Ordered Network Analysis in CS Education: Unveiling Patterns of Success and Struggle in Automated Programming Assessment

Andres Felipe Zambrano†
Graduate School of Education
University of Pennsylvania
Philadelphia, PA, United States
afzambrano97@gmail.com

Maciej Pankiewicz
Institute of Information Technology
Warsaw University of Life Sciences
Warsaw, Poland
maciekpankiewicz@gmail.com

Amanda Barany
Graduate School of Education
University of Pennsylvania
Philadelphia, PA, United States
amanda.barany@gmail.com

Ryan S. Baker
Graduate School of Education
University of Pennsylvania
Philadelphia, PA, United States
ryanshaunbaker@gmail.com

ABSTRACT

Computer science (CS) education at the university level is often challenging, particularly for students with no prior programming experience. To help scaffold students' CS learning, instructors often utilize systems for automated assessment of programming assignments, where students can individually learn online using automatically generated feedback. However, despite the growing usage of these systems, learning outcomes are often mixed and not all students benefit equally from using these applications. In this study, we utilize Ordered Network Analysis (ONA) to examine data from a system for automated assessment of programming assignments and compare platform activity between novice students (N=110) achieving high (N=43) and low (N=67) scores on the final test of an introductory CS course. We identify and visualize differences in the activity patterns between the groups. High performing novice students tend to request feedback more often, while low performing students more often leave the assignment unsolved after experiencing an unsuccessful attempt. These findings show that Ordered Network Analysis can serve as a useful tool for understanding student behaviors, facilitating the design of targeted interventions that might support learners at key moments in their programming engagement towards task success.

†Corresponding Author

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07
DOI: https://doi.org/10.1145/3649217.3653613

CCS CONCEPTS

• Applied computing → Education → Interactive learning environments.

KEYWORDS

Computer science education, Programming, Automated assessment, Ordered network analysis, Digital learning platforms.

ACM Reference format:

Andres Felipe Zambrano, Maciej Pankiewicz, Amanda Barany, and Ryan S. Baker. 2024. Ordered Network Analysis in CS Education: Unveiling Patterns of Success and Struggle in Automated Programming Assessment. In *Proceedings of the 29th annual ACM conference on Innovation and Technology in Computer Science Education (ITiCSE'24), July 8-10, 2024, Milan, Italy. ACM, New York, NY, USA, 7 pages.* https://doi.org/10.1145/3649217.3653613

1 Introduction

In recent years, increasing recognition of the relevance of Computer Science (CS) skills has resulted in global initiatives to expand and refine CS Education [9, 23]. In pursuit of this goal, CS Education research and practice has long included the use of automated assessment tools for both the improvement of grading efficiency and the design of timely and personalized feedback that can support programming success when repeated attempts are necessary [5, 10, 11]. While summative evaluations of automated programming assessments have broadly demonstrated their utility and efficacy for learning (e.g., [3, 10]) such benefits are often inconsistent in practice based on the specific actions taken or strategies enacted by novice learners. For example, behavioral self-regulation skills [4, 8, 19] and metacognitive awareness of the steps needed to complete a problem [13, 14] have been identified as key

mediators of ultimate success. Students who can successfully apply such skills often demonstrate markedly different performance from those who cannot. Ultimately, the ways in which such processes are enacted remains relatively underexplored in CS education, suggesting the need for research on the differences in activity patterns associated with novices who struggle or ultimately succeed in a programming task.

To address this need, this work leverages Ordered Network Analysis (ONA) [21] to examine data from a system for automated assessment of programming assignments and compare platform activity between novice students in an introductory CS course who achieved high and low scores on the final test. Specifically, our research question is: What are the differences in the patterns of programming actions made by high and low achieving students using an automated assessment programming assignment platform?

1.1 Epistemic Network Analysis

Epistemic Network Analysis (ENA) is a methodological framework used for identifying and quantifying connections within coded qualitative data, visually representing the strength of associations between the codes as they appear across a discourse using network models [18]. A growing number of researchers have employed ENA in recent years to visualize and compare the relationships between variables in complex phenomena such as self-regulation (e.g., [15, 25]), affect (e.g., [6]) and collaboration practices (e.g., [27]). The structure of epistemic networks also supports visual and statistical comparisons of network patterns across defined groups (e.g., high versus low achieving students, see [18]).

Two recent innovations to the application of ENA in the research community have helped to improve the relevance of applying this approach in the context of programming education where systems for automated assessment of programming assignments are used. First, there is now increasing understanding of how to apply ENA to interaction data, where events in log files may serve as the codes themselves [17]. Examples of this approach in practice have included epistemic networks that compared patterns of player actions in a puzzle-based video game [7], and the use of a series of in-game actions from log data to identify significant differences in how players responded to game events when they replay a scientific educational game [16]. The second advancement is the expansion of ENA techniques to also calculate the direction of code relationships in chronological data (Ordered Network Analysis, described in detail in *Methods* section; [21]). ONA has demonstrated utility for data and contexts in which it is important to understand the order of events as they unfold over time, such as instances of collaborative problem solving [21], or player actions in games [26]. Taken together, these innovations position this approach as well-suited for comparing activity patterns over time for high and low achieving learners using data from an automated programming assessment tool.

1.2 ENA for Analyzing Computer Education

Research in the field of CS education has begun to apply ENA to examine complex phenomena, with a particular emphasis on collaborative learning practices. In K-12 learning settings, Su and colleagues [20] and Vandenberg and colleagues [22] used epistemic networks to relate elements within conversations within pairs of students using pair programming [20] and course learning topics [22]. Research in CS education at the university level has also leveraged ENA to compare computational thinking approaches enacted by low and high performing novice groups in a university-level CS course based on collaborative problem-solving discussion data [24]. Pinto and colleagues' [12] extended the application of ENA to understand student programming actions more directly. These authors built network models to investigate the evolution of debugging behaviors in code submission assignments, ultimately showcasing the unique approaches enacted by more and less experienced programmers. Our work is an effort to extend this innovation by using Ordered Network Analysis to understand what patterns of events emerge when high and low performing novice students engage in programming tasks.

2 Methods

2.1 Data collection

The data used in this study was collected in fall semester 2022/2023 as a part of the "Introduction to programming" course: a mandatory first-semester course for computer science students at a large university in Poland. It consists of answers submitted in a questionnaire at the beginning of the semester, the results of the pre-test conducted directly after submitting a questionnaire, activity within an online platform for automated assessment of programming assignments collected during the semester, and the results of the final test submitted as a graded assignment at the end of the semester. Consent was obtained from students (N=198) prior to joining this study.

This work examines the patterns of behaviors made by learners we conceptualize as novice students who indicated having little to no programming experience prior to joining the course. The novice categorization was based on student responses to a survey item deployed during the first classes: "On the scale 1-5, where 1 is zero experience, and 5 is a lot of experience, please rate your basic programming knowledge (types, variables, conditional statement, recursion, loops, arrays)." We consider novices to be students that responded with a 1 or 2 (N=110) in the questionnaire and excluded more experienced students that responded with a 3-5 (N=88). We validated students' self-reports of expertise by asking students to take a pre-test assessing their general programming

knowledge. This test was administered immediately after students submitted the questionnaire. The difference between the novice group (Mdn=16%) and the group of more experienced students (Mdn=71%) was statistically significant (W=518, p<0.001) for a non-parametric Mann-Whitney U test.

The final test was administered at the end of the semester and evaluated student knowledge in the common introductory programming concepts such as types, variables, conditional statement, recursion, arrays, and loops. Within the novice group, we further categorized students based on their performance on the final test to create the high (N=43) and low (N=67) performing groups, in order to understand the difference between students who learned more or less. Novices with final test scores greater than the test median (Mdn=55%) were assigned to the high-performing group (hp-novices), and the rest of students were assigned to the low-performing group (lp-novices). To evaluate the potential impact of the prior knowledge, we compared results of the pre-test for both groups. For the pre-test, no statistically significant difference was found between the lp-novices (Mdn=22%) and hp-novices (Mdn=21%) groups, (W=1596.5, p=0.340), for a non-parametric Mann-Whitney U test.

The online platform for automated assessment of programming assignments was used as a learning resource within the course. Students uploaded their C# code for a total of 146 tasks covering the following introductory programming topics: (1) types and variables (33) – tasks in this section require the usage of basic operators on variables of different types (multiplication, text concatenation), (2) conditional statement (25), (3) recursion (28), and (4) arrays and loops (60) – tasks in this section require the development of simple algorithms, such as ordering elements in an array according to a specific condition. Usage of the platform was not mandatory, and the number of submissions was not limited. Novice students generated 44,448 code submissions on the platform (23,038 in low performing and 21,410 in the high performing group) during the course of the study. There was no statistically significant difference in the number of submissions between the lp-novices (Mdn=79) and the hp-novices (Mdn=80) groups (W=1450, p=0.956), for a non-parametric Mann-Whitney U test.

2.2 Categorization of Events

To model and study student behavior, we focus on 6 events that can be directly extracted from the log data of their interactions with the platform. Initially, upon viewing a task, a student has two choices: either submit code to test its correctness or quit and proceed to the next task without solving the current one. When they submit a solution, the code is compiled and then run against a set of unit cases. As a result of the submission, three outcomes are possible: the solution might contain a compiler error (meaning the code fails to execute), the code might execute but produce an incorrect result (meaning that at least one unit test failed), or it might successfully deliver the correct outcome. In instances of a compiler error, the system provides an error message. If the code executes but is incorrect, students have several options: they can request feedback, submit another solution without seeking feedback, or choose to move on to the next task without solving the current one. Once students correctly solve a task, they can either submit more solutions for the same task or, more commonly, progress to the next task. The 6 events used in this study encapsulate these decision points and actions. Detailed definitions of these events are presented in Table 1.

Event	Definition
Compiler Error (CE)	Student submitted code containing a compiler error.
Unit Tests Failed (UTF)	Student submitted compiling code but failing to pass at least one unit test.
Feedback Requested (FR)	Student requested feedback after submitting code that failed to pass all unit tests.
No Feedback Requested (NFR)	Student did not request feedback after submitting code that failed to pass all unit tests.
Correct Submission (CS)	Student submitted compiling code that passed all unit tests.
Quit Task (QT)	Student moved to a new task without solving the current one.

Table 1: Categories of events.

2.3 Ordered Network Analysis

In our study, Ordered Network Analysis (ONA) accumulates connections across events (stored in separate lines) in student log data, using a moving window to connect each line of chronological data to the preceding lines within the window. The length of the moving window is determined through examination of the dataset to estimate how far back each line should refer. In our case, we chose a narrow window size of 2, to visualize only the relationship between two consecutively observed events for a student. Connection accumulations are only calculated across lines associated with the same student and task in the tool, i.e. from a student's start to successful completion (or quitting) of a task.

ONA accounts for the order in which the connections occur by constructing an asymmetric adjacency matrix for the data associated with each unit of analysis. For this research, the unit of analysis is the series of events associated with a single task attempted by an individual student. Units are then categorized based on student assignment into the high and low performing groups.

In ONA, connections have directions and therefore the number of connections from code A to code B may be different than the number of connections from B to A. The ONA

algorithm transforms the matrix for each unit (events for a task-student pair) into a single high dimensional asymmetric adjacency vector, which is then normalized and centered. The algorithm performs a dimensionality reduction using singular value decomposition (SVD) and a means rotation (MR) when comparing two groups (e.g., high performers and low performers). The ONA projects the units' means of each individual unit-network into a 2-dimensional space so that the network graph visualizations can meaningfully reflect the mathematical properties of each unit. Each unit's mean (for each task-student pair) is then represented as a point in the low dimensional space (the coordinate plane). The network nodes in ONA are positioned in the space using an optimization algorithm that minimizes the distance between the centroids of the networks obtained with those nodes and the actual unit means obtained after the dimensional reduction [2]. As a result, the ONA metric space can be interpreted based on the locations of the nodes. Units on the right side of the space have more frequent connections between the codes (nodes) on the right side of the space, and units with points on the left have more frequent connections between the codes on the left side of the space (see [18] for further details about ENA interpretation, [2] for mathematical details about ENA, and [21] for specific details about ONA).

For each of the hp-novice and lp-novice groups, a network graph (not displayed in this article) shows the strength and directionality of the connections made between events. In this article, we focus on (and display) the difference between these 2 individual networks of each group. In our visualizations, only those transitions where the line weight exceeds 0.005 are shown, to improve clarity. For the implementation of the ONA, we employed the R package developed by Tan et al. [21].

Once the network models were generated, we applied a mixed-effects rank regression model to determine if the students in each group showed statistically significant differences in their behavior at the task submission level. Variables were transformed to ranks due to substantial non-normality in most of the variables. This approach involves comparing the unit mean ranks along the MR axis (x-axis of the two-dimensional reduction) for all task-student pairs in both groups. We chose a mixed-effects approach due to the non-independence in task submissions of the same student. This procedure was replicated across three distinct scenarios. Firstly, we applied the ordered network analysis considering all student data to identify key patterns throughout the semester. Secondly, we focused exclusively on data related to the *conditional statement* (if … else) tasks, a fundamental topic presented in an early stage of the semester. Lastly, we examined data involving *loops*, which correspond to a more challenging content introduced in the second half of the semester. This approach allowed us to track the evolution of patterns over the semester across various topics.

3 Results

Figure 1 shows directed event patterns comparing the two novice groups across all the tasks during the semester. This model (denominated difference model) is generated as the subtraction of the individual models of each of the two groups. Line weights (lw) of these directed connections between nodes of the individual and difference model are presented in Table 2.

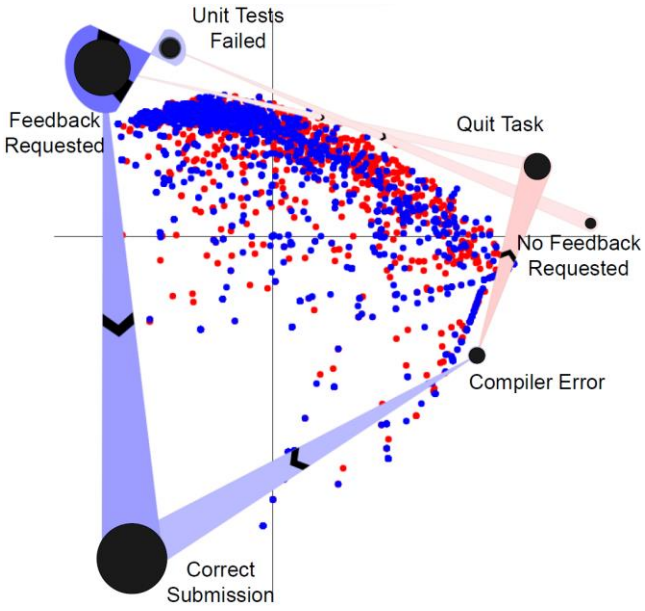

Figure 1: Difference Model comparing programming novices in low (red) and high (blue) performing groups (events during solving all tasks). The order of events goes from the thin to the thick part of the edge. Node size represents the relative response strength of each event. See [21] **for more details.**

Transition	High (Blue)	Low (Red)	Difference
UTF→ FR	0.395	0.357	0.038
FR → CS	0.145	0.120	0.026
CE → CS	0.187	0.169	0.018
FR → UTF	0.145	0.129	0.016
CE → QT	0.031	0.043	-0.012
UTF→ NFR	0.076	0.082	-0.006

Table 2: Weights of transitions with absolute difference > 0.005 considering all events during the semester.

Along the X axis (SVD1 with MR), a mixed-effects linear model showed marginally significant differences between the hp-novice (N=43) and the lp-novice groups (N=67, $\boldsymbol{\beta}$=411.47, p=0.060). The main difference between the two groups was their behavior in seeking feedback after unit tests fail. While both groups commonly requested feedback following such errors, high-performing novices (lw=0.395) did so more frequently than their low-performing counterparts (lw=0.357).

Notably, high performers generally had a higher rate of correct submissions after both types of error, particularly if they actively searched for feedback (feedback after unit test failures is presented upon the student's request after a student clicks a selected unit test). Interestingly, high performers also more often experienced unit test failures after seeking feedback compared to low performers (line weights of 0.145 and 0.129, respectively). This suggests that while seeking feedback might initially lead to additional errors, it may eventually contribute to a higher success rate for these students. In contrast, low performers did not show a significant increase in any specific transition compared to high performers. However, there was a tendency among them to avoid seeking feedback and to abandon tasks without submitting a correct answer, instead moving on to new tasks.

To analyze how learning patterns differed across various content areas, we focused on a *conditional statement* (introduced early in the semester) and *loops* tasks (a more complex topic in the second half of the semester). Figure 2 and Table 3 specifically examine conditional statement tasks. On the X axis (SVD1 with MR), a mixed-effects linear model showed a statistically significant difference between the unit mean for the hp-novices group (N=43) and the unit mean for the lp-novices group (N=67) ($\boldsymbol{\beta}$=97.05, p=0.008). These patterns mirror those seen in Figure 1 for the entire semester.

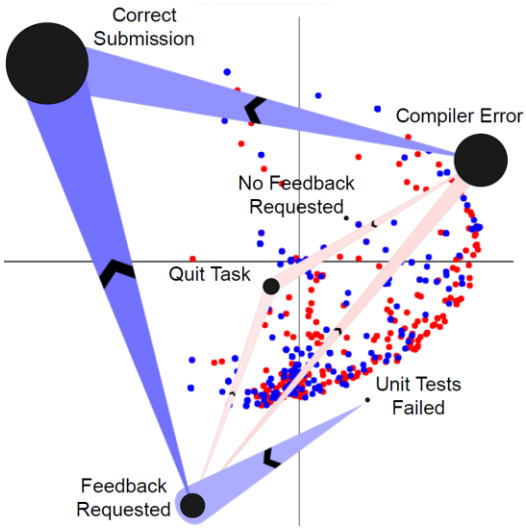

Figure 2: Difference Model comparing programming novices in low (red) and high (blue) performing groups (events for conditional statement tasks).

The high-performing group consistently more often submitted correct answers after facing both error types (requesting feedback for unit test failures). In contrast, low performers showed a slight tendency to experience compiler errors more often after requesting feedback (difference of line weights Δlw=-0.008) and quitting the task after requesting feedback (Δlw=-0.006) or having a compiler error (Δlw=-0.007). These trends suggest that low performers might more often face compiler errors and quit tasks at higher rates, whereas high performers are more likely to complete tasks successfully, especially when they request feedback.

Transition	High (Blue)	Low (Red)	Difference
FR → CS	0.163	0.127	0.037
CE → CS	0.276	0.248	0.028
UTF→ FR	0.331	0.310	0.021
FR → CE	0.030	0.039	-0.008
CE → QT	0.032	0.040	-0.007
FR → QT	0.031	0.037	-0.006

Table 3: Weights of transitions with absolute difference > 0.005 for events during solving the conditional statement tasks.

Figure 3 and Table 4 specifically focus on the differences observed between the two groups for *loops* tasks. Here, along the X axis (SVD1 with MR), a mixed-effects linear model showed a significant difference between the hp-novices group (N=43) and the lp-novices group (N=67) ($\boldsymbol{\beta}$=205.04, p=0.010). These results show patterns consistent with the previous models but with greater differences between high and low performers for this content. Once again, high performers were observed to request feedback more often and submit correct answers more often. Similarly, low performers tend to quit the task and/or avoid requesting feedback more often, which is a potential cause of their lower performance in the final test of the semester. This area presented the greatest differences between the individual networks of the two groups. As the last subject before the final test, the group differences appear to have become increasingly pronounced over the semester. These differences likely correlate more with the differences seen in the final test than with the content taught earlier in the semester.

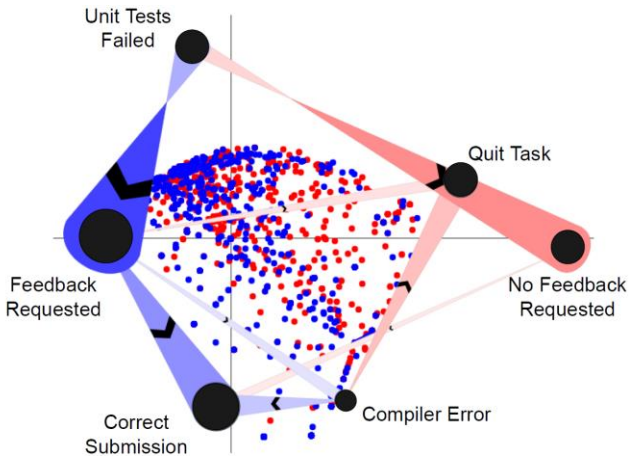

Figure 3: Difference model comparing programming novices in low (red) and high (blue) performing groups (events for loops tasks).

Transition	High (Blue)	Low (Red)	Difference
UTF→ FR	0.508	0.458	0.050
UTF→ NFR	0.070	0.099	-0.030
FR → CS	0.170	0.141	0.029
FR → UTF	0.194	0.173	0.021
CE → QT	0.014	0.030	-0.016
CE → CS	0.121	0.106	0.015
NFR → QT	0.009	0.020	-0.012
NFR → UTF	0.033	0.044	-0.010
FR → QT	0.079	0.087	-0.008
FR → CE	0.053	0.046	0.007
NFR → CS	0.018	0.023	-0.006

Table 4: Weights of transitions with absolute difference > 0.005 for events during solving loops tasks.

4 Discussion & Conclusions

Overall, these findings highlight the utility of ordered networks for illustrating behavioral differences between low performing and high performing groups when engaging in programming tasks. We identify three main advantages of employing this technique. First, ordered networks offer insight into the directionality of associations between events, suggesting chronological event relationships that can help explain user patterns of behavior. For example, a more strongly associated pathway for high performers in several models involves users having a unit test failure, followed by requesting feedback on the nature of the error, followed by a correct submission. Perhaps unsurprisingly, this suggests that reviewing feedback when an error is made helps to support later programming success.

Second, Ordered Network Analysis allows for examination of the same user data at different levels of granularity. Figure 1 highlights summative trends that are similar and different in high and low performing novices' actions as they work toward correct responses using the automated programming assessment tool, while Figures 2 and 3 illustrate the more granular patterns that emerge when learners engage in conditional statement and loops tasks. While the Figure 1 summary model reveals a stronger association between requesting feedback and higher performance in the final tests, the more granular models reveal that requesting (or not requesting) feedback explains a greater variance between the two performing groups for loops tasks than for conditional statement tasks.

Third, the use of difference models highlights the specific network associations that contributed to statistically significant differences between user groups, offering insights into what patterns of behavior set low performing and high performing users apart. Including performance indicators in the models (e.g., errors, correct submission, tasks quitting) shows not only which task-level outcomes were achieved more frequently by group, but also which patterns of events preceded their onset. For example, when learners made compiler errors for loops tasks (Figure 3), high performers were more likely to request feedback before ultimately achieving a correct response, while low performers were more likely to not request feedback on the error or quit the task.

The observed differences between high-performing novices and their less successful peers highlight the critical role of self-regulatory behaviors in computer science education. In line with prior work [e.g., 8, 13, 14] our results particularly emphasize the value of fostering a reflective attitude towards past mistakes and the importance of perseverance and persistence rather than abandoning the task when facing consecutive errors. The differences in these self-regulated skills already manifested at an early stage of the semester (for the *conditional statement* tasks) and became stronger at the end of the semester (for the *loops* tasks), suggesting that the low-performing group of novices did not learn these skills during the semester. Such findings can guide and support the design of targeted interventions that might assist learners and reinforce self-regulatory skills at key moments in their programming engagement toward task success.

Some possible interventions to enhance learning using this platform could include automatically showing students feedback about their last submission, including the specific unit tests it failed. Additionally, after several successive incorrect attempts, platforms could offer customized hints or motivational messages tailored depending on the nature of the errors and the duration between each submission. When implementing these interventions, designers should be mindful of encouraging self-regulation and genuine engagement with the material rather than motivating counterproductive behaviors such as students submitting code repeatedly merely to access hints, thereby attempting to complete tasks more quickly without understanding the content (e.g. gaming the system [1]).

In conclusion, Ordered Network Analysis can be a useful method for understanding how trajectories of behavior differ between groups of learners. By understanding those trajectories, we can find strategies used by more successful students and identify opportunities for interventions that shift less successful students' behaviors and support their learning.

ACKNOWLEDGMENTS

Andres Felipe Zambrano thanks the Ministerio de Ciencia, Tecnología e Innovación and the Fulbright-Colombia commission for supporting his doctoral studies through the Fulbright-MinCiencias 2022 scholarship.

REFERENCES

[1] Ryan S.J.d. Baker, Adriana M.J.B. De Carvalho, Jay Raspat, Vincent Aleven, Albert T. Corbett, and Kenneth R. Koedinger. 2009. Educational software features that encourage and discourage "gaming the system." In *Proceedings of the 14th international conference on artificial intelligence in education*, 2009. 475–482.

[2] Dale Bowman, Zachari Swiecki, Zhiqiang Cai, Yeyu Wang, Brendan Eagan, Jeff Linderoth, and David Williamson Shaffer. 2021. The mathematical

foundations of epistemic network analysis. In *Advances in Quantitative Ethnography: Second International Conference, ICQE 2020, Malibu, CA, USA, February 1-3, 2021, Proceedings 2*, 2021. Springer, 91–105.

[3] Li-Chen Cheng, Wei Li, and Judy CR Tseng. 2023. Effects of an automated programming assessment system on the learning performances of experienced and novice learners. *Interact. Learn. Environ.* 31, 8 (2023), 5347–5363.

[4] Katrina Falkner, Rebecca Vivian, and Nickolas JG Falkner. 2014. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014. 291–296.

[5] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, 2010. 86–93.

[6] Shamya Karumbaiah and Ryan S Baker. 2021. Studying affect dynamics using epistemic networks. In *Advances in Quantitative Ethnography: Second International Conference, ICQE 2020, Malibu, CA, USA, February 1-3, 2021, Proceedings 2*, 2021. Springer, 362–374.

[7] Xiner Liu, Basel Hussein, Amanda Barany, Ryan S Baker, and Bodong Chen. 2023. Decoding Player Behavior: Analyzing Reasons for Player Quitting Using Log Data from Puzzle Game Baba Is You. In *International Conference on Quantitative Ethnography*, 2023. Springer, 34–48.

[8] Dastyni Loksa, Lauren Margulieux, Brett A Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2022. Metacognition and self-regulation in programming education: Theories and exemplars of use. *ACM Trans. Comput. Educ. TOCE* 22, 4 (2022), 1–31.

[9] Qizhong Ou, Weijie Liang, Zhenni He, Xiao Liu, Renxing Yang, and Xiaojun Wu. 2023. Investigation and analysis of the current situation of programming education in primary and secondary schools. *Heliyon* 9, 4 (2023).

[10] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ. TOCE* 22, 3 (2022), 1–40.

[11] Maciej Pankiewicz, Ryan Baker, and Jaclyn Ocumpaugh. 2023. Using intelligent tutoring on the first steps of learning to program: affective and learning outcomes. In *International Conference on Artificial Intelligence in Education*, 2023. Springer, 593–598.

[12] Juan D Pinto, Qianhui Liu, Luc Paquette, Yingbin Zhang, and Aysa Xuemo Fan. 2023. Investigating the Relationship Between Programming Experience and Debugging Behaviors in an Introductory Computer Science Course. In *International Conference on Quantitative Ethnography*, 2023. Springer, 125–139.

[13] 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*, 2019. 531–537.

[14] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive difficulties faced by novice programmers in automated assessment tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018. 41–50.

[15] John Saint, Dragan Gašević, Wannisa Matcha, Nora'Ayu Ahmad Uzir, and Abelardo Pardo. 2020. Combining analytic methods to unlock sequential and temporal patterns of self-regulated learning. In *Proceedings of the tenth international conference on learning analytics & knowledge*, 2020. 402–411.

[16] Jennifer Scianna, David Gagnon, and Bryan Knowles. 2021. Counting the Game: Visualizing Changes in Play by Incorporating Game Events. In *Advances in Quantitative Ethnography*, Andrew R. Ruis and Seung B. Lee (eds.). Springer International Publishing, Cham, 218–231. https://doi.org/10.1007/978-3-030-67788-6_15

[17] Jennifer Scianna, Xiner Liu, Stefan Slater, and Ryan S Baker. 2023. A Case for (Inter) Action: The Role of Log Data in QE. In *International Conference on Quantitative Ethnography*, 2023. Springer, 395–408.

[18] David Williamson Shaffer, Wesley Collier, and Andrew R Ruis. 2016. A tutorial on epistemic network analysis: Analyzing the structure of connections in cognitive, social, and interaction data. *J. Learn. Anal.* 3, 3 (2016), 9–45.

[19] Leonardo Silva, António Mendes, Anabela Gomes, and Gabriel Fortes. 2023. Fostering regulatory processes using computational scaffolding. *Int. J. Comput.-Support. Collab. Learn.* 18, 1 (2023), 67–100.

[20] Yu-Sheng Su, Shuwen Wang, and Xiaohong Liu. 2023. Using Epistemic Network Analysis to Explore Primary School Students' Computational Thinking in Pair Programming Learning. *J. Educ. Comput. Res.* (2023), 07356331231210560.

[21] Yuanru Tan, Andrew R Ruis, Cody Marquart, Zhiqiang Cai, Mariah A Knowles, and David Williamson Shaffer. 2022. Ordered network analysis. In *International Conference on Quantitative Ethnography*, 2022. Springer, 101–116.

[22] Jessica Vandenberg, Collin Lynch, Kristy Elizabeth Boyer, and Eric Wiebe. 2023. "I remember how to do it": exploring upper elementary students' collaborative regulation while pair programming using epistemic network analysis. *Comput. Sci. Educ.* 33, 3 (2023), 429–457.

[23] E Vegas, M Hansen, and B Fowler. Building skills for life: how to expand and improve computer science education around the world (2021).

[24] Bian Wu, Yiling Hu, Andrew R Ruis, and Minhong Wang. 2019. Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *J. Comput. Assist. Learn.* 35, 3 (2019), 421–434.

[25] Mengqian Wu, Jiayi Zhang, and Amanda Barany. 2022. Understanding Detectors for SMART Model Cognitive Operation in Mathematical Problem-Solving Process: An Epistemic Network Analysis. In *International Conference on Quantitative Ethnography*, 2022. Springer, 314–327.

[26] Andres Felipe Zambrano, Amanda Barany, Jaclyn Ocumpaugh, Nidhi Nasiar, Stephen Hutt, Alex Goslen, Jonathan Rowe, James Lester, Eric Wiebe, and Bradford Mott. 2023. Cracking the code of learning gains: Using Ordered Network Analysis to Understand the Influence of Prior Knowledge. In *International Conference on Quantitative Ethnography*, 2023. Springer, 18–33.

[27] Si Zhang, Qianqian Gao, Mengyu Sun, Zhihui Cai, Honghui Li, Yanling Tang, and Qingtang Liu. 2022. Understanding student teachers' collaborative problem solving: Insights from an epistemic network analysis (ENA). *Comput. Educ.* 183, (2022), 104485.