Developing Computational Thinking through a Virtual Robotics Programming Curriculum

EBEN B. WITHERSPOON, ROSS M. HIGASHI, CHRISTIAN D. SCHUNN, and
EMILY C. BAEHR, University of Pittsburgh
ROBIN SHOOP, The Robotics Institute, Carnegie Mellon University

Computational thinking describes key principles from computer science that are broadly generalizable. Robotics programs can be engaging learning environments for acquiring core computational thinking competencies. However, few empirical studies evaluate the effectiveness of a robotics programming curriculum for developing computational thinking knowledge and skills. This study measures pre/post gains with new computational thinking assessments given to middle school students who participated in a virtual robotics programming curriculum. Overall, participation in the virtual robotics curriculum was related to significant gains in pre- to posttest scores, with larger gains for students who made further progress through the curriculum. The success of this intervention suggests that participation in a scaffolded programming curriculum, within the context of virtual robotics, supports the development of generalizable computational thinking knowledge and skills that are associated with increased problem-solving performance on nonrobotics computing tasks. Furthermore, the particular units that students engage in may determine their level of growth in these competencies.

CCS Concepts: • **Applied computing → Interactive learning environments**;

Additional Key Words and Phrases: Computational thinking, robotics, programming, K-12, curriculum design

ACM Reference format:
Eben B. Witherspoon, Ross M. Higashi, Christian D. Schunn, Emily C. Baehr, and Robin Shoop. 2017. Developing Computational Thinking Through a Virtual Robotics Programming Curriculum. *ACM Trans. Comput. Educ.* 18, 1, Article 4 (October 2017), 20 pages.
https://doi.org/10.1145/3104982

1 INTRODUCTION

In the last decade, computational thinking has gained a great deal of attention in K-12 computing education. It is typically construed as an essential 21st-century skill that draws on algorithmic thinking and design processes, but especially in ways that may be generalizable across various contexts (Grover and Pea 2013; Wing 2006). In 2011, a committee of computer science (CS) experts, examining the role that CS would play in bringing computational thinking to K-12, broadly

This work is supported by the National Science Foundation, under grant DRL1418199.
Authors' addresses: E. B. Witherspoon, R. M. Higashi, C. D. Schunn, and E. C. Baehr, Learning Research and Development Center, University of Pittsburgh, 3939 O'Hara Street, Pittsburgh, PA 15260; emails: {ebw13, rmh57, schunn, ecb42}@pitt.edu; R. Shoop, National Robotics Engineering Center, 10 40th Street, Pittsburgh, PA 15201; email: rshoop@andrew.cmu.edu.

https://doi.org/10.1145/3104982

defined computational thinking as "an approach to solving problems in a way that can be solved by a computer…a problem solving methodology that can be transferred and applied across subjects" (Barr and Stephenson 2011). In 2015, national educational policies in the United States specifically included references to CS as part of a well-rounded science, technology, engineering, and mathematics (STEM) education, and the term "computational thinking" was added to the Next Generation Science Standards as a core scientific practice that could be applied across many science content areas (Weintrop et al. 2015). Despite its increasing prevalence as a general skill in K-12 education policies, relatively few empirical studies have been conducted to evaluate if instruction in computational thinking within a robotics environment can reliably produce computational skills that are useful for problem solving in other contexts.

1.1 Developing Tools for Generalizable Computational Thinking Instruction

Students can be exposed to aspects of computational thinking, including algorithmic thinking, by engaging in computer programming through diverse means such as website design, modeling and data analysis tools in science, and robotics (Lye and Koh 2014). While measures of transferrable CT have been developed in a variety of specific contexts, few have been applied within middle school robotics (Koh and Motter 2013; Werner and Kawamoto 2012). Recent studies of computational thinking that make use of visual programming languages and virtual learning environments show evidence of the development of computational practices like abstraction and algorithm development as well as increased interest in computing in those contexts (Hambrusch and Hosking 2009; Werner et al. 2012). Advances in visual environments (e.g., Alice) and graphical programming languages (e.g., Scratch) have led to a renewed interest in understanding the kinds of generalizable skills that programming might develop for a broader range of students across disciplines (Brennan and Resnick 2012; Lye and Koh 2014). Visual languages are thought to reduce the cognitive load required for novice programmers and allow for learners to attend to higher-level computational principles (Kelleher and Pausch 2005). By making output more visually concrete or reducing unnecessary syntax and effectively "chunking" textual elements through the visual representations of certain coding functions, students are able to focus more closely on the structural logic of programming (Robins and Rountree 2010). Providing rich, diverse programming contexts, as well as new technologies that provide a "low threshold" and "high ceiling" (Repenning and Ioannidou 2010) for novice programming students, could teach computing more effectively and approachably.

As momentum builds around including computational thinking in K-12 education, it will be necessary to design CS curricula that use new technologies in ways that adequately prepare and engage all students in developing the necessary computational skills for a 21st-century CS workforce (National Research Council 2010). Technological advances in most workplaces over the last 25 years have led to an overall decline in the number of jobs requiring rote skills and an increase in the need for workers who are prepared for complex, unstructured problem-solving tasks (Lee and Apone 2014). The field of CS has been no different; applications of computational problem solving are required in a much broader range of careers outside of those traditionally related to CS training. Indeed, now individuals and employers are required to dynamically use STEM skills in a variety of contexts that may be very different from that of their initial training (National Science Board 2015). This suggests that the most valuable kinds of skills are those that are generalizable and generative, rather than specific technical knowledge that quickly becomes obsolete. Therefore, while more specialized CS courses currently exist at the high school level, providing a more generalizable base of computational knowledge and skills at the middle school level may offer a productive opportunity for intervention. Furthermore, although CS is one of the fastest-growing STEM fields, there is a widening gap in the percentage of women, individuals from low socioeconomic backgrounds, and underrepresented minority populations entering CS careers (Atkinson and Mayo 2011). Curriculum and instruction that provide all middle school students with equitable opportunities to engage in computational thinking in varied contexts could increase overall participation and promote innovation in the field by including more diverse perspectives (ACM Education Policy Committee 2014).

Ideally, then, a tool for teaching computational thinking should be designed to take advantage of advances in new cognitively beneficial technologies, be accessible to learners with limited resources and diverse interests, and, perhaps most importantly, develop a generalizable computational thinking skill set in learners.

1.2 Robotics as a Context for Learning Computational Thinking

Educational robotics programs have experienced renewed interest and popularity since the early 1990s as engaging and motivating learning contexts that encourage a broad population of students to pursue STEM career pathways (Melchior and Leavitt 2005). Technological advances have made robotics more widely accessible to students both by lowering costs and by increasing mechanical and programmable customizability. In learning contexts, students often use robots as concrete tools for formalizing unfamiliar and abstract STEM concepts (Alimisis 2012; Okita 2013). In fields like mathematics and physics, concrete contextualization and interdisciplinary applications of robotics have been linked with improved domain-specific skills, as well as more general skill development in problem solving, logic, and scientific inquiry (Benitti 2012). Similarly, early research on computer programming in the context of robotics has been shown to offer unique opportunities for students to learn CS and develop computational thinking practices (Eguchi 2014; Grover and Pea 2013; Major, Kyriacou and Brereton 2012).

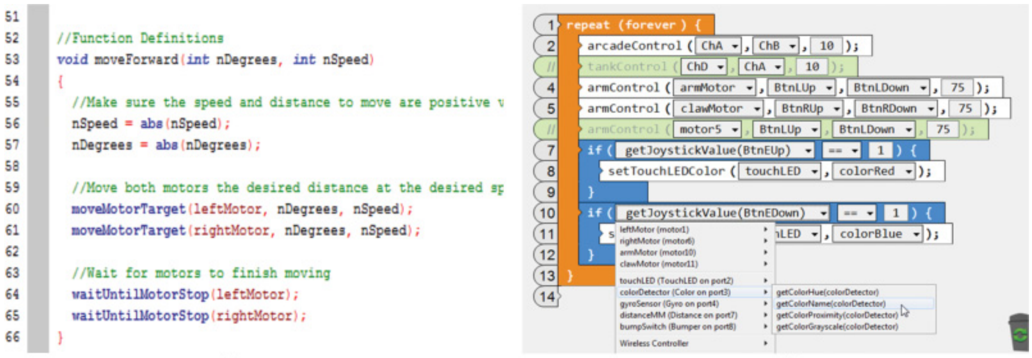
However, it is uncertain whether the knowledge and skills that students acquire in educational robotics contexts are generalizable to dissimilar contexts in the way that policymakers and advocates of computational thinking intend. Since the 1960s, studies of concrete LOGO programming tools in K-12 settings have produced varied and sometimes conflicting empirical results regarding the acquisition of generalizable computer programming skills (Clements and Gullo 1984; Klahr and Carver 1988; Pea 1983). Specific instruction in LOGO and other programming languages is generally more successful in achieving transfer of specific declarative skills (e.g., language functions and syntax), with fewer studies successfully demonstrating acquisition of generalizable procedural skills (e.g., debugging heuristics (Palumbo 1990)). Salomon and Perkins (1989) propose that these differences in acquisition arise from the existence of two types of transfer: low road and high road. Low-road transfer characterizes habituated behavioral patterns that become automatized through long-term, repeated practice in varied contexts and are subsequently triggered when sufficiently similar conditions arise. High-road transfer occurs through "deliberate, conscious abstraction from one context to another" (p. 152) and is more typical of the intentional application of skills from one context to another. Key to the idea of high-road transfer is the concept of "mindful abstraction," where students must possess both a decontextualized representation of a principle or procedure and an understanding of the particular situations in which it can be instantiated (Salomon and Perkins 1989, p. 126). Furthermore, providing students with the opportunity to construct these abstractions autonomously through active learning processes is also believed to facilitate high-road transfer (Mayer 1974; Mayer and Greeno 1972). Therefore, while low-road transfer is likely to be developed only after years of programming practice, high-road transfer could be more efficiently achieved through instruction that teaches students deliberate, metacognitive strategies for the application of generalizable computational thinking principles. Specifically, learning experiences that provide students with opportunities to actively construct abstract representations of computational principles and engage in problem-solving tasks with multiple solution pathways are more likely to facilitate transfer of those general principles.

When applied to typical K-12 robotics programs in their current form, this theoretical framework raises some concerns about the prospect of students successfully acquiring generalizable computational skills. Duration is a potential problem—robotics programs in middle schools are frequently implemented as short (e.g., 6-week) electives that are unlikely to offer sufficient opportunities for students to practice programming to a degree that would facilitate low-road transfer. Depth and the explicit emphasis on content are also issues. Software and hardware development in robotics contexts are often inextricable from one another. However, when the mechanical tasks are nontrivial, the emphasis in many environments seems to remain largely on correcting mechanical error and construction, and a relatively small portion of students are directly involved in programming (Alimisis 2012; Melchior et al. 2005). Furthermore, robotics programming requirements in these environments are often very basic, consisting largely of prescribed sequences of commands (Alimisis and Kynigos 2009; Liu et al. 2013). A narrow focus on a specific band of basic content, applied in a single concrete context, would not likely provide opportunities for high-road transfer. Lack of access to appropriate instructional support may also impede opportunities for transfer. A 2007 Computer Science Teachers Association report showed that only 12 states required a CS certification at the middle school level (Khoury 2007). As relative novices themselves, nondomain expert educators are more inclined to focus on rote, surface-level aspects of instruction, while de-emphasizing the structural features integral to learners' development of generalizable practices (Kurland et al. 1986; Catrambone 1998). In informal environments like the popular FIRST LEGO League robotics competition, a different problem occurs: mentors with professional engineering experience, while possessing substantial content knowledge, are unlikely to receive any pedagogical training (Melchior et al. 2005). Thus, without the appropriate curricular supports, robotics instructors frequently lack the necessary combination of content and pedagogical knowledge (Shulman 1986) required to teach CT-relevant material effectively, and in a conceptually rich way.

In short, programming education using robots has shown a great deal of potential, including successful learning outcomes in specifically targeted content areas. However, limited opportunities to engage in meaningful computer science learning, reinforced by a shortage of teachers trained in both content and pedagogy, leave it a crucial empirical question whether robotics programming learning contexts are ultimately capable of building generalized computational thinking skills.

1.3 Research Objectives

The goal of the current study is to evaluate if participation in a visual programming curriculum is related to measurable gains in students' ability to apply generalizable computational thinking skills to dissimilar problem-solving tasks. This curriculum is designed to scaffold the use of technologies such as graphical programming languages and virtual robotics simulations to produce optimal conditions for the "mindful abstraction" of generalizable computational thinking skills. First, using a graphical programming language and providing videos that explicitly decontextualize a core computational concept supports students in extracting the computational principles involved in programming, by using both of the *product* and *process* of abstraction (Salomon and Perkins 1989). Second, offering a variety of increasingly complex problem-solving scenarios in which students autonomously apply those principles facilitates the "controlled" metacognitive process of active learning that theoretically predicts more transfer than passively receiving direct instruction (Mayer and Greeno 1972; Salomon and Perkins 1989). Preliminary studies with earlier versions of this curriculum were more narrowly evaluated for programming skills and demonstrated significant gains (Liu et al. 2013). Here, we hope to expand upon this previous research to determine if a virtual programming environment can also demonstrate significant learning gains in broader computational thinking principles, especially as applied in nonrobotics contexts.



```
//Function Definitions
void moveForward(int nDegrees, int nSpeed)
{
  //Make sure the speed and distance to move are positive v
  nSpeed = abs(nSpeed);
  nDegrees = abs(nDegrees);

  //Move both motors the desired distance at the desired sp
  moveMotorTarget(leftMotor, nDegrees, nSpeed);
  moveMotorTarget(rightMotor, nDegrees, nSpeed);

  //Wait for motors to finish moving
  waitUntilMotorStop(leftMotor);
  waitUntilMotorStop(rightMotor);
}
```

(a)

```
repeat (forever ) {
  arcadeControl ( ChA , ChB , 10 );
// tankControl ( ChD , ChA , 10 );
  armControl ( armMotor , BtnLUp , BtnLDown , 75 );
  armControl ( clawMotor , BtnRUp , BtnRDown , 75 );
// armControl ( motor5 , BtnLUp , BtnLDown , 75 );
  if ( getJoystickValue(BtnEUp) == 1 ) {
    setTouchLEDColor ( touchLED , colorRed );
  }
  if ( getJoystickValue(BtnEDown) == 1 ) {
    s nLED , colorBlue );
  }
}
```

leftMotor (motor1)
rightMotor (motor6)
armMotor (motor10)
clawMotor (motor11)
touchLED (TouchLED on port2)
colorDetector (Color on port3)
gyroSensor (Gyro on port4)
distanceMM (Distance on port7)
bumpSwitch (Bumper on port8)
Wireless Controller
getColorHue(colorDetector)
getColorName(colorDetector)
getColorProximity(colorDetector)
getColorGrayscale(colorDetector)

(b)

Fig. 1. Examples of the ROBOTC text-based (a) and ROBOTC graphical (b) programming languages.

2 METHODS

2.1 Overview

This article presents two studies of the ongoing evaluation and iterative design of a robotics programming curriculum. The first study reported here piloted the curriculum within a single district. This initial study was used as an opportunity to get feedback on the curricular design, debug the technical aspects of the curriculum and assessments, and perform an initial exploratory case study evaluation of the learning gains of the intervention. The second study implemented the curriculum in 26 classrooms across four different school districts, using three separate versions of an assessment for computational thinking skills and practices as measures of learning gains. In this study, we tracked each class's progress through the curriculum and used a pre- and posttest design with alternative versions to assess if learning gains in computational thinking were related to amount of participation in the virtual robotics curriculum. It is important to note that while the programming activities and lessons reported here allowed the curriculum to be run entirely in the virtual environment, the curriculum was developed to replicate existing physical robotics hardware; therefore, the capability remained to download and test programs on physical robots for those teachers who had access to them. While it is possible that some of the teachers took advantage of this capability, we are confident that these physical activities remain supplemental and did not compose a majority of instruction.

2.2 Materials

2.2.1 Robotics Programming Curriculum. The online robotics curriculum, developed by Carnegie Mellon University and Robomatter, involves a sequence of lessons in robotics programming using a unique programming language, ROBOTC Graphical. ROBOTC Graphical is based on an existing text version of the ROBOTC programming language (see Figure 1(a)) that is designed for more advanced robotics students. ROBOTC Graphical, however, uses a graphical interface intended to support novice students, by focusing on the broader logic of programming while de-emphasizing the particular syntactic requirements of more traditional programming languages (see Figure 1(b)).

The curricular materials capitalize on the engaging aspects of robotics competitions, while emphasizing the practice of specific programming skills. The curriculum consisted of four instructional units: Intro to Programming, Basic Movement, Sensors, and Program Flow (see Table 1). The lessons within each unit were sequenced to develop students' understanding of basic programming

Table 1. Curricular Units with Number of Lessons and Content Topics Addressed

Unit Name	Lessons	Activities	General Topics Addressed
Intro to Programming	4	1	Setting up your computer with ROBOTC System requirements and configuration Tutorial on operating the ROBOTC platform
Basic Movement	6	16	Measurement (e.g., centimeters, degrees, rotations) Basic forward movement command sequences Basic turning command sequences
Sensors	10	19	Uses of sensors in robotics Equations with inequalities Boolean logic If-then conditional statements in programming
Program Flow	10	18	Repeat and infinite loops If-else conditional statements in programming Repeated decisions (if-else nested within loops)

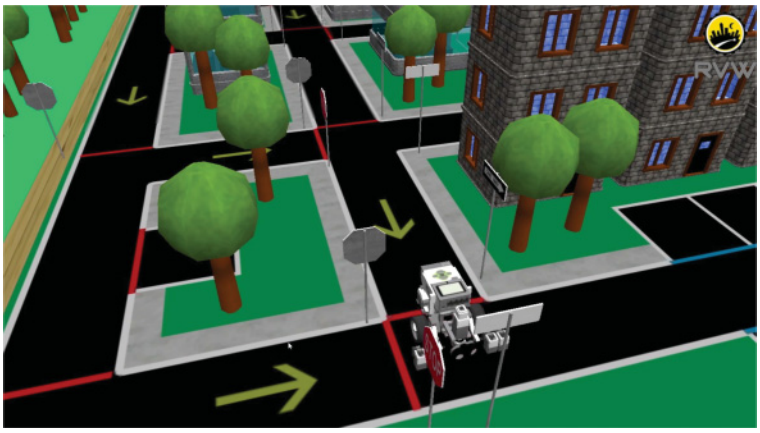


Fig. 2. An example of a game-like Robot Virtual Worlds challenge.

concepts (e.g., sequences, conditionals, and looping) as well as key computational concepts in programming as they appear in a robotics context (e.g., “Programming Is Precise,” “Sensors, Programs and Actions,” and “Make Sense of Systems”).

In each unit, students progress through a sequence of virtual robotics problem-solving tasks, using the ROBOTC Graphical language to develop programmed solutions. Students engaged with the curriculum through Robot Virtual Worlds, a simulated 3D game-like virtual environment designed to emphasize the programming aspects of robotics, while aiming to increase student interest and engagement (see Figure 2).

Students can iteratively test modular programmed solutions with simulated robots in a three-dimensional virtual platform. Finally, these pieces of code are reused and “remixed” (Brennan and

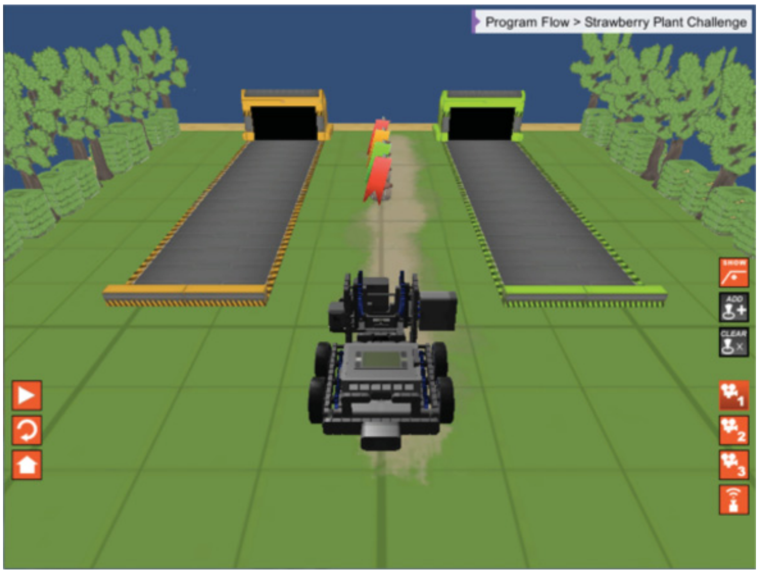


Fig. 3. An example of a programmable problem-solving task in the virtual platform. Using if-else statements, loops, and sensors, students program the robot to sort flags onto the left or right conveyor belt based on their color.

Resnick 2012) to create compound solutions to more complex virtual challenges (see Figure 3) in which learners must apply their previous programming knowledge to problem-solving tasks that foreground computational thinking principles like abstraction, decomposition, and systems thinking.

By representing robotics challenges in a virtual environment, this curriculum reduces the influence of mechanical errors that often frustrate novices and can distract from basic programming concepts. This feature enables programming students to instead focus on higher-level computational principles. Simulating robot movement within a virtual environment also reflects a common practice of robotics engineers today. Participating in this process provides students an opportunity to learn through participating in the authentic practices of a professional community (Lave and Wenger 1991). Virtual robots are also less expensive than physical ones, allowing the benefits of the curriculum to reach a broader population where the costs of physical robotics curricula can be prohibitive. Furthermore, an earlier study by Liu and Shoop (2013) found that students using an earlier version of this technology achieved learning gains in programming content equivalent to students using physical robots, but in less time.

The design of the curricular materials reflects a constructivist approach to instruction in which learners use worked examples, scaffolding, and reflection to build increasingly complex programs. Each lesson begins with a video of a robot performing an important real-world task; students are then asked to program their robots to solve an analogous “game board” version of the task. This sets up topic areas, tasks, and task contexts to put teachers and students “on the same page” (i.e., establish *intersubjectivity* (Puntambekar and Hubscher 2005)). Furthermore, it centers the lesson around student construction of a solution to the problem, which requires them to construct an understanding of the requisite programming principles (Papert and Harel 1991).

A series of instructional videos helps students to get started by providing step-by-step direction on how to code a set of related robot behaviors (but not the solution to the main challenge). These

Table 2. CT Constructs Measured in the Assessment, Number of Items, and Example Item Descriptions

Computational Thinking Constructs	Number of Items	Description of Example Items
Developing algorithms	4	Develop an algorithm by completing a missing step or correctly combining two algorithms
Evaluating algorithms	7	Determine if an algorithm will accomplish the task or identify an inaccuracy or inefficiency
Developing abstractions	3	Describe how an algorithm or piece of code will need to be adjusted to respond to the introduction of a new variable
Documentation and process	3	Identify procedures for iterative development, data collection, and documentation

videos use several multimedia approaches for reducing extraneous cognitive processing, managing essential processing, and fostering generative processing (Mayer 2008). For example, the videos encourage learners to self-pace by pausing the videos (allowing them to manage cognitive processing) and use a conversational tone (creating a sense of social partnership with the narrator, which encourages them to try harder to make sense of what the narrator is saying). These features allow viewers to more effectively process and understand the information being conveyed in the videos. Meaningful connections between the related-behavior videos and the challenge task are heavily implied but left to learners to construct.

The rest of the lesson structure uses a partial scaffolding approach (Puntambekar and Hubscher 2005) to assist learners in constructing a solution to the problem-solving task. The videos act as a direct instructional anchor to a range of incrementally harder virtual challenges (e.g., "Check Your Understanding" questions, leading to Mini-Challenges, and ultimately the chapter-level Challenge). Direct instruction is *faded* in later steps (requiring learners to apply skills more independently) and also across lessons (e.g., using simplified movement functions in early lessons but requiring control of independent motors in later ones). Collectively, these resources provide teachers a connected strand of resources to support learners with varying initial competencies, as they develop their understanding toward the level needed to construct a solution to the final challenge.

2.2.2 Computational Thinking. Assessments for computational thinking were distributed to students via an online curriculum platform, as part of an evaluation of the online robotics curriculum being tested in these classrooms. For the studies reported here, three different assessment versions were created, each using different cover scenarios to present 17 structurally isomorphic Computational Thinking questions, with only minor adaptations made to make the items sensible for each scenario (see Appendix A for sample assessment items). The purpose of alternative versions was to allow for longitudinal assessment of the same learners while avoiding large test/retest effects. The assessment items consisted of 17 multiple-choice questions that measured students' understanding of CS principles, such as algorithm development, iteration, and Boolean logic (see Table 2). Each item was scored out of one point, for a possible score range of 0 to 17. Armor's theta for the 17 items on these assessments was 0.77. Armor's theta is similar to Cronbach's alpha, but is more appropriate for binary data (item correct vs. incorrect). An Armor's theta value greater than 0.7 is considered "adequate" for demonstrating internal reliability.

Each assessment presented two real-world scenarios. The first scenario assessed students' understanding of computational thinking concepts within a nonrobotics context that was relatively

Table 3. Pre- and Posttest Means for Sixth Graders in Study 1 by Gender, with SD, Sample Size and 95% Confidence Intervals for Mean Difference

	Assessment						95% CI for Mean Diff.
	Pre			Post			
	M	*SD*	*n*	*M*	*SD*	*n*	
Male	6.1	3.9	64	8.2	3.2	62	1.1, 3.1
Female	6.6	3.1	59	8.3	3.1	59	0.7, 2.7
Total	6.3	3.5	123	8.2	3.1	123	1.2, 2.6

similar to that presented within the robotics curriculum (e.g., GPS navigation for self-driving cars). The second scenario assessed the application of the same computational thinking concepts in a more dissimilar context (e.g., filling tables at a restaurant from a queue). The increasing contextual distance of the items (from robotics) was intended to assess whether participation in the robotics curriculum developed problem-solving strategies that could transfer to nonrobotics tasks.

3 STUDY 1: PILOT PROGRAM AND INITIAL EVALUATION

3.1 Sample

Our initial study consisted of a cohort of sixth graders ($N = 123$) at a single suburban school district in southwestern Pennsylvania. The sample was relatively equally divided by gender (male = 52%, female = 48%). The sample was drawn from a school district that is primarily (99%) white and has a relatively low percentage of students eligible for free or reduced lunch (12%). A single teacher elected to be part of the study and taught the same curriculum to all students across five classes throughout the semester. All classes were 40 minutes long and held 5 days a week.

3.2 Procedure and Analysis

The pretest was administered to students at the end of their fifth-grade school year, after completing the Basic Movement unit (i.e., before having reached the more complex content). The following school year, students were enrolled in the robotics course as sixth graders, this time beginning the curriculum with the Sensors unit and continuing until completing the Program Flow unit. Students were then assessed on an analogous posttest assessment after participating in a semester-long robotics course using the curriculum; equivalence of the posttest is established in the second study discussed later. A paired-samples *t*-test was used to determine if there were significant differences between pre- and posttests.

3.3 Results

Results from the *t*-test showed that there were significant differences overall between pre- and posttest scores, $t(122) = 5.44$, $p < .001$, $d = 0.57$. Important for our goals of expanding access of computational thinking, these gains were also not differential by gender; there were no significant differences found between male and female students (see Table 3) on pretest, $t(121) = 0.73$, $p = .47$, or posttest, $t(143) = 0.05$, $p = .96$. It is important to note that for this initial study, the expanded time of implementation could have increased the effect size. We also do not know if a significant number of students participated in any robotics enrichment activities (i.e., summer camps, robotics competition teams) over the summer break between the pre- and posttest assessment dates, which could feasibly impact gains found on the posttest. However, we believed that the results found in this initial evaluation were meaningful enough to warrant further studies of the association between the curriculum and gains on the assessments of computational thinking.

4 STUDY 2: ASSESSMENT VALIDATION AND LEARNING GAINS

4.1 Sample

The sample for the second study consisted of sixth-, seventh-, and eighth-grade ($N = 441$) students in multiple robotics classrooms across four schools in southwestern Pennsylvania. Students were distributed across 26 different robotics classrooms within four suburban school districts and were split relatively evenly by gender, with slightly more males (56%) than females (44%). These districts serve student populations that are majority white (87%–99%) and represent a moderate range of free or reduced lunch eligibility (5%–35%). The four teachers participating elected to be part of the study and to use the tests and the associated online curriculum within their regularly scheduled robotics classes. All classes were between 40 and 60 minutes long and held 5 days a week. Most class sessions ran for about 5 to 7 weeks, with a mean of 34 class days. The teachers had each received approximately 32 hours of professional development (about 4 days) on the implementation of the curriculum.

4.2 Procedure and Analysis

4.2.1 Computational Thinking Assessments. Students were randomly assigned to one of three analogous versions of the computational thinking assessment ("Version A," "Version B," or "Version C") for the pretest, and were later assigned to take an alternate version as a posttest. Pretests were administered prior to the start of instruction with the online curriculum materials; posttests were administered at the end of the class instructional period, which ranged from 5 to 7 weeks. For both pre- and posttests, observations in which at least half of the items were not completed were excluded from analysis, as too little information was collected to make a valid inference and there were likely additional factors (e.g., attendance, engagement) influencing student responses. If students completed more than half of the items but left some items unanswered, those responses were marked as a zero.

The assessment data were first analyzed by version using a one-way ANOVA to determine if there were statistically significant differences in pretest scores between Version A, Version B, and Version C of the computational thinking assessments. Next, students' pre- and posttest data were matched and a paired-samples t-test was performed to determine if there were significant differences in individual gains in pre- and posttest scores overall across grade levels. Finally, an ANCOVA was performed to test if there was a significant association between student completion of additional units of the curriculum and learning gains on the pre- and posttests. Effect sizes are reported as Cohen's D, using the conventional thresholds for small ($d = .20$), medium ($d = .50$), and large ($d = .80$) effects (Cohen 1992).

4.2.2 Progress. Progress was measured based on completion of assignments and formative quizzes, which were then compiled into a total progress score for each unit. As lessons included both optional and repeated content, completing a unit did not require finishing every available activity. If a student completed more than 50% of the available activities for a particular lesson within a unit, that student was marked as completing that particular lesson. Because some units offered multiple opportunities to practice similar lessons and teachers may have students skip over these, students who completed at least 40% of the lessons in the unit were considered to have fully engaged with the concepts in that unit, and therefore were marked as having completed it. Progress was then recoded as an ordinal variable indicating increasing progress through the curriculum (Introduction = 1, Basic Movement = 2, Sensors = 3, and Program Flow = 4).

Table 4. Pre- and Posttest Means in Study 2, with SD, Sample Size and 95% Confidence Intervals for Mean Difference

	Pre			Post			95% CI for
	M	*SD*	*n*	*M*	*SD*	*n*	Mean Diff.
Total	7.2	3.1	364	7.8	3.4	364	0.3, 0.9

The Introduction unit primarily consisted of non-programming-related instruction of how to set up the software and navigate the course menus (see Table 1). As a result, after their first group of students, many teachers felt they had sufficient understanding and decided to skip this unit for subsequent groups. Therefore, in the analyses presented here, progress measures are reported for only the content-based units of Basic Movement, Sensors, and Program Flow. Observations reporting no progress were used only to check for test-retest effects.

4.3 Results

4.3.1 Testing the Equivalence of Versions. For the initial analysis, a one-way ANOVA of pretest scores across all students who completed the test ($n = 426$) was conducted to determine the reliability of the different versions of the assessment, by comparing the relative performance for students who took Version A ($n = 97$, $M = 7.6$, $SD = 3.2$), Version B ($n = 208$, $M = 7.1$, $SD = 3.2$), and Version C ($n = 121$, $M = 6.8$, $SD = 2.8$). Results showed no significant differences in means between any of the test versions, $F(2{,}425) = 1.96$, $p = .14$. Therefore, as these appear to be relatively analogous measures of computational constructs, results from all three versions will be collapsed in further analyses. To check for the possibility of test-retest effects, we also examined pre/post differences between $n = 26$ students who reported no progress through the curriculum. Overall, this group of students had slightly lower pretest scores from students who completed the curriculum, $t(326) = 2.06$, $p <$ .05, $d = 0.40$, but were otherwise comparable (i.e., from the same classrooms and school districts as the group who made progress through the curriculum). Results for the no-progress group show marginal gains, $t(26) = 2.02$, $p = .054$, $d = 0.40$, representing a small test-retest effect. However, as explained further in the results later, this effect size is comparable to those from the Basic Movement unit and does not account for larger gains found in the Sensors and Program Flow units.

4.3.2 Overall Gains. Next, we tested overall mean differences in gains from pre- to posttests by examining matched samples of students across all four schools that completed both pre- and posttests ($n = 364$). Results showed there was a small but statistically significant overall mean gain of 0.57 points from pretest to posttest, $t(363) = 3.62$, $p < .001$, $d = 0.18$ (see Table 4).

While the overall mean difference in pre-/posttest gains found here is a relatively small effect size ($d = 0.18$) compared to the medium effect size for pre-/posttest gains found in Study 1 ($d =$ 0.57), we believe that the wider range of students introduced in this study, as well as the shorter timeline for implementation, could have played a role in the decreased pre-/posttest gains found. It is important to note that while Study 1 gains were observed over a full semester plus summer break, implementation in Study 2 took place over just 5 to 7 weeks. Therefore, gains in Study 2 could be smaller simply because this also includes students who only reached the more basic units. Follow-up analyses were conducted to examine posttest scores by different levels of progress through the course, to provide insight into how additional course units may be associated with different learning gains for students in Study 2.

4.3.3 Mean Gains by Amount of Curriculum Progress. For this analysis, only students who had completed material beyond the introductory setup lessons were included, $n = 315$. Of these students, $n = 142$ progressed through the Basic Movement unit, $n = 165$ students progressed

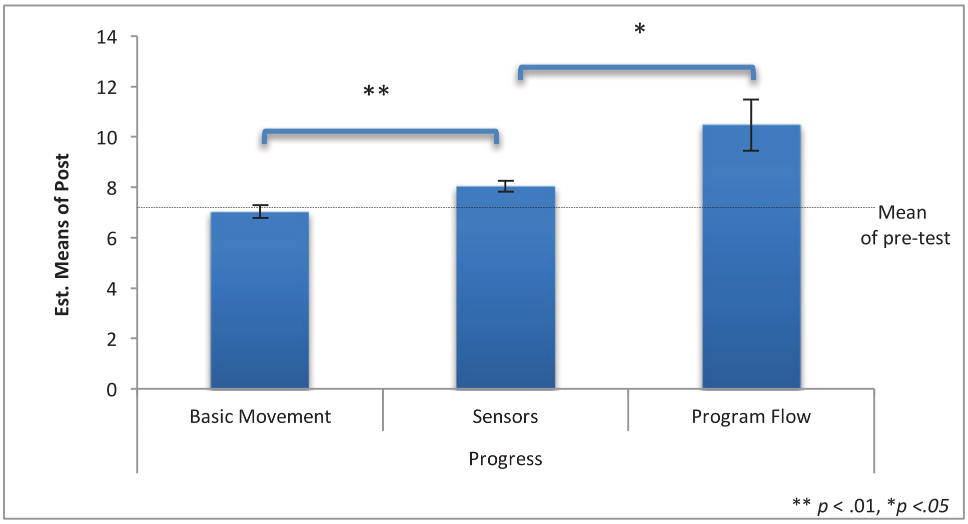


Fig. 4. Estimated marginal means of posttest by curriculum progress in Study 2, controlling for pretest scores.

through the Sensors unit, and $n = 8$ students completed the final Program Flow unit. An ANOVA of pretest scores between these three groups found significant differences, $F(2, 296) = 8.09$, $p < .001$. A post hoc Tukey test showed small differences between the Basic Movement and Sensors groups ($p < .05$), and between the Basic Movement and Program Flow groups ($p < .05$). Differences between the Sensors and Program Flow groups were not significant ($p = .45$). These baseline differences motivated the use of an ANCOVA on posttest scores, to adjust for these pre-existing differences. The ANCOVA on students' posttest scores tested for significant differences by the levels of progress they made through the curriculum, while controlling for pretest scores. The ANCOVA model showed there to be significant association of progress with posttest scores, $F(2, 258) = 8.14$, $p < .001$ (see Figure 4).

A post hoc Tukey test showed that there were significant differences of about 1 point between posttest scores for Basic Movement and Sensors at the $p < .01$ level ($d = .34$), as well as significant differences of about 2.4 points between Sensors and Program Flow at the $p < .05$ level ($d = .86$). The differences of about 3.4 points between Basic Movement and Program Flow were found to be significant at the $p < .01$ level ($d = 1.16$). Therefore, while there are significant gains observed for groups of students who complete each additional unit of the curriculum, the largest differences in scores were observed between groups of students who completed the Basic Movement and Program Flow units. Furthermore, Figure 4 also shows mean pretest levels, revealing that those who had completed only Basic Movement had made no gains, those who had completed Sensors had made modest gains, and those who had completed Program Flow showed sizeable gains. This pattern is to be expected, since Basic Movement involves little complexity, while Program Flow is where computational thinking is required most intensively. Importantly, differential gains by progress in the curriculum rules out the possibility that the gains observed are merely a result of test-retest effects, such as pretest exposure to the structure of the assessment.

5 GENERAL DISCUSSION

In the two studies reported, we contribute to a growing body of literature around new educational technologies aiming to provide programming instruction that teaches generalizable computational thinking skills. Specifically, we evaluate the association between implementation of a particular online virtual robotics curriculum and students' computational problem-solving abilities in both similar and dissimilar programming contexts. Results from the analysis of mean pre- to posttest gains in both Study 1 and Study 2 showed relatively small, yet significant, overall gains in performance on nonrobotics computational thinking tasks. When examining these effects by the amount of progress that students are able to make through the curriculum, however, we observed that significantly larger learning gains occurred for groups of students who reach the more content-rich Sensors and Program Flow units. Thus, students were able to learn generalizable skills, despite being embedded in a context that placed strong emphasis on a particular context (i.e., robotics).

5.1 Limitations

The results of this study are limited by a number of factors. First, the lack of a random-assignment control group and the correlational relationship between progress and gains does not address causality. Therefore, we cannot be certain that it was in fact exposure to the curriculum that caused the observed gains in student scores. For example, it is possible that other unobserved factors, such as supplemental materials developed by the teacher, or other student characteristics such as age or class attendance accounted for students' ability to move further into the curriculum and score higher on the posttest. In addition, the extent to which teachers who had access to and incorporated physical robotics into the virtual curriculum is likely to have varied greatly, as school districts had differential access to these resources, and some teachers may have been affiliated with physical robotics teams at their schools. Indeed, it will be important for future studies of this curriculum that teacher input is gathered in order to determine what additional teacher materials and supports used or developed during implementation might also have contributed to the successful implementation of the curriculum found here.

5.2 Practical Considerations

One issue raised by our results is that unlike in the relatively controlled environment of Study 1, many classrooms implementing the curriculum in Study 2 were unable to consistently progress past more fundamental units, often stopping before fully completing either the Basic Movement or Sensors units. While this speaks to the real-world validity of our data, the phenomenon itself suggests that additional issues will need to be addressed before widespread implementation is feasible. Informal exploratory classroom observations throughout the implementation of Study 2 suggest a few possible hypotheses.

First, a major barrier in reaching these more conceptually rich units may be teacher content knowledge, as well as a lack of specific pedagogical preparation to teach higher-order programming concepts. Few states in the United States, including Pennsylvania, where this study was conducted, require teachers to obtain specific CS certifications, and often courses like robotics that do teach CS and programming skills fall under general technical education electives as disparate as Business or Career, Technical, and Agricultural Education (Guzdial and Ericson 2014). For example, many teachers implementing the curriculum in Study 2 were certified under Pennsylvania's Technology Education certification. The 2015 Educational Testing Service's Praxis certification exam for Technology Education contains no computer-programming-specific questions, instead primarily assessing topics like Manufacturing and Construction Technologies or user-facing computing applications like how to operate software programs (Educational Testing Service 2015). Informal observations showed that generally teachers in Study 2 progressed linearly though each activity, lesson, and unit, focusing on completion of each section rather than emphasizing conceptual understanding of the particular programming concept being targeted by each lesson. Additionally, much of the feedback provided to students struggling with more complex programming challenges consisted of line-by-line comparison of students' code with example solutions that were either previously developed by the teachers or provided within the curriculum.

We hypothesize that as novice programmers themselves, teachers' lack of understanding of the underlying principles of programming could reduce their capacity to notice and use general structural similarities between "correct" code and student code (Catrambone 1998; Kurland et al. 1986; Robins et al. 2010). Furthermore, teachers' interpretation of the pedagogical requirements for ambitious CS instruction may also be focused on the surface level, rather than structural features of these reform curriculum materials (Coburn 2001; Spillane 2000). This can both lead to a focus on superficial features of the robotics programming tasks that extend the time of implementation and inhibit teachers from moving students into more demanding and rich aspects of the curriculum where they are less confident in their abilities. Existing research on teacher professional development in problem-based learning and technology-rich contexts offers some suggestions on how to support teachers' implementation of reform curricula using content-based collaborative inquiry (CBCI), where teachers collaboratively construct their own pedagogical content knowledge through engaging in authentic classroom experiences (Doppelt et al. 2009; Harris 2011; Zech 2000). CBCI professional development experiences could develop teacher programming in ways that shift instruction away from rote line-by-line programming and focuses more on the underlying computational thinking practices. Additional research is needed to better understand these and other pedagogical barriers that prevent students from progressing into more complex units.

A second pattern observed in classrooms during implementation of the curriculum in Study 2 was teachers' intermittent use of physical robotics to supplement the virtual curriculum. The amount to which teachers incorporated physical robotics in the classroom and the reasons they provided for using them differed; however, two main themes emerged. First, many teachers claimed that the use of physical robotics improved students' engagement and motivation to participate in class. Second, they attributed increased learning gains to the additional practice of programming skills in a similar context using physical robots. There is a growing body of literature suggesting a relationship between teachers' beliefs about technology, pedagogy, and content and their ability to effectively integrate technology in their practice (Ge 2001; Kim 2013; Wang 2004). Therefore, many teachers may continue to intersperse physical robotics activities throughout the virtual curriculum as a means of re-engagement for students, which could further increase the time required for students to progress through units containing higher-order computational concepts. The additional cost of purchasing physical robot hardware also decreases access to this type of curriculum for underserved students. It is unclear from the data collected here what effects on student learning were associated with the differential use of physical robotics in conjunction with the virtual curriculum. While previous studies with a similar virtual robotics curriculum showed no significant differences between learning gains for students who participated in a virtual or physical version of the curriculum (Liu 2013), other researchers have suggested that the embodied nature of physical robotics may confer learning benefits for novices in programming and mathematics (Bruner and Anglin 1973; Silk 2009). Further studies are needed to demonstrate if similar results exist for differences in computational thinking gains as a function of physical or virtual representation of this particular programming curriculum, and if there might in fact be higher levels of motivation and engagement observed with students in hybrid physical and virtual robotics environments.

Lastly, a related hypothesis as to why many classes struggled to reach more conceptually rich programming units is that while presenting programming material in a robotics context may help maintain higher levels of engagement in students, curricular activities that address robotics-specific features are often time consuming, and occur at the expense of more computationally rich programming activities. Even in the virtual context, robotics-specific content such as sensor function and motor port location may direct students' attention away from programming skills and toward more mechanical aspects of the robot's operation, and add additional time to the curriculum without necessarily increasing their exposure to computational thinking content. For example, understanding mechanically how a touch sensor receives and transmits data to a robot may be a less powerful introduction to key computational thinking concepts for students than understanding how the data that it receives is represented as a Boolean value within a program. As with many interdisciplinary curricula, certain lessons and sections of units may be more focused on teaching critical aspects of one discipline over another, and may have different goals of developing student engagement, specific content understanding, or broader skills and practices. It is important, then, in developing and preparing interdisciplinary curricula, to select and optimize units to directly target key concepts. Concretely, in the tested curriculum, it is likely more students could have progressed to the Program Flow unit by simplifying the number of sensors included in the Sensors unit.

6 CONCLUSION

Currently, there is an effort within the CS community, most recently demonstrated in the announcement of the Computer Science for All initiative in the United States, to broaden the scope of CS learning opportunities in K-12, citing both economic- and equity-based concerns from the field (Ericson 2014; Goode 2011). Our studies suggest that curricula using new technologies such as visual programming languages and simulated robots to teach programming in concrete contexts may also facilitate learning of generalizable CS skills. Further investigation into how those affordances interact with other features of robotics learning environments, such as the incorporation of physical robots and differential levels of teacher pedagogical support, could offer additional insight into how to develop rich, authentic CS learning environments that produce the ambitious and dynamic learning outcomes necessary for 21st-century CS learners.

APPENDIX A: SAMPLE ASSESSMENT ITEMS

Scenario 1A:

AutoAutomobile is a company developing a system that will convert normal cars into self-driving, "autonomous" cars.

Section 1: Navigation

When the driver puts in where they want to go, the system uses GPS to find five possible paths from the car's current location to the goal location. The system then uses the distance and speed limit to calculate the expected travel time for each path.

1. Which of the following equations correctly calculates the expected travel time, based on the distance and average speed limit along each path?

Select one:

a. Average speed limit × Total time × Path distance
b. Average speed limit ÷ Path distance
c. Path distance ÷ Average speed limit
d. Path distance × Average speed limit

2. The system now knows the expected travel time along five different paths. It must then pick the one with the shortest travel time.

Will the following set of steps (called an *algorithm*) correctly identify the path with the shortest travel time?

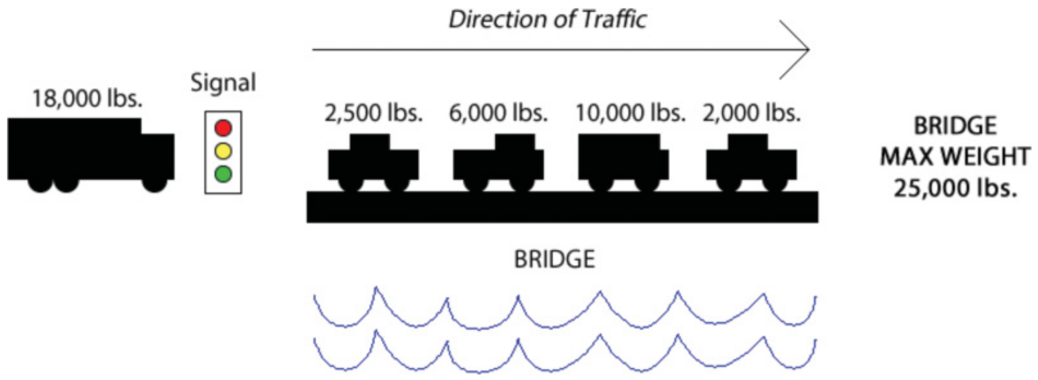
- **Step 1.** Number the five paths 1–5.
- **Step 2.** Initially, mark Path #1 as the "shortest path."
- **Step 3.** Then, for each Path from #2 through #5:
  - **Step 3a.** Check whether that path is shorter than the existing "shortest path."
  - **Step 3b.** If it is, mark this as the new "shortest path."
  - **Step 3c.** Repeat this process for the next path, until they have all been checked.
- **Step 4.** Whichever path is marked the "shortest path" at the end is the shortest path.

Select one:

a. Yes, this algorithm will correctly identify the shortest path
b. No, it will always declare Path #1 to be the shortest, even if it is not
c. No, it will always declare the last path to be the shortest, even if it is not
d. The algorithm only works if the paths are already sorted in order of travel time

Scenario 2A:

A civil engineer is developing a system for a new bridge that uses automated traffic signals. The traffic signal is designed so that the total weight of trucks and cars allowed on the bridge never exceeds a certain amount. A weight sensor measures the weight of each car or truck BEFORE it gets onto the bridge, and a second (simpler) sensor detects when the car or truck reaches the other side and gets off the bridge.



3. In the previous picture, the large truck on the left is waiting to get on. The maximum weight allowed on the bridge is 25,000 lbs. When should the light turn green?

Select one:

a. After the next car has gotten off
b. After the next two cars have gotten off
c. After the next three cars have gotten off
d. After all four cars have gotten off

4. The system should give a car the green light to get onto the bridge as soon as...?

Select one:

a. There is enough space in front of the car that it will not collide with the car in front of it
b. The additional weight of the new car does not cause the total weight on the bridge to exceed the safe maximum

c. The weight of the new car does not exceed the maximum total weight
d. The light has been red for at least 10 seconds

APPENDIX B: SAMPLE ANALOGOUS ASSESSMENT ITEMS

Scenario 1B:

Excellent Electric designs all-electric cars. Some of these cars have new kinds of automation.

Section 1: Power Consumption
There are very few electric car charging stations, so electric cars must be careful about the amount of electricity used in a trip. So, the computer system calculates six different paths to a destination to find the path that will use the least electricity.

1. Which of the following calculations correctly determines the expected amount of electricity used, based on those parameters?

Select one:

a. Average electricity used per mile × Total time × Path distance
b. Path distance ÷ Average electricity used per mile
c. Path distance × Average electricity used per mile
d. Average electricity used per mile ÷ Path distance

2. The system now knows the electricity used along each of the paths. It must then recommend the route that uses the least electricity.

Will the following set of steps (called an *algorithm*) correctly identify the path that uses the least electricity?

- **Step 1.** Number the six routes 1–6.
- **Step 2.** Initially, mark Path #1 as the "least electricity used."
- **Step 3.** Then, for each Path from #2 through #6:
  - **Step 3a.** Check whether that path's electricity use is *bigger* than the existing "least electricity used."
  - **Step 3b.** If it is, mark this as the new "least electricity used."
  - **Step 3c.** Repeat this process for the next path, until they have all been checked.
- **Step 4.** Whichever path is marked the "least electricity used" at the end is the path that uses the least electricity.

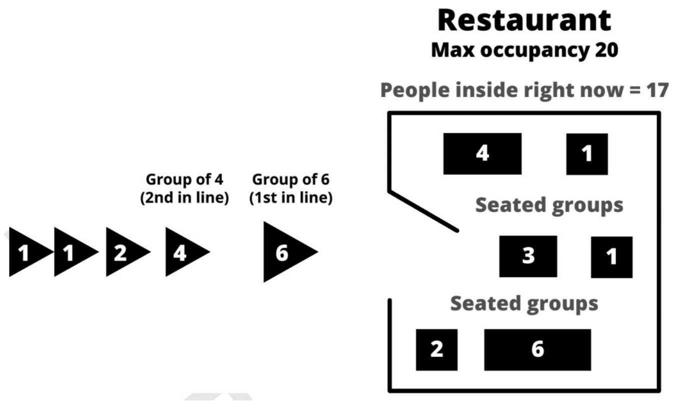
Select one:

a. Yes, this algorithm will correctly identify which path uses the least electricity
b. No, it will always say Path #1 uses the least electricity, even if it doesn't
c. No, it will always declare the last path to use the least electricity, even if doesn't
d. No, the algorithm will find the path the uses the MOST electricity

Scenario 2B:

A popular restaurant has long lines around lunchtime. In order to be fair, the manager of the restaurant starts a "first come, first served" policy where groups of diners are served <u>in the order they arrive</u>.

Seats can be moved around, so there is no need to wait for larger or smaller tables. When a group is done eating, they pay and leave together, although <u>groups do not always leave in the same order they arrived.</u>



3. The restaurant (shown previously) has room for 20 people. The line of groups waiting outside is shown, along with the number of people in each group, and the order that the parties are lined up in. There are currently 17 people in the restaurant, in groups as shown. When should the first group in line be let in?

Select one:

a. As soon as the next group leaves the restaurant
b. As soon as the next two groups leave the restaurant
c. After groups totaling three or more people have left
d. After all 17 people inside the restaurant leave

4. In general, a restaurant should let the next group in as soon as…?

Select one:

a. There is enough space in the doorway for the next group to stand in it
b. The additional people in the new group do not cause the total number of people in the restaurant to exceed the maximum allowed
c. The size of the new group is less than the allowed maximum
d. The group has been in line for at least 15 minutes

REFERENCES

ACM Education Policy Committee. 2014. *Rebooting the Pathway to Success: Preparing Students for Computing Workforce Needs in the United States*. Retrieved from http://pathways.acm.org/ACM_pathways_report.pdf.

D. Alimisis and C. Kynigos. 2009. Constructionism and robotics in education. In D. Alimisis (Ed.), *Teacher Education on Robotics-Enhanced Constructivist Pedagogical Methods*. School of Pedagogical and Tech. Education, Athens, Greece, 11–26.

D. Alimisis. 2012. Robotics in education & education in robotics: Shifting focus from technology to pedagogy. In *Proceedings of the International Conference on Robotics in Education*. Matfyz Press, 7–14.

R. D. Atkinson and M. Mayo. 2011. *Refueling the U.S. Innovation Economy: Fresh Approaches to Science, Technology, Engineering and Mathematics (STEM) Education*. The Information Technology and Innovation Foundation. Retrieved from http://www.itif.org/files/2010-refueling-innovation-economy.pdf.

V. Barr and C. Stephenson. 2011. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads* 2, 1, 48–54. DOI: 10.1145/1929887.1929905

F. B. V. Benitti. 2012. Exploring the educational potential of robotics in schools: A systematic review. *Computers and Education* 58, 3, 978–988. DOI: 10.1016/j.compedu.2011.10.006

K. Brennan and M. Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the American Educational Research Association*. 1–25. Retrieved from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf.

R. Catrambone. 1998. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General* 127, 4, 355–376.

D. H. Clements and D. F. Gullo. 1984. Effects of computer programming on young children's cognition. *Journal of Educational Psychology* 76, 6, 1051–1058. DOI:10.1037/0022-0663.76.6.1051

J. Cohen. 1992. A power primer. *Psychological Bulletin* 112, 1, 155–159. DOI:10.1037/0033-2909.112.1.155

Y. Doppelt, C. D. Schunn, E. M. Silk, M. M. Mehalik, B. Reynolds, and E. Ward. 2009. Evaluating the impact of a facilitated learning community approach to professional development on teacher practice and student achievement. *Research in Science & Technological Education* 27, 3, 339–354. DOI:10.1080/02635140903166026

Educational Testing Service. 2015. *The Praxis Study Companion: Technology Education.* Educational Testing Service, Princeton, NJ. Retrieved from https://www.ets.org/s/praxis/pdf/5051.pdf.

A. Eguchi. 2014. Learning experience through robocupjunior: Promoting engineering and computational thinking through robotics competition. In *Proceedings of the American Society for Engineering Education.* Retrieved from https://peer.asee.org/20743.

B. Ericson and M. Guzdial. 2014. Measuring demographics and performance in computer science education at a nationwide scale using AP CS data. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14).* ACM Press, New York, 217–222. DOI:10.1145/2538862.2538918

J. Goode and J. Margolis. 2011. Exploring computer science. *ACM Transactions on Computing Education* 11, 2, 1–16. DOI:10.1145/1993069.1993076

S. Grover and R. Pea. 2013. Computational thinking in K-12: A review of the state of the field. *Educational Researcher* 42, 1, 38–43. DOI:10.3102/0013189X12463051

M. Guzdial, B. Ericson, T. Mcklin, and S. Engelman. 2014. Georgia computes! An intervention in a US state, with formal and informal education in a policy context. *ACM Transactions on Computing Education (TOCE)* 14, 2, 13.

S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking. 2009. A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin* 41, 183. DOI:10.1145/1539024.1508931

J. B. Harris and M. J. Hofer. 2011. Technological pedagogical content knowledge (TPACK) in action: A descriptive study of secondary teachers' curriculum-based, technology-related instructional planning. *Journal of Research on Technology in Education* 43, 3, 211–229.

C. Kelleher and R. Pausch. 2005. Lowering the barriers to programming. *ACM Computing Surveys* 37, 2, 83–137. DOI:10.1145/1089733.1089734

G. Khoury. 2007. *CSTA Certification Committee Report: Computer Science State Certification Requirements.* Computer Science Teachers Association, New York. Retrieved from http://www.csta.acm.org/ComputerScienceTeacherCertifi cation/sub/TeachCertRept07New.pdf.

C. Kim, M. K. Kim, C. Lee, J. M. Spector, and K. DeMeester. 2013. Teacher beliefs and technology integration. *Teaching and Teacher Education* 29, 1, 76–85. DOI:10.1016/j.tate.2012.08.005

D. Klahr and S. M. Carver. 1988. Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology* 20, 3, 362–404. DOI:10.1016/0010-0285(88)90004-7

K. H. Koh, A. Repenning, H. Nickerson, Y. Endo, and P. Motter. 2013. Will it stick? Exploring the sustainability of computational thinking education through game design. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE'13).* ACM Press, New York, 597–602. DOI:10.1145/2445196.2445372

R. Kurland, C. Pea, and Clement R. Mawby. 1986. A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research* 2, 4, 429–458

J. Lave and E. Wenger. 1991. *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press, Cambridge, UK.

I. Lee, F. Martin, and K. Apone. 2014. Integrating computational thinking across the K–8 curriculum. *ACM Inroads* 5, 4, 64–71. DOI:10.1145/2684721.2684736

A. S. Liu, J. Newsom, C. Schunn, and R. Shoop. 2013. Students learn programming faster through robotic simulation. *Tech Directions* 72, 16–19. Retrieved from http://www.education.rec.ri.cmu.edu/content/educators/research/files/p16-19 Shoop et al.pdf.

A. S. Liu, C. D. Schunn, J. Flot, and R. Shoop. 2013. The role of physicality in rich programming environments. *Computer Science Education* 23, 4, 315–331. DOI:10.1080/08993408.2013.847165

S. Y. Lye and J. H. L. Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41, 51–61. DOI:10.1016/j.chb.2014.09.012

L. Major, T. Kyriacou, and O. P. Brereton. 2012. Systematic literature review: Teaching novices programming using robots. *IET Software* 6, 6, 502. DOI:10.1049/iet-sen.2011.0125

R. E. Mayer. 1974. Acquisition processes and resilience under varying testing conditions for structurally different problem-solving procedures. *Journal of Educational Psychology* 66, 5, 644–656. DOI:10.1037/h0037482

R. E. Mayer. 2008. Applying the science of learning: evidence-based principles for the design of multimedia instruction. *American Psychologist* 63, 8, 760–769. DOI:10.1037/0003-066X.63.8.760

R. E. Mayer and J. G. Greeno. 1972. Structural differences between outcomes produced by different instructional methods. *Journal of Educational Psychology* 63, 12, 165–173. DOI:10.1037/h0032654

A. Melchior, F. Cohen, T. Cutter, and T. Leavitt. 2005. *More Than Robots: An Evaluation of the FIRST Robotics Competition Participant and Institutional Impacts*. Brandeis University Heller School for Social Policy and Management, Waltham, MA.

National Research Council, Committee for the Workshops on Computational Thinking. 2010. *The Scope and Nature of Computational Thinking*. National Research Council, Washington, DC. Retrieved from http://www.nap.edu/catalog/12840.html.

National Science Board. 2015. *Revisiting the STEM Workforce, A Companion to Science and Engineering Indicators 2014*. National Science Board, Arlington, VA. Retrieved from http://www.nsf.gov/nsb/publications/2015/nsb201510.pdf.

S. Y. Okita. 2013. The relative merits of transparency: Investigating situations that support the use of robotics in developing student learning adaptability across virtual and physical computing platforms. *British Journal of Educational Technology* 45, 5, 844–862. DOI:10.1111/bjet.12101

D. B. Palumbo. 1990. Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research* 60, 1, 65–89. DOI:10.3102/00346543060001065

R. Pea. 1983. LOGO Programming and problem solving. In *Proceedings of the American Educational Research Association (AERA'83)*. Retrieved from https://web.stanford.edu/~roypea/RoyPDF%20folder/A39_Pea_87d_CCT_TR_MS.pdf.

S. Puntambekar and R. Hubscher. 2005. Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed? *Educational Psychologist* 40, 1, 1–12. Retrieved from http://doi.org/10.1207/s15326985ep4001.

A. Repenning, D. Webb, and A. Ioannidou. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10)*. ACM, New York, 265–269. DOI:10.1145/1734263.1734357

A. Robins, J. Rountree, and N. Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education* 13, 2, 137–172. DOI:10.1076/csed.13.2.137.14200

G. Salomon and D. N. Perkins. 1987. Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research* 3, 149–170.

L. S. Shulman. 1986. Those who understand: knowledge growth in teaching. *Educational Researcher* 15, 2, 4–14. DOI:10.3102/0013189X015002004

E. Silk, C. Schunn, and R. Shoop. 2009. Synchronized robot dancing: Motivating efficiency & meaning in problem solving with robotics. *Robot Magazine* 17, 42–45.

L. Wang, P. A. Ertmer, and T. J. Newby. 2004. Increasing preservice teachers' self-efficacy beliefs for technology integration. *Journal of Research on Technology in Education* 36, 3, 231–250. DOI:10.1080/15391523.2004.10782414

D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, and U. Wilensky. 2016. Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology* 25, 1, 127–147. DOI:10.1007/s10956-015-9581-5

L. Werner, J. Denner, S. Campe, and D. C. Kawamoto. 2012. The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*. ACM, New York, 215–220. DOI:10.1145/2157136.2157200

J. M. Wing. 2006. Computational thinking. *Communications of the ACM* 49, 3, 33. DOI:10.1145/1118178.1118215

L. K. Zech, C. L. Gause-Vega, M. H. Bray, T. Secules, and S. R. Goldman. 2000. Content-based collaborative inquiry: A professional development model for sustaining educational reform. *Educational Psychologist* 35, 3, 207–217. DOI:10.1207/S15326985EP3503_6

Received December 2016; revised May 2017; accepted May 2017