Immediate Data-Driven Positive Feedback Increases Engagement on Programming Homework for Novices

Samiha Marwan
North Carolina State Univ.
Raleigh, NC, USA
samarwan@ncsu.edu

Thomas W. Price
North Carolina State Univ.
Raleigh, NC, USA
twprice@ncsu.edu

Min Chi
North Carolina State Univ.
Raleigh, NC, USA
mchi@ncsu.edu

Tiffany Barnes
North Carolina State Univ.
Raleigh, NC, USA
tmbarnes@ncsu.edu

ABSTRACT

Learning theories and psychological research show that positive feedback during practice can increase learners' motivation, and correlates with their learning. In our prior work, we built a system that provides immediate positive feedback using *expert-authored* features, and found a promising impact on students' performance and engagement with the system. However, scaling this expert-feedback system to new programming tasks requires extensive human effort. In this paper, we present a system that provides *automated, data-driven, immediate positive feedback (DD-IPF)* to students while programming. This system uses a data-driven feature detector that automatically detects feature completion in the current student's code based on features learned from historical student data. To explore the impact of DD-IPF on students' programming behavior, we performed a quasi-experimental study across two semesters in a block-based programming class. Our results showed that students with DD-IPF were more engaged, as measured by time spent on the programming task, and also showed marginal improvement in their grades, compared to students in a prior semester solving the same task without feedback. This suggests that positive feedback based on data-driven feature detection can provide benefits in student engagement and performance. We conclude with design recommendations for data-driven programming feedback systems.

1. INTRODUCTION

Intelligent tutoring systems (ITSs) are software systems that have been shown to improve student performance and learning by providing individualized feedback to each student, as human tutors do [26]. One particularly effective form of *human tutor* feedback is *positive* feedback, which can improve students' affective outcomes [5], and correlates with their learning [10]. Mitrovic et al. noted that ITSs "that teach primarily by addressing errors and misconceptions might become yet more helpful if extended with a positive feedback capability [18]." However, there are two barriers to using positive feedback effectively in ITSs.

First, little work has empirically evaluated the effect of positive feedback in ITSs, especially in traditional classrooms, and it is unclear how it will affect students in practice. On one hand, one might think that students who have already correctly completed a step will not need additional feedback since they are already on the right track. On the other hand, considering that novices have not yet established their programming understanding, it might be particularly beneficial for them to get positive reinforcement when they create a correct step. Novices have a range of prior knowledge and confidence levels, and unexpected results in the programming environment may cause students to question their knowledge. Providing students with feedback that confirms their correct steps could, therefore, reduce their uncertainty [18], and this may also improve their performance [17], but further empirical evaluations are needed to test this. Second, positive feedback is rarely present in current programming ITSs, since correct steps can be difficult to detect in programming problems, due to their huge, possible solution strategies and sparse, diverse data from students.

In this work, we present a system that provides data-driven immediate positive feedback (DD-IPF) to novice students while programming. In this system, we combine both *automated data-driven feature detection* with *low-effort expert human labelling* to provide high quality data-driven feedback. To do so, the DD-IPF system uses a data-driven feature detector algorithm that learns common code structures (features) that are present in correct solutions from prior student data, and then detects when these features are completed or broken in a student's code. We combined these features into a set of objectives with meaningful labels designed by human experts. We integrated our DD-IPF system into a block-based programming environment. As shown in Figure 1, the interface of our DD-IPF system has two components: a *progress panel* that displays human labels of the data-driven features of a programming task and shows when they are completed, and *pop-up messages* triggered based on the completion of these features or lack of progress.

We performed a quasi-experimental pilot study across two semesters to investigate our primary research question: How does the data-driven immediate positive feedback system impact student engagement and performance while programming? Through log data analysis, we found that students who received DD-IPF spent significantly more time with the system, and had a somewhat better performance than students who did not work with the DD-IPF system. We argue that this increase in time shows that students were more engaged to work more on the programming assignment, which had historically low engagement, and this resulted in an increase in students' scores.

In summary, this work makes **contributions** to educational data mining and computing education through: (1) an approach that combines a data-driven feature detector with human labelling to provide data-driven immediate positive feedback (DD-IPF), (2) a controlled quasi-experimental study that suggests that DD-IPF in classrooms can increase students' engagement with the programming environment, and has the potential to improve their performance as well, and (3) recommendations on the design of data-driven feedback systems for programming.

2. RELATED WORK ON POSITIVE FEEDBACK

Intelligent tutoring systems are developed to adopt human strategies to improve students' learning, especially through adaptive feedback [9, 27, 12]. For example, positive feedback is given to students when they complete a problem-solving step appropriately, e.g. "Good move" [3, 10, 12]. Empirical studies of human tutoring dialogues show that positive feedback occurs eight times as often as negative feedback [10], and correlates with learning [13, 10, 6, 7]. More importantly, human tutors find positive feedback to be an effective motivational strategy that improves student confidence [16]. In programming, while various learning environments provide feedback through compiler messages [22, 4], or error detectors [1, 25], or autograders [2, 14], or hints [23, 20], far less work has been devoted to integrate features that detect students' correct moves and provide positive feedback as human tutors do.

To our knowledge, only two studies have focused on automated positive feedback for programming: Fossati et al.'s iList[12] tutor for data structures, and Mitrovic et al.'s SQL tutor, for databases [18]. In the iList tutor, Fossati et al. provided students with positive feedback by calculating the goodness and uncertainty of students' moves. Authors detected a good move if it improves the student's probability to reach the correct solution, while the uncertainty is detected if a student spent more time than the time taken by prior students at this point. If a student's move is detected as a good move and uncertain, then iList will provide positive feedback. Fossati et al. found that the iList tutor with positive feedback improved learning, and students liked it more than iList without positive feedback. The SQL tutor is a constraint-based tutor using a knowledge base of 700 constraints. When a student submits their solution, the detected satisfied constraints can trigger positive feedback to students. An evaluation of SQL tutor showed that positive feedback helped students master skills in less time. However, in both iList and SQL tutor studies, positive feedback is just one of several supports provided, and their studies do not attempt to separate the impact of the positive feedback from the overall system.

In addition, the timing of positive feedback can be important for retention. The SQL tutor, and programming autograders like Lambda for Snap*!* [2], provide positive feedback only when students submit their code, and usually this feedback indicates which constraints or test cases are satisfied. This means that students who become discouraged or confused during programming may never receive any positive feedback at all. While prior work shows that immediate feedback allows students to finish problems quickly [8], there is less evidence on the impact of immediate *positive* feedback on students in programming. Our DD-IPF system continuously and adaptively confirms when students complete (or break) meaningful objectives through its progress panel. Our system also includes personalized pop-up messages addressing the user as "you", and tailored to our student population, since personalization is key in effective human tutoring dialogs [5, 10] and has been shown to improve novices' learning [15, 19].

3. DATA-DRIVEN FEATURE DETECTOR (DDFD) ALGORITHM

In our prior work [29], we introduced a data-driven feature detector (DDFD) algorithm for block-based programming exercises. In this context, a *feature* corresponds to a meaningful objective or property of a correct solution. In brief, this algorithm works as follows: First, it learns common features from existing, correct student solutions. Each feature is represented as a set of code blocks, referred to as code shapes. For example, using a `'pen down'` block, followed by a `'move'` block, is a necessary feature for any drawing task. Second, the algorithm detects the presence or the absence of each feature in each student's code, generating a boolean array that represents its *feature state*. For example, if the algorithm learns four features for a given exercise, and for a given student's code it detects only the first and last features, then the algorithm will output {*1, 0, 0, 1*} as the code's feature state.

This DDFD algorithm motivated us to build a system that provides *immediate positive* feedback that can be easily *scaled* to various programming tasks because of three reasons. First, this DDFD algorithm can detect features at any phase of student code, whether the code is complete, or incomplete, and therefore can provide *immediate* feedback. Second, the presence of features represents student *progress* towards the correct solution, and therefore can provide *positive* feedback. Third, it is designed to generate features for a variety of programming tasks, as long as historical student data exists, making it *scalable* to various contexts and tasks. However, the DDFD algorithm suffers two limitations that make it hard to deploy in practice. First, because features are code shapes that are generated automatically, they are not labelled with meaningful names for students to understand how they are making progress. Second, the generated features are too specific and may need to be further clustered into a smaller set of features, to limit their number and allow more concrete positive feedback. In the next section, we describe how we took advantage of the DDFD algorithm, and addressed its limitations to develop a system that pro-

vides data-driven immediate positive feedback to students while programming. Rather than being fully data-driven or fully expert-authored, this system applies a small amount of expert curation to a largely data-driven process to achieve scalable, higher-quality results.

4. DATA-DRIVEN IMMEDIATE POSITIVE FEEDBACK (DD-IPF) SYSTEM

To build the DD-IPF, we first had to apply the DDFD algorithm to a new dataset and overcome DDFD's two main limitations of having unlabeled features and having too many to present to students. To do so, we first used three semesters of previous correct students' solutions to generate a set of code shapes (features) of correct solutions of a programming exercise (described in Section 5.2). As expected, (as shown in the first column of Table 1), each of the seven generated features were too small to be used as an objective (e.g. F1: create a procedure), and lacked labels to break the larger task down into smaller meaningful objectives. Therefore, the first author combined these seven features into four features, and provided each with a label. The last author briefly reviewed the combination and suggested a few minor wording changes. Table 1 shows the mapping of the seven data-driven features to four meaningful objectives, with human labels. The features were combined, ordered, and labeled based on their relevance and importance. Our goal was to make the label clear, meaningful and concrete. Note that not every aspect of the data-driven features is reflected in the objective labels, so there is not a direct correspondence between the objective label provided to students and the process that the detector applies to detect it. We decided to limit the number of objectives to 4, and that it was more important to have independent objectives that students could understand, than it was to tell students exactly what constructs make up each objective. As we discuss in more detail later, listing just a few objectives will necessarily leave out information, there is no way to build an algorithm that can correctly detect every possible solution, and there is more complex information in the detectors than novice programmers can understand. Given these important and inherent limitations of automated feedback, we felt clarity and brevity were of high value.

We strove to design the interface of the DD-IPF system to help students' track their progress, with the goal of improving their performance, and increasing their motivation to complete the programming task. The DD-IPF system consists of two main features that continuously provide adaptive, positive feedback to students based on their code edits: 1) a progress panel and 2) pop-up messages, shown in Figure 1. These two components are designed together to comprise a positive feedback system for open-ended programming. The progress panel shows students the human labels of data-driven features (objectives) for a task, and whether they are complete, or broken, since prior research suggests that students who were uncertain often delete their correct code [11]. Initially, all the objectives are deactivated. Once an objective is detected to be completed, it provides positive feedback by becoming green, but if it is detected to be broken, it changes to red, as shown in the bottom right of Figure 1. The pop-up messages provide positive messages on students' accomplishments, i.e. whenever they completed an objective or fixed a broken one, as shown in the top left of Figure 1. To increase students' motivation, the system also provides motivational pop-up messages if a student makes no progress for more than three minutes (a threshold based on instructors' feedback). We integrated the DD-IPF system into iSnap, a block-based programming environment that extends Snap*!* programming environment [20].

We note that our current version of DD-IPF system has a similar interface to our adaptive feedback system in [17], but the approach to generate positive feedback is different. Specifically, in our prior work we used expert-authored autograders to monitor students' progress instead of the data-driven feature detector algorithm presented in our current work (Section 3). This adaptation is important for scaling the system to more programming tasks.

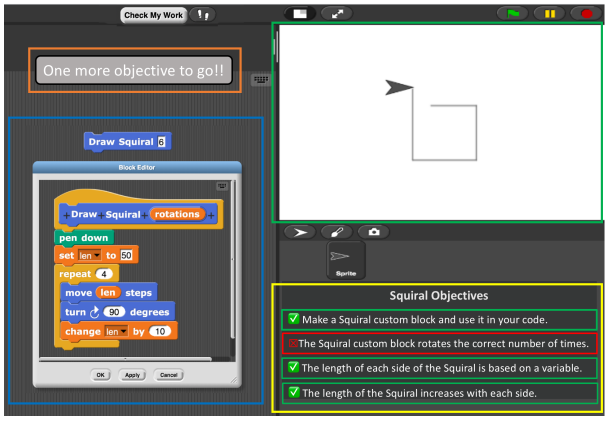

Figure 1: Data-driven immediate positive feedback (DD-IPF) system integrated to iSnap, a block-based programming environment. The script area – where students add code blocks – is shown in the bottom-left (blue box), the pop-up message is shown in top-left (orange box), the stage where students can see their output is shown in the top-right (green box), and progress panel is shown in the bottom-right (yellow box).

5. CLASSROOM STUDY

Our goal in this study is to evaluate the impact of data-driven immediate positive feedback on students in a classroom setting. This study seeks to answer our primary research question: How does the data-driven immediate positive feedback system impact student engagement and performance while programming?

5.1 Population

The participants of this study were enrolled in two semesters of an introductory programming course for computer science non-majors, both taught by the same instructor at a large southeastern university in the United States. The Spring 2019 class had 48 students and the Spring 2020 class had 42 students. We adopted a quasi-experimental design, where the *experimental* (Spring 2020) group used iSnap with access to the DD-IPF system on one assignment (Squiral, described below), and the *control* (Spring 2019) group completed the same assignment in iSnap but *without* the DD-IPF system. Due to deployment issues, the *first* 15 students to complete the assignment (36%) in the Spring 2020 class did not use

Table 1: Generated Data-driven features, and their Corresponding Objectives and Human Labels.

Data-Driven Features	Combined Features (Objectives)	Human Label
F1. Create a procedure OR '`ReceiveGo`' block.	Objective 1 = F1 + F2	Make a Squiral custom block and use it in your code.
F2. Add procedure on stage.		
F3. Have a '`multiple`' block with a variable in a '`repeat`' block OR two nested '`repeat`' blocks.	Objective 2 = F3	The Squiral custom block rotates the correct number of times.
F4. Add parameter in the procedure.	Objective 3 = F4 + F5	The length of each side of the Squiral is based on a variable.
F5. Add a variable in '`move`' block AND a variable in '`repeat`' block.		
F6. Have a '`move`' and '`turn`' block inside a '`repeat`' block AND a '`pen down`' block.	Objective 4 = F6 + F7	The length of the Squiral increases with each side.
F7. Change a variable value inside a '`repeat`' block.		

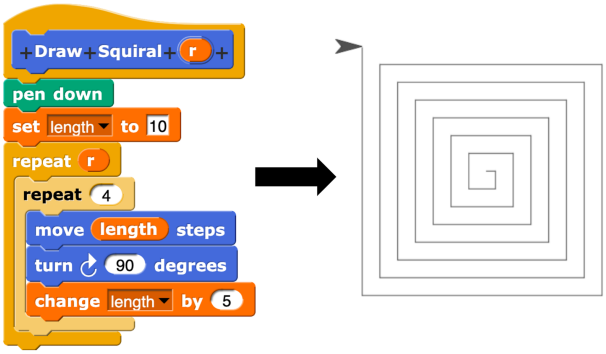

Figure 2: One possible solution of Squiral programming exercise (on the left), and its output (on the right).

the DD-IPF system, and therefore we excluded them, resulting in 27 students in the experimental group. Since this exclusion may have biased our sample in the experimental group, we also removed the first 36% of students (17) in the Spring 2019 class, resulting in 33 students in the control group.

5.2 Procedure

This study took place during the second programming homework in the CS0 classroom. The programming task is called *Squiral*, which asks students to create a method with one parameter, r, that draws a square-shaped spiral with r rotations. Figure 2 shows one possible solution to *Squiral*, and its output. The instructor gave students one week to submit this homework. In both semesters, the programming environment provides students with access to on-demand next-step hints, which provide students with single edits that can possibly bring their code closer to a correct solution, but this was independent of their access to the DD-IPF. Our prior evaluations of expert-authored positive feedback suggests it provides complementary benefits to hints, but does not conflict with them [17].

5.3 Results

In this section we report the impact of using our DD-IPF system on the *time* students took to finish the homework exercise, and the *score* of their submitted solution, as assessed by a rubric.

Time: We measured a student's total time from when they began to program to the time when they either successfully completed the programming task, or submitted it incorrectly. We did this because some students who completed the task continued to work afterwards, so we did not include the afterward time in their total time. We found the average time (in minutes) spent by students in the experimental group ($Med = 34.75$; $M = 42.29$; $IQR = 28.04$) was much greater than that spent by the control group ($Med = 13.54$; $M = 20.14$; $IQR = 17.81$), as shown in Plot A of Figure 3. A t-test[1] shows that this difference is significant with a strong effect size ($t(40.78) = 3.96$; $p < 0.01$; Cohen's $d = 1.08$).

Score: To grade students' submissions, we used a rubric for the *Squiral* exercise created by researchers in prior work. This rubric consists of six items, each with two points, making a total of 12 points. Two researchers in block-based programming graded students' submitted code across the two semesters[2]. We found that the average score of students in the experimental group ($Med = 91.7\%$; $Mean = 86.4\%$; $SD = 14.5\%$) was more than that in the control group ($Med = 83.3\%$; $Mean = 76.9\%$; $SD = 25.5\%$), as shown in Plot B of Figure 3. A Mann-Whitney U test[3] shows that this difference is not significant, but has a medium effect size ($p = 0.19$; Cohen's $d = 0.45$).

6. DISCUSSION

In this section we discuss our primary research question: How does the data-driven immediate positive feedback system impact student engagement and performance while programming? We found that the DD-IPF system increased the amount of time students spent engaged with the programming homework, and we found suggestive evidence that it can improve students' programming performance as well.

In our study we found students who used the DD-IPF system (*experimental* group) took more than double the time on average compared to students in the *control* group to complete their homework. While increasing time on task is sometimes considered a negative outcome (i.e. decreased learning efficiency), for this homework, we believe that our results suggest that the DD-IPF system *increased* students'

[1]Our use of t-tests indicates the data was normally distributed.
[2]We note that we found all students submitted their code for grading.
[3]Our use of non-parametric tests indicates the data was non-normal.

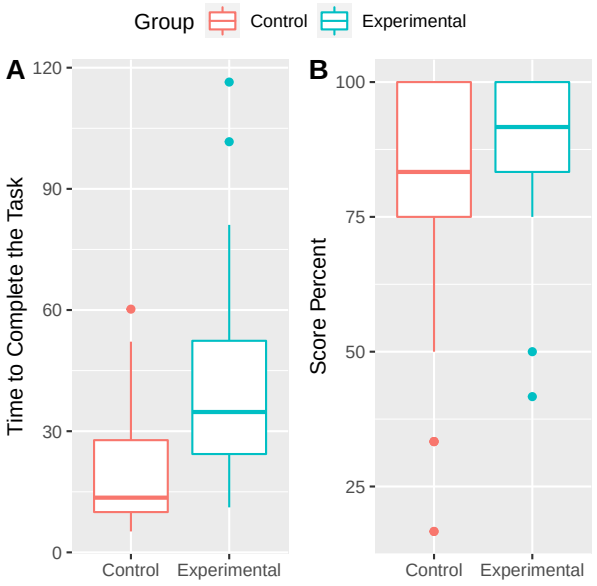

Figure 3: Boxplots comparing time (in minutes) to complete the task (left) and score percent (right) between the control and experimental groups.

engagement with the assignment. *Squiral* is a challenging assignment for students, which should take students more than the median 13.5 minutes spent by the control group to compete correctly. However, this task was a programming homework, where students were not observed and did not have easy access to an instructor to get assistance. We found that of the 15 students in the control group who submitted their homework in less than the median time of 13.5 minutes, 11 of these students (73.3%) had incorrect submissions. This suggests that many students spent too little time and submitted incomplete or incorrect work in the control group. Overall, the control group had lower performance (average grade 77%), in comparison with the experimental group (average grade 86%). By contrast, only 2 students in the experimental group spent less than 13.5 minutes. We hypothesize that the DD-IPF system helped keep students engaged by continuously informing students in the experimental group about their progress, such as how far they are from completing the homework, which might have motivated them to keep working to try to get all the objectives marked correct in the progress panel. However, we cannot directly investigate this hypothesis with the current study (e.g. by correlating student time and performance). For example, higher-performing students may take less time to complete the assignment (creating a negative correlation), even if any individual student may perform better by taking more time.

However, we note that some of the increase in students' time engaged with the assignment was not productive, (e.g. 2 students spending over 90 minutes). Some of this may have resulted from errors in the data-driven feature detection. As shown in Table 1, a few of the data-driven features do not correspond well to a concrete assignment objective. For example, data-driven Feature 1 requires the use of `'ReceiveGo'` block (which can be used to start a script) to complete Objective 1; however this is not necessary according to the instructions. It is a feature that most students did in the dataset used to train the DD-IPF system. Based on our manual investigation of the *experimental* group log data, we found a number of times where the system *mislabeled* a student's progress. For example, some students finished the programming task, but not all the objectives were detected by the system. As a result, a few students kept working for more time, despite having completed the task, which was not the case for students in the *control* group. We provide specific case studies on instances at which the DD-IPF system provides incorrect feedback, and how students responded in [24].

We also found that students with the DD-IPF system may have had improved performance, since students in the *experimental* group achieved higher average scores (almost 9.5% higher) than those in the *control* group. We conclude that the DD-IPF system may have increased students' scores *because* it increased their working time in the programming environment. The ability of the progress panel to confirm correct steps might have increased students' motivation and reduced their uncertainty about their moves, leading to improved performance. These results are consistent with the "uncertainty reduction" hypothesis presented by Mitrovic et al. [18], suggesting that the positive feedback helps students to continue working because it reduced their uncertainty about their code edits.

While the availability of on-demand hints might have affected our results somewhat, students in both semesters had access to hints, so the differences between semesters were due primarily to the DD-IPF system. Interestingly, when we compared the hint usage across the two semesters, we found suggestive evidence that the DD-IPF system might have increased students' hint usage. We found that the average number of hints requested by the experimental group ($Mean = 17$; $Med = 11$) was higher than that of the control group ($Mean = 10$; $Med = 6.5$). In addition, we found that the average percent of followed hints in the experimental group ($Mean = 66.64\%$; $Med = 70\%$) was significantly higher than that in the control group ($Mean = 39.91\%$; $Med = 38.3\%$). We hypothesize a few possible implications from these results: First, the progress panel may have motivated students to seek out and follow hints in order to achieve the incomplete objectives it showed. Second, students in the experimental group might have followed more hints since the progress panel helps them to understand how hints relate to a specific objective. Lastly, students might have trusted the system more because the DD-IPF helped them see some intelligence and intentionality behind the system. This addresses a concern raised by prior work, which suggests that students do not trust automated feedback because they did not believe the system really understands what students are doing, or its ability to offer useful help [21].

7. RECOMMENDATIONS FOR DATA-DRIVEN PROGRAMMING FEEDBACK

Based on our current and prior work, we have several recommendations for designing data-driven feedback for programming [29, 17, 24]. Because of the very large, sparse solution spaces for programming, we should always expect to have

some inadequacies in any data-driven features or detectors for programs. For example, in our current work dataset, we found cases where the DD-IPF system incorrectly detected the completion of an objective and others where the student completed an objective, but it was not detected. These flaws point to a tradeoff between the more easily-generated, data-driven features used in this work and the expert-authored features that we used in our prior work [17]. A data-driven positive feedback system is likely to learn less generalizable features from prior students, and still needs to be labeled and curated for presentation, but it can be scaled to various programming exercises with less human effort. In contrast, an expert-authored feedback can be designed to have understandable and generalizable features, but it is hard to scale to various programming exercises, and requires extensive time and effort to create autograders (e.g. with static analysis [17, 28, 2]). In this work we combined both approaches, using data-driven feature detection to account for the diversity of student solutions, and a human labeling process to make the features more general, independent, and understandable.

We recommend that data-driven feedback approaches should be designed iteratively, learning and implementing an initial set of feature detectors from a given dataset, and iteratively detecting new features after each new dataset is added. For example, despite extracting features from over 100 prior student solutions to the *Squiral* programming exercise (which can be solved in 7-11 lines of code), in this study, we found two students who used a completely new strategy, not represented in the prior data. In particular, one of these two students created a procedure (i.e. a custom block) to draw a side of a Squiral, and called this procedure in an inner loop of the main procedure, and as a result, the DD-IPF system only detected the completion of two objectives that matched the features it learned from prior students' data, as shown in Figure 4. This behavior should be expected and planned for, since the process of providing automated data-driven feedback is inherently uncertain. To mitigate this, we propose to combine iterative cycles of data-driven feature detection with expert authoring to achieve the best of both worlds - building a system that can intelligently address the diverse but correct ways that students solve problems (fitting correct prior solutions), while benefiting from human expertise in communication (labeling the objectives). There is also a need to investigate how and when to communicate the fact that feature detection will always be imperfect for open-ended tasks in programming. We plan to explore ways to explain how the system works to future students, both to promote learning and to mitigate potential harms from incorrect feedback.

We also recommend learning features from student data in an offline, section-by-section fashion, grouping students who took the same course with the same instructor at the same time. Objective features should be reviewed and edited to be as independent as possible, and labeled so that students can understand what each one means. Note, however, that we do not believe that every feature detected must be fully explained by experts. Detectors should be trained on prior data, and cross validated with testing and training groups from different sections. This is because we do not anticipate being able to generate highly accurate data-driven feature

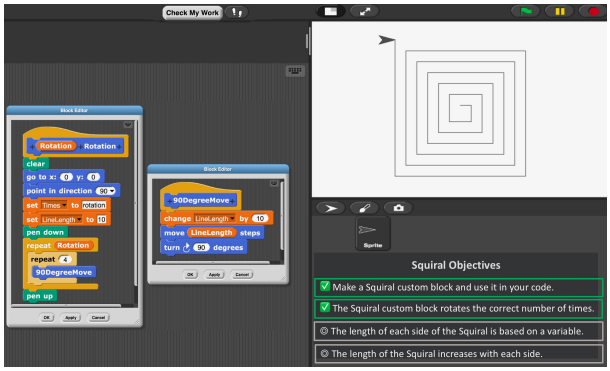

Figure 4: A novel student solution, with a new and correct, but undetected, strategy, where only 2 objectives were detected (see progress panel on the right).

detectors that could be added to the system without expert review. Therefore, validation should be conducted to match the nature of how our system is used in practice: trained on one dataset and used in a separate, later section. Furthermore, instructors can have a large impact on how students approach problems, so each section of a class is very likely to differ significantly due to that factor alone.

8. LIMITATIONS & CONCLUSION

This study has four primary limitations. First, this was a quasi-experimental study and therefore there might be other differences between semesters that affected our results. However, when we compared the students excluded from both semesters (due to deployment problems as mentioned in Section 5.1), we found that the time taken by the excluded students (the first 36% to complete or submit the programming task) in Spring 2019, as well as their scores, were very close to that of the excluded (first 36%) students in Spring 2020. This suggests that the differences we found in our study results were likely due to the DD-IPF system, rather than inherent differences between semesters. The second limitation is that, due to the structure of the programming course, we were not able to measure learning with pre/post tests. We hypothesize that the DD-IPF system can improve learning, since it provides immediate feedback that confirms that the programming steps a student just completed are those that contribute to the specific objective. The third limitation is that we evaluated the DD-IPF system on only one programming homework. The second and third limitations, however, are somewhat addressed in our prior work [17], making us optimistic that these limitations can be successfully addressed in future studies. Our prior work shows that, when compared to students in the control group with no positive feedback, students who used the expert-authored positive feedback system performed better on two tasks when they had access to the feedback, and continued to perform better in a third, more difficult task, without positive feedback [17]. Finally, we acknowledge that our DD-IPF system includes other features than positive feedback: it breaks down the programming task into smaller objectives and provides corrective feedback when objectives are broken. Our study presents the results of the whole system, but we argue that the most salient aspect of the

system was its focus on providing immediate positive feedback during problem solving. In our future work, we hope to conduct a larger-scale study with different treatment groups to evaluate individual features of the DD-IPF system.

To conclude, we developed a system that combines a data-driven feature detector with human labelling to provide data-driven immediate positive feedback (DD-IPF) in a block-based programming environment. We conducted a quasi-experimental classroom study to evaluate the impact of DD-IPF on students while programming homework. We found evidence that the DD-IPF system increased students' engagement with the programming task, and it has the potential to improve students' programming performance. We also provided recommendations to the computing education researchers on how to design better data-driven feedback systems. In our future work, we plan to improve the accuracy of the DD-IPF system, test it across several programming exercises, and evaluate its impact on students' cognitive and affective outcomes. We also plan to research ways to counteract the inadequacies of automated feedback with interface design and opportunities for self-explanation prompts to promote user trust and learning.

9. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under grant 1623470. The authors would also like to thank Preya Shabrina for her help on data-analysis.

10. REFERENCES

[1] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207, 1995.

[2] M. Ball. Lambda: An autograder for snap. *Masterscriptie. EECS Department, University of California, Berkeley*, 2018.

[3] D. Barrow, A. Mitrovic, S. Ohlsson, and M. Grimley. Assessing the impact of positive feedback in constraint-based tutors. In *International Conference on Intelligent Tutoring Systems*, pages 250–259. Springer, 2008.

[4] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney. Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 26(2-3):148–175, 2016.

[5] K. E. Boyer, R. Phillips, M. D. Wallis, M. A. Vouk, and J. C. Lester. Learner characteristics and feedback in tutorial dialogue. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, pages 53–61. Association for Computational Linguistics, 2008.

[6] W. L. Cade, J. L. Copeland, N. K. Person, and S. K. D'Mello. Dialogue modes in expert tutoring. In *International Conference on Intelligent Tutoring Systems*, pages 470–479. Springer, 2008.

[7] L. Chen, B. Di Eugenio, D. Fossati, S. Ohlsson, and D. Cosejo. Exploring effective dialogue act sequences in one-on-one computer science tutoring dialogues. In *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 65–75. Association for Computational Linguistics, 2011.

[8] A. Corbett and J. R. Anderson. Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes. In *Proceedings of the SIGCHI Conference on Human Computer Interaction*, pages 245–252, 2001.

[9] B. Di Eugenio, D. Fossati, S. Haller, D. Yu, and M. Glass. Be brief, and they shall learn: Generating concise language feedback for a computer tutor. *International Journal of Artificial Intelligence in Education*, 18(4):317–345, 2008.

[10] B. Di Eugenio, D. Fossati, S. Ohlsson, and D. Cosejo. Towards explaining effective tutorial dialogues. In *Annual Meeting of the Cognitive Science Society*, pages 1430–1435, 2009.

[11] Y. Dong, S. Marwan, V. Catete, T. Price, and T. Barnes. Defining tinkering behavior in open-ended block-based programming assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1204–1210. ACM, 2019.

[12] D. Fossati, B. Di Eugenio, S. Ohlsson, C. Brown, and L. Chen. Data driven automatic feedback generation in the ilist intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, 10(1):5–26, 2015.

[13] D. Fossati, B. Di Eugenio, S. Ohlsson, C. W. Brown, L. Chen, D. G. Cosejo, et al. I learn from you, you learn from me: How to make ilist learn from students. In *AIED*, pages 491–498, 2009.

[14] D. E. Johnson. Itch: Individual testing of computer homework for scratch assignments. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 223–227. ACM, 2016.

[15] M. J. Lee and A. J. Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*, pages 109–116. ACM, 2011.

[16] M. R. Lepper, M. Woolverton, D. L. Mumme, and J. Gurtner. Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. *Computers as cognitive tools*, 1993:75–105, 1993.

[17] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the International Computing Education Research Conference (forthcoming)*, 2020.

[18] A. Mitrovic, S. Ohlsson, and D. K. Barrow. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education*, 60(1):264–272, 2013.

[19] R. Moreno and R. E. Mayer. Personalized messages that promote science learning in virtual environments. *Journal of educational Psychology*, 96(1):165, 2004.

[20] T. W. Price, Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, 2017.

[21] T. W. Price, Z. Liu, V. Catete, and T. Barnes. Factors Influencing Students' Help-Seeking Behavior while

Programming with Human and Computer Tutors. In *Proceedings of the International Computing Education Research Conference*, 2017.

[22] P. C. Rigby and S. Thompson. Study of novice programmers using eclipse and gild. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 105–109. ACM, 2005.

[23] K. Rivers and K. R. Koedinger. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017.

[24] P. Shabrina, S. Marwan, T. W. Price, M. Chi, and T. Barnes. The impact of data-driven positive programming feedback: When it helps, what happens when it goes wrong, and how students respond. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'20*, 2020.

[25] D. Sleeman, A. E. Kelly, R. Martinak, R. D. Ward, and J. L. Moore. Studies of diagnosis and remediation with high school algebra students. *Cognitive Science*, 13(4):551–568, 1989.

[26] K. Vanlehn. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, 16:227–265, 08 2006.

[27] K. VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

[28] W. Wang, R. Zhi, A. Milliken, N. Lytle, and T. W. Price. Crescendo : Engaging Students to Self-Paced Programming Practices. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, 2020.

[29] R. Zhi, T. W. Price, N. Lytle, and T. Barnes. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'18*, 2018.