Extending Scratch Framework to Improve the Analytical Skills of Undergraduate Students

Ubaid Ul Akbar
Department of Computer Science, City University of Science and Information Technology
Peshawar, Pakistan
ubaidulakbar@gmail.com

Saeed Akbar
School of Computer Science and Technology, Zhejiang Normal University, Jinhua 321004, China
saed.akbr@zjnu.edu.cn

Mumtaz Ali
Department of Computer Science, City University of Science and Information Technology
Peshawar, Pakistan
mumtazali@cusit.edu.pk

Rahmat Ullah
School of Computer Science and Electronic Engineering, University of Essex, UK
rahmat.ullah@essex.ac.uk

Rizwan Khan
School of Computer Science and Technology, Zhejiang Normal University, Jinhua 321004, China
imrizwankhan@gmail.com

Ivandro Ortet Lopes
School of Cyber Science & Engineering, Huazhong University of Science & Technology, Wuhan, China
ivandro.lopes@gmail.com

ABSTRACT

Scratch is an innovative and the most popular block-based Visual Programming Language designed for beginners to learn programming effectively. However, it lacks the capacity to allow users to learn and solve larger real-world problems such as the Travelling Salesman Problem (TSP). Hence, there is a need for an accessible and effective tool to assist beginners and novice programmers in learning and tackling the TSP. This paper introduces ScratchTSP, an extension of the Scratch programming Framework, specifically designed for the TSP. It offers an opportunity for beginners and novice programmers to grasp the TSP and the various algorithms used to solve it. For evaluation, we conduct a survey among undergraduate students to assess the usability and usefulness of the proposed ScratchTSP extension. Survey findings indicate that ScratchTSP offers a user-friendly and effective tool for beginners or novice programmers, aiding them in learning the TSP while developing their critical thinking and problem-solving skills.

KEYWORDS

Block-based Visual Programming Languages, Problem-solving skills, Scratch, Travelling Salesman Problem

ACM Reference Format:
Ubaid Ul Akbar, Saeed Akbar, Mumtaz Ali, Rahmat Ullah, Rizwan Khan, and Ivandro Ortet Lopes. 2024. Extending Scratch Framework to Improve the Analytical Skills of Undergraduate Students. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programming is vital across various engineering disciplines such as software engineering, electrical, and industrial engineering. Before Visual Programming Languages (VPLs), students found programming challenging and uninteresting [17]. VPLs make programming simpler and fun for beginners. Among VPLs, Scratch [13] is the most prominent visual programming framework, primarily designed to increase beginner's motivation towards programming. Although Scratch does not provide any native support for transitioning to high-level languages [12], students having a background in Scratch appear to perform well in C programming [6]. The scratch framework helps students in developing their problem-solving skills [8], computational thinking [5], and logical reasoning [20].

Despite the numerous advantages that Scratch offers, one of its major limitations is that the set of features offered by Scratch is not mature enough [4] to help in teaching, learning, and solving complex programming tasks [15] such as the Travelling Salesman Problem (TSP) [16], software project scheduling [2] and deep learning [3]. The TSP is a well-known combinatorial optimization problem with numerous practical applications in various fields [16]. These applications include logistics, vehicle routing, and circuit design. It serves as a benchmark problem in the study of algorithms and optimization techniques. Therefore, there is a need to design and develop a Scratch-based extension to help beginner/novice programmers learn, understand, and solve the TSP.

In this paper, we fill the aforementioned gap by introducing a Scratch extension called the ScratchTSP. It allows beginner-level or novice programmers to gain a better understanding of the TSP and how various algorithms solve it using Scratch. We design and develop custom blocks for the said purpose and add them to the basic vocabulary of the Scratch framework. We implement and integrate well-known optimization algorithms such as Brute Force, Branch and Bound, and Simulated Annealing. The code is publicly available on Github [1].

The remaining sections are ordered as: Section 2 summarises the existing literature. Section 3 presents the ScratchTSP. Section 4 and Section 5 discuss the survey results and the conclusion, respectively.

[1] https://github.com/ubaidulakbar/scratchtsp

2 RELATED WORK

Alturayeif et al. [3] introduce DeepScratch, a deep learning model extension to Scratch that provides powerful language elements to facilitate designing and building deep learning models. The extension provides two options to implement deep learning models: training a neural network based on built-in data sets and using pre-trained deep learning models. Park et al. [15] came up with the idea of novel Scratch programming blocks for web scraping, which allows students to scrape the contents of HTML elements using CSS Selector and XPATH and automating their keyboard and mouse in several ways.

Park et al. [14] design Tooee, a Scratch extension that allows K-12 students to create big data and artificial intelligence applications using text-based visual blocks. The proposed extension integrates WebSockets into the Scratch environment. Scratch is primarily designed to increase beginner's motivation towards programming. Recent studies have tried to add secondary features related to programming such as the syntax validator [18], hint generator [9], and logical errors detector [10] to the Scratch framework.

Research gap: Research community offers several extensions for the Scratch programming environment to enable its users teach, learn, and solve complex problems such as training a deep neural network [3]. However, existing studies overlook some of the most important real-world problems such as the TSP [16] and task scheduling [1]. TSP is one of the most widely studied problems due to its numerous practical applications in real life. Therefore, there is a need for a Scratch-based extension to help its users learn, understand, and solve the TSP.

3 THE PROPOSED EXTENSION

The proposed ScratchTSP extension adds additional blocks and commands to the Scratch framework. Furthermore, the ScratchTSP implements three well-known algorithms for solving the TSP that the Scratchers can experiment with: (1) the Brute Force algorithm [11], (2) the Branch and Bound algorithm [19], and (3) the Simulated Annealing algorithm [7]. The extension vocabulary is divided into two types of blocks:

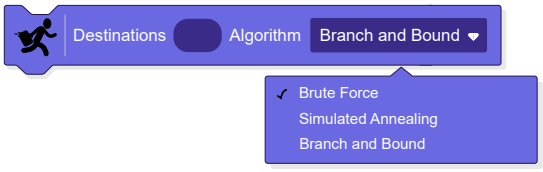

Figure 1: ScratchTspVisual block

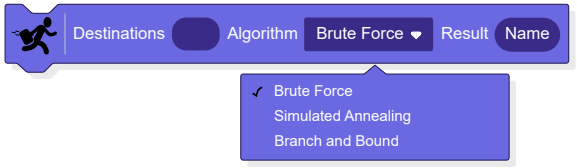

Figure 2: ScratchTspData block

3.1 ScratchTSP Blocks

Before conducting experiments with the Traveling Salesman Problem (TSP), it is essential for the user to delineate the destinations to be traversed and the algorithm to identify the optimal path connecting the designated points. This can be achieved through the utilization of the custom Stack blocks provided by the ScratchTSP extension.

3.1.1 ScratchTspVisual Block. ScratchTspVisual block allows the user to specify destinations and select the algorithm to find the optimal path from a drop-down list as shown in Figure 1.

It simulates the given scenario and visualizes the output as an un-directed graph. This block has two parts: (1) *Input List* – It takes a list defining the position of nodes (destinations) in terms of x-axis and y-axis values as an input as shown in Figure 3, and (2) *Algorithm Selection* – Another essential element of the block is the algorithm drop-down menu. It allows users to select algorithms such as the Brute Force algorithm, the Branch and Bound algorithm, and the Simulated Annealing algorithm to find the optimal path for the given destinations.

3.1.2 ScratchTspData Block. Similar to the ScratchTspVisual block, the ScratchTspData block allows users to specify destinations and select an algorithm to solve the given input. However, unlike ScratchTspVisual, ScratchTspData solves a given scenario and returns a list (Result) containing the optimal path found by the selected algorithm (as shown in Figure 2) instead of visually presenting the simulation results in an un-directed graph. ScratchTspData block is useful if the users tend to use the results of the selected algorithm in their custom applications. This block has three parts: (1) *Input List* – Similar to the ScratchTspVisual block, it takes a list defining the position of nodes (destinations) in terms of x-axis and y-axis values as input, (2) *Algorithm Selection*: Similar to the ScratchTspVisual block, this block has a drop-down menu allowing users to select an algorithm to solve the TSP problem with input defined in the input list, and (3) *Result* – In addition to Input List and Algorithm Selection, the ScratchTspData block allows the user to specify the name of the list where it needs to store the optimal path returned by the selected algorithm.

3.2 Variable Block

The ScratchTSP extension contains a variable block called the Distance block, as depicted in Figure 4. The distance block returns the total distance travelled after running the simulation. The Variable block helps track the progress of the solution and displays information to the user.

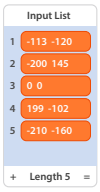

Figure 3: Defining positions for the nodes

Figure 4: Distance variable block

4 SURVEY RESULTS AND DISCUSSION

We employ both pre-lecture and post-lecture surveys in our study. The pre-lecture survey is used to collect demographic data on students, including age, gender, prior degree grades, programming proficiency, and understanding of the TSP. To assess the usability and effectiveness of our extension, the post-lecture survey is conducted. Figure 5 provides a visual representation of the procedural sequence. The process commences with the formation of two student groups (G1 and G2) and the administration of the pre-lecture survey. Subsequently, analysis of the pre-lecture survey results informs the development of lectures tailored to content relevance and sufficiency. Following a two-week lecture series, the post-lecture survey assesses the ScratchTSP extension's effectiveness. The feedback provided by students guides the evaluation of usability and effectiveness, as well as suggestions for potential enhancements. In the following subsections, we present the survey methodology, participant's detailed demography, data collection and analysis method, survey results and discussion, and finally summarize the main findings and their implications in detail.

4.1 Survey Design and Methodology

We design two surveys: (1) pre-lecture survey, and (2) post-lecture survey. The combination of pre and post-lecture surveys ensured a holistic approach to data collection, aligning the lectures with the students' diverse proficiency levels and enhancing the ScratchTSP extension based on their insights.

4.1.1 Pre-lecture Survey. The pre-lecture survey is meticulously designed to understand the programming proficiency levels of participating students and tailor the lectures accordingly. Another important objective is to assess the students' baseline understanding of the Traveling Salesman Problem (TSP). This survey was instrumental in ensuring that the subsequent lectures were adjusted to meet the diverse needs of the attendees.

4.1.2 Post-lecture Survey. The post-lecture survey is structured to collect feedback from students who had participated in the ScratchTSP lectures. Its primary objectives is to assess the effectiveness of the ScratchTSP extension, gather insights into its ease of use, and solicit suggestions for potential improvements. The survey aims to gauge the students' overall satisfaction with the extension and understand how it met their educational needs.

4.2 Participants

To evaluate the usability and effectiveness of the ScratchTSP extension, we conduct a survey of 65 undergraduate students. The majority of the students are male (54), and the remaining 11 students are female. The participating students are from Engineering, Medical, and Computer Science, with ages ranging from 20 to 25 years. Moreover, students are classified based on their proficiency level in programming as beginner-level (newbies), novice, intermediate, proficient, and expert programmers (Table 1 and 2). Moreover, Figure 6 offers insights into students' levels of understanding of the Traveling Salesman Problem (TSP), segmented by their programming proficiency.

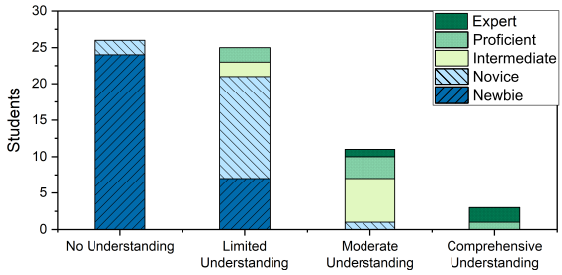

Figure 6: Pre-lecture survey – TSP understanding

According to Figure 6, a significant number of students, especially beginners and novices, report having no understanding of TSP. This underscores the need for educational interventions to introduce these students to the concept effectively. The category of "Limited Understanding" is prevalent among novices and intermediates. This group acknowledges some familiarity with TSP but is still in the early stages of comprehension. Students with a "Moderate Understanding" of TSP are primarily intermediates, proficients, and experts. These students have a relatively deeper grasp of TSP, indicating that as proficiency increases, so does the understanding of complex concepts like TSP. The category of "Comprehensive Understanding" is primarily represented by proficients and experts. These students demonstrate a high level of comprehension in TSP, showcasing the rewards of advanced programming proficiency.

4.3 Data Collection and Analysis

The data collection is done through the administration of physical in-person surveys, both before and after a series of educational lectures. These surveys are designed to gather valuable insights from the participating students. The pre-lecture survey comprising multiple-choice questions serves as a vital tool to profile the students' backgrounds and proficiency levels. It encompasses inquiries about previous academic performance, age, programming skills, and understanding of the TSP. These details help in categorizing the students into distinct proficiency levels, enabling a comprehensive understanding of their diverse demographics.

Table 1: Participants

Group	Students	Male/Female	Descipline
1	43	37/6	Computer Science
2	7	5/2	Medical
3	15	12/3	Engineering
Total	65	54/11	

Table 2: Programming proficiency of Participants

Proficiency	Definition	Students	M/F
Newbie	No prior Experience	31	24/7
Novice	Experience <= 1 year	17	14/3
Intermediate	Experience <= 2 years	8	7/1
Proficient	Experience <= 3 years	6	5/1
Expert	Experience > 3 years	3	3/0

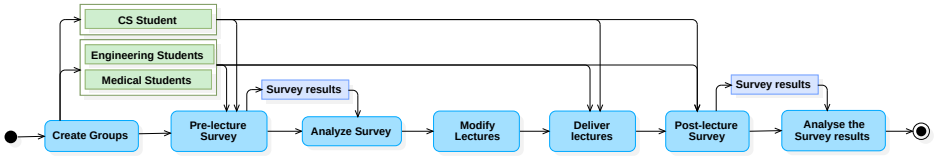

Figure 5: Activity diagram describing the workflow of the survey

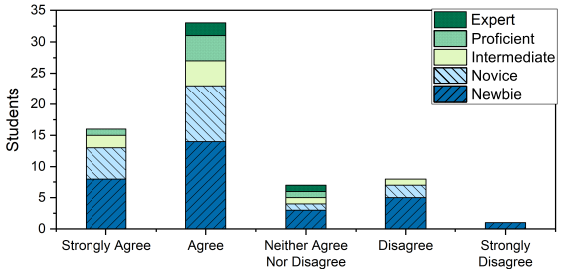

Figure 7: ScratchTSP effectiveness

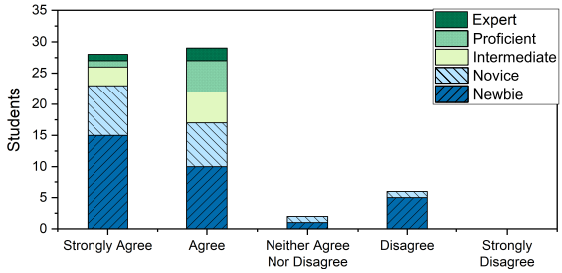

Figure 8: ScratchTSP ease of use

In contrast, the post-lecture survey primarily focuses on assessing the effectiveness of the proposed ScratchTSP extension, gauging students' satisfaction, and soliciting feedback for potential improvements. The Likert scale-based questions in this survey facilitate a nuanced evaluation of the student's satisfaction with the proposed extension. Additionally, an open-ended question provides the students with the opportunity to share qualitative feedback regarding any areas where they believe enhancements could be made.

For the analysis of survey results, bar charts are used to present the data graphically, enabling the depiction of patterns and trends with clarity. These charts provide a visual representation of the student's responses, making it easier to identify variations and correlations within the data. The use of graphical elements in data presentation adds a layer of accessibility and enhances the interpretability of survey results. This comprehensive data collection and analysis methodology allowed for a robust examination of students' demographics, their interactions with the proposed extension, and their feedback for potential improvements, fostering a more informed and tailored approach.

4.4 Post-lecture Evaluation

The post-lecture survey focuses on gathering feedback from the students to evaluate the effectiveness and ease of use of the proposed ScratchTSP extension. In the following text, we visualize and describe the survey results to gain useful insights about the feedback received from the students.

4.4.1 Effectiveness. Figure 7 offers a detailed view of students' perceptions regarding the effectiveness of the ScratchTSP, categorized by their programming proficiency levels. The Likert scale, encompassing options from "Strongly Agree" to "Strongly Disagree," served as a robust tool to capture the nuances of their feedback.

Across all proficiency levels, there is a substantial number of students who either "Strongly Agree" or "Agree" with the effectiveness of the ScratchTSP. This suggests that the majority of students, regardless of their programming expertise, found the extension to be highly effective or at least moderately so. Most importantly, all the Proficient and Expert programmers believe that the proposed extension is effective or highly effective. It is also important to mention that Beginners largely express favorable views.

A significantly smaller proportion of students choose "Neither Agree Nor Disagree." The segments that "Disagree" or "Strongly Disagree" are notably smaller compared to the segments that "Strongly Agree" or "Agree". A relatively small number of students, especially Beginners and Novices, hold negative views. Only one of the Intermediate programmers expresses some disagreement.

4.4.2 Ease of Use. We also evaluate how easy is to use the proposed ScratchTSP extension. Figure 8 offers a comprehensive view of students' perceptions regarding the ease of use of the ScratchTSP, categorized by their programming proficiency levels. Utilizing a Likert scale ranging from "Strongly Agree" to "Strongly Disagree," the data captures the nuanced feedback from students.

A significant number of students across all proficiency levels, including beginners, novices, intermediates, proficients, and experts, either "Strongly Agree" or "Agree" that the ScratchTSP extension is easy to use. This indicates a consistent positive sentiment across the entire spectrum of programming expertise. Even beginners, who typically have less experience, express strong agreement with the extension's ease of use.

The category of "Neither Agree Nor Disagree" contains a minimal number of responses, reflecting a slight degree of neutrality about the ease of use. Only 2 students express neutral feedback regarding the ease of use of the proposed extension.

The segments of students who "Disagree" or "Strongly Disagree" with the extension's ease of use are extremely small. Most of the students are Beginners, while only one is a Novice. No Intermediate, Proficient, or Expert programmer expresses negative thoughts. This trend suggests that the ScratchTSP extension generally avoids strong negative perceptions regarding its usability.

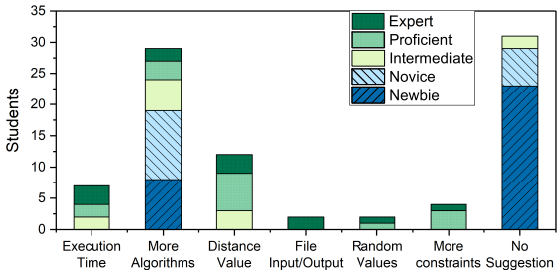

Figure 9: Suggested improvements

4.4.3 Suggestions for Potential Improvements. In addition to collecting feedback regarding the effectiveness and usability of the proposed extension, we also added an open-ended question to allow the users to suggest further potential advancements. Figure 9 reflects students' suggestions for improvements in the proposed ScratchTSP extension, categorized by programming proficiency levels. The dataset is structured with feature names as the first column, allowing us to examine how students across different proficiency levels perceive potential enhancements to the extension.

A large number of students from various proficiency levels have suggested the inclusion of more algorithms in the ScratchTSP extension. This aligns with the generally positive perceptions of the extension's effectiveness and ease of use observed in the previous datasets. It indicates an interest in enhancing the extension's functionality to cater to the diverse needs and expectations of students at different proficiency levels.

Intermediate and proficient programmers are the ones who most commonly suggest including features related to distance values, which is consistent with their deeper understanding of related concepts. This suggestion aligns with the effectiveness and ease of use data, as students at these proficiency levels have previously demonstrated a more favorable perception of the extension.

The suggestion to add features related to file input/output and random values is mainly from proficient and expert programmers. This aligns with the positive perception of the extension's ease of use among these proficiency levels. On the other hand, the suggestion to add more constraints to the extension primarily comes from proficient programmers.A significant number of beginners and novices do not provide any specific suggestions.

5 CONCLUSION

In this work, we develop ScratchTSP, a Scratch-based extension that aims to help novice programmers learn and understand the traveling salesman problem and its solution algorithms. The motivation for the development of the ScratchTSP extension stems from its potential to bridge the gap between theoretical knowledge and practical implementation of the TSP. This facilitates a more intuitive understanding of TSP and its applicability in real-world scenarios. The ScratchTSP can be used as a teaching aid in computer science and software engineering classes to introduce optimizations and algorithm design principles.

To evaluate the efficacy of the proposed extension, we conduct pre-lecture and post-lecture surveys. The pre-lecture survey is used to understand the demography and programming proficiency of participating students and tailor the lectures accordingly. The post-lecture survey is conducted to evaluate the proposed extension in terms of effectiveness and ease of use. We have proven through post-lecture surveys that our work has the potential to benefit the Scratch programming community by providing a better understanding of the TSP, a well-known combinatorial optimization problem.

REFERENCES

[1] Saeed Akbar, Ubaid Ul Akbar, Rahmat Ullah, and Zhonglong Zheng. 2024. A Caolitional Game-Based Adaptive Scheduler Leveraging Task Heterogeneity For Greener Data Centers. *IEEE Transactions on Green Communications and Networking* (2024), 1–1.

[2] Saeed Akbar, Muhammad Zubair, Rizwan Khan, Ubaid Ul Akbar, Rahmat Ullah, and Zhonglong Zheng. 2024. Weighted Multi-Skill Resource Constrained Project Scheduling: A Greedy and Parallel Scheduling Approach. *IEEE Access* (2024).

[3] Nora Alturayeif, Nouf Alturaief, and Zainab Alhathloul. 2020. DeepScratch: scratch programming language extension for deep learning education. *International Journal of Advanced Computer Science and Applications* 11, 7 (2020).

[4] Nayeon Bak, Byeong-Mo Chang, and Kwanghoon Choi. 2020. Smart Block: A visual block language and its programming environment for IoT. *Journal of Computer Languages* 60 (2020), 100999.

[5] Kay-Dennis Boom, Matt Bower, Jens Siemon, and Amaël Arguel. 2022. Relationships between computational thinking and the quality of computer programs. *Education and information technologies* 27, 6 (2022), 8289–8310.

[6] Jesennia Cárdenas-Cobo, Amilkar Puris, Pavel Novoa-Hernández, Águeda Parra-Jiménez, Jesús Moreno-León, and David Benavides. 2021. Using scratch to improve learning programming in college students: A positive experience from a non-weird country. *Electronics* 10, 10 (2021), 1180.

[7] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. 2019. Simulated annealing: From basics to applications. *Handbook of metaheuristics* (2019).

[8] Osman Erol and Neşe Sevim Çırak. 2022. The effect of a programming tool scratch on the problem-solving skills of middle school students. *Education and Information Technologies* 27, 3 (2022), 4065–4086.

[9] Benedikt Fein, Florian Obermüller, and Gordon Fraser. 2022. CATNIP: An Automated Hint Generation Tool for Scratch. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. 124–130.

[10] Gordon Fraser, Ute Heuer, Nina Körber, Florian Obermüller, and Ewald Wasmeier. 2021. Litterbox: A linter for scratch programs. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 183–188.

[11] Marijn JH Heule and Oliver Kullmann. 2017. The science of brute force. *Commun. ACM* 60, 8 (2017), 70–79.

[12] Yuhan Lin and David Weintrop. 2021. The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages* 67 (2021), 101075.

[13] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[14] Youngki Park and Youhyun Shin. 2021. Tooee: A novel scratch extension for K-12 big data and artificial intelligence education using text-based visual blocks. *IEEE Access* 9 (2021), 149630–149646.

[15] Youngki Park and Youhyun Shin. 2022. Novel scratch programming blocks for web scraping. *Electronics* 11, 16 (2022), 2584.

[16] Petrică C. Pop, Ovidiu Cosma, Cosmin Sabo, and Corina Pop Sitar. 2024. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research* 314, 3 (2024), 819–835.

[17] Alexander Repenning. 2017. Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *J. Vis. Lang. Sentient Syst.* 3, 1 (2017), 68–91.

[18] Alexander Repenning and Ashok Basawapatna. 2021. Smacking screws with hammers: Experiencing affordances of block-based programming through the hourglass challenge. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 267–273.

[19] Caio Paziani Tomazella and Marcelo Seido Nagano. 2020. A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem. *Expert Systems with Applications* 158 (2020).

[20] Albert Valls Pou, Xavi Canaleta, and David Fonseca. 2022. Computational Thinking and Educational Robotics Integrated into Project-Based Learning. *Sensors* 22, 10 (2022).