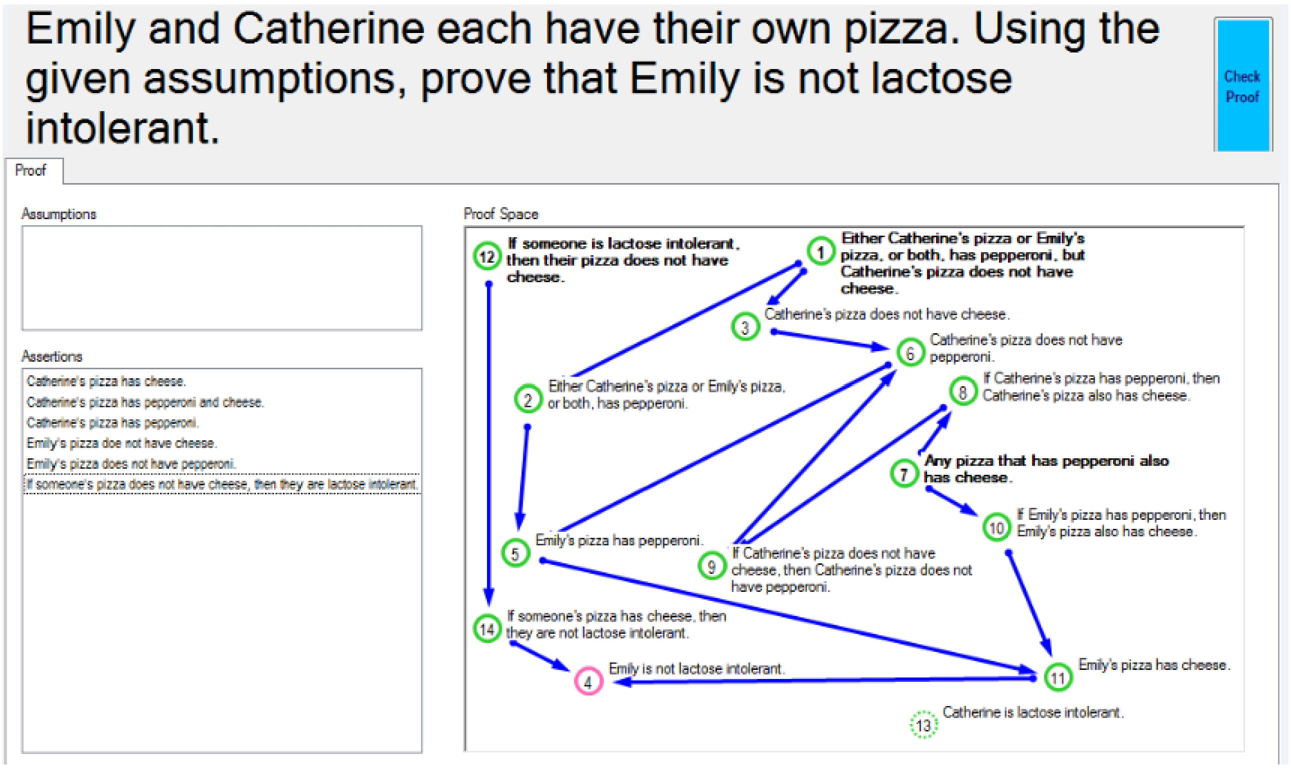
(1) "Either Catherine's pizza or Emily's pizza, or both, has pepperoni, but Catherine's pizza does not have cheese."
(2) "Any pizza that has pepperoni also has cheese."
(3) "If someone is lactose intolerant, then their pizza does not have cheese."

Assumptions and assertions are used to complete the proof. An assumption is a statement considered valid without need of substantiation, such as an axiom. An assertion is a statement that needs substantiation before it is considered valid. Assertions may be validated by connecting them with assumptions and other assertions.

3.1 Placing the Dots

To solve the problem, a student will select assumptions and assertions from prescribed lists on the screen. The student will drag assertions and assumptions from the lists into a proof space shown on the screen. When an assumption or assertion is moved into the proof space, a specific dot will appear with written text stating the assumption or assertion. The student will click individual dots to select them and connect them to other dots with arrows in the proof space.

Our platform allows a student to start the proof construction with any assumption or assertion and complete the proof in a roundabout


Emily and Catherine each have their own pizza. Using the given assumptions, prove that Emily is not lactose intolerant.

Catherine's pizza has cheese.
Catherine's pizza has pepperoni and cheese.
Catherine's pizza has pepperoni.
Emily's pizza doe not have cheese.
Emily's pizza does not have pepperoni.
If someone's pizza does not have cheese, then they are lactose intolerant.

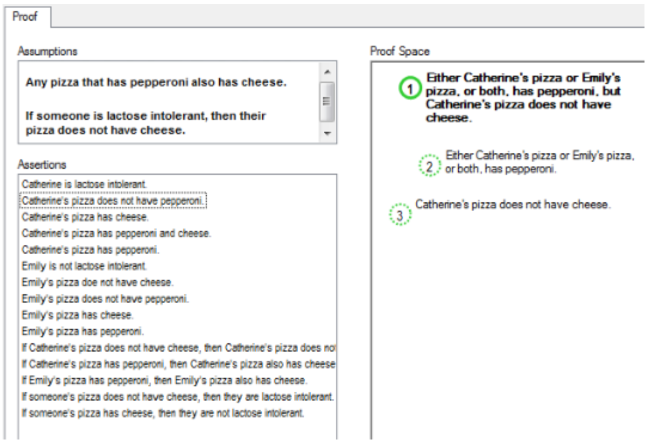
1 **Either Catherine's pizza or Emily's pizza, or both, has pepperoni, but Catherine's pizza does not have cheese.**
2 Either Catherine's pizza or Emily's pizza, or both, has pepperoni.
3 Catherine's pizza does not have cheese.
4 Emily is not lactose intolerant.
5 Emily's pizza has pepperoni.
6 Catherine's pizza does not have pepperoni.
7 **Any pizza that has pepperoni also has cheese.**
8 If Catherine's pizza has pepperoni, then Catherine's pizza also has cheese.
9 If Catherine's pizza does not have cheese, then Catherine's pizza does not have pepperoni.
10 If Emily's pizza has pepperoni, then Emily's pizza also has cheese.
11 Emily's pizza has cheese.
12 **If someone is lactose intolerant, then their pizza does not have cheese.**
13 Catherine is lactose intolerant.
14 If someone's pizza has cheese, then they are not lactose intolerant.

Figure 1: Screenshot of completed Pizza Proof Problem with text toggled on, assumption text is in bold print.

manner, like approaching the proof backward or using a mixture of going forward and backward to solve the proof problem.

After studying the three assumptions and the assertions on the prescribed lists adjacent to the proof space, in Figure 2, suppose the student decides to drag the assumption "Either Catherine's pizza or Emily's pizza, or both, has pepperoni, but Catherine's pizza does not have cheese." into the proof space. Notice that when this assumption is moved into the proof space, the text of the assumption appears with a dot "1" adjacent to the text. Also notice the text of the assumption is displayed in bold print to distinguish it from assertions which are displayed in regular print.

The student then determines what should follow this assumption. The student peruses the list of assertions, in Figure 2, decides the assertions "Either Catherine's pizza or Emily's pizza, or both, has pepperoni." and "Catherine's pizza does not have cheese." could follow the assumption in the proof space, and drags these two assertions into the proof space. The text of the two assertions appear with partial dots "2" and "3". A partial dot means the associated assertion needs substantiation to be validated. When an assertion is validated, its partial dot will change to a complete dot.



Any pizza that has pepperoni also has cheese.

If someone is lactose intolerant, then their pizza does not have cheese.

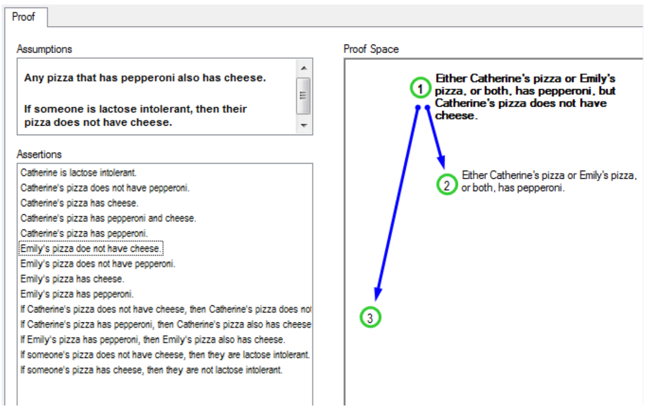
Catherine is lactose intolerant.
Catherine's pizza does not have pepperoni.
Catherine's pizza has cheese.
Catherine's pizza has pepperoni and cheese.
Catherine's pizza has pepperoni.
Emily is not lactose intolerant.
Emily's pizza doe not have cheese.
Emily's pizza does not have pepperoni.
Emily's pizza has cheese.
Emily's pizza has pepperoni.
If Catherine's pizza does not have cheese, then Catherine's pizza does no
If Catherine's pizza has pepperoni, then Catherine's pizza also has cheese
If Emily's pizza has pepperoni, then Emily's pizza also has cheese.
If someone's pizza does not have cheese, then they are lactose intolerant.
If someone's pizza has cheese, then they are not lactose intolerant.

1 **Either Catherine's pizza or Emily's pizza, or both, has pepperoni, but Catherine's pizza does not have cheese.**
2 Either Catherine's pizza or Emily's pizza, or both, has pepperoni.
3 Catherine's pizza does not have cheese.

Figure 2: Moving assumptions and assertions into proof space, assumptions are complete dots and assertions are partial dots.

3.2 Connecting the Dots

To see if the decision that the two assertions follow the assumption was correct, the student left clicks dot "1" (assumption) followed by a left click of partial dot "2" (assertion). An arrow will appear to connect the dots with the arrow pointing at dot "2" as shown in Figure 3. Also notice that the partial dot "2" has changed to a complete dot "2" indicating the assertion was validated by the connection with the assumption dot "1". The student repeats this process to connect dots "1" and "3". As shown in Figure 3, an arrow connects the two dots and the partial dot "3" is changed to a complete dot indicating the assertion associated with dot "3" was substantiated and validated by the connection.



Any pizza that has pepperoni also has cheese.

If someone is lactose intolerant, then their pizza does not have cheese.

Catherine is lactose intolerant.
Catherine's pizza does not have pepperoni.
Catherine's pizza has cheese.
Catherine's pizza has pepperoni and cheese.
Catherine's pizza has pepperoni.
Emily's pizza doe not have cheese.
Emily's pizza does not have pepperoni.
Emily's pizza has cheese.
Emily's pizza has pepperoni.
If Catherine's pizza does not have cheese, then Catherine's pizza does not
If Catherine's pizza has pepperoni, then Catherine's pizza also has cheese
If Emily's pizza has pepperoni, then Emily's pizza also has cheese.
If someone's pizza does not have cheese, then they are lactose intolerant.
If someone's pizza has cheese, then they are not lactose intolerant.

Either Catherine's pizza or Emily's pizza, or both, has pepperoni, but Catherine's pizza does not have cheese.

Either Catherine's pizza or Emily's pizza, or both, has pepperoni.

Figure 3: Connecting the dots, validated assertions become complete dots, toggling text and relocating dots.

Notice in Figure 3 that dots "2" and "3" have moved from their original locations shown in Figure 2. Our platform permits the student to drag the dots to different locations in the proof space to arrange the proof spatially in the way that makes the most sense to them.

Also notice the text associated with dot "3" is not shown in Figure 3. The platform allows the student to make the text associated with a dot disappear and reappear by double clicking the dot. This feature allows the student to hide the text of dots they are not considering at the moment. This can reduce cognitive load and promote better focus and efficiency for the student to solve the part of the problem they are currently working on [3]. Additionally, when students hover their mouse pointer over any dot, it will intermittently display the associated assertion, if it has been hidden.

3.3 Completing the Proof

The student continues selecting assertions and assumptions from the lists, moving them into the proof space, and connecting them until the proof is completed. The proof is completed when the goal assertion is validated (its partial dot has been converted into a complete dot) and all assertions leading to the goal assertion have also been validated. Notice that Figure 1 shows the goal assertion dot "4" and all of the other connected dots are complete dots. The tutoring system will also display a message on the computer screen that the proof has been successfully completed.

Notice the remaining assertions in the list even after the proof is completed in Figure 1. These assertions are *distractors*, unnecessary to complete the proof. Some of the distractors are valid assertions that can actually be justified in the proof space, but will lead down paths that do not connect with the goal. Others are "bugs" that get denoted with hashed dots when dragged into the proof space.

3.4 Authoring New Problems

New problems can be authored for this platform using the Python scripting language. The following is example code for the syllogism, *"All men are mortal. Socrates is a man. Therefore, Socrates is mortal."*

```
ProblemDescription.Text = "Prove Socrates is mortal."
ax1 = ProofItem("All men are mortal.")
ax1.isAssumption = True
ax1.show()
ax2 = ProofItem("Socrates is a man.")
ax2.isAssumption = True
ax2.show()
goal = ProofItem("Socrates is mortal.")
goal.Requirements.Add(ax1)
goal.Requirements.Add(ax2)
goal.show()
```

In the future, we are considering creating a graphical tool for constructing new problems.

4 EVALUATION

Over the course of two semesters, we did a study to evaluate students using our proof construction platform to solve problems in two different Computer Science courses at an R1 public university. Overall, there were 59 students who got to test the platform and their feedback is reported here.

We started with a pilot experiment in the Fall 2016 semester in a Junior-level algorithms course. The experiment commenced near the end of the semester, and we gave participants a problem related to the topic of NP-Completeness. This was one of the last topics covered in the course, and students had virtually no practice with or assessment on this topic before they participated in the study. A total of 26 students used the platform in this experiment, and an additional 22 students were part of a control group.

In Spring 2017, we continued with an experiment in a Sophomore-level discrete math course. This experiment also occurred near the end of the semester. We gave participants three pure logic problems. Two were written in plain English, and a third used a mixture of English and symbolic formulas. These problems were similar to many practice problems that the students had already completed in their homework. They had also been given a similar problem on their midterm exam. A total of 33 students used the platform in this experiment, and additional 36 students were part of a control group.

4.1 Methodology

Students received a small amount of extra credit on their final exam for participating in the study, but were not required to participate and could drop out at any time without penalty. We followed the same general protocol for both Fall 2016 and Spring 2017 experiments.

We announced our study at the beginning of a lecture. We told the students general details about the study, but to avoid biasing the distribution of volunteers, we did not tell students what kind of problems they would be given or even the nature of the software they would be using, besides indicating that it was designed to help them in the course.

Volunteers were assigned to one of two groups, experimental or control. We asked the instructor to ensure that both groups were equal in size and were randomly selected in a way to ensure that both groups had the same grade distributions based on their performance from a midterm exam. After the groups were selected, some students dropped out, resulting in the groups being slightly unequal in size but this did not change the distribution much. We believe that slightly more students may have dropped from the experimental groups because they could not run the software if they did not have Windows.

The experimental group solved problems with the proof construction platform. They were instructed to download the software with the selected problems, watch a short instructional tutorial[1] on the software, and attempt to solve the problems. While they were using the platform, we automatically video recorded their screen interactions and asked them to upload the videos to us. After uploading their data, they were asked to do an online survey.

The control group worked the same problems, but without the platform, either writing or typing their solutions as they would a normal homework assignment. They uploaded PDF copies of their written work to us, and we returned feedback similar to what they would receive on a homework assignment in the course.

Both groups were graded on how well they performed on the problems in the experiment. The control group was graded using the same standards normally used for homework problems in the course. The experimental group was graded based on what we observed in the videos, with variable scores assigned for incomplete proofs based on how many connections were missing.

We took extreme care to ensure that all participants in the study were anonymous to us. They generated random identifiers that tracked them anonymously throughout the study. We setup an anonymous file drop service for them to upload their data to us, which was either videos files for the experimental group or PDF files for the control group. The feedback given to the control group was also delivered via anonymous email drops. Most importantly, the survey responses were also anonymous and only linked to the anonymous identifiers.

4.2 Description of Problems

The following problem was given for the experiment in the algorithms course: "*Let PATH =* $\{\langle G, s, t\rangle \mid G$ *is a graph with a path from s to t}. We know PATH is in P. Prove that if* $P = NP$*, PATH is NP-Complete.*"

We had three logic problems used in the discrete math course, which we refer to as the "Pizza Problem", the "Muddy Dog Problem" and the "Murder Mystery Problem". The "Pizza Problem" is described in Section 3, and we omit the full descriptions of the others to save space. "Muddy Dog" is moderately difficult, requiring 42 dot connections for successful completion. The "Murder Mystery Problem" required 55 dot connections for successful completion, and was challenging because it mixed symbolic formulas with English descriptions. The others were written completely in English.

4.3 Results and Discussion

The video recordings submitted by the experimental students were analyzed to determine if the students encountered difficulties in using the software or user interface, and if they were able to solve the proof problems with the tutoring system. No significant difficulties were observed in the students using the software. Table 1 gives statistics for how the control and experimental groups performed on

[1]https://youtu.be/3jQM_T1qZvY

Table 1: Performance comparison of Experimental and Control Groups. N = NP-Completeness Problem, P = Pizza Problem, D = Muddy Dog Problem, M = Murder Mystery Problem. For each problem, scores ranged from 0-100. The "Perfect" row indicates the percentage of students who got a perfect score amongst those who attempted the problem.

	N		**P**		**D**		**M**	
	Experimental	Control	Experimental	Control	Experimental	Control	Experimental	Control
Mean	94	59	95	96	79	82	49	46
Median	100	55	100	100	95	100	50	50
Perfect	88	27	90	92	55	60	44	31

the problems. For most of the problems, the statistics are very similar for both groups, with the exception of the "NP-Completeness Problem". Students in the control group had significant conceptual difficulties with that problem, and often conflated the concepts of NP and NP-Complete in their proofs. Many tried to argue that PATH was NP-Complete because it can be verified in polynomial time, which indicates that PATH is in NP but not that it is NP-Complete. We think that the guided help that the platform gives the experimental group kept them from getting lost, and helped them solve what is an easy problem for students who have no conceptual difficulties with the material.

However, for the other problems given to the students in the discrete math course, we do not believe that many of the students in either group had conceptual difficulties. The differences in performance seem to be more indicative of the innate challenge provided to students in figuring out the proof, not difficulty understanding the material. Given that the statistics are so similar for both groups, we theorize that our puzzle-based method of discovering proofs is just as challenging and rewarding as the normal pen and paper method.

The data from the surveys were analyzed and the results that mainly pertain to the usefulness of the platform are shown in Figure 4. Student comments from the surveys are italicized below.

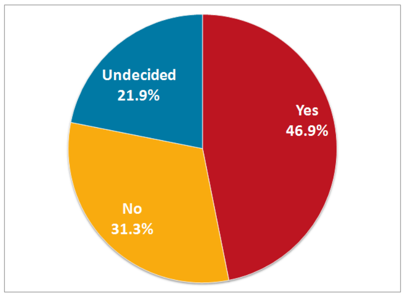
One attraction of the proof construction platform is it could be amusing and entertaining to the students using it. One student stated that it was a *"fun way to learn"* and another said, *"It is way more interesting than traditional homework."* But the main benefit of the proof construction platform is exemplified by this survey comment: *"I think if I had more practice using this application, it would be beneficial to my proof construction skills."*

Other students elaborated on what they found helpful:

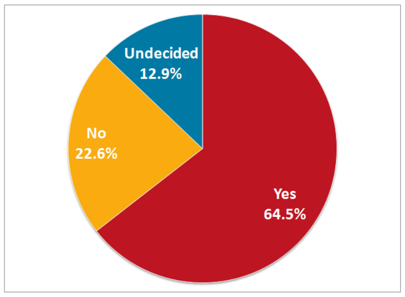
"It's very helpful in visualizing the proofs."

"Having the pieces already helps to understand what parts are required whereas with a pen and paper you are on your own."

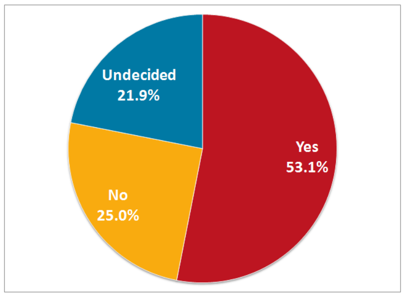
"One thing that is useful is being able to do the forward backward proof method. It was very straightforward to do backward proofs. Another thing that I thought was useful was being able to solve the problem in chunks. E.g. I could solve two different parts of a problem then use the 'backward' proof method to link the two parts together."



(a) Did you find the tutoring system helpful in your learning proof construction?



(b) Do you want the tutoring system to be available in other courses with proof construction?



(c) Would you recommend the tutoring system to others learning proof construction?

Figure 4: Responses from Questionnaires

Many of the surveys comment on how the platform forced them to be meticulous: *"It was good at making me remember the smaller steps. You aren't allowed to jump, you really have justify everything."*, and *"It takes much more time than do it by hand."* With pen and paper, many students skip lots of steps. In the future, a feature may be developed for the platform to assess a student's ability to solve specific types of proofs by looking at past performance. When a student has mastered a particular proof type, we can reduce the granularity of proof steps to allow abbreviated proofs for those types of proofs.

There were many comments requesting the provision of hints when using the proof construction platform; e.g., *"If there is a hint while I am stuck in a problem, it will be great. Otherwise I will just give up after many tries."* This is also a feature we plan to introduce in the future.

The results of the experiments show the proof construction platform is indeed helpful to students learning proof construction.

5 CONCLUSION AND FUTURE WORK

In this paper, we have presented and evaluated a new tool for helping students learn the process of constructing proofs from assertions in two different computer science theory classes. Overall feedback from students who used the tool was positive, and our empirical results indicate that students who use our software were able to overcome conceptual difficulties that the control group struggled with, as illustrated with the NP-completeness problem. In the future, we plan to look at how the use of the software affects exam performance as an indicator of retention of critical proof construction skills. We also plan to add a hint feature to guide students in the construction process, and a search feature that will allow students to find specific assertions more rapidly. We aim for our tool to become an indispensable asset to students of computer science.

Acknowledgements: We thank Dave Barrington for providing assistance to facilitate the experiments.

REFERENCES

[1] Peter B Andrews, Chad E Brown, Frank Pfenning, Matthew Bishop, Sunil Issar, and Hongwei Xi. 2004. ETPS: A system to help students write formal proofs. *Journal of Automated Reasoning* 32, 1 (2004), 75–92.

[2] David Barker-Plummer, Jon Barwise, and John Etchemendy. 2011. *Language, Proof, and Logic: Second Edition* (2nd ed.). Center for the Study of Language and Information/SRI.

[3] Roland Brünken, Roxana Moreno, and Jan L Plass. 2010. *Cognitive Load Theory.* Cambridge University Press.

[4] Bjorn Carey. 2014. Stanford's Maryam Mirzakhani wins Fields Medal. (12 Aug. 2014). Retrieved August 31, 2017 from http://news.stanford.edu/news/2014/august/fields-medal-mirzakhani-081214.html

[5] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research.* ACM, 113–124.

[6] Nickolas Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. 2010. Puzzle-based learning for engineering and computer science. *Computer* 43, 4 (2010), 20–28.

[7] João Carlos Gluz, Fabiane Penteado, Marcel Mossmann, Lucas Gomes, and Rosa Vicari. 2014. A student model for teaching natural deduction based on a prover that mimics student reasoning. In *International Conference on Intelligent Tutoring Systems.* Springer, 482–489.

[8] David Gries and Fred B Schneider. 1994. *A new approach to teaching mathematics.* Technical Report TR94-1411. Department of Computer Science, Cornell University.

[9] Philip N Johnson-Laird. 2010. Mental models and human reasoning. *Proceedings of the National Academy of Sciences* 107, 43 (2010), 18243–18250.

[10] Richard Jay Lipton. 2015. Why Is Discrete Math Hard To Teach? (27 Dec. 2015). Retrieved August 31, 2017 from https://rjlipton.wordpress.com/2015/12/27/why-is-discrete-math-hard-to-teach/

[11] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. 2017. Generating hints and feedback for Hilbert-style axiomatic proofs. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.* ACM, 387–392.

[12] Mark McCartin-Lim. 2016. Complexity Tutor: Developing an Interactive Tutoring System for Computational Complexity (Abstract Only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16).* ACM, New York, NY, USA, 494–494. https://doi.org/10.1145/2839509.2850524

[13] David McMath, Marianna Rozenfeld, and Richard Sommer. 2001. A computer environment for writing ordinary mathematical proofs. In *Logic for Programming, Artificial Intelligence, and Reasoning.* Springer, 507–516.

[14] Erica Melis and Jörg Siekmann. 2005. e-Learning Logic and Mathematics: What We Have and What We Still Need. *Essays in Honor of Dov Gabbay* (2005).

[15] Zbigniew Michalewicz and Matthew Michalewicz. 2007. Puzzle-based learning. In *Proceedings of the 2007 AaeE Conference.* 1–8.

[16] Behrooz Mostafavi and Tiffany Barnes. 2017. Evolution of an intelligent deductive logic tutor using data-driven elements. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 5–36.

[17] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.* Australian Computer Society, Inc., 157–163.

[18] Marc Prensky. 2001. *Digital Game-based Learning.* McGraw-Hill.

[19] Marvin R. G. Schiller. 2010. *Granularity Analysis for Tutoring Mathematical Proofs.* Ph.D. Dissertation. Saarland University, Saarbrücken, Germany. Advisor(s) Siekmann, Jörg.

[20] Wilfried Sieg. 2007. The AProS project: Strategic thinking & computational logic. *Logic Journal of IGPL* 15, 4 (2007), 359–368.

[21] Daniel J. Velleman. 1994. *How to Prove It: A Structured Approach.* Cambridge University Press.