Received: 4 October 2021 | Revised: 26 April 2022 | Accepted: 7 August 2022

DOI: 10.1111/jcal.12720

ARTICLE

Journal of Computer Assisted Learning WILEY

Discovering computational thinking in everyday problem solving: A multiple case study of route planning

Ndudi O. Ezeamuzie[1] | Jessica S. C. Leung[1] | Raycelle C. C. Garcia[2] | Fridolin S. T. Ting[2]

[1]Faculty of Education, University of Hong Kong, Pokfulam, Hong Kong

[2]Department of Applied Mathematics, The Hong Kong Polytechnic University, Kowloon, Hong Kong

Correspondence
Ndudi O. Ezeamuzie, Faculty of Education, University of Hong Kong, Pokfulam, Hong Kong.
Email: amuzie@connect.hku.hk

Funding information
Research Grants Council, University Grants Committee, Grant/Award Number: HKBU1/T&L/16-19

Abstract

Background: The idea of computational thinking is underpinned by the belief that anyone can learn and use the underlying concepts of computer science to solve everyday problems. However, most studies on the topic have investigated the development of computational thinking through programming activities, which are cognitively demanding. There is a dearth of evidence on how computational thinking augments everyday problem solving when it is decontextualized from programming.

Objectives: In this study, we examined how computational thinking, when untangled from the haze of programming, is demonstrated in everyday problem solving, and investigated the features of such solvable problems.

Methods: Using a multiple case study approach, we tracked how seven university students used computational thinking to solve the everyday problem of a route planning task as part of an 8-week-long Python programming course.

Results and Conclusions: Computational thinking practices are latent in everyday problems, and intentionally structuring everyday problems is valuable for discovering the applicability of computational thinking. Decomposition and abstraction are prominent computational thinking components used to simplify everyday problem solving.

Implications: It is important to strike a balance between the correctness of algorithms and simplification of the process of everyday problem solving.

KEYWORDS
abstraction, algorithm, computational thinking, problem solving, programming

1 | INTRODUCTION

Wing (2006) described computational thinking (CT) as a problem solving approach that incorporates 'concepts fundamental to computer science' (p. 33). Wing's conceptualisation, often regarded as the reference point for CT in the 21st century (Grover & Pea, 2013; Shute et al., 2017), elicited several positive benefits associated with CT. Amongst these, the perception that anybody can apply computer scientists' cognition styles to everyday problems has resonated distinctly among educators, policymakers, and society. Why? The promise that CT may support problem solving, is an objective that speaks directly to the purpose of education. According to Gagné (1985), the crux of education is to make people better problem solvers. Everyone encounters problems daily: What clothes should I wear? Which mode of transport should I take? What topic should I teach? The list of problems we encounter is limitless and includes the regular 'solve these problems' when assessing students' knowledge. In fact, 'most educators regard problem solving as the most important learning outcome for life' (Jonassen, 2000, p. 63).

However, problem solving is a complex and amorphous construct, especially in educational settings where the concept is used loosely to refer to a wide variety of tasks. Although it is still theoretically difficult

to outline all the dimensions of problem solving, a certain level of clarity has emerged over time. For example, problems may be classified into well-structured or ill-structured (Jonassen, 1997) and routine or non-routine (Mayer, 1998). In the early 21st century, Jonassen (2000) argued that such dichotomous classifications of problems could be further refined, and he identified how problems varied in terms of their structuredness, complexity, and domain specificity.

Therefore, when Wing (2006) challenged colleagues to share the treasures of computer science with every student through CT, the challenge resonated with most educators, who understood CT to be another skill that could give learners the ability to succeed in life's fundamental learning outcome, problem solving. This heightened importance on CT triggers the need to understand how CT can achieve this goal. To deconstruct what it means to think like a computer scientist, the active CT research community has continued to clarify its framing (e.g., Barr et al., 2011; Brennan & Resnick, 2012; Csizmadia et al., 2015; Korkmaz et al., 2017; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

Most empirical studies measuring participants' development of CT have operationalized it as a composite of programming activity (Ezeamuzie & Leung, 2022). Often, these studies had no clear model of CT distinguishing CT from programming (Ezeamuzie & Leung, 2022; Lye & Koh, 2014). Outside of programming, what types of problems can we solve with CT? While this question resonates deeply with us, we believe it is not peculiar to us but an honest need-to-know by computing educators. Often, students and instructors have confronted us with this basic question. Unfortunately, we have been unable to provide a sound theoretical response. Despite the explicit positioning of CT as 'conceptualizing, not programming' (Wing, 2006, p. 35), there is a dearth of understanding about the nature of the problems that can be solved with CT. More so, Wing (2006) argued, 'To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability' (Wing, 2006, p. 33). How can we justify the elevation of CT's importance to the same level as these time-honoured literacies if the very nature of its solvable problems cannot be clearly expounded? Of course, to construe CT as a panacea for all problems seems delusional. Although some CT empirical investigations have adopted an unplugged approach, programming remains the major way that CT is expressed (Lye & Koh, 2014).

For many, it is hard to articulate the need for CT when its difference from computer programming is nearly undiscernible in practice. Without diminishing the plethora of benefits of learning how to program a computer, the emphasis on CT stems from its affordance to augment problem solving in everyday activities. Yes, at minimum, programming is a beneficial skill and is useful for helping students understand the operations of the pervasive technologies around us. However, the cognitive demand to learn programming is high, and its elevation to a skill that should be learned by every student would probably be excessive. In fact, it was the challenge of learning programming and the concomitant high drop-out from computer science that necessitated Wing's (2006) challenge of computer science educators to teach every student how to use the thinking style of a computer scientist to solve problems. To deliver the primary objective of CT, understanding the nature of problems that are solvable by CT is a gap worth investigating. From an instructional design perspective, it is obvious that not all problems can be solved through CT. Therefore, it is important to discover the features of problems that make them suitable to and solvable through CT. This knowledge will also inform how educators can assess students' CT skills.

We acknowledge upfront that diving into the nature of problem solving is a complex venture. Hence, we are hesitant to generalize our findings, and this is not our aim. Rather, the purpose of this study is to stimulate research, discussion, and empirical validation of the nature of problems that can be solved with CT. This paper reports on a qualitative study that identified the features of everyday problems as framed by two research questions:

> **RQ1.** *How is computational thinking demonstrated in solving everyday problems without programming?*

> **RQ2.** *What are the features of everyday problem solving that uses computational thinking?*

The findings from this review will provide a platform for educators, including those who are interested in joining the CT community, to clearly grasp the conceptualisation of CT and gain *perspectives on how to design pedagogical activities*. In addition, because CT has been positioned as an interdisciplinary cognitive ability, clarifying how CT infuses with the everyday problem will help to make the boundaries between CT research and adjacent fields more permeable, and thus facilitate access to researchers from other fields, who may not be well informed about programming and theoretical computer science.

2 | THEORETICAL FRAMEWORK

In this section, we describe the theoretical context of our investigation by highlighting the key associations between CT and problem solving.

2.1 | Computational thinking

The field of CT assumes that everybody should learn to apply the cognitive strategies of computer scientists when solving problems. Although most educators regard Wing's (2006) discussion of CT as a pivotal moment in the 21st century CT discourse, a report on the historical development of the concept by Tedre and Denning (2016) offered deeper insight into the topic. Specifically, the works of Perils, Knuth, and Papert are worth mentioning. Although he did not use the term 'computational thinking' in 1962, Alan Perils (the first recipient of the renowned A. M. Turing Award from the Association for Computing Machinery) promoted the inclusion of programming in liberal education (Perils, 1962, as cited in Guzdial, 2008). About a decade later, Donald Knuth (a renowned computer scientist) theorized that people would have a better grasp of concepts if they could teach a

computer to perform a task (Nardelli, 2019). In Seymour Papert's seminal work on procedural thinking and constructionism, he argued that children can teach computers to 'think' through programming (Papert, 1980). Together with Wing's CT model, the contributions of Perils, Knuth, and Papert embody the spirit of CT and underscore the association between computer science and human cognition.

According to Wing (2006), CT is reflected through problem solving, system design, recursive thinking, parallel processing, data interpretation, abstraction, decomposition, and heuristic reasoning. This conceptualisation has drawn criticism for being overly broad and somewhat vague (Mannila et al., 2014). Several frameworks have emerged over the years to further clarify how CT can be modelled; these include the International Society for Technology in Education framework (Barr et al., 2011; Barr & Stephenson, 2011), Computing at School (Csizmadia et al., 2015), and others (e.g., Brennan & Resnick, 2012; Korkmaz et al., 2017; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

These multiple conceptualisations are helpful for understanding CT, but their differences can make cross-study comparisons difficult. For example, Aho (2012) defined CT as the cognitive skill required to develop algorithmic solutions for problems, where an algorithmic solution is the 'series of steps that control some abstract machine or computational model without requiring human judgement' (Denning, 2017, p. 33). In contrast with Aho (2012) conceptual definition, a review by Selby and Woollard (2013) described CT as a composite skill having five components: abstraction, decomposition, algorithms, evaluation, and generalization. Other researchers have also modelled CT differently with some overlapping components and pushing for a consensus on CT would be both an uphill task and precarious venture (Ezeamuzie & Leung, 2022). In this study, we relied on Wing's (2006) description of CT as thinking like a computer scientist. This holistic approach circumvents the limitations of other definitions, such as narrow views about CT (Denning et al., 2017; Tedre & Denning, 2016).

2.2 | The nature of problem solving

There is no one-size-fits-all model for framing problem solving. In his seminal work on thinking, Dewey (1910) described problem solving as a five-step process of harmonizing two non-congruous purposes (i.e., a problem): suggestion, intellectualisation, hypothesizing, reasoning, and verification. According to Dewey, the cognitive activities involved in these steps overlap with each other when a person is solving a problem. More recently, the Organization for Economic Co-operation and Development (OECD) identified the core ingredient of problem solving: an activity is considered problem solving when the method of solving the problematic situation is not immediately apparent to the solver (OECD, 2013). There are other significant theories about problem solving, such as the General Problem Solver for information-extracting problems (Simon & Newell, 1971) and the IDEAL (Identify, Define, Explore, Act, Look) problem solver (Bransford, 1993). Although a detailed review of the theories about problem solving is beyond the scope of this investigation, we found that the work of Jonassen in theorizing the nature of problems provided an invaluable model for constructing the meaning of problem solving and its dimensions (Jonassen, 1997, 2000).

Jonassen (1997) identified two broad classifications of problems that humans encounter in their lives: well-structured problems and ill-structured problems. Problems are well-structured when they have clearly defined specifications (i.e., problem statements), a predictive solution process (using rules and principles), and a finite solution. Let us take, for example, a textbook problem that asks students to find the roots of a quadratic equation (e.g., $x^2 - 4 = 0$). The specifications for the problem are well-defined, the process of solving the problem requires applying a finite set of rules (e.g., using the method of completing the square), and the problem has an unknown yet solvable solution ($x = 2$ or -2). These problems are also referred to as word problems, story problems, or classroom problems (Jonassen et al., 2006). In contrast, ill-structured problems have undefined specifications (e.g., missing parameters or unclear goals), divergent solution processes (multiple solution paths or no solution path), and indeterminate solutions (i.e., what is deemed correct is subjective). Ending global warming, conserving the biodiversity of marine life, and choosing a career are examples of ill-structured problems. According to Jonassen (2000), in everyday settings such as the workplace, we encounter ill-structured problems (e.g., Jonassen et al., 2006) more often than we encounter well-structured problems posed in educational contexts.

The classification of problems as either ill-structured or well-structured has several limitations. First, the idea of solving well-structured problems seems to contradict the OECD's (2013) position that problem solving is the search for solutions that are not obvious to the solver. Following from this, deciding whether solving a quadratic equation should be regarded as problem solving would be dependent on the solver's prior knowledge and experience. In addition, the definition of an ill-defined problem fails to distinguish between the problem of designing a stadium to maximize seating without overcrowding and the problem of tackling global warming. In other words, this binary classification of problems based on their structuredness alone does not account for the individual differences between problem solvers or the complexity of problems. Informed by these limitations, Jonassen (2000) elaborated on the nature of problem solving for instructional designers in his demonstration of how problems differ in terms of their structuredness, complexity, domain specificity, representation, and individual differences. Depending on the problem solver's properties (e.g., domain-specific knowledge and experience), problems have variable levels of complexity that scale from simple to complex.

2.3 | CT approach to problem-solving

How has CT been used to solve problems? Some practical examples of embedding CT in classroom problem solving were documented by Yadav et al. (2016). These included using big data analytics in a social studies class to examine the linguistic composition of US presidential inaugural speeches across four centuries and analysing publicly available data in a science class to make sense of greenhouse emissions.

These examples illustrate the variation and complexity in describing the association between CT and problem solving. Although some studies (e.g., Román-González et al., 2017) have reported a strong correlation between CT and problem solving, others (e.g., Çiftci & Bildiren, 2020; Psycharis & Kallia, 2017) have found no significant influence of CT on students' problem solving abilities. However, this disparity is not unexpected due to the diverse conceptualisations of these concepts. For example, unlike the operationalization of problem solving through mathematics test (Psycharis & Kallia, 2017), Román-González et al. (2017) modelled CT as an aggregate of sequence, loop, conditionals, and functions in a multiple-choice test and construed problem solving from an off-the-shelf test that measures learners' speed and flexibility in manipulating logic. Therefore, it is important to understand the context when interpreting the relationship between CT and problem solving. Taking the broad definition of a problem (i.e., harmonizing two non-congruous purposes), CT studies that measured learners' performance offer insights into the nature of problem solving. As the pool of such studies is large, we restrict our discussion below to those studies that targeted the acquisition of problem solving and CT as their learning outcomes.

2.3.1 | Problem solving as a learning outcome

Consistent with the finding that computer programming is the primary way of learning CT (Lye & Koh, 2014), problem solving was identified as the core learning outcome in a review that investigated what students learn through programming (Popat & Starkey, 2019). Problem solving in the reviewed studies was conceived either as the application of mathematical concepts to coding tasks or the enhancement of mathematical skills through programming. For example, in a case study investigating the development of mathematical sense, problem solving was conceived as the ability of kindergarteners to control the behaviour of a virtual character (a ladybug, similar to the Logo programming language turtle) by constructing paths and navigating a maze programmatically (Fessakis et al., 2013). Other studies conceived problem solving based on kindergarteners' performance in a 50-item multiple-choice questionnaire (Çiftci & Bildiren, 2020), primary school students' perception of problem solving in a 24-item self-reported scale (Kalelioglu & Gülbahar, 2014), and high school students' solutions to items in a government-standardized mathematics test (Psycharis & Kallia, 2017). Beyond the mathematical contexts, an ethnographic study by Melander Bowden (2019) investigated epistemics-in-interaction in Scratch programming. In the study, children were instructed to modify the visual and auditory aspects of a game as the assigned problem-solving activity. These studies showed the varied ways that problem solving could be conceived both within and in between domains.

2.3.2 | CT as the operationalized learning outcome

Most CT studies assessed learners' CT skills as the manifestation of problem solving ability. For example, in investigating learners' CT development, problem solving was framed through testing students' ability to draw electronic circuits in a maker activity (Yin et al., 2020), assessing in-service teachers' understanding of CT concepts and practices (Kong et al., 2020), and testing students' ability to create projects in an outreach program (Panskyi et al., 2019). These examples depict the diverse ways that empirical studies construed problem scenarios. Weintrop et al. (2016) discussed how problem solving could be operationalized through four practical areas of CT (i.e., data practices, modelling and simulation, computational problem solving, and system thinking) in science and mathematics classes.

Apart from CT as learning outcomes in diverse problem scenarios, problem solving has been conceptualized as a learning process. This was the case when Ma et al. (2021) explored how a five-tiered problem-solving instructional model influenced the CT skills of primary school students. In the model, students in the experimental group were taught to solve problems following a linear order: (i) identify the problem, (ii) gather data, (iii) generate a solution, (iv) implement the solution, and (v) assess the solution. Students in the experimental group recorded significant improvements in both CT skills and self-efficacy. However, in this model, the nature of problems that were solvable by CT was unclear.

2.4 | CT in everyday activities

Detaching all elements of programming from CT may not be feasible in some scenarios. Nonetheless, the problems we encounter in daily life often do not require complex representations in forms such as programming or arithmetic. In CT discourse, one of the commonly used sets of problems that is closest to everyday problem solving is the Bebras challenge (Dagienė & Sentance, 2016), which is a collection of short age-adaptive questions that model real and non-domain specific problems. Implicitly, studies that have measured CT using the Bebras tasks have strongly associated problem solving with daily activities; notably, these questions could be adapted for learners with low epistemic inclination to programming. For example, Chiazzese et al. (2019) demonstrated how CT correlated with everyday problems by selecting 10 Bebras questions to investigate the effectiveness of robotics laboratory training in developing CT among primary school students. Of course, other tasks that measure performance as problem solving and CT skills contain elements of everyday problems. For example, the challenge to construct paths and navigate a maze programmatically in Fessakis et al. (2013) was an implementation of route planning, which is an everyday problem. However, the way that the solution was represented (i.e., programming) is not a conventional approach that most people would use in their daily lives.

To raise awareness about the use of CT in everyday problem solving, Standl (2017) designed the 'pack your moving box' activity as an example of an everyday problem and challenged students to sketch their idea using a five-step process (understand, abstract, decompose, design, and test) to solve the problem of how to pack and move their boxes efficiently. Although the problem solving challenge improved students' awareness of CT problem solving, Standl found that students struggled in abstraction and decomposition. Other studies

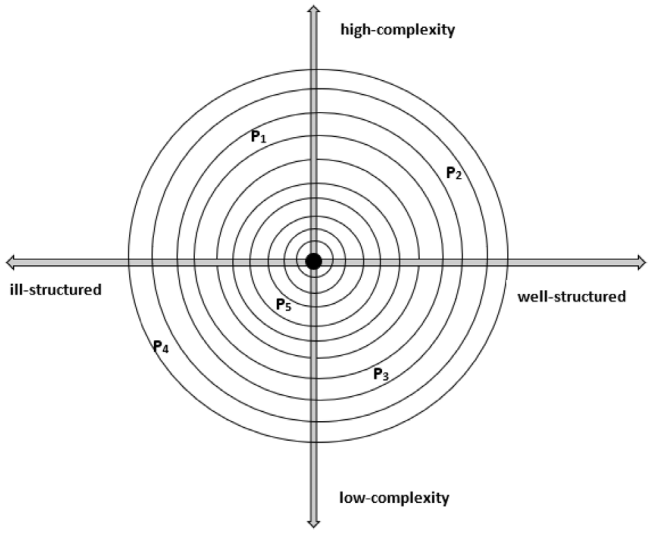

FIGURE 1 Problem space: Mapping of problems across the dimensions of structuredness and complexity

(e.g., Chen et al., 2017; Shen et al., 2020) also included everyday problem solving in their investigations into CT. In the study by Chen et al. (2017), everyday problems, such as finding an optimal way to load a washing machine, were operationalized as multiple-choice questions. Shen et al. (2020) operationalized everyday problem solving through an exercise in which participants were asked to deduce the most cost-effective route between two locations, given a map showing a network of flights and information about costs and travel time. Although the task of finding the most cost-effective flight was open-ended and offered some insight into the use of CT in everyday problems, the researchers' analysis focused on comparing the pre-test and post-test improvements after a robotic intervention. Standl's (2017) study was the closest to our study in its consideration and analysis of everyday problem solving detached from programming.

As a summary, Figure 1 shows the problem space in which problems are mapped according to the dimensions of structuredness and complexity. Problems are depicted on a continuum in the overlapping axes of structuredness and complexity (Jonassen, 2000; Simon, 1973). Specifically, the position of a problem in the problem space is not static. This implies that a particular problem could be perceived differently by different problem solvers (e.g., the P_1–P_5 in Figure 1). The perceived position of a problem is determined by the problem representation and the problem solver's familiarity, knowledge, metacognition, epistemological beliefs, attitude, and motivation about and towards the problem (Jonassen, 2000).

3 | METHODOLOGY

3.1 | Multiple case study of a routing problem

Tying back to the aim of this study (i.e., understanding the features of problems solvable by CT), a case study was deemed a suitable approach for launching an in-depth investigation. A case study is a form of research design that allows for the investigation of complex issues by probing participants' views (Yin, 2018). As established in the theoretical framing above, problems are complex and hard to generalize. Thus, the multiple case study approach (Stake, 1995), a variant of the case study approach, was considered to be even more desirable because it allows for the harvesting of rich data from multiple participants, thereby supporting a deeper understanding through cross-case analyses of an issue of interest. In this study, we probed how seven university students solved a routing problem (an instance of an everyday problem).

Routing and navigation problems are familiar routine challenges that we encounter daily. Every day, people choose routes and modes of transportation that offer the most effective paths to work, school, and home. What constitutes 'effectiveness' varies according to different variables, including time, distance, and cost. In fact, routing challenges, although often infused into programming, are CT problems that are commonly encountered when seeking to control physical robots or virtual agents, such as when writing programs to construct paths for an agent to navigate a maze (Fessakis et al., 2013), or when designing code sequences to make a robot travel via a certain path (Chen et al., 2017). Similar to Chen et al. (2017) and Fessakis et al. (2013) that implemented CT solution by constructing paths for agents but programmatically, a routing problem was selected for this study to investigate both the features and implementation of CT when decontextualized from programming.

3.2 | Context

Funded by the government, this study was part of a large-scale collaborative project between five universities in the region. In view of the ubiquity and increasing impact of technology on how we live, work, and study, the project aimed to expose all the university students (irrespective of their program of study and academic level) to the potentials afforded by programming and to an understanding of how programming underlies the operation of technology. Every academic year, three cohorts were recruited to participate in a non-credit-bearing Python programming course. By the end of the course, students were expected to appreciate the importance of programming and ideally use the acquired skills to create innovative applications with environmental, social, and educational benefits.

For this study, data were collected from the January 2021 cohort. Due to the COVID-19 pandemic restrictions at that time, a virtual flipped learning model was adopted for the training, which spanned 8 weeks. Students attended four online lessons via Zoom (a group videoconferencing application; https://zoom.us/). These lessons were conducted fortnightly, with each lesson lasting for 1.5 h. At the end of every class, students were assigned take-home tasks (problems) to attempt and submit before the next lesson. Students were encouraged to explore other resources that could help them solve the take-home challenges. We used Blackboard (a learning management system; https://www.blackboard.com/) and Google Colaboratory (a free, scalable and browser-based Python code

TABLE 1 Course learning objectives

Lesson	Objective
One	Know that Python is a high-level interpreted language Be familiar with the Google Colaboratory environment Identify and distinguish between data types (integers, floats, strings, booleans) Understand and apply variables in a script Use the built-in functions such as *input()* to receive user entries within a program
Two	Use comments in a script Identify how conditionals and logical operators are used in a program, and apply them in basic scenarios Use for and while loops to efficiently program repetitive tasks
Three	Identify, distinguish, and understand what data structures (lists, tuples, dictionaries) are and how to access and manipulate their elements Use in-built and user-created functions to program basic app features Find and apply Python modules from the Standard Python Library
Four	Understand how to read and write into text files, and apply this in a script Know that Python has packages and modules for data analysis and manipulation See an example of NumPy and Pandas to read data and make basic observations

editor; https://colab.research.google.com/) to organize the learning activities.

3.3 | Learning activities and material

Because the participating students came from diverse backgrounds and the course was open to all students without any prerequisite courses, we assumed that majority of the cohort would be beginners with little or no knowledge of programming. The learning activities were collaboratively designed by the instructor and the project team, which comprised experts in educational technology and computer scientists. Programming problems with different levels of complexity were also designed to account for students with greater exposure, interest, and programming abilities.

Table 1 summarizes the focal objectives of the lessons. Although the learning objectives were depicted as a list, they were taught in the context of problem solving. The course instructor emphasized the concepts by walking through take-home tasks and posing problems for students to 'learn by doing', an instructional model patterned after Papert's (1980) constructionism but guided with close-ended problems. A pool of 15 take-home tasks with varying levels of complexity was developed. Although the tasks were not compulsory, students were encouraged to attempt as many problems as they could. One important feature of these tasks was that the problems were masked from common solutions found on the web. For example, instead of asking students to design a search or sort algorithm, for which they could easily find standard solutions on the web, we modified the tasks to draw them into deeper learning (see Appendix for examples of the take-home tasks).

Throughout the training, the students were reminded that CT could be demonstrated if they used the fundamental concepts of computer science acquired in learning programming to solve everyday problems.

TABLE 2 Participants and their academic programs and levels

Pseudonym	Gender	Academic program	Academic level
Arthur	Male	Computing	Fourth year
Ben	Male	Science	Fourth year
Charlie	Male	Accounting	Fourth year
Dan	Male	Engineering	Second year
Eddy	Male	Computing	Postgraduate
Ethan	Male	Engineering	First year
Felicia	Female	Mathematics	Postgraduate

3.4 | Participants and case selection

The January 2021 cohort comprised 82 students from different academic programs and levels at one of the participating universities. The free Python programming course was advertised through the university's online portal and bulk email and posted on physical noticeboards across the campus for ~6 weeks before the commencement date. Registration was opened only to students on a first-come, first-served basis. Participating students signed up through the university's authentication portal. Before the study, ethical clearance was obtained from the institutional review board. Participants were informed about their rights, the privacy policies, and the data management plan. Of the 82 students registered in the course, seven submitted their responses to the CT open-ended question (see Section 3.5). Subsequently, these seven students (see Table 2), who constituted the cases in this study, volunteered to participate in a follow-up interview.

3.5 | Data collection

3.5.1 | Open-ended CT tasks

A routing problem in the form of food delivery service task was assigned to the students. The students played the role of a restaurant manager whose daily duties included ensuring that delivery motorbikes delivered food to customers efficiently. They were presented with isomorphic representations of a map showing different sets of travel times between the restaurant (R) and the locations A, B, and C (see Figures 2–4). The numbers in the circles represented the average time (in minutes) required to travel by motorbike between two locations (e.g., it would take 3 min to travel from B to R in Figure 2). In the fictional scenario, three customers at locations A, B, and C simultaneously requested the restaurant to deliver food to them.

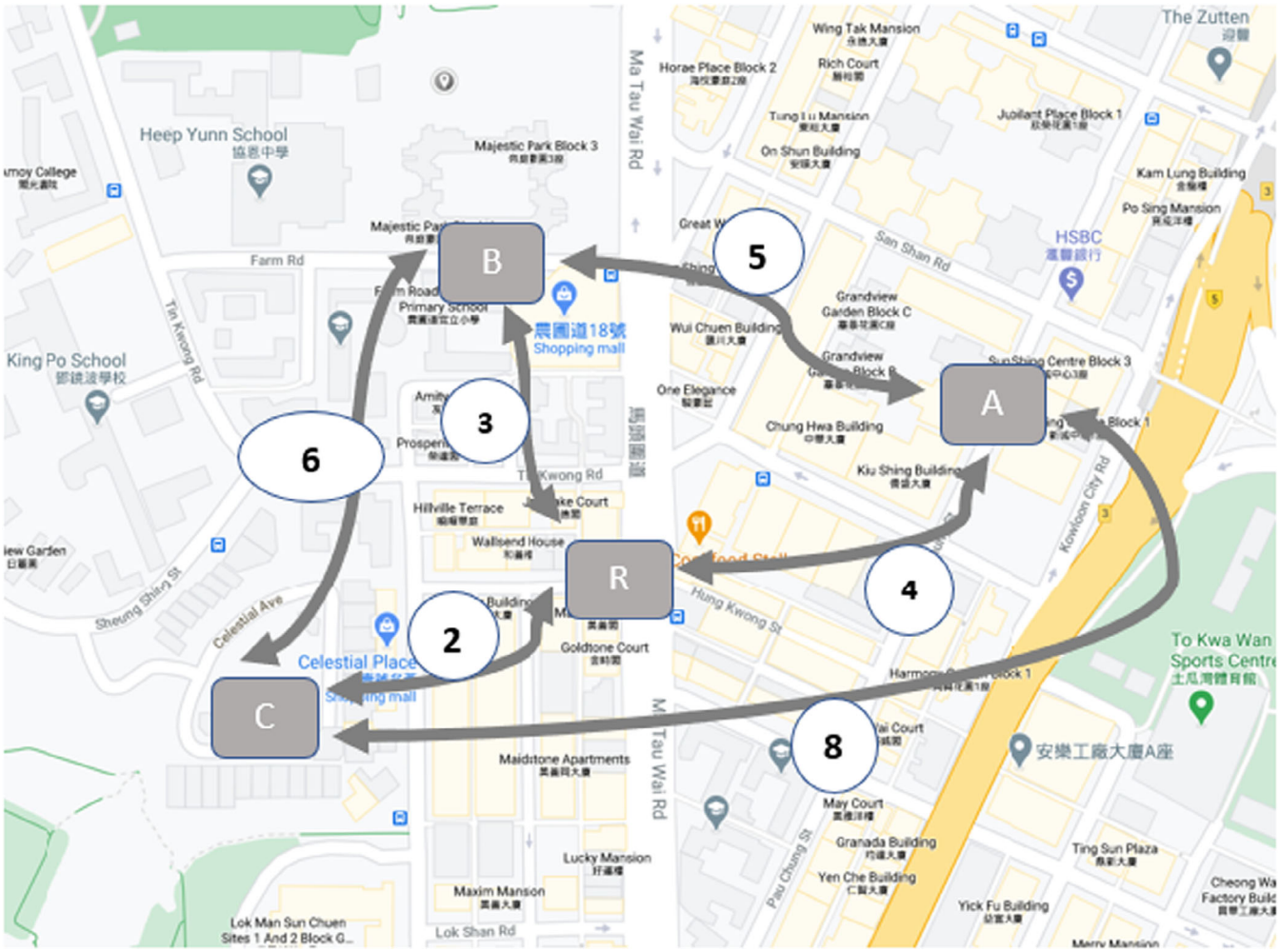

FIGURE 2 Normal routing of the food delivery service in Task 1

Three open-ended questions were posed as follows:

Task 1: Study Figure 2 and suggest the most efficient route for delivering food to the three customers, starting from R and returning to R (e.g., RABCR, RBRCAR, RACRBR). Also, describe to your primary school-aged niece how you decided on the route.

Task 2: After 3 months, due to road improvement work around the restaurant, the travel times to the different locations have changed, as shown in Figure 3. In light of the new situation, please suggest the most efficient route for delivering food to the three customers, starting from R and returning to R. Also, describe to your niece in primary school how you decided on the new route.

Task 3: As a result of your excellent performance in coordinating food delivery efficiently, you are promoted. However, in the new role, you are tasked with training the incoming manager on how to optimize the delivery service. Because the travel times to different locations (A, B, and C) can change in the future, they are represented using unknown variables (u, v, w, x, y, z), as shown in Figure 4. Describe to the incoming manager how to find efficient routes.

For these tasks, students were reminded that it was not sufficient to merely say, 'I chose a route (e.g., RABCR) because it was the fastest'. A qualified response would also explain why the other routes were not chosen. They were encouraged to use any type of representation or annotation to explain their solutions, such as words, texts, diagrams, flowcharts, computer code, or a pseudo-code.

3.5.2 | Class observation and follow-up interviews

Apart from the participants' answers to the food delivery task, the students' responses during the programming class and one-to-one interviews with the seven cases provided invaluable Data S1 for triangulation. For example, students were asked at separate times in the programming class to describe how they came to school. On each occasion, a varying level of structuredness, in the form of prompts and cues, was added to elicit their CT problem solving.

The follow-up interviews, which were conducted after 3 weeks of the course had been completed, lasted for 15 min per case. The interviews were designed to meet two aims: (a) to allow the participants to

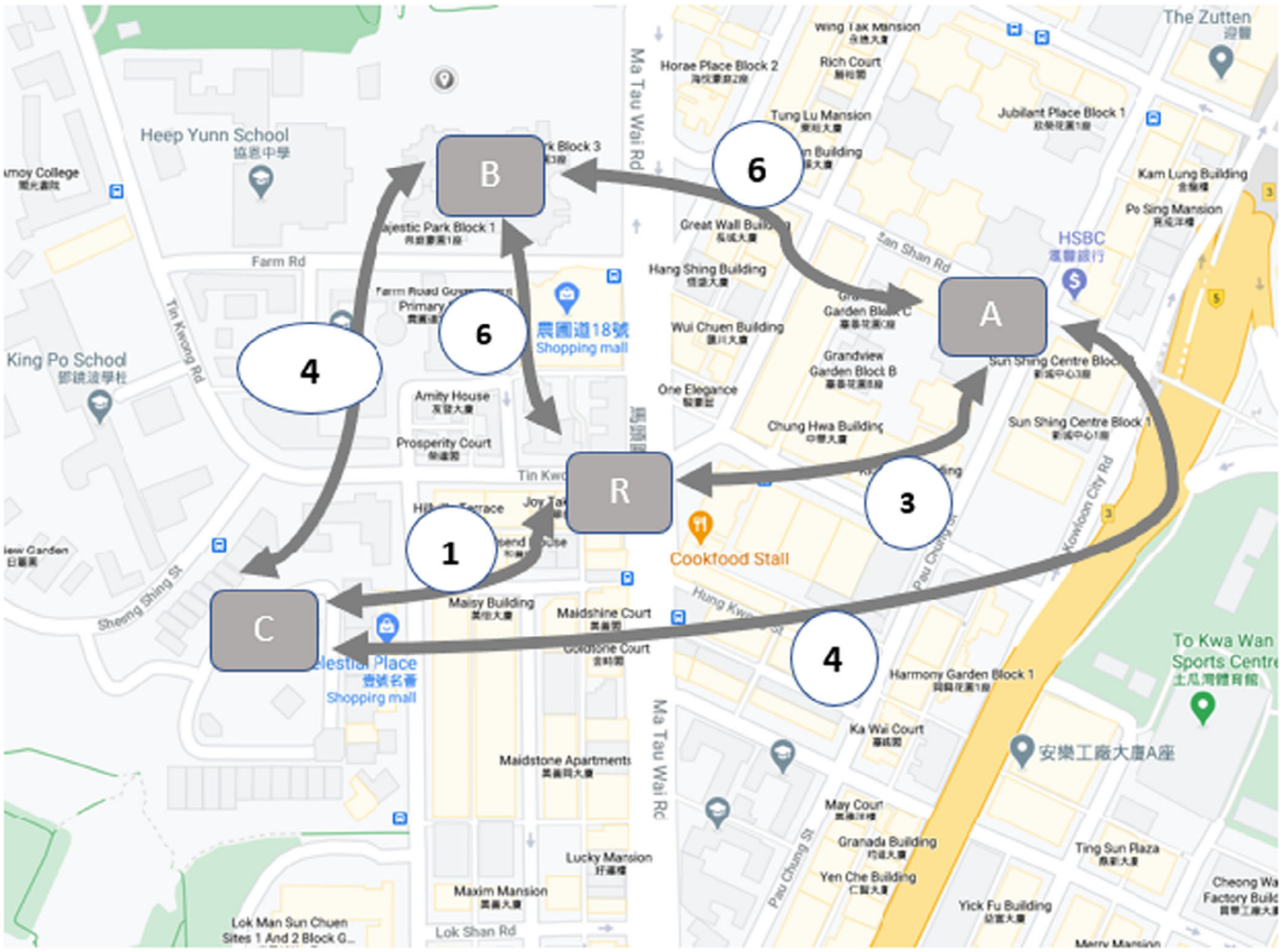

FIGURE 3 Modification of the food delivery service routing after 3 months due to road construction and improvement work in Task 2

elaborate on their food delivery solution and (b) to probe their conceptual understanding of CT (and how it differed from programming).

3.6 | Data analysis

In the theoretical framing of this study, we noted the complexities in modelling problem solving and the multiple conceptualisations of CT (e.g., Barr et al., 2011; Brennan & Resnick, 2012; Csizmadia et al., 2015; Korkmaz et al., 2017; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016). Although the diversity of interpretations of CT had some benefits, we were confronted with the challenge of analysing the data without a consensus on the framing of CT. Therefore, we relied on Wing's (2006) seminal description of CT as the use of fundamental concepts of computer science to solve everyday problems. Because it is not clear what constitutes CT, the grounded theory approach (Glaser & Strauss, 1968, 2017) was most suitable for analysing our data. The grounded theory approach relies on data to uncover the theme and constructs pertaining to the issue of interest. This approach fit our exploratory enquiry about the features of CT solvable problems. Therefore, we coded the submitted data by identifying the computer science concepts that were reflected in the solutions.

4 | RESULTS

4.1 | Summarizing the problem solving of the cases

In this section, we briefly present the participants' solutions to Tasks 1, 2, and 3 to elicit the diverse approaches to everyday problem solving. The full documents submitted by the participants (Arthur, Ben, Charlie, Dan, Eddy, Ethan, and Felicia) are available as Data S1. Several references to the locations R, A, B, and C are made in presenting these results and also in the discussion section. Therefore, familiarity with the maps (Figures 2–4) is helpful for understanding some of the technicalities in the reporting of the results and the discussion.

Arthur presented the solutions to the routing problems by describing the process in textual form. According to him, RCRABR (16 min) and RABCR (14 min) were the optimal paths (i.e., the routes

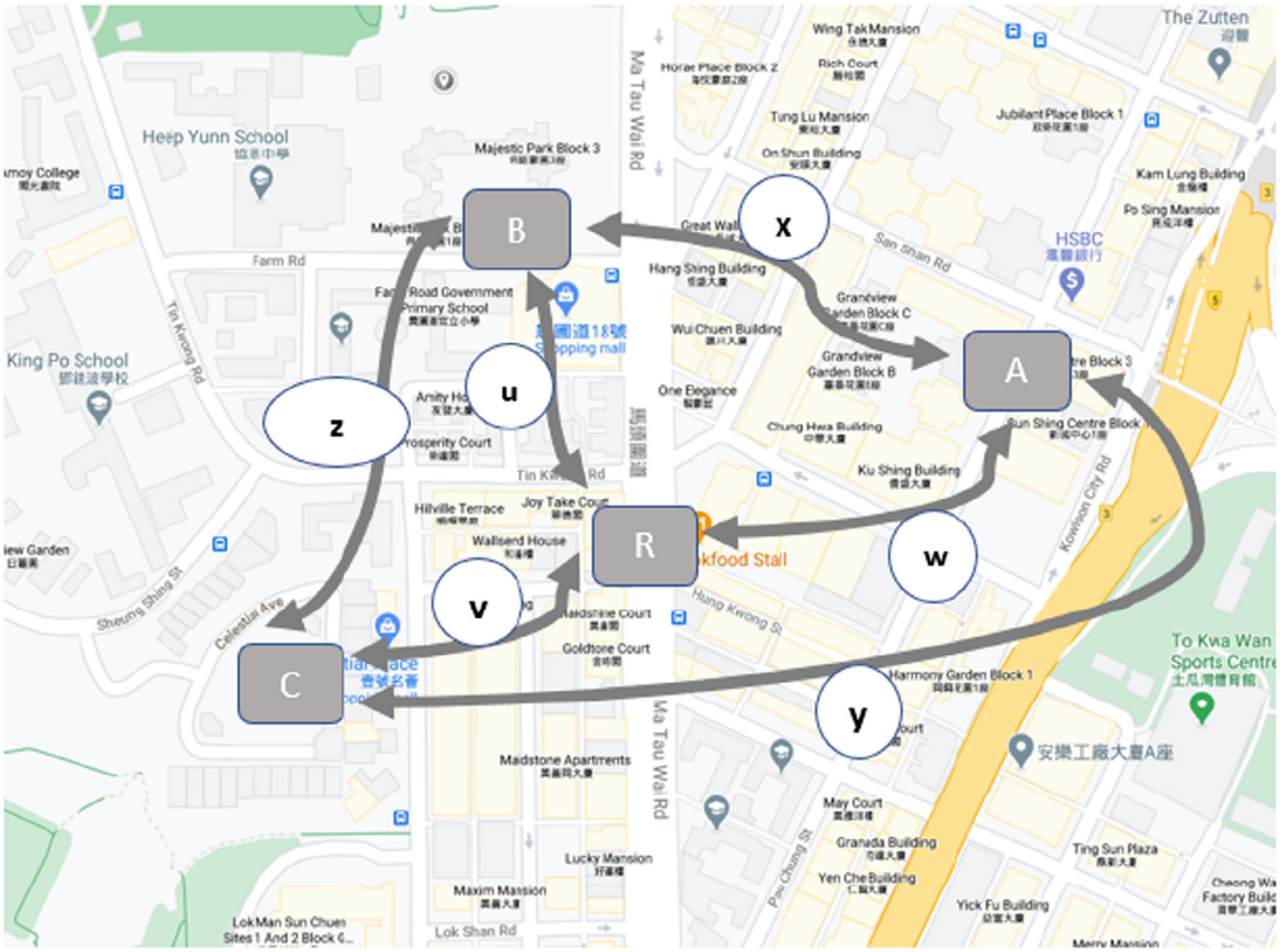

FIGURE 4 Design of generic solution for optimal and efficient food delivery in Task 3

with the shortest travel times) in Task 1 and Task 2, respectively. Arthur did not attempt the third question.

Ben solved the problems by programming with Python. Using 404 lines of code, he designed an automated solution and represented the travelling time between adjacent locations using placeholders (i.e., 'variables' in programming terminology). For Task 1, Ben's program returned four efficient pathways (i.e., RABRCR, RBARCR, RCRABR, and RCRBAR), with a total travel time of 16 min for each pathway. When the travel times between adjacent locations in Task 2 were assigned to the placeholders, Ben's program found two optimal routes (RABCR and RCBAR), with a travel time of 14 min. Although the number of routes that emerged from Ben's programmed solution was different from the number arrived at in Arthur's textual solution, a close examination revealed that the routes in the two solutions were the same except that Ben's solution captured different combinations of the optimal routes.

Charlie had an interesting approach. Presenting his solution in the form of written text, he noted that 24 routes should be compared to determine the route with the shortest travel time. Charlie did not substantiate how the epistemics of permutation and combination supported his claims, and he successfully listed only 20 of his claimed 24 routes. Although Charlie was unable to list all 24 of the viable routes, his solution found four and two optimal routes in Task 1 and Task 2, respectively. These solutions were the exact routes that Ben found in his programmed solution. For Task 3, which was meant to evaluate the student's ability to develop a generic and automated solution, Charlie expressed his solution using mathematical equations. The proof of correctness is beyond the scope of this study.

Dan developed his solution by describing the process in textual form. Like Arthur, Dan found RABRCR (16 min; the same route as Arthur's RCRABR but with a different combination) to be the optimal route for Task 1. His solution to Task 2 was the same as Arthur's RABCR (14 min). For Task 3, Dan formulated a series of mathematical equations (see Figure 5).

Eddy and Ethan collaborated with each other on the tasks. Their approach was similar to Charlie's listing of the possible combinations of routes in Task 1 and Task 2. However, unlike Charlie, who claimed that it was necessary to compare 24 pathways, Eddy and Ethan compared only six routes, as shown in Figure 6. Based on a close examination of their approach and the follow-up interview with them, we

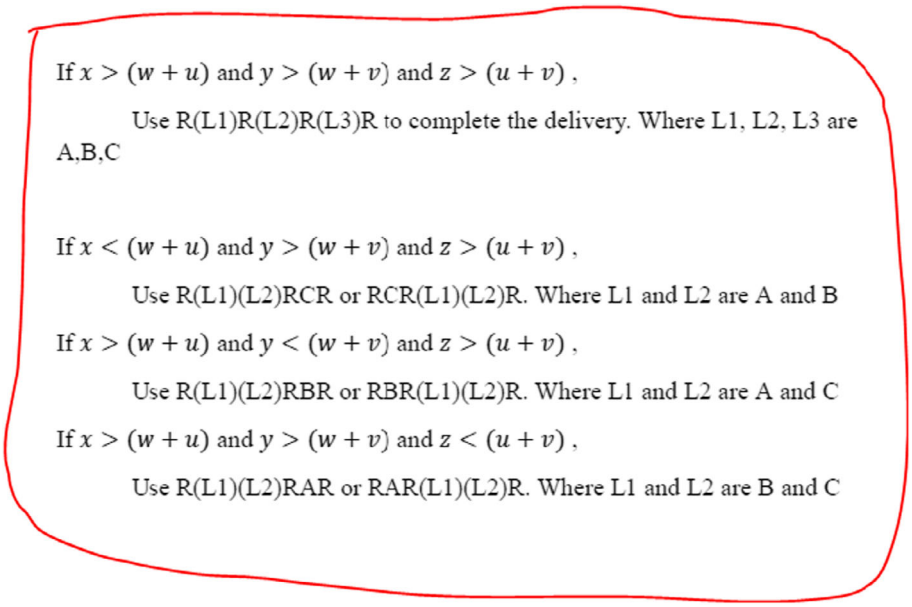

FIGURE 5 Snippet of Dan's mathematical equations for the optimal routing solution

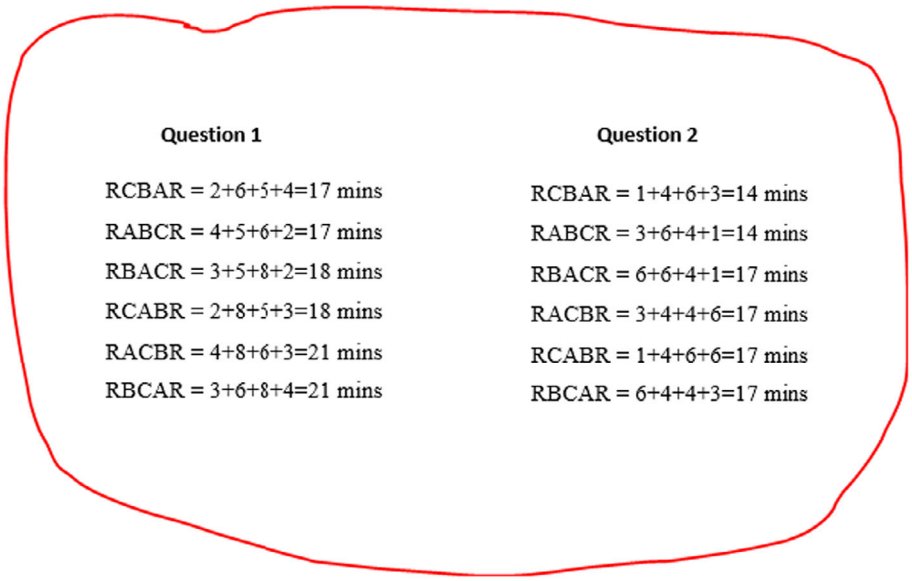

FIGURE 6 Eddy and Ethan's solution with six pathways

discovered that they had treated location R only as the start and final location, ignoring routes that could have included R as a possible intermediary location. Unlike other participants who found a 16-min optimal route in Task 1, Eddy and Ethan returned two routes (RCBAR and RABCR), with a travel time of 17 min for each route. Their solution for Task 2 was consistent with other participants' solutions, finding two optimal routes (RABCR and RCBAR) each with a travelling time of 14 min. For Task 3, Eddy and Ethan stated that an optimal solution could be generated using Dijkstra's algorithm. However, they did not substantiate how the algorithm could be applied to this problem. Dijkstra's algorithm (Dijkstra, 1959), named after the designer and recipient of the Association of Computing Machinery A. M. Turing Award, is a fundamental algorithm for finding the shortest paths between nodes in a graph with wide application in the design of computer networks. A discussion about Dijkstra's algorithm is beyond the scope of this paper, but its application is significant when considering the shortest path between two nodes in a graph, which is synonymous with the shortest travel time between two locations in a road network. Although Eddy and Ethan's idea to use Dijkstra's algorithm was commendable, the nature of the everyday routing problem in Task 3 imposed the condition of visiting three locations (A, B, and C) before returning to the starting R and therefore did not fit directly with Dijkstra's algorithm (except with appropriate modification).

Felicia, the only female participant, listed seven plausible routes before comparing them to determine the route with the shortest travel time. Similar to Eddy and Ethan's list of six plausible pathways, it was unclear how Felicia had determined the seven routes, especially in light of Charlie's realization that there were 24 viable pathways. Nevertheless, whether by serendipity or thoughtful design, Felicia found that RCRABR (16 min) and RABCR (14 min) were the optimal

FIGURE 7 Felicia's mathematical expression of the total travel time

route	time
RABCR	w+x+z+v
RACBR	w+y+z+u
RBACR	w+x+y+v
RCRBRAR	2v+2u+2w
RCRBAR	2v+u+x+w
RBRACR	2u+w+y+v
RARBCR	2w+u+z+v

paths in Task 1 and Task 2, respectively. For Task 3, Felicia represented the total travel time of each of the seven routes as a mathematical expression of the missing variables, as shown in Figure 7, and submitted that the route with the shortest travel time denoted the efficient solution.

4.2 | Cross-case analysis: CT practices in everyday problem solving

In relation to the first research question (i.e., to understand how CT was demonstrated in solving everyday problems), the findings from the cross-case analysis are discussed in this section. Because the primary focus of the study was CT without programming, all the solutions except for Ben's Python program were included in the analysis.

4.2.1 | Decomposition

Decomposition is the breaking down of a process or object into smaller components, with the overarching goal of making tasks manageable and simplifying their solutions. Arthur and Dan examined the travel times between the four locations (A, B, C, and R). In total, in Task 1, there were six point-to-point links with travel times of 2, 3, 4, 5, 6, and 8 min. Arthur and Dan decomposed (broke down) the problem by visualizing the routing as the aggregate of the travel times of constituent direct paths between adjacent locations (e.g., RC = 2 min, AC = 8 min). Therefore, it was easy for Arthur and Dan to avoid the paths that would have taken a longer time. By choosing this approach, they were able to determine how to directly reduce the travel time between two locations. Specifically, they avoided the two longest paths from C (i.e., AC = 8 min and BC = 6 min).

Charlie, Eddy, Ethan, and Felicia demonstrated decomposition differently. Whereas Arthur and Dan had broken down the problem into point-to-point routes, Charlie, Eddy, Ethan and Felicia demonstrated decomposition by breaking down the problem into possible route combinations. Charlie and Felicia found 20 and 7 routes, respectively, whereas Eddy's and Ethan's collaboration yielded 6 routes as the output of their decomposition.

When a complex system is reduced to its components, it is easier to understand the scope, relationship, and pattern that exist between the components. With the problem decomposed as the aggregate of plausible routes, it was easy to compare them to find the route with the shortest travel time.

4.2.2 | Abstraction

The essence of abstraction is focusing on important components and ignoring irrelevant details. In Task 1, Arthur considered paths AC and BC to be irrelevant for finding the optimal path. He flagged the important aspect of the problem by rephrasing and narrowing the problem to the challenge of finding the optimal route between R, B, and A only. Clearly, Arthur was practising abstraction by removing non-relevant elements (AC and BC) and focusing on the essential components (R, B, A). In addition, abstraction has an interesting relationship with decomposition. Decomposition is the primary enabler of abstraction. For example, it was through decomposition that the differences in the paths' travel times were highlighted in Task 1. Consequently, Arthur excluded the two longest paths (i.e., AC = 8 min and BC = 6 min) and identified the only optimal path to C (i.e., RCR of 4 min must be a subpath of the optimal solution). With the focus narrowed to locations R, B, and A, Arthur discovered that he was left with two possible options: either a circular path of RABR/RBAR (12 min) or a path

transversing through location R in the sequence RARBR/RBRAR (14 min). The former had the shortest travel time and was chosen as the subpath of the total solution yielding RCRABR (16 min) as the optimal solution for the routing problem. The intertwined relationship between abstraction and decomposition validates the designation of decomposition as a precursor to abstraction in CT problem solving (Ezeamuzie et al. 2022).

Another variant of abstraction was demonstrated in the solutions by Charlie, Felicia, and Eddy and Ethan. Although abstraction is popularly conceived as the act of focusing on the important components in a system, these participants demonstrated abstraction as an alternate representation of an object/process. They transformed the problem into a form of mathematical representation using combination and permutation, which was detached from the original nature of the problem. This is similar to the process of data transformation featured in the study by Kong and Lao (2019), in which in-service teachers demonstrated abstraction by mathematically representing the total queueing times of customers in a supermarket as part of an algorithm for finding the fastest service counter.

4.2.3 | Algorithm—A derivative of pattern recognition and automation

The position of algorithm in CT discourse was aptly captured in Aho's (2012) definition of CT as the skill required to develop algorithm. Task 3 was posed to elicit the students' ability to design functional algorithms.

To design an algorithm for Task 3, Felicia found the repeating pattern in her solutions to Task 1 and Task 2 to be useful. This was evident from her use of the same approaches in the two tasks that generated the 7 possible routes. 'I extended the pattern in solving Task 3', Felicia said during the interview. Here, it is not suggested that Felicia's algorithm yielded the correct solution. Rather, her problem solving highlighted how the identification of patterns aided the design of the algorithm. This is not always the case. Charlie and Eddy and Ethan used a similar approach and found a common pattern between Task 1 and Task 2. However, their algorithmic solutions for Task 3, which took the forms of a mathematical equation and Dijkstra algorithm, respectively, were not informed by their solutions to Task 1 and Task 2.

Arthur did not attempt Task 3. In the interview, we discovered that his inability to design an algorithmic solution was linked to his use of different approaches in Task 1 and Task 2. Although Task 1 and Task 2 were similar problems except for the specified travel times, the differences in Arthur's approaches hindered him from discovering the commonality in the tasks. In Task 1, Arthur used an elimination approach to discard paths AC and BC. However, in Task 2, he adopted an inclusion approach by selecting ARC as a subpath of the total solution: 'Path ARC and AC have the same length, but ARC also connected to R, so ARC is a better option than AC'. According to Arthur, 'I relied on my guts to optimize the solutions for Task 1 and Task 2'. His inability to discover any underlying rules meant that the solutions could not be automated. As with Arthur, Dan also used different approaches to solve Task 1 and Task 2. Hence, he also failed to discover the underlying pattern.

We thus found that the participants' ability to design algorithms, often described as a step-by-step solution, was hinged on discovering the underlying patterns and organizing the identified patterns into automated rules. What constitutes patterns from decomposing a problem becomes vivid when such discoveries can be articulated in the form of concrete rules. The product of the algorithm becomes obvious from automation, the ability to use the generated solution in different scenarios.

4.3 | Cross-case analysis: Features of everyday problem solving

In answer to the second research question, we found three characteristics of everyday problem solving using CT.

4.3.1 | Simplification trumps formal proof of correctness in everyday problem solving

Irrespective of the representation, applying CT to solve everyday problems was not trivial. Whereas most of the participants, except Eddy and Ethan, found the correct optimal path, they all expressed uncertainty about the correctness of their solutions.

For example, the analysis of Arthur's solution shows the complexity of solving a problem. In Task 1, although Arthur intuitively avoided the paths with longer travel times to reduce the total time, it was unclear why he chose to exclude only two paths and not others such as AB, which had a travel time of 5 min. Implicitly, whether or not Arthur's solution was correct, it became obvious that solving everyday problems falls short of a formal proof of correctness because other problem solving methods, such as human intuition and guessing, co-exist in the solutions. When Arthur's focus was narrowed down to three locations (R, A, and B), he was able to construct an instant mental solution from the emerging pattern. This was not the case when he considered four locations (R, A, B, C). It appears that Arthur's cognition could not decode underlying patterns with certainty when traversing four locations. Hence, he relied on decomposition and abstraction to simplify the problem.

Arthur's approach was consistent with the discovery that emerged when participants were asked to think about how their solutions would be affected if extra locations were to be added to the routing map (e.g., choosing an efficient path through five locations). Eddy, Ethan, and Ben noted that additional locations would result in an exponential increase in complexity and conceived that the solution would be much harder for humans. Other participants' solutions demonstrated varying levels of complexity. Charlie's and Dan's mathematical equations, Eddy and Ethan's Dijkstra algorithm, and Ben's programming solution embodied the complex knowledge and skills that were applied to solve the everyday problem with CT.

Is it necessary to apply these complex schemas in solving everyday problems? There is no simple binary answer to this question, because what we conceived as everyday problems varied in a continuum. When the participants focused on achieving proven correctness in the algorithm, the complexity of problem solving became enormous. However, several everyday problems do not require such a high level of correctness. Ethan's response to the question of how he would apply CT captured this aptly: 'When I do a job such as processing documents, I will consider ways to smartly save my time and finish the job well'. Decomposition and abstraction are more useful in these scenarios than the proven correctness of the algorithm.

4.3.2 | Structuring and restructuring of problems increases the applicability of CT

Generally, everyday problems require structure to enable students to recognize how CT can be applied in their solutions. The higher the level of structuredness, the easier and more obvious the application of CT becomes. For example, as part of the observation, we posed these two questions during the online training classes: (a) describe how you come to school and reflect on how CT facilitated your choice, and (b) describe how you chose a checkout counter in a department store and reflect on how CT facilitated your choice. In response to the first question, most of the students gave simple answers such as 'by bus', 'using the train' or 'I walk to school'. For the second question, all the students replied that they chose counters that had the fewest people in the queue.

However, students responded differently when more structure was added to the question, such as the distance of the school from home, the presence of alternative modes of transport, travel times, and the costs of various modes of transport. In this structured context, they explained the factors they considered when choosing their means of transport to school, and their explanations included more detail, such as comparisons of alternatives. Similarly, in response to the second question, students responded with improved clarity and logic when additional cues were added, such as the number of items in the customers' carts and the availability of express counters. Considering that the students were not bound by time to elaborate on their solutions, a plausible explanation for the difference in their responses is that students are not naturally acquainted with the propensity to apply CT in everyday problem solving unless it is structured.

In addition, the students noted that they had never used the CT approach explicitly to solve everyday problems; this was aptly captured in Ben's interview: 'These kinds of solutions usually come from my mind'. Therefore, students should be taught and encouraged to explicitly think of CT as a problem solving approach. To successfully apply CT in everyday problem solving, students need to 'see' the problem from a wider perspective, determine the factors that could influence the solutions, and intentionally apply the discovered structures.

4.3.3 | Everyday problem solving encompasses latent CT practices

In Section 4.2, we identified decomposition, abstraction, pattern recognition, algorithm, and automation as dimensions of CT exhibited in students' everyday problem solving. Except for algorithm, which is linked to sequencing and decision making, the participants neither explicitly nor consciously applied these CT practices to their solutions.

When Ethan was asked if he had ever applied CT consciously in solving everyday problems, his response was consistent with this finding. According to him, 'In my daily life, I think I have adopted some of them but not the theory; like going to work and office, I have thought about which route is the best for me and save more time'.

4.4 | Decontextualizing everyday problem solving with CT from programming

The underpinning aim of this study was to understand how CT could be practised without the cognitive burden of programming. As we stated earlier, unless the research community can delineate these overlaps, arguments for CT that are detached from programming will continue to be vague, especially in light of the dominance of computer science and programming education as well-established domains.

Because the introductory programming class was open to students from every discipline, participants were instructed to solve the problem with or without programming (e.g., using a textual description or a flowchart). We anticipated that participants would either be novices or have sparse exposure to programming. We presumed that they would not be able to solve the problem programmatically. However, we were wrong. Ben, a fourth-year science student, wrote a computer program as the solution, which allowed for further investigation into the varied representation and complexities of everyday problem solving. Originally, our focus was to answer the two research questions stated in the introduction section. However, the data provided by Ben's programming solution prompted us to ask and discuss a third question:

> **RQ3.** *What are the differences and similarities (if any) between the computer programmed and non-programmed CT solutions to everyday problems?*

Ben wrote 404 lines of Python code! Using programming as a representation in problem solving lies beyond the usual scope of approaches that people use to solve everyday problems. Ben relied on several Python modules that were not discussed in the learning activities. In a follow-up interview, Ben acknowledged that although he had not programmed in the last 3 years prior to the coding workshop, he relied on his past C++ programming experience to solve the routing task using Python. Testing for the correctness of Ben's program was beyond the scope of this paper. However, when Task 1 and Task 2 were used as test cases, the outputs were correct. Also, in the interview, Ben expressed strong optimism that his program would solve

similar problems with different point-to-point travelling times, provided that the travelling map was not altered.

What were the differences in practices between Ben's programmed solution and the solutions that had not used programming? Ben used several loops (i.e., the 'for' and 'while' keywords in Python), conditionals (i.e., 'if–else' decision constructs), and programming variables. However, the application of loops and the use of variables were conspicuously absent in the non-programmed solutions. Although conditionals were used in the non-programmed solutions, they were implicit, unlike their explicit use in Ben's programmed solution. For example, when Charlie compared the 24 possible paths to deduce the optimal route, he was implicitly demonstrating the application of the 'if–else' conditional. In contrast, Ben's program showed the detailed steps that yielded the final solution, which represented the algorithm. However, Ben was unable to explain how he used decomposition and abstraction practices to simplify his solution. Because he did not create any user-defined function or class, we could not further decode his planning and design strategies.

Differences in the programmed and non-programmed solutions were also evident in terms of the nature of the practices of abstraction, decomposition, and algorithm design. In the programmed solution, these concepts were practised on a micro scale, such as how the data were manipulated in the arrays and interoperabilities of the application's programming interface. These findings are consistent with studies that found differences in human cognition and programming, such as the unnaturalness for humans of solving computational tasks using looping and multi-branched conditionals (Miller, 1981; Pane & Myers, 2001).

5 | DISCUSSION

We embarked on this investigation to understand how CT could be used in everyday problem solving. Our motivation stemmed from the concern that the blurry boundaries between programming and CT may be impeding the development of CT. Wing's (2006) seminal article stated clearly that CT is about conceptualisation, rather than programming, and the context of Wing's account of CT was based on the belief that the wealth of the cognitive practices of computer scientists should be transferred to learners outside of computer science. CT is still an actively developing field with immense potential, as seen in the increasing integration of education robotics, virtual reality, and artificial intelligence in mainstream learning. Also, the proliferation of age-adaptive visual platforms for programming, game development, and simulation demonstrates the vast learning opportunities in CT. However, irrespective of how research and practice have presented CT as detachable from programming, an underlying bias for programming remains. This is consistent with Lye and Koh's (2014) findings from their review of literature that programming is the primary medium for learning CT.

To be clear, we believe that learning to program is a useful skill, especially in the 21st century. As computer science practitioners, we would be delighted to see programming integrated into core subjects to solve multidisciplinary problems. However, the reality is that programming demands high levels of cognitive processing, which would suit the interests only of a limited set of students. Although there are hosts of benefits to learning to write computer programs, CT cannot be positioned as a skill for everyone if learners need to write Ben's code or implement Eddy and Ethan's Dijkstra algorithm to solve an everyday problem. Rather, we must discover how computer science practices can augment everyday problem solving, such as making decisions about efficient travelling paths, arranging seats in a classroom for easier access and less distraction, classifying books on a shelf, and arranging booths in an exhibition fair.

We deemed it important to distinguish between *following* and *designing* algorithms in everyday problem solving. For example, should following the step-by-step instructions of a cooking recipe be regarded as an algorithmic practice? In this study, the use of an algorithm in solving problems was interpreted as discerning and designing an algorithm as part of the solution, which was distinguished from following a given sequence in the process of problem solving. We discovered that the correctness of algorithms in everyday activities is subjective, which is consistent with Standl's (2017) finding when students were tasked to solve the real-life challenge of packing moving boxes. This approach is different from theoretical computing, in which an algorithm is not just any step-by-step process but rather a series of steps that control an abstract machine without the use of human judgement (Denning, 2017). When an algorithm is conceived from the theoretical lens (e.g., Dijkstra, Sort and Search algorithms), the proof of correctness is a criterion for its validity. However, such proofs were never the focus when students solved everyday problems; this supports the perception of CT as systems thinking through multiple variables that prioritize a holistic approach to problem solving (Weintrop et al., 2016).

Abstraction and decomposition often overlap in problem solving (Ezeamuzie et al. 2022). A similar investigation of everyday problem solving found that students struggled to describe how they abstracted and decomposed when solving a bin packing problem (Standl, 2017). Aided by the interviews, we elicited practices that conformed to decomposition and abstraction to solve the routing problem. However, the participants were not able to explain whether these practices were deliberate or not, and their answers mirrored Standl's (2017) designation of them inseparable components in his modified three steps of CT problem solving.

Decomposition is not just any random splitting of a problem; rather, it is the act of intentionally dividing complex problems into smaller functional parts that fit squarely into the larger system (Shute et al., 2017). According to Wing (2008), abstraction is the most important component of CT and was found to be the most frequently featured dimension of CT in extant empirical studies (Ezeamuzie & Leung, 2022). To advance the application of CT in solving everyday problems, abstraction and decomposition should be explicitly demonstrated using everyday examples. Pedagogical practices that highlight and encourage learners to apply these latent practices will increase the use of such cognition in everyday problem solving.

This study generated valuable findings: the need to provide a balance between formal correctness and simplification in problem solving, an understanding of the latent nature of CT in problems, and the need to provide intentional structure when presenting CT students with everyday problems. These findings are not without limitations, which we acknowledge herein. This investigation was based on case studies. This entailed analysis of qualitative data which interpretations are informed through researchers' lens and experience. Implicitly, the risk for observation bias and subjectivity is high, impeding the generalization of results to the wider population. Also, the nature of the problem investigated in this study is limited and neither explains if similar CT practices are transferable to other problem solving scenarios nor similarity of performance with a different group. Despite these limitations, we believe that we uncovered valuable qualitative information for advancing research and practice in CT as well as computing education. As we noted in our introductory section, this study did not aim for a generalization of its results. Rather, it aimed to fill an obvious gap in understanding of the nature of CT problem solving when practised without programming through close observation of learners' practices.

Scherer (2016) recognized the precise need for empirically validated research on the transfer effect of computer programming. Unfortunately, this need remains unmet. Although the participants in this study participated in a programming course, we found no tangible evidence to support the notion that the knowledge gained from the lessons aided their solution of the everyday problem using the CT approach. Nonetheless, the lessons drove the students' 'thinking and awareness' that they now possessed an additional problem solving approach that could be used to solve everyday problems, which is consistent with Standl's (2017) findings. For example, Ben enthusiastically said, 'If I choose to buy a product, and there are multiple companies that provide the products, I will use CT concepts to think which product has the best properties'. How and when these CT concepts can be used is still largely unknown and untested.

Based on what we unearthed about the nature of solving everyday problems with CT and the affirmation that CT is about conceptualizing, not programming (Wing, 2006), this study found that CT practices are latent in everyday problems. Therefore, it is hard for students to discover how CT can augment their ability to solve such problems. Pedagogies that promote CT should consider leading the learners to both discover the latent CT practices and structure everyday problems intentionally as a valuable premise for discovering the applicability of CT. Although algorithm, the step-by-step approach to problem solving, is a conspicuous CT practice for everyday problems such as route planning, decomposition and abstraction, though latent, are prominent CT components for simplifying everyday problem solving. Future studies need to explicitly demonstrate concrete and relatable examples of CT application in everyday problem solving.

CONFLICT OF INTEREST

The authors declare no conflict of interest to disclose.

PEER REVIEW

The peer review history for this article is available at https://publons.com/publon/10.1111/jcal.12720.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

ORCID

Ndudi O. Ezeamuzie https://orcid.org/0000-0001-8946-5709
Jessica S. C. Leung https://orcid.org/0000-0002-6299-8158
Raycelle C. C. Garcia https://orcid.org/0000-0001-9400-6566
Fridolin S. T. Ting https://orcid.org/0000-0001-7432-0187

REFERENCES

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning Leading with Technology*, *38*(6), 20–23.

Barr, V., & Stephenson, C. (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. https://doi.org/10.1145/1929887.1929905

Bransford, J. (1993). *The IDEAL problem solver: A guide for improving thinking, learning, and creativity* (2nd ed.). W.H. Freeman.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association* (Vol. 1, pp. 1–25). AERA. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162–175. https://doi.org/10.1016/j.compedu.2017.03.001

Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V., & Tosto, C. (2019). Educational robotics in primary school: Measuring the development of computational thinking skills with the bebras tasks. *Informatics*, *6*(4), 43. https://doi.org/10.3390/informatics6040043

Çiftci, S., & Bildiren, A. (2020). The effect of coding courses on the cognitive abilities and problem-solving skills of preschool children. *Computer Science Education*, *30*(1), 3–21. https://doi.org/10.1080/08993408.2019.1696169

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking: A guide for teachers. https://eprints.soton.ac.uk/424545/

Dagienė, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In A. Brodnik & F. Tort (Eds.), *Informatics in schools: Improvement of informatics knowledge and perception* (pp. 28–39). Springer. https://doi.org/10.1007/978-3-319-46747-4_3

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39. https://doi.org/10.1145/2998438

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Communications of the ACM*, *60*(3), 31–33. https://doi.org/10.1145/3041047

Dewey, J. (1910). *How we think*. D.C. Heath & Company https://pure.mpg.de/rest/items/item_2316308/component/file_2316307/content

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271. https://doi.org/10.1007/BF01386390

Ezeamuzie, N. O., & Leung, J. S. C. (2022). Computational thinking through an empirical lens: A systematic review of literature. *Journal of Educational Computing Research*, *60*(2), 481–511. https://doi.org/10.1177/07356331211033158

Ezeamuzie, N. O., Leung, J. S. C., & Ting, F. S. T. (2022). Unleashing the potential of abstraction from cloud of computational thinking: A systematic review of literature. *Journal of Educational Computing Research*, *60*(4), 877–905. https://doi.org/10.1177/07356331211055379

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers and Education*, *63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Gagné, R. M. (1985). *The conditions of learning and theory of instruction*. Holt, Rinehart and Winston.

Glaser, B. G., & Strauss, A. L. (1968). *The discovery of grounded theory: Strategies for qualitative research*. Weidenfeld and Nicolson.

Glaser, B. G., & Strauss, A. L. (2017). *Discovery of grounded theory: Strategies for qualitative research*. Routledge.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27. https://doi.org/10.1145/1378704.1378713

Jonassen, D., Strobel, J., & Lee, C. B. (2006). Everyday problem solving in engineering: Lessons for engineering educators. *Journal of Engineering Education*, *95*(2), 139–151. https://doi.org/10.1002/j.2168-9830.2006.tb00885.x

Jonassen, D. H. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational Technology Research and Development*, *45*(1), 65–94. https://doi.org/10.1007/BF02299613

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development*, *48*(4), 63–85. https://doi.org/10.1007/bf02300500

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Kong, S.-C., Lai, M., & Sun, D. (2020). Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy. *Computers and Education*, *151*, 103872. https://doi.org/10.1016/j.compedu.2020.103872

Kong, S.-C., & Lao, A. C.-C. (2019). Assessing in-service teachers' development of computational thinking practices in teacher development courses. In E. K. Hawthorne, M. A. Pérez-Quiñones, S. Heckman, & J. Zhang (Eds.), *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 976–982). ACM. https://doi.org/10.1145/3287324.3287470

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

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

Ma, H., Zhao, M., Wang, H., Wan, X., Cavanaugh, T. W., & Liu, J. (2021). Promoting pupils' computational thinking skills and self-efficacy: A problem-solving instructional approach. *Educational Technology Research and Development*, *69*(3), 1599–1616. https://doi.org/10.1007/s11423-021-10016-5

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. In A. Clear & R. Lister (Eds.), *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference* (pp. 1–29). ACM. https://doi.org/10.1145/2713609.2713610

Mayer, R. E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science*, *26*(1–2), 49–63. https://doi.org/10.1023/A:1003088013286

Melander Bowden, H. (2019). Problem-solving in collaborative game design practices: Epistemic stance, affect, and engagement. *Learning, Media and Technology*, *44*(2), 124–143. https://doi.org/10.1080/17439884.2018.1563106

Miller, L. A. (1981). Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, *20*(2), 184–215. https://doi.org/10.1147/sj.202.0184

Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM*, *62*(2), 32–35. https://doi.org/10.1145/3231587

Organisation for Economic Co-operation and Development (2013). Problem-solving framework. In *PISA 2012 assessment and analytical framework: Mathematics, reading, science, problem solving and financial literacy* (pp. 119–137). OECD Publishing. https://doi.org/10.1787/9789264190511-6-en

Pane, J. F., & Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human–Computer Studies*, *54*(2), 237–264. https://doi.org/10.1006/ijhc.2000.0410

Panskyi, T., Rowinska, Z., & Biedron, S. (2019). Out-of-school assistance in the teaching of visual creative programming in the game-based environment—Case study: Poland. *Thinking Skills and Creativity*, *34*, 100593. https://doi.org/10.1016/j.tsc.2019.100593

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365–376. https://doi.org/10.1016/j.compedu.2018.10.005

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583–602. https://doi.org/10.1007/s11251-017-9421-5

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Scherer, R. (2016). Learning from the past—The need for empirical evidence on the transfer effects of computer programming skills [Editorial Material]. *Frontiers in Psychology*, *7*, 1–4. https://doi.org/10.3389/fpsyg.2016.01390

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*. University of Southampton https://eprints.soton.ac.uk/356481/

Shen, J., Chen, G., Barth-Cohen, L., Jiang, S., & Eltoukhy, M. (2020). Connecting computational thinking in everyday reasoning and programming for elementary school students. *Journal of Research on Technology in Education*, *1–21*, 205–225. https://doi.org/10.1080/15391523.2020.1834474

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence*, *4*(3–4), 181–201. https://doi.org/10.1016/0004-3702(73)90011-8

Simon, H. A., & Newell, A. (1971). Human problem solving: The state of the theory in 1970. *American Psychologist*, *26*(2), 145–159. https://doi.org/10.1037/h0030806

Stake, R. E. (1995). *The art of case study research*. Sage.

Standl, B. (2017). Solving everyday challenges in a computational way of thinking. In V. Dagienė & A. Hellas (Eds.), *Informatics in schools: Focus on learning programming* (pp. 180–191). Springer. https://doi.org/10.1007/978-3-319-71483-7_15

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. In J. Sheard & C. S. Montero (Eds.), *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 120–129). ACM. https://doi.org/10.1145/2999541.2999542

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

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical Engineering Sciences*, *366*(1881), 3717–3725.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K–12 classrooms. *TechTrends*, *60*(6), 565–568. https://doi.org/10.1007/s11528-016-0087-7

Yin, R. K. (2018). *Case study research and applications: Design and methods* (6th ed.). SAGE.

Yin, Y., Hadad, R., Tang, X., & Lin, Q. (2020). Improving and assessing computational thinking in maker activities: The integration with physics and engineering learning. *Journal of Science Education and Technology*, *29*(2), 189–214. https://doi.org/10.1007/s10956-019-09794-8

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Ezeamuzie, N. O., Leung, J. S. C., Garcia, R. C. C., & Ting, F. S. T. (2022). Discovering computational thinking in everyday problem solving: A multiple case study of route planning. *Journal of Computer Assisted Learning*, *38*(6), 1779–1796. https://doi.org/10.1111/jcal.12720

APPENDIX

EXAMPLES OF THE TAKE-HOME PROGRAMMING TASKS

Question 1

In this exercise, the 26 letters in English will be mapped to a number. The mapping is not case sensitive (i.e., lowercase, and uppercase of a letter will map to the same number)

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

Write a Python program that will accept users' surname and calculate the sum, product, and average of the characters as shown in the letter-to-number mapping above.Example: For an input 'Cheung',

$$\text{Sum} = C + H + E + U + N + G = 3 + 8 + 5 + 21 + 14 + 7 = \mathbf{58}$$

$$\text{Product} = 3 \times 8 \times 5 \times 21 \times 14 \times 7 = \mathbf{246,960}$$

$$\text{Average} = \text{sum}/\text{number of character} = 58/6 = 9.67$$

```
Please enter a surname. Accepts only English alphabets: Cheung
Surname:  cheung
Sum:  58
Product:  246960
Average:  9.67
```

```
Please enter a surname. Accepts only English alphabets: Biden
Surname:  biden
Sum:  34
Product:  5040
Average:  6.8
```

Question 2

In cryptography, we hide and distort messages so that adversaries or third parties cannot understand our communication with a trusted partner.

Write a program that can encrypt and decrypt a message by swapping every character with the fifth letter (e.g., a → f, b → g, u → z, w → b). The characters are non-case sensitive (i.e., uppercase and lowercase characters will be treated as the same). Numbers and other special characters will remain unchanged.

```
Enter 'E' for Encryption and 'D' for Decryption: e
Please enter a message for encryption or decryption: I love you
Message:  i love you
********** Encrypted Message **********
n qtaj dtz
**************************************
Enter 'E' for Encryption and 'D' for Decryption: d
Please enter a message for encryption or decryption: n qtaj dtz
Encrypted Message:  n qtaj dtz
********** Decrypted Message **********
i love you
**************************************
Enter 'E' for Encryption and 'D' for Decryption: E
Please enter a message for encryption or decryption: Programming is fun but requ
ires deep thinking
Message:  programming is fun but requires deep thinking
********** Encrypted Message **********
uwtlwfrrnsl nx kzs gzy wjvznwjx ijju ymnspnsl
**************************************
Enter 'E' for Encryption and 'D' for Decryption: d
Please enter a message for encryption or decryption: uwtlwfrrnsl nx kzs gzy wjvz
nwjx ijju ymnspnsl
Encrypted Message:  uwtlwfrrnsl nx kzs gzy wjvznwjx ijju ymnspnsl
********** Decrypted Message **********
programming is fun but requires deep thinking
**************************************
Enter 'E' for Encryption and 'D' for Decryption:
```