Exploring Novice Programmers' Hint Requests in an Intelligent Block-Based Coding Environment

Joseph B. Wiggins[1], Fahmid M. Fahid[2], Andrew Emerson[2], Madeline Hinckle[2], Andy Smith[2], Kristy Elizabeth Boyer[1], Bradford Mott[2], Eric Wiebe[2], and James Lester[2]
jbwiggi3@ufl.edu,ffahid@ncsu.edu,ajemerso@ncsu.edu,mthinckl@ncsu.edu,pmsmith4@ncsu.edu,
keboyer@ufl.edu,bwmott@ncsu.edu,wiebe@ncsu.edu,lester@ncsu.edu
[1]University of Florida, Gainesville, Florida
[2]North Carolina State University, Raleigh, North Carolina

ABSTRACT

Block-based programming environments are widely used by novices who are learning computer science. However, even in block-based coding environments that have been carefully developed to serve novices, students frequently struggle and require additional support. A promising avenue to provide this support is the use of intelligent tutoring systems, which offer adaptive hints to assist learners. In order to provide students with the adaptive hints they need, we must investigate their help-seeking behaviors and identify patterns surrounding their need for support. In this experience report, we examine data collected from 174 college students in an introductory engineering course, who used an intelligent block-based coding environment to learn computer science. These students made more than 1,000 hint requests, which we represent in two-dimensional space along axes of elapsed time and code completeness. Analysis revealed five major clusters of hint requests, which we further characterized through qualitative examination of the coding trajectories that preceded each hint request. We also analyzed how students' incoming knowledge and perceived computer skill were related to their help-seeking behaviors. Students with higher incoming knowledge requested hints when their code was more complete than students with lower incoming knowledge. Students with high perceived computer skill asked for hints when their code was less complete than those with low perceived computer skill. The results presented here provide insight into student help-seeking behavior in computer science education, informing CS educators and system designers on how best to develop support strategies.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Applied computing** → **Education**.

KEYWORDS

Help-seeking behavior; CS1; intelligent tutoring system

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432538

ACM Reference Format:
Joseph B. Wiggins[1], Fahmid M. Fahid[2], Andrew Emerson[2], Madeline Hinckle[2], Andy Smith[2], Kristy Elizabeth Boyer[1], Bradford Mott[2], Eric Wiebe[2], and James Lester[2]. 2021. Exploring Novice Programmers' Hint Requests in an Intelligent Block-Based Coding Environment. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432538

1 INTRODUCTION

Novice programmers have historically struggled in introductory computer science courses and suffer from high attrition rates [3]. In recent years, block-based programming languages have become frequently used in introductory programming courses because they eliminate many of the common challenges novice programmers encounter [31]. The use of block-based programming languages in introductory courses has many benefits, such as reduced cognitive load [33], improved understanding of programming structure [32], and increased efficiency while completing programming tasks [24].

Despite the advantages that block-based environments offer, learning to program presents novices with significant challenges [20]. An important open area of research involves how to best provide support, such as hints, to novice programmers [5]. Utilizing instructional strategies such as scaffolding, which slowly reduces student assistance as mastery is achieved, has been shown to improve metacognitive skills and increase learning gains [15, 26]. Several decades of work in the intelligent tutoring systems research field provide a basis for investigating this open area by showing, for example, the importance of timely feedback [2], the prevalence of students abusing hinting systems to get answers without having to invest in the task [29], the importance of help seeking in self-regulated learning [1], and how intelligent tutoring systems can be integrated into block-based programming environments [25].

In this experience report, we present a data-driven exploration of novice students' hint requests using a block-based learning environment for undergraduate introductory computer science. We clustered students' hint requests in two-dimensional space along axes of elapsed time (since the student began coding) and code completeness (which we calculate by evaluating the abstract syntax tree similarity between the student's code and expert-authored code) at the moment they requested the hint. Quantitative analysis of the hint requests revealed there were relationships between students' CS concept knowledge, perception of their computer skills, and hint-requesting behavior. Qualitative analysis of the resulting clusters revealed patterns in the code completeness, knowledge level,

and perceived computer skill of students who were likely to request hints in those clusters. For each cluster of hint requests, we discuss approaches to provide further support in those contexts. This work can inform CS educators and system designers on patterns in students' hint requests and suggestions for support strategies that would improve novice programmers' experiences.

2 RELATED WORK

A growing body of research has investigated the impact of different types of support during programming for novices. Price et al. [25] created a block-based programming environment, iSnap, that automatically generates contextual hints for students based on their detailed action logs. The authors found that most students were willing to use hints and that hints can lead to positive learning outcomes. Zhi et al. [34] compared instances of when a student received worked examples in their programming tasks, showing that worked examples do not result in significant learning improvements but can help reduce cognitive load and time to solve programming problems. In addition to reducing time spent on problems, support in block-based programming keeps students engaged [30].

Automated feedback techniques are increasingly being utilized to assist teachers in large class settings. A primary goal of automated feedback is to give insightful hints that then result in fewer errors and misconceptions [12]. To properly address errors and misconceptions, it is important to first identify students in need of help. Recent studies have identified students that are at-risk or struggling through programming trajectory analysis [7], early prediction models such as neural networks [4], and keystroke-based models of student performance [21].

Previous work has also aimed to improve existing automated feedback systems by addressing errors in student code that prevents expert-designed test cases from being executed [22]. Several other studies have aimed to assist teachers by effectively grouping student programs to facilitate feedback. For example, subgraph matching techniques have been used to match natural language feedback to code in introductory programming MOOCs [17]. Creating effective automated hinting systems for block-based programming environments is still an active area of research. For example, a recent study by Leite et al. [16] compared machine-generated feedback to human-authored feedback, showing that human feedback helped students obtain better conceptual understanding of programming concepts and improved student performance on class exams. This work illustrates that more research is needed to improve machine-generated feedback mechanisms.

As more research in block-based programming languages is being conducted, large volumes of data are being generated which create opportunities to observe novel patterns. Data mining techniques, such as cluster analysis, can provide a way to visually inspect student experiences to generate more relevant feedback [9]. Similarly, clustering can be used to group common student misconceptions to better understand where students struggle in both general block-based programming and specific programming activities [6, 13]. The work presented in this paper uses cluster analysis to focus on students' help-seeking behaviors while coding.

Code State	Hint
No addition of user inputs	"To increase the current value of a variable, you can use the math operation block with a set variable block"
Same variable is being used to receive input twice	"If you try to read user input again, the old input value will be overwritten. Store the user's input in a separate variable to keep track of what the user has entered"
No subtraction in countdown code	"Make sure you are decreasing the countdown inside of the repeat loop"

Table 1: Sample hints and the conditions that generate them.

3 INTELLIGENT BLOCK-BASED LEARNING ENVIRONMENT

We have been developing an adaptive block-based programming environment to support novice programmers in their learning of introductory computer science. The goal of this system is to serve both as a tool that educators can use during the first few weeks of their introductory programming courses and, alternatively, as a standalone environment that self-directed learners can utilize to learn programming concepts.

In the system, students build their programs using a customized version of Google's Blockly block-based programming plugin [8]. Learners proceed through twenty programming activities, where each successive activity builds on the skills learned from the previous one. These activities are organized into three units: 1) Environment tutorial, Input/Output, Numeric data types, Expressions, Variables, Iteration; 2) Abstraction, Functions, Parameters; 3) Boolean data types, Conditionals, Indefinite iteration, Debugging. This experience report focuses on three programming activities from Unit 1 that introduce students to loops. The first activity has the students gathering five numbers and then displaying the sum of those numbers, the second activity has the students introducing a "repeat" block (a block that repeats a set of code some constant amount of iterations), and finally the third has the students count down to zero from a number entered by the student. We focus on these activities because they are sufficiently challenging that many students requested hints while attempting the programming activities.

3.1 Intelligent Hinting System

The intelligent hinting system is designed with a fixed progression of programming activities and a button to request hints from the system. As shown in Figure 1, the hint button is in the upper right corner of the web interface, accessible to the student at any time. Each time that button is pressed, the system compares the student's code state to a set of expert-authored regular expressions which were refined over several iterations, and then delivers a hint to the student. Hints were either a suggestion for a next step ("Now that there's a number stored in the variable, see what happens when you print the variable") or extra details about a block that was relevant to the current task ("The repeat block lets you take a segment of code and repeat it as many times as you want"). For some examples of hints, see Table 1.

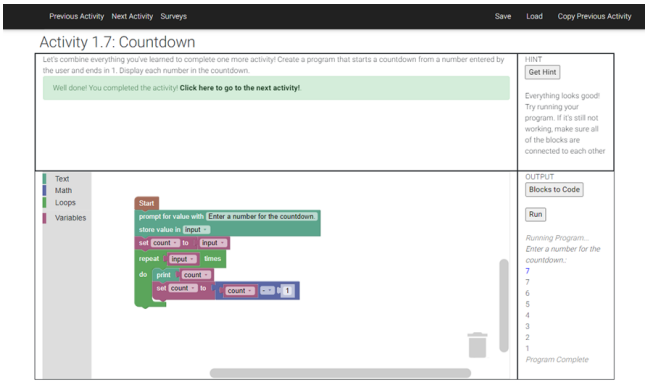

Figure 1: The block-based learning environment interface is divided into four sections: the programming activity description in the top left, the coding pane in the bottom left, the hint request button in the top right, and the output in the bottom right.

3.2 Study Design

We conducted the study using the system at a large public university in the United States in Fall 2019. The participants in this IRB-approved study were from one of two online sections for an introductory engineering course and received extra credit for completing the study. The introductory engineering course reviewed some computing concepts such as file systems and html, but did not cover any programming concepts. A total of 342 consenting participants logged on to the system, and of those, 174 participants attempted or completed at least one of the three activities. These participants had an average age of 18.76, with 34.29% of them reporting their gender as female and 65.71% as male. Students reported majoring in primarily non-CS such as engineering disciplines, math, agricultural science, or undecided (79.43%); computer science (20.57%). Of these students, 71.26% reported their ethnicity as White, 14.94% as Asian, 4.02% as African American, 4.02% as Hispanic and 3.35% as Other. The rest did not report their ethnicity (2.41%).

3.3 Survey Data

We collected measurements of students' incoming knowledge of computer science through block-based test questions, and their perceived computer skill. The concepts in the pretest were mapped to computer science skills, knowledge, and abilities (FKSAs) present in the exercises, using a process similar to Grover and Basu [10, 11]. We assessed perceived computer skill by asking the following question: How skilled are you with computers, compared to the average person? (much more skilled, somewhat more skilled, average, somewhat less skilled, much less skilled). Students had an average pretest score of 55.75 out of 100 (standard deviation 20.22) and an average perceived computer skill of 3.38 out of 5 (standard deviation 0.86).

3.4 Hint-Related Metrics

The students in this study made a total of 1,063 hint requests during the three programming activities. To characterize their help-seeking behaviors, we represented each hint request as a two-dimensional point along axes of elapsed time and code completeness, as we now detail.

Elapsed time indicates the number of seconds since the student started the programming activity. The calculation of elapsed time in a web-based environment can be complicated due to the following behaviors: students may shuffle between activities (likely checking the previous programming activity descriptions), and students may leave the web interface open in a tab for long periods of time without working on the programming activity. In the elapsed time calculation, we do not include brief periods of time in which no other actions were taken. To correct for students leaving the browser open for a long time without progress, we first calculated the median time between student actions in the block-based programming interface. This median was 29.96 seconds. For any gap longer than 5 minutes between actions, we replaced those times with the median gap time. The 5-minute threshold was selected based on the 3rd quartile value of gaps between actions in the interface (30.06 seconds) because visually inspecting the data revealed that time gaps above the 3rd quartile became much longer and were likely associated with the student no longer working in the interface for that time. This heuristic approach has a limitation in that if a student were still actively considering the task but not taking any actions, we could artificially decrease their elapsed time.

The second characteristic that we calculated for each hint request was code completeness. This was calculated by first converting the student's block-based code to standard Python code and then measuring the abstract syntax tree (AST) similarity (Damerau–Levenshtein distance) with an expert-authored solution using the analysis package pyastsim[1]. This measure allows for the comparison of the code between different moments in time, and trends in the trajectory of the code to be observed (e.g. the code is getting closer to the solution over time).

4 RESULTS

The goal of this experience report is to analyze patterns in student help-seeking behaviors to understand commonalities in the conditions under which they ask for hints. To accomplish this, we first compared those students who did and did not ask for hints during the three programming activities. After uncovering features that distinguish hint requesters from non-hint requesters, we conducted a cluster analysis on only the hint requesters.

4.1 How Do Hint Requesters Differ from Non-Hint Requesters?

During the completion of the three programming activities, many students requested hints from the system. Those requests are the focal point of this paper, but before we excluded students who did not ask for help, we first analyzed how these groups differed. Table 2 highlights some differences in incoming characteristics of each group.

There were significant differences ($p < 0.05$) in pretest scores between hint requesters and non-hint requesters using a two-tailed t-test, with hint requesters having lower pretest scores. Similarly, the hint requesters also reported significantly lower perceived computer skills than those students who did not ask for hints. We found

[1] https://pypi.org/project/pyastsim/

	Hint Requesters (n = 130)	Non-Hint Requesters (n = 44)	p-Value
Pretest Scores (out of 100)	53.37 (19.30)	65.78 (20.00)	<0.001
Perceived Computer Skill (out of 5)	3.27 (0.87)	3.72 (0.75)	0.003

Table 2: Comparison of students who requested hints and those who did not (p-Value calculated using a two-tailed t-test).

Features	Correlation	p-Value
Perceived Computer Skill & Code Completeness at Hint Request Time	-0.1887	0.0316
Pretest Score & Code Completeness at Hint Request Time	0.2781	0.0014
Pretest Score & Computer Skill	-0.2057	0.0189

Table 3: Correlations between students incoming characteristics and their average hint requesting metrics (p-Value calculated using Pearson's R).

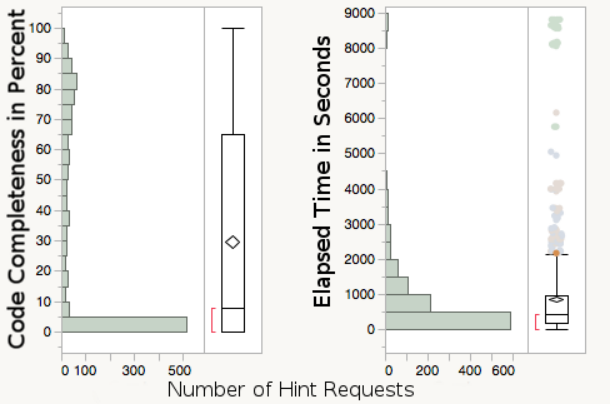

Figure 2: Histogram and box plot of code completeness (left) and elapsed time (right) for the requested hints.

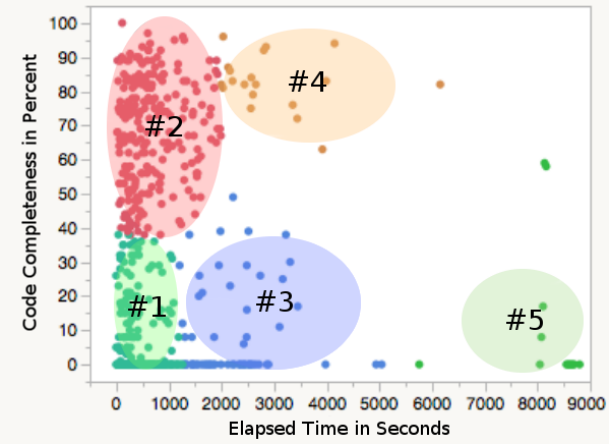

Figure 3: Clusters of hint asking behavior.

no demographic differences between these groups related to gender or ethnicity.

4.2 How Do Student Characteristics Correlate with Their Hint-Requesting Behaviors?

This analysis focuses on the two metrics calculated for each hint request: elapsed time and code completeness. In the cluster analysis we will consider these as two dimensions that represent an instance of help-seeking. Before we review their two dimensional distributions, we examine each metrics' individual distributions (Figure 2).

As shown in Figure 2, students most frequently asked for help early in their coding session, before making substantial progress toward the solution. It is also important to consider how these metrics are influenced by student pretest score and perceived computer skill. Table 3 displays the correlations between pretest scores, perceived computer skill, elapsed time, and code completeness. To avoid overrepresentation of students who asked for more hints, we averaged the elapsed times of each individual student's set of hint requests and averaged the code completeness level across all of that student's hint requests.

Pearson's R revealed significant correlations ($p < 0.05$) between pretest score, perceived computer skill, and the code completeness at the time of the hint requests. Perceived computer skill had a negative correlation with the code completeness at the time of the hint requests. In other words, students with lower perceived computer skill completed more code before requesting a hint than students with higher perceived computer skill. Pretest score had a positive correlation with code completeness; in other words, students with higher pretest scores completed more code before asking for a hint.

4.3 What Clusters of Hint Asking Behavior Arise?

To cluster similar hint requests, we used K -means clustering in the JMP package[2]. K-means clustering partitions data points into k clusters by optimizing a metric of cluster separation (distance between different clusters) and cohesion (points within one cluster being close to each other). We utilized cubic clustering criterion as this metric [27]. The ideal k is often not known a priori. We explored k=3, 4, 5, and 6, and found that five clusters resulted in the optimal clustering based on the clustering criterion. The five clusters of hints are shown in Figure 3.

4.3.1 Cluster 1 - Where Do I Start? The cluster containing the largest number of hint requests is characterized by a code completeness score of 4%, meaning that the code had a very low similarity to functional code and was most likely only one block. This cluster's typical hint request had an elapsed time of 352 seconds, which is only a few minutes after the start of the programming activity. Figure 4 displays a typical code trajectory preceding a hint request

[2]https://www.jmp.com/en_us/offers/statistical-analysis-software.html

in this cluster, in which only one to two blocks were added before turning to the hint button. These hint requests may represent a "Where Do I Start?" attitude toward utilizing the hint feature, as we further discuss in Section 5.

4.3.2 Cluster 2 - Does This Look Right? The second largest cluster of hint requests is characterized by a code completeness score of 70% and an elapsed time of 679 seconds. The requests for help in this cluster were representative of students who were fairly close to completion in the first 11 minutes. Figure 4 displays a typical code trajectory preceding a hint request in this cluster. The code displayed in the graph started improving within the first 250 seconds and had a fairly steady progression upwards. These hint requests may represent a "Does This Look Right?" mindset, as we further discuss in Section 5.

4.3.3 Cluster 3 - I Think I'm Missing Something. The third largest cluster of hint requests is characterized by a code completeness score of 8% and an elapsed time of 2142 seconds. This cluster seemed to have experienced difficulty getting started on the programming activity or had attempted a few solutions and later deleted their code. Figure 4 shows a typical code trajectory preceding a hint request in this cluster. The student has managed to make some progress towards the goal and made a step backwards. They also have been frequently asking for hints up until the hint request. These hint requests may represent an "I Think I'm Missing Something" mindset, as we further discuss in Section 5.

4.3.4 Cluster 4 - This Time For Sure… Right? The second smallest cluster of hint requests is characterized by a code completeness score of 84% and an elapsed time of 3330 seconds. Figure 4 displays a typical code trajectory that leads to hint requests in this cluster. This sample displays a slow progression followed by a plateau that eventually results in a request for help. This cluster is very similar to Cluster 2, with the programming activity being engaged with for a longer period of time. These hint requests may represent a "This Time For Sure… Right?" mindset, as we further discuss in Section 5.

4.3.5 Cluster 5 - I'm Out of Ideas. Help, Please. The final, and smallest, cluster of hint requests is characterized by a code completeness score of 11% and an elapsed time of 8193 seconds. This cluster is characterized by frequent disengagement from the activity, sometimes leaving the activity open for 14 hours at a time (although these actions were not considered in the calculation of elapsed time, as described in Section 3.4). These behaviors seem to have been related to this cluster getting stuck on the programming activity because of prolonged confusion leading to disengagement with the system. Figure 4 displays a typical code trajectory that leads to hint requests in this cluster. As shown in Figure 4, hint requests in this cluster also frequently asked for help in clusters 1 and 3. These students had started many different solutions and ultimately restarted their code several times. These hint requests may represent a "I'm Out Of Ideas. Help, Please." mindset, as we further discuss in Section 5.

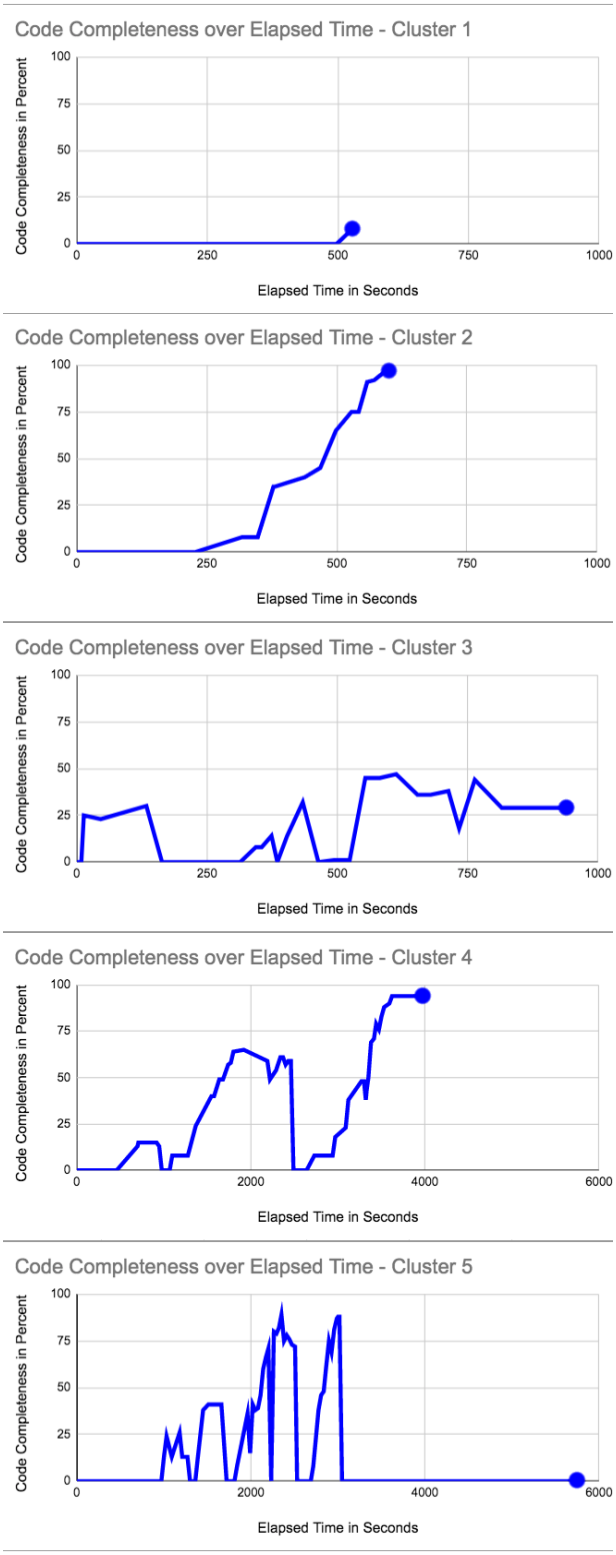

Figure 4: Sample student code trajectory preceding a hint in clusters 1, 2, 3, 4, and 5.

5 DISCUSSION

The results reveal that incoming knowledge and perceived computer skill were correlated with code completeness at the time of the hint: students with higher incoming knowledge completed more code before asking for hints. Students with higher perceived computer skill asked for hints when their code was less complete. Cluster analysis revealed five distinct clusters of hint requests, each characterized by differing levels of code completeness and elapsed time.

Cluster 1 - Where Do I Start? The largest cluster of hint requests (562 hints) happened early in the programming activity and with a very low level of code completeness. This behavior likely indicates students not knowing how to start the activity. Redirecting the students towards the programming activity description to help them identify phrases that they should attempt to translate into code first may be an effective first response [23]. Providing more granular instructions or even code starters to help them overcome these early hurdles to progress through the exercise [14] have both been pointed out in the literature as effective approaches.

Cluster 2 - Does This Look Right? The second largest cluster (359 hints) was typified by a slightly larger elapsed time than the first cluster, but at a much higher code completeness level of 70%. The literature suggests that rather than highly directive "next-step" hints, feedback after significant progress could focus on statements affirming students' progress with positive feedback and prompting them with a reflective question about their code [19, 28].

Cluster 3 - I Think I'm Missing Something. The third largest cluster (86 hints) occurred much later in elapsed time than the prior two clusters but had a typical code completeness score of only slightly better than the first cluster, at 8%. Feedback after students restart their code could highlight the successful elements in their previous attempts and provide help based on what they had previously done correctly, rather than starting from the beginning as may have been more helpful for Cluster 1.

Cluster 4 - This Time For Sure... Right? The second smallest cluster (38 hints) occurred slightly later than Cluster 3, and at the highest code completeness of any cluster, at 84%. The sample code trajectory displays the code completeness gradually increasing over time until a prolonged plateau preceding the hint request. When a hint is requested in this cluster, the student may be more likely to be frustrated by more reflective prompts, and may respond better to more direct feedback. A strategy that has shown to improve learning in some students is to have the tutor deliver some extra educational content and then question the student [18]. As these students are close to the solution, more analysis identifying the specific states students are getting stuck on will be important for generating feedback to effectively overcome their specific issues.

Cluster 5 - I'm Out of Ideas. Help, Please. The smallest cluster (19 hints) had the lowest code completeness (11%) and almost three times longer elapsed time than Cluster 4. To respond to hint requests at this stage, the literature would suggest using worked examples and bottom-out hints [34] since the requests likely stem from being stuck and having immense difficulty completing the programming activity. This hinting strategy has been shown to reduce cognitive load and help students finish their code, but is unlikely to result in learning gain [34]. Ideally, a system would be capable of proactively intervening before a student reaches this point, whereas the current system could only give hints when requested.

Limitations. A limitation of this analysis and data set is that the help requesters are self-selecting, although this is unavoidable to study the natural occurrence of help-seeking behaviors. It is also important to consider that marginalized groups in computing may rate their skills and confidence lower than their peers and this could impact results surrounding perceived computer skill. Students in our sample were also not graded on the number of completed activities, so not all students completed the first activity represented in this work, meaning our sample may not contain some of the students with the highest need for support. Another consideration is the well-documented phenomenon of hint abuse and hint refusal [29], both of which may skew this data and not make it representative of all students.

6 CONCLUSION

Intelligent tutoring systems designed for computer science learning can utilize a wide range of approaches for providing hints to students. Investigating when, and under what conditions, students request hints in an intelligent tutoring system allows for prioritization in what kinds of feedback should be focused on. In order to provide high-quality hints that will be useful for students, it is critical to develop a nuanced understanding of students' help-seeking behavior.

The goal of this analysis was to discover patterns in student help-seeking behaviors as they interacted with the intelligent block-based coding environment. From the dataset of 174 students, 130 of them requested at least one hint from the system during the programming activities. By considering both code trajectories and elapsed time on programming activity, these 1,063 hint requests were represented in two-dimensional space by five clusters. Based on empirical attributes of the hint requests, as well as qualitative analyses of code trajectories preceding the hint request, this paper discussed recommendations for how systems or instructors could effectively tailor support.

Future work should focus on implementing and evaluating context-sensitive hint strategies that take elapsed time and code completeness into account. Additionally, we should consider the interplay of student characteristics and help-seeking behaviors. In this work, incoming knowledge and perceived computer skill were shown to differ significantly between hint requester and non-hint requester, and have significant correlations with code completeness at the time of each hint request. These characteristics, along with others, likely play a more nuanced role in how hints are requested and overall system interaction. This line of investigation may benefit educators and system designers by providing insights on how to build better hints, and how to more effectively build automated help systems.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation through the grants DUE-1626235 and DUE-1625908. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Vincent Aleven, Ido Roll, Bruce M McLaren, and Kenneth R Koedinger. 2016. Help helps, but only so much: Research on help seeking with intelligent tutoring systems. *International Journal of Artificial Intelligence in Education* 26, 1 (2016), 205–223.

[2] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. 1995. Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences* 4, 2 (1995), 167–207.

[3] Theresa Beaubouef and John Mason. 2005. Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bulletin* 37, 2 (2005), 103–106.

[4] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 111–116.

[5] Paul Denny, Brett A Becker, Michelle Craig, Greg Wilson, and Piotr Banaszkiewicz. 2019. Research this! Questions that computing educators most want computing education researchers to answer. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 259–267.

[6] Andrew Emerson, Andy Smith, Fernando J Rodríguez, Eric N Wiebe, Bradford W Mott, Kristy Elizabeth Boyer, and James C Lester. 2020. Cluster-based analysis of novice coding misconceptions in block-based programming. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 825–831.

[7] Anthony Estey, Hieke Keuning, and Yvonne Coady. 2017. Automatically classifying students in need of support by detecting changes in programming behaviour. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 189–194.

[8] Neil Fraser et al. 2013. Blockly: A visual programming editor. *URL: https://code. google. com/p/blockly* 42 (2013).

[9] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 1–35.

[10] Shuchi Grover. 2020. Designing an assessment for introductory programming concepts in middle school computer science. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 678–684.

[11] Shuchi Grover and Satabdi Basu. 2017. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 267–272.

[12] Georgiana Haldeman, Andrew Tjang, Monica Babeş-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D Nguyen. 2018. Providing meaningful feedback for autograding of programming assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 278–283.

[13] David Joyner, Ryan Arrison, Mehnaz Ruksana, Evi Salguero, Zida Wang, Ben Wellington, and Kevin Yin. 2019. From clusters to content: Using code clustering for course improvement. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 780–786.

[14] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2019. How teachers would help students to improve their code. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. 119–125.

[15] Joo Yeun Kim and Kyu Yon Lim. 2019. Promoting learning in online, ill-structured problem solving: The effects of scaffolding type and metacognition level. *Computers & Education* 138 (2019), 116–129.

[16] Abe Leite and Saúl A Blanco. 2020. Effects of human vs. automatic feedback on students' understanding of AI concepts and programming style. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 44–50.

[17] Victor J Marin, Tobin Pereira, Srinivas Sridharan, and Carlos R Rivero. 2017. Automated personalized feedback in introductory Java programming MOOCs. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1259–1270.

[18] Christopher M. Mitchell, Eun Young Ha, Kristy Elizabeth Boyer, and James C. Lester. 2013. Learner characteristics and dialogue: Recognizing effective and student-adaptive tutorial strategies. *International Journal of Learning Technology (IJLT)* 8, 4 (2013), 382–403.

[19] Antonija Mitrovic, Stellan Ohlsson, and Devon K Barrow. 2013. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education* 60, 1 (2013), 264–272.

[20] Jan Moons and Carlos De Backer. 2013. The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education* 60, 1 (2013), 368–384.

[21] Jonathan P Munson and Joshua P Zitovsky. 2018. Models for early identification of struggling novice programmers. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 699–704.

[22] Sagar Parihar, Ziyaan Dadachanji, Praveen Kumar Singh, Rajdeep Das, Amey Karkare, and Arnab Bhattacharya. 2017. Automatic grading and feedback using program repair for introductory programming courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 92–97.

[23] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 531–537.

[24] Thomas W Price and Tiffany Barnes. 2015. Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. 91–99.

[25] Thomas W Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards intelligent tutoring in novice programming environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 483–488.

[26] Siti Nurulain Mohd Rum and Maizatul Akmar Ismail. 2017. Metocognitive support accelerates computer assisted learning for novice programmers. *Journal of Educational Technology & Society* 20, 3 (2017), 170–181.

[27] Warren S Sarle. 1983. *Cubic clustering criterion*. SAS Institute.

[28] Alexandria Katarina Vail and Kristy Elizabeth Boyer. 2014. Identifying effective moves in tutoring: On the refinement of dialogue act annotation schemes. In *International Conference on Intelligent Tutoring Systems*. Springer, 199–209.

[29] Jason A Walonoski and Neil T Heffernan. 2006. Detection and analysis of off-task gaming behavior in intelligent tutoring systems. In *International Conference on Intelligent Tutoring Systems*. Springer, 382–391.

[30] Wengran Wang, Rui Zhi, Alexandra Milliken, Nicholas Lytle, and Thomas W Price. 2020. Crescendo: Engaging students to self-paced programming practices. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 859–865.

[31] Christopher Watson and Frederick WB Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. 39–44.

[32] David Weintrop and Uri Wilensky. 2015. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs.. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, Vol. 15. 101–110.

[33] Benjamin Xie and Hal Abelson. 2016. Skill progression in MIT app inventor. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 213–217.

[34] Rui Zhi, Thomas W Price, Samiha Marwan, Alexandra Milliken, Tiffany Barnes, and Min Chi. 2019. Exploring the impact of worked examples in a novice programming environment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 98–104.