Journal of Pedagogical Research
Volume 7, Issue 2, 2023
https://doi.org/10.33902/JPR.202318518

Research Article

What challenges emerge when students engage with algorithmatizing tasks?

John Griffith Tupouniua

Massey University, Auckland, New Zealand (ORCID: 0000-0001-6541-4229)

A critical part of supporting the development of students' algorithmic thinking is understanding the challenges that emerge when students engage with algorithmatizing tasks—tasks that require the creation of an algorithm. Knowledge of these challenges can serve as a basis upon which educators can build effective strategies for enhancing students' algorithmic thinking skills. This paper presents three illustrative cases of emergent challenges evident as students grapple with the process of creating an algorithm. The first challenge highlights discrepancies between the method with which students solve a problem and the algorithm they create, and claim would, when implemented, solve the same problem. The second challenge pertains to the persistence of students' normatively incorrect algorithms, despite going through multiple iterations of testing and revising. Finally, the third challenge concerns issues around the use of test problems for supporting students in their creation of generalized algorithms. These three challenges are discussed using student data (illustrative cases) from three different mathematical algorithmatizing tasks. Suggestions are put forth for addressing some of these challenges, with a particular emphasis on practical pedagogical suggestions for cultivating students' mathematical thinking in the context of algorithmatizing tasks.

Keywords: Algorithmic thinking; Algorithmatizing tasks; Student-invented algorithms; Mathematical thinking

Article History: Submitted 13 October 2022; Revised 9 March 2023; Published online 10 June 2023

1. Introduction

Computational thinking (CT) was defined by Wing (2006) as a process that "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). Stephens and Kadijevich (2019) posited that CT had four main aspects: decomposition, abstraction, algorithmization, and automation. Furthermore, they argued that algorithmic thinking (AT) differs from CT only in terms of automation (i.e., CT has the additional concern of representing solutions and strategies in ways that computers can process them). More specifically, AT involves the processes of creating, testing, and revising an algorithm—''a precisely-defined sequence of rules telling how to produce specified output information from given input information in a finite number of steps'' (Knuth 1974, p. 323). Furthermore, AT can be regarded as a type of mathematical thinking, centered around algorithms, in the same way there are other types of mathematical thinking such as functional thinking,

Address of Corresponding Author

John Griffith Tupouniua, Ph.D, Massey University, Albany, Auckland, New Zealand.

✉ j.g.tupouniua@massey.ac.nz

How to cite: Tupouniua, J. G. (2023). What challenges emerge when students engage with algorithmatizing tasks?. *Journal of Pedagogical Research, 7*(2), 93-107. https://doi.org/10.33902/JPR.202318518

statistical thinking, geometrical thinking and so forth (Stephens & Kadijevich, 2019).

Efforts to enhance students' algorithmic thinking skills in mathematics education come in different forms, one of which revolves around students engaging with *algorithmatizing tasks* that require creating and articulating an algorithm for solving relatively novel problems (e.g., Marrongelle, 2007; Rasmussen et al., 2005; Tupouniua, 2019). Although having students work on algorithmatizing tasks has its benefits in terms of enhancing students' algorithmatizing skills and conceptual understanding of traditional algorithms (Carroll, 2000; Futscheck & Moschitz, 2010; Son, 2016; Tupouniua, 2019), challenges exist with respect to how educators can support students as they are working on these algorithmatizing tasks (see Moala et al., 2019; Peressini & Knuth, 1998; Tupouniua, 2019). Merely asking students to work on an algorithmatizing task does not necessarily mean that students will engage in the processes of algorithmatizing or develop their algorithmatizing skills (Tupouniua, 2019). Hence, a critical part of supporting the development of students' algorithmatizing skills is understanding the challenges that emerge when students engage with algorithmatizing tasks. Understanding such challenges can serve as a basis upon which educators can build effective strategies for supporting students.

This paper presents three illustrative cases corresponding to three emergent challenges that some students encountered as they grappled with creating their own algorithm. These three challenges are ones that have been evident throughout my research in mathematics education, which primarily focuses on exploring a diverse range of aspects pertaining to algorithmic thinking (e.g., Moala et al., 2019; Tupouniua, 2019, 2020, 2023). The first challenge highlights discrepancies between the method with which students solve a problem versus the algorithm they create, and claim would, when implemented, solve the same problem. The second challenge pertains to the persistence of students' (normatively) incorrect algorithms despite going through multiple iterations of testing and revising. The third challenge concerns issues around the use of test problems for supporting students in their creation of generalized algorithms. With respect to relevant literature, these three challenges are discussed using student data from three different mathematical algorithmatizing tasks. Some suggestions are put forth for addressing some of these challenges, with an emphasis on practical pedagogical suggestions for cultivating students' mathematical thinking in the context of algorithmatizing tasks.

2. Research Design

The data presented in the three illustrative cases were collected as part of a larger ongoing self-designed study (Tupouniua, 2023) exploring various aspects of students' algorithmic thinking across different educational sectors in New Zealand. This larger study is a continuation of work that began during my doctoral study (Tupouniua, 2019) which examined the mechanisms by which students' algorithms emerge. Data for the present study are collected via task-based interviews (Maher & Sigley, 2020), in which students work either in pairs or individually on an algorithmatizing task in one-hour sessions (with no more than one session on any given day). Past studies have shown that algorithmatizing tasks are useful for eliciting students' algorithmic thinking and providing educators with ideas for enhancing students' algorithmic thinking (e.g., Marrongelle, 2007; Rasmussen et al., 2005; Tupouniua, 2019).

Students are given 45 minutes to work on the task, then a 15-minute follow-up semi-structured interview is conducted after every task to clarify and gain more insight into students' thinking. At the time of writing this paper, data had been collected from 43 students (23 university students and 20 secondary school students). The 43 students were divided into: eight pairs of university students, seven individual university students; seven pairs of secondary school students; and six individual secondary students. Each group/individual worked on three different tasks in three different sessions. More specific information regarding the tasks presented in each of the illustrative cases chosen for this paper are presented in the respective sections (cases) below.

The topics of all the tasks used in the larger study were chosen so that they were familiar and readily accessible, relative to the participants' mathematical background. The decision to have

readily accessible topics meant that the students spent more of the limited time (45 minutes) creating and refining their algorithms rather than trying to understand the problem. The appropriateness of each task chosen for the study (i.e., its effectiveness for eliciting students' algorithmic thinking) was justified either by its successful implementation in past literature (e.g., Tupouniua, 2020a; Zazkis & Chernoff, 2008) or through pilot tests in which the tasks were implemented on students from relatively similar mathematical backgrounds to the participants. These pilot tests often led to minor revisions of the tasks, mainly in terms of clarifying the instructions, and removing unnecessary steps to reduce the time it took for students to reach the stages of creating and refining their algorithms. It is worth noting that the three tasks presented in the illustrative cases were not specifically designed for the purpose of eliciting the respective emergent challenges. The tasks were designed with the foremost goal of eliciting students' engagement in the process of algorithmic thinking more generally. And, although it was hypothesized based on pilot tests and previous research (e.g., Moala et al., 2019; Tupouniua, 2020b) that the tasks would elicit some challenges, the precise nature of these challenges were not known a priori.

In addition to the foregoing information regarding task design, the following criteria were used to choose the three tasks presented in the three illustrative cases for this paper: 1) given the space limitations in this paper, the task instructions and algorithms elicited are short and concise; and 2) the content of each task is straightforward and readily accessible for readers. These criteria also emphasize that algorithmic thinking can be encouraged, supported, and elicited in relatively basic mathematics tasks. This in turn supports the arguments that algorithmic thinking can be, and perhaps should be, developed from early years of school, and that there is a need to explicate the reciprocal relationship between computational and mathematical thinking, especially in the earlier years of schooling (Stephens & Kadijevich, 2019; Wu & Yang, 2022).

Each task session was video recorded and transcribed. The overall analysis of data, conducted with the help of three research assistants, involved *open interpretation* (Clement, 2000) and *thematic analysis* using both inductive and deductive code generation methods (Braun & Clarke, 2012). This open interpretation method aligns with the openness of the overarching research question of the larger study: *What challenges emerge when students engage with algorithmatizing tasks?* Before proceeding any further, it is worth explaining what the term "challenges" means in the context of this study. Challenges refer to two types of situations: 1) situations in which the participants produce normatively incorrect algorithms; 2) situations in which the participants indicated that they did not how to proceed. As such, two main goals guided the data analysis: to identify the places in students' work that corresponded to challenges; and to establish plausible reasons and explanations for these challenges.

The three ensuing sections respectively present the three illustrative cases and discuss the challenges evident in each case.

3. Challenge 1: Students find correct solutions but propose incorrect algorithms

The first challenge pertains to discrepancies between the method with which students solve a problem and the algorithm students create, and claim would, when implemented, solve the same problem (cf. Darminto, 2020; Moala, 2019; Tupouniua, 2020b). Note, this challenge differs from a similar, more common, situation whereby students can solve specific problems, but are not able to create an algorithm that would solve the general class of problems, to which the specific problems belong. For instance, a student may be able to solve specific multi-digit addition problems correctly (e.g., *23 + 145*; *567 + 453*; *1354 + 761*) but is unable to create and articulate a correct algorithm for adding any two multi-digit numbers. To clarify, the situation of interest in this section pertains to that in which students solve a problem correctly but then create an algorithm that cannot solve the same problem. For example, a student might be able to solve *23 + 145* correctly, but then produces an algorithm that when implemented fails to produce the correct solution for *23 + 145*.

This challenge is illustrated below using excerpts from the work of two groups of students'—Jon and Mac (Year 11 students); Rio and Sid (first-year undergraduate students)—on *The shuttle relay problem* (Tupouniua, 2020b).

3.1. Task and student data

Figure 1
The shuttle relay problem (Tupouniua, 2020b)

In 1975 at age 79 Dame Whina Cooper led a hīkoi (march) with 5000 people 1054 km in protest over the historic sale of Māori land and the control of land still in Māori hands. Marching from Te Hapua situated right near the top of the North Island to the Beehive in Wellington. For students to honour her commitment, sacrifice and quantify the distance she marched, the Year 9 students are going to do a shuttle relay in teams of 4. The first set will be: Student 1 will jog 1 x 25 metres; Student 2 will jog 2 x 25 metres; Student 3 will jog 3 x 25 metres; Student 4 will jog 4 x 25 metres. The second set will be: Student 1 will jog 5 x 25 metres; Student 2 will jog 6 x 25 metres, Student 3 will jog 7 x 25 metres, Student 4 will jog 8 x 25 metres. The remaining sets repeats the pattern above. Create an algorithm (method) that allows you to calculate the total distance the team has jogged, after any number of sets.

After reading the task instructions, the students and the interviewer discussed the problem to ensure that the students understood how the relay worked and what they needed to create. Below are excerpts from the two groups' work:

Jon and Mac

1. Mac: One set is…[*writes down 25 x 1 + 25 x 2 + 25 x 3 + 25 x 4*].
2. Jon: Two fifty…twenty-five times ten.
3. Mac: Yeah, two hundred and fifty meters.
4. Jon: Then, two sets is [*writes down 25 x 1 + 25 x 2 + 25 x 3 + 25 x 4 + 25 x 5 + 25 x 6 + 25 x 7 + 25 x 8*].
5. Mac: Yes, that's correct, so…[*uses calculator*]…nine hundred meters.
6. Interviewer: Can I just ask…what would an algorithm be for calculating the total distance covered over two sets?
7. Mac: We went up to four in the first set, and then eight after two sets, so you're just doubling the distance of the first set.
8. Jon: Yeah, for two sets it's just [writes *25 x 10 x 2…Mac nods his head in agreement*].

Rio and Sid

1. Sid: [*writes 25 x (1 + 2 + 3 + 4) = 25 x 10 = 250*]
2. Rio: That's the first set [*Sid nods in agreement*]. Then, next set is [writes *25 x (5 + 6 + 7 + 8) = 25 x 26 = 650*]…third set is…
3. Sid: This [*points to what he wrote*: *25 x (9 + 10 + 11 + 12) = 25 x 42 = 1050*].
4. Rio: Cool. So total distance over three sets is…just add 'em up [*writes 25 x 10 + 25 x 26 + 25 x 42 = 1950*].
5. Sid: Yes yes.
6. Interviewer: OK, so you've calculated the total distance run over three sets. Can you tell me what an algorithm could be for…finding the total distance that the students have run over three sets would be?
7. Sid: You start off with ten twenty-five-meter sprints…[*points at what Rio wrote in Line 4*] then each next set, you add on 16, so for three sets the algorithm is [*writes 25 x 10 + 25 x (16 x 2)*].

8. Rio: Right, that's more efficient, then what I did [*laughs*]…two hundred fifty, then twenty-five by sixteen twice, for second and third sets.

3.2. Analysis and discussion of challenge 1

In interpreting the excerpts above, as done in previous work (e.g., see Moala, 2019; Tupouniua, 2020b) I use the term "procedure" to refer to how the students solved a problem (i.e., how they found a solution), and the term "algorithm" to refer to what they present as a way of solving the problem. The use of these different terms is not merely for ease of discussion, but rather to highlight a crucial distinction between two acts (Moala, 2019; Tupouniua, 2019, 2020b): 1) the act of solving a given problem correctly; and 2) the act of creating and articulating an algorithm that the creator claims would, when implemented, solve the given problem correctly.

In both excerpts above, discrepancies are evident between how the students found a solution to a problem (i.e., their procedure), and the algorithm that they presented for the same problem. For instance, Jon's and Mac's procedure for finding the total distance covered in two sets was $25 \times 1 + 25 \times 2 + 25 \times 3 + 25 \times 4 + 25 \times 5 + 25 \times 6 + 25 \times 7 + 25 \times 8$, but their algorithm was $25 \times 10 \times 2$. Of course, the discrepancy between procedure and algorithm is not a problem per se, because a discrepancy between the procedure and the algorithm can be expected to some extent. That is, upon reflecting on one's procedure, one might create a correct algorithm that is a more concise and polished form of the procedure. However, the focus of the discussion at hand, is those situations in which there is a correct procedure but an incorrect algorithm. So, what are some plausible explanations for, and how might students be supported in, such situations?

Firstly, in alignment with findings from past research (e.g., Moala, 2019; Tupouniua, 2019, 2020b), the students in the present study (including the two groups above) were at times unaware that the algorithm they put forth differed from the procedure they had used to solve a specific problem. In other instances, students were aware that their algorithm was not equivalent to their procedure (line for line), but they claimed that their algorithm was a refined version of their procedure, and that the algorithm would produce the same correct result. In either scenario, students seldomly verified whether their algorithm was indeed the same as their procedure or, if different, whether the algorithm produced the same correct result as their procedure. Therefore, it is important that students are supported in moving beyond assuming that their algorithm is correct simply because their procedure is correct. In other words, students need support with verifying and validating their algorithms.

One way educators might provide such support, especially in situations where students claim that their algorithm differs from their procedure but still produces the same result, is by prompting students to implement both the procedure and the algorithm on specific cases. This is often useful because it can uncover the fact that the procedure and the algorithm do not necessarily yield the same result (e.g., Moala, et al., 2019; Tupouniua, 2020b). However, for situations in which students are unaware of discrepancies between their procedure and their algorithm, the aforementioned prompt to implement both the procedure and the algorithm may be ineffective because some students will simply implement either the procedure or the algorithm twice, as found in past research (Tupouniua 2019, 2020b). In such situations, it may be useful to encourage students to write down their procedure while they are solving the problem. Then, after creating the algorithm, they can compare it with the procedure to notice discrepancies (Moala, 2014; Tupouniua, 2019).

Supporting students' validation and verification processes can also counter students' assumptions of linearity, which can hinder the development of a correct algorithm despite having a correct procedure (Tupouniua, 2019, 2020b). For instance, in Jon and Mac's case above, they assumed that the total distance increases in constant increments of 250 meters. Similarly, Sid and Rio's algorithm contained the assumption that the total distance increases after the first set in constant increments of (25×16) meters. Such assumptions of linearity have been discussed in past studies and recognized as a common obstacle in students' mathematical thinking (e.g., De Bock et al., 2002; Van Dooren et al., 2003). Researchers argue that this assumption becomes ingrained in

students' intuitive knowledge making it persistent in the face of conflicting evidence and correct without a need for justification (Fischbein, 1987). The influence that the assumption of linearity has on students' thinking is particularly interesting in the above excerpts because both groups of students had a correct procedure for finding the total distance covered over two and three sets, respectively. Yet, in both cases, the students examined their procedure, then very quickly described it in a linear manner (i.e., linear growth of distance covered in relation to the number of sets). Thus, it is fascinating to note how the students' assumptions of linearity influenced not the act of solving the problem (i.e., the procedure) as demonstrated in past studies (e.g., Van Dooren et al., 2003; Tupouniua, 2020b) but rather the act of describing how they thought they had solved the problem (i.e., the algorithm).

Furthermore, students' conceptions of the term "algorithm" can also contribute to the creation of an incorrect algorithm despite the use of a correct procedure (e.g., see Moala et al., 2019; Tupouniua, 2019, 2020b). For instance, in the follow-up interviews, Rio and Sid declared an algorithm to be: "an easy way of finding the distance for many sets." Jon and Mac described it as "a quick way to find the total distance for, like, when you have big sets, like twenty sets, it'll take too long to add up each one!" Such definitions of "algorithm" can be problematic when students are prompted to implement and validate their algorithm on a specific small(er) case(s), because the students implement their procedure instead of their algorithm (Moala et al., 2019; Tupouniua, 2020b). This suggests that some students view the procedure as something to use in solving small cases (e.g., one set, two sets) while an algorithm is used for solving the large cases (Moala et al., 2019, Tupouniua, 2020b). Consequently, students might not validate their algorithm on small cases, when asked to do so, because it is reserved for large(r) cases. Based on previous research (Moala, 2019; Moala et al., 2019; Tupouniua, 2019, 2020b), two suggestions for addressing this issue are: a) broadening students' conceptions of algorithms and their relationship with/to procedures; and b) choosing appropriate problems on which students are asked to validate their algorithm—e.g., students who perceive algorithms as reserved for larger cases, should be given relatively large cases to test their algorithm.

4. Challenge 2: Students' incorrect algorithms persist through multiple revisions

The process of creating an algorithm is iterative—an initial algorithm is proposed, tested on certain problems, and revised according to the results of the tests (Campbell et al., 1998; Marrongelle, 2007; Mingus & Grassl, 1998; Rasmussen et al., 2005). The end goal of this iterative process is to produce an algorithm that correctly solves all the problems that it is expected to solve. Yet, research has found that students can often engage in multiple iterations of testing and revising, without producing a correct algorithm (Ashlock, 2001; De Bock et al., 2002; Moala et al., 2019). This challenge is illustrated below, using data from the work of two students (Ben and Kai; Year 9 and 10 respectively) on a task about comparing simple fractions.

4.1. Task and student data

The task (Zazkis & Chernoff, 2008) involves creating an algorithm for determining which of two simple fractions has a larger magnitude. At the beginning of every session involving this task, the participating students listed the different algorithms they knew for comparing two fractions: 1) converting fractions into ones with a common denominator, and then comparing the numerators; 2) subtracting one fraction from the other ($a - b$); if $a - b$ was positive then a was larger, and if negative, then b was larger; 3) converting fractions into decimals/percentages and comparing them. The participants were informed that all these algorithms were valid, but for this task they needed to devise a different algorithm.

A chronological summary of Ben and Kai's algorithms for comparing two fractions is presented in Table 1. Ben and Kai began with an initial algorithm, and then engaged in three cycles of testing, validating, and revising their algorithm with respect to specific problems for which their current algorithm did not produce a correct answer.

Table 1
Ben and Kai's algorithms for comparing two simple fractions

Algorithm (as expressed by students verbatim)	*Algorithm (as interpreted by researchers)*	*Problem(s) on which the students validated their algorithm (i.e., problems identified for which the current algorithm produced a correct answer)*	*Problem(s) that motivated a revision of the algorithm (i.e., problems identified for which the current algorithm produced an incorrect answer)*
"Take away numerator from denominator. The fraction with the smaller difference is bigger."	Algorithm 1: For each fraction, subtract the numerator from the denominator. The fraction corresponding to the smaller difference is the larger fraction.	*3/5* vs *4/5*	*2/6* vs *3/8* *4/5* vs *8/10*
"Multiply the numerator and denominator. The bigger fraction has the bigger product."	Algorithm 2: For each fraction, multiply the numerator and the denominator. The fraction corresponding to the bigger product is the larger fraction.	*2/6* vs *3/8* *3/6* vs *4/6*	*1/3* vs *2/6*
"Use the same algorithm [*Algorithm 2*], but only for non-equal fractions. If equal, then the fractions are equal."	Algorithm 3: For each fraction, multiply the numerator and the denominator. The fraction corresponding to the bigger product is the larger fraction. This algorithm will work only if the two fractions are not equal.	*7/8* vs *6/7* *3/4* vs *8/10*	*19/23* vs *23/29*
"Find difference between numerator and denominator. The smaller the difference, the bigger the fraction."	Algorithm 4: For each fraction, subtract the numerator from the denominator. The fraction corresponding to the smaller difference is the larger fraction.	*19/23* vs *23/29*	N/A (the session ended before students made any further revisions)

4.2. Analysis and discussion of challenge 2

As summarized, in Table 1, Ben and Kai went through multiple iterations of testing and revising their algorithm, but after 45 minutes, they still had an algorithm that could not produce the correct answer for any pair of simple fractions. In fact, after forty-five minutes of testing and revising, Ben and Kai ended up with the same incorrect algorithm they had initially created (i.e., Algorithm 1 and Algorithm 4 are the same). Of course, having an incorrect algorithm at the end of multiple iterations is not unusual (Tupouniua, 2019). However, examining the ways in which Ben and Kai revised their algorithms provides some plausible explanations for why incorrect algorithms might persist through multiple iterations of testing and revising.

One plausible explanation revolves around the mechanism of *localized considerations* (Moala et al., 2019), In the context of algorithmatizing, localized considerations refers to the practice of evaluating the aptness (Kontorovich et al., 2012) of an algorithm, or a feature thereof, with respect to a relatively limited subset of information. For instance, in Ben and Kai's work, each algorithm was created and validated on a very limited sample of problems. And, at no point during the 45-minute session did they explicitly verify whether their algorithm worked for *all* simple fractions. Localized considerations was apparent in the first revision that Ben and Kai conducted on Algorithm 1—they acknowledged that their algorithm did not work on both of *2/6* vs *3/8* and *4/5* vs *8/10*, however their revised algorithm (Algorithm 2) was only validated on the former.

Note, localized considerations is not always disadvantageous. At the initial stages of algorithm design, it can be beneficial to start by creating an algorithm that works on specific (local) problems (Futschek & Moschitz, 2010), and then to refine this algorithm multiple times so that it would ultimately work for a larger group of problems. As such, the issue in Ben and Kai's case was not so much that they employed localized considerations per se, but rather that they did not effectively transition from local to more global problems in their considerations. As aforementioned, Ben and Kai's first revision resulted in an algorithm that worked for only one of the problems that motivated the revision of Algorithm 1. This was a common occurrence across a large majority of participants who worked on this task—even though students revised an algorithm because it did not work for certain problems, the revised algorithm only worked for some of the problems that motivated the revisions.

Another plausible explanation for the persistence of incorrect algorithms lies in the lack of validation and verification. For instance, a student might propose the following algorithm for subtracting one positive integer from another: *subtract the smaller digit from the corresponding (same place value) larger digit*. This algorithm correctly solves problems such as *56 – 25 = 31*, but not problems such as *42 – 13* for which it would yield *31*. Some would argue that such a student has not verified their algorithm, by for example checking whether *13 + 21* yields *42* (e.g., Booth et al., 2014; Brown & Quinn, 2006). However, in examining Ben and Kai's case, it can be argued that the issue resides not in whether they validated their algorithm, but rather the way in which they validated their algorithm(s) (Czocher, 2018; Czocher et al., 2018). Ben and Kai validated their algorithm in each iteration by identifying at least one problem for which the algorithm produced the correct answer. However, the specific way in which Ben and Kai validated their algorithm is problematic (e.g., the algorithm was proven to work only on a subset of problems that motivated its revision). As such, it may be useful to encourage students to verify whether the revised algorithm correctly solves, at the very least, the problem(s) that motivated the revision (Moala et al., 2019; Tupouniua, 2020a). Moreover, students need to be supported with the process of determining whether the problems on which they validate and verify their algorithms are sufficiently diverse and represent the global set of problems that the algorithm is expected to solve (Moala, 2019; Moala et al., 2019; Tupouniua, 2020a). To this effect, the literature and pedagogy around proofs/refutations and example/counterexample generation may be useful to consider (e.g., Lew & Zazkis, 2019; Pinto & Cooper, 2023; Sinclair et al., 2011).

5. Challenge 3: Students "struggle" to create a generalized algorithm

In the previous section, Ben and Kai were given problems for which their current algorithm did not produce the correct solution. These problems motivated them to revise their algorithm. In past research (e.g., Moala et al., 2019; Tupouniua, 2020a) these problems have been referred to as *counterexamples to students' algorithms.* In theory, these counterexamples are meant to motivate students to construct a generalized algorithm. However, I have also argued (Tupouniua, 2020, 2023) that past studies commonly construe "generalized algorithm" as an algorithm that correctly solves all problems that the teacher/researcher wants the student(s) to solve. Consequently, past studies consider all students' revisions that do not result in a generalized algorithm to be incorrect, and they deem all counterexamples that do not motivate a revision that yields a generalized algorithm as ineffective. Under such interpretations of "generalized algorithms," I would be inclined to say that a large of majority of students that I have observed engaging in algorithmic thinking failed, because they did not create a generalized algorithm. However, as discussed below, a slight change in one's frame of thinking in relation to students' algorithms and counterexamples, can result in not only recognizing the legitimacy of students' work, but also realizing opportunities to support and build on students' work more effectively.

5.1. Task and student data

The data presented below come from the work of Liv and Mel (pseudonyms). At the time of data collection, Liv and Mel were primary school teachers enrolled in a masters-level course in mathematics education at a New Zealand tertiary institution. Liv and Mel had two and five years teaching experience respectively at the primary level. The task (Tupouniua, 2023) at hand invited the students to create an algorithm for expanding the expression $(a+b)^2$ where a and b can be any two real numbers.

At the beginning of the session, a discussion between the interviewer and the students took place around simplifying and expanding expressions. From this discussion, I gathered that both students were able to expand single-variable expressions. But they found it difficult to expand multi-variable expressions especially involving exponents greater than one. The table below provides a chronological summary of the development of Liv and Mel's algorithm. Liv and Mel began with an initial algorithm, and then engaged in six cycles of testing, validating, and revising with respect to specific problems/counterexamples (see right-most column in Table 2).

As alluded to above, a slight change in one's frame of thinking can potentially lead to a greater appreciation of students' algorithmatizing processes. Along these lines, I (Tupouniua, 2019, 2020a) interpret the notion of *student-invented algorithm* as having two parts: firstly, a set of instructions (SoI) that is intended to enable one to solve specified problems; and secondly, a domain of validity (DoV)—a set of problems for which the set of instructions would, according to the student, yield the correct answer. The following analysis and discussion highlight some of the affordances of the aforementioned interpretation, including a crucial distinction between the problems which students intend to solve with their algorithms, and the problems that external sources (e.g., teachers, interviewers) want them to solve.

5.2. Analysis and discussion of challenge 3

One might say that Liv and Mel, as evidenced by their work in Table 2, made little to no progress because they could not get past their initial algorithm, $a^2 + b^2$, which does not yield the correct solution for the general class of problems that they were asked to solve—all pairs of real numbers a and b. However, as alluded to above, this "general class of problems" tends to be pre-determined and fixed by an external source (e.g., teacher, researcher) and does not necessarily align with the problems that students intend to solve with their algorithms (Tupouniua, 2020a, 2023). The external source's fixed view of "problems to be solved," especially when it does not align with the students' domain of validity, can be problematic (Tupouniua, 2023). For instance, problems that

Table 2

Summary of Liv and Mel's algorithms for expanding $(a + b)^2$

Algorithm (as expressed by student(s) verbatim)	*Set of instructions (SoI)*	*Domain of validity (DoV)*	*Problem(s) on which students validated their current SoI*	*Problems/counterexamples that motivated students to revise their current algorithm*
"...square a and b, and then add together…for any two numbers."	$a^2 + b^2$	All pairs of a and b	None	$a = 2, b = -2$
"…square a and b, add 'em together, but only when a and b don't have the same absolute value."	$a^2 + b^2$	All pairs of a and b, such that $\|a\| \neq \|b\|$	None	$a = 1, b = 1$
"…square a and b, and then add together, when a and b have different absolute value."	$a^2 + b^2$	All pairs of a and b, such that $\|a\| \neq \|b\|$	None	$a = 2, b = 1$
"When a and b have unequal absolute value, add a square to b square, and then add four."	$a^2 + b^2 + 4$	All pairs of a and b, such that $\|a\| \neq \|b\|$	$a = 2, b = 1$	$a = 0, b = 4$
"…a square plus b square…should work for all numbers of a or b."	$a^2 + b^2$	All pairs of a and b	$a = 0, b = 4$	$a = -1, b = 2$
"…a square plus b square, when one of a or b is positive and the other one is zero"	$a^2 + b^2$	All pairs of a and b, such that one is positive, and the other is zero	$a = 0, b = 4$	$a = 0, b = 0$
"…anytime one or both of the two numbers is zero, then square a plus square b."	$a^2 + b^2$	All pairs of a and b, such that at least one of a and b is zero	N/A	N/A (session ended before any further development to algorithm)

are counterexamples to the external source, in the sense that they are part of the general class of problems which the student's algorithm does not produce a correct solution, might not be perceived as a counterexample by the students because it does not exist in their DoV (Tupouniua, 2020a, 2023).

This is evident in the second-third iteration of Liv and Mel's algorithm. The students were given what the interviewer deemed to be a counterexample ($a = 1$, $b = 1$) to their algorithm. However, this problem was not a counterexample to the students because it was not an element of their current DoV (i.e., all pairs of a and b, such that $|a| \neq |b|$). On the one hand, one might say that the students ignored and dismissed the counterexample they were given (e.g., Zazkis & Chernoff, 2008). On the other hand, one could argue that the interviewer had failed to acknowledge the DoV of the students' algorithms, thus giving them a problem which they did not intend for their algorithm to solve. Failing to recognize and acknowledge students' algorithms in terms of their set of instructions and domain of validity can result in the external source missing out on the small, nuanced gains that students make as they revise their algorithm in response to counterexamples (Tupouniua, 2019, 2020a, 2023). This point is discussed further in the following paragraphs.

Throughout their work, Mel and Liv revised the DoV of their algorithm by progressively recognizing and identifying exceptions (counterexamples) within their current DoV. Because these revisions did not result in an algorithm capable of solving the entire pre-determined general set of problems (i.e., all pairs of real numbers a and b), one might deem these revisions unproductive and the counterexamples ineffective because they were merely treated as exceptions (Moala et al., 2019; Selden & Selden, 1998; Zazkis & Chernoff, 2008). However, I argue that such revisions are legitimate, and a sign of progress (Tupouniua, 2020a, 2023). Liv and Mel made progress in their work by narrowing the domain of validity for their algorithm to the point where they had identified the normatively correct DoV for their algorithm (i.e., $(a + b)^2 = a^2 + b^2$ for all pairs of a and b, such that at least one of a and b is zero).

Mel and Liv's progress can go unnoticed, unsupported, and unrewarded if the external source maintains a fixed notion of "the problems to be solved" and disregards the dynamic nature of the DoVs of students' algorithms. This type of revision that Mel and Liv conducted, treating counterexamples as exceptions, is similar to the process of *exception-barring* (Lakatos, 1976)—a process in which counterexamples are treated as exceptions to a theorem, to delineate a safe domain of validity for the theorem. It also aligns with Sierpinska's (1994) description of conceptual understanding, which involves not only looking for universally valid ways of thinking, but also identifying the domain of validity of locally correct ways of thinking.

Moreover, past studies suggest that students' prior experiences with examples can explain why they dismiss counterexamples as mere exceptions that do not affect the validity of their conjectures (e.g., Selden & Selden, 1998; Zazkis & Chernoff, 2008). That is, some students understand that a few examples do not suffice to prove the universal validity of a conjecture. Consequently, some students believe that a few counterexamples do not suffice to disprove a conjecture. Selden and Selden (1998, p.1) suggest that "this can happen when a counter-example is perceived as 'the only one that exists', rather than being seen as generic".

However, if *generic* is interpreted as *a representative of a larger class of objects*, then it can be argued that Liv and Mel considered the counterexamples they were given as generic. For example, they viewed $a = 2$ and $b = -2$ as a representative of all pairs of a and b, such that $|a| = |b|$. Liv and Mel removed the classes represented by the counterexample from the current DoV to obtain a more normatively correct DoV of their algorithm. As such, Liv and Mel's work suggests that in some cases, the issue might not be that students do not consider a counterexample as generic, but rather that the general class of which the students perceive a counterexample to be a representative does not align with the general class of problems that the external source (e.g., teacher, researcher) has in mind (e.g., Tupouniua, 2020a, 2023). Once again, paying attention to the dynamic DoVs of students' algorithms rather than maintaining a fixed view of the problems to solve affords

opportunities for educators to build on students' thinking and offer appropriate counterexamples to further support the development of their algorithms (Tupouniua, 2020a, 2023; Zazkis & Chernoff, 2008).

6. Concluding remarks and future directions

A critical aspect of supporting the development of students' algorithmic thinking is understanding the challenges that emerge when students engage with algorithmatizing tasks. This paper presented and discussed three examples of such challenges, which can serve as a basis upon which educators can build effective strategies for supporting students. I conclude this paper by highlighting some key findings and potential directions for future research.

This study revealed, among other things, some affordances of paying attention to the processes by which students create their algorithms (the how and why), rather than merely the end-product. Examining these processes provides valuable insight into the challenges that students encounter. For example, as illustrated in Section 4, instead of merely asking whether students have validated their algorithm, it is also useful to ask how they are validating their algorithms. Also, as discussed in Section 5, changing one's frame of thinking about notions such as "generalized algorithm" and the goals toward which we often expect students to work can uncover certain nuances in students' thinking and reasoning, which in turn provides alternative ways of understanding and supporting students' algorithmic thinking (Tupouniua, 2020a, 2023).

This study also echoes claims from past research about how algorithmic thinking can complement the teaching and learning of mathematical concepts; and similarly, mathematical concepts can be taught and learned in a way that develops students' algorithmic thinking (e.g., Rasmussen et al., 2005; Ross, 1998; Stephens & Kadijevich, 2019). The three illustrative cases provide some evidence that getting students to engage in the iterative process of creating their own algorithms, through algorithmatizing tasks, can not only be a means of developing students' conceptual understanding of mathematical concepts (e.g., comparing fractions, describing rules for growing patterns, expanding expressions) but also gives students valuable experience in engaging with fundamental constructs and processes of mathematical reasoning such as counterexamples and validation. Numerous studies have found that, in terms of problem-solving skills, primary and secondary school students who are given opportunities to create their own algorithms outperform students who are taught conventional algorithms, particularly in activities which present students with relatively novel scenarios (e.g., Clarke, 2005; Futschek & Moschitz, 2010; Madell, 1985; Ross, 1998). Hence, there are well-grounded reasons to make more prevalent the use of algorithmatizing tasks in mathematics teaching (Ross, 1998; Stephens & Kadijevich, 2008).

Lastly, the present study focused primarily on the activity and perspective of students. Additionally, the results (challenges) presented in this study emerged in non-classroom settings. However, if the results of this study are to have any practical impact on pedagogy, then there is a need to discuss the challenges described in this study with teachers, and to collaborate on ways to effectively enact (if possible) the corresponding pedagogical suggestions. This is a potential avenue for future research. Groth (2007) pointed point out that some teachers prefer teaching conventional algorithms over giving students opportunities to create their own. Reasons for this preference include teacher beliefs such as: the teacher's primary role is to present ready-made algorithms for students to practice; and mathematics development takes place according to a fixed sequence of levels. Furthermore, some studies claim that teachers' insistence on teaching conventional algorithms and rejecting the importance of student-invented algorithms can be attributed to teachers having low confidence in assessing, and supporting the development of, students' novel and idiosyncratic algorithms (e.g., Groth, 2007; Morrow & Kenney, 1998). Hence, supporting students overcome students' challenges such as those presented in this paper will require further research into, among other things, developing teachers' content and pedagogical knowledge with respect to algorithmic thinking.

Funding: No funding source is reported for this study.

Declaration of interest: No conflict of interest is declared by author.

References

Ashlock, R. (2001). *Error patterns in computations: Using error patterns to improve instruction*. Prentice Hall.

Booth, J. L., Barbieri, C., Eyer, F., & Paré-Blagoev, E. J. (2014). Persistent and pernicious errors in algebraic problem solving. *The Journal of Problem Solving*, *7*(1), 3. https://doi.org/10.7771/1932-6246.1161

Braun, V. & Clarke, V. (2012) Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology, Vol. 2: Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). American Psychological Association.

Brown, G., & Quinn, R. J. (2006). Algebra students' difficulty with fractions: An error analysis. *Australian Mathematics Teacher, 62*(4), 28-40.

Campbell, P. F., Rowan, T. E., & Suarez, A. R. (1998). What criteria for student-invented algorithms? In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics, 1998 yearbook* (pp. 49–55). NCTM.

Carroll, W. M. (2000). Invented computational procedures of students in a standards-based curriculum. *The Journal of Mathematical Behavior*, *18*(2), 111-121. https://doi.org/10.1016/S0732-3123(99)00024-3

Clement, J. (2000). Analysis of clinical interviews: Foundation and model viability. In A. E. Kelly, & R. Lesh (Eds.). *Handbook of research design in mathematics and science education* (pp. 547–589). Lawrence Erlbaum Associates. https://doi.org/10.4324/9781410602725

Czocher, J. A. (2018). How does validating activity contribute to the modeling process? *Educational Studies in Mathematics*, *99*(2), 137-159. https://doi.org/10.1007/s10649-018-9833-4

Czocher, J., Stillman, G., & Brown, J. (2018). Verification and validation: what do we mean? In Hunter, J., Perger, P., & Darragh, L. (Eds.), *Proceedings of the 41st annual conference of the mathematics education research group of Australasia* (pp. 250-257). MERGA.

Clarke, D. M. (2005). Written algorithms in the primary years: Undoing the good work. In M. Coupland, J. Anderson, & T. Spencer (Eds.), *Making Mathematics Vital* (pp. 93-98). AAMT.

Darminto, B. (2020). The Influence of Mathematical Logic Ability Toward Computer Programming to Solve Mathematical Problems. In *Proceedings of the 2nd international conference on education, ICE 2019*. Universitas Muhammadiyah Purworejo, Indonesia. http://doi.org/10.4108/eai.28-9-2019.2290992

De Bock, D., Van Dooren, W., Janssens, D., & Verschaffel, L. (2002). Improper use of linear reasoning: An in-depth study of the nature and the irresistibility of secondary school students' errors. *Educational Studies in Mathematics, 50*(3), 311–334. https://doi.org/10.1023/A:1021205413749

Fischbein, E. (1987). *Intuition in science and mathematics.* D. Reidel. https://doi.org/10.1007/0-306-47237-6

Futschek, G., & Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. *Proceedings of the 2010 constructionist approaches to creative learning, thinking and education: Lessons for the 21st century* (pp. 1-10). Comenius University, Slovakia.

Groth, R. (2007). Understanding teachers' resistance to the curricular inclusion of alternative algorithms. *Mathematics Education Research Journal*, *19*(1), 3-28. https://doi.org/10.1007/BF03217447

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323-343. https://doi.org/10.1080/00029890.1974.11993556

Kontorovich, I., Koichu, B., Leikin, R., & Berman, A. (2012). An exploratory framework for handling the complexity of mathematical problem posing in small groups. *The Journal of Mathematical Behavior*, *31*(1), 149-161. https://doi.org/10.1016/j.jmathb.2011.11.002

Lakatos, I. (1976). *Proofs and refutations*. Cambridge University Press. https://doi.org/10.1017/CBO9781139171472

Lew, K., & Zazkis, D. (2019). Undergraduate mathematics students' at-home exploration of a prove-or-disprove task. *The Journal of Mathematical Behavior*, *54*, 100674. https://doi.org/10.1016/j.jmathb.2018.09.003

Madell, R. (1985). Children' s natural processes. *Arithmetic Teacher, 32,* 20-22. https://doi.org/10.5951/AT.32.7.0020

Maher, C. A., & Sigley, R. (2020). Task-based interviews in mathematics education. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 821-824). Springer. https://doi.org/10.1007/978-3-030-15789-0_147

Marrongelle, K. (2007). The function of graphs and gestures in algorithmatization. *The Journal of Mathematical Behavior*, *26*(3), 211–229. https://doi.org/10.1016/j.jmathb.2007.09.005

Mingus, T. T., & Grassl, R. M. (1998). Algorithmic and recursive thinking: Current beliefs and their implications for the future. In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics, 1998 yearbook* (pp. 32-43). NCTM.

Moala, J. G. (2014). *On being stuck: a preliminary investigation into the essence of mathematical problem solving*. Master's thesis. The University of Auckland.

Moala, J. G. (2019). Creating algorithms by accounting for features of the solution: the case of pursuing maximum happiness. *Mathematics Education Research Journal*, *33*(2), 263-284. https://doi.org/10.1007/s13394-019-00288-9

Moala, J.G., Yoon, C., & Kontorovich, I. (2019). Localized considerations and patching: Accounting for persistent attributes of an algorithm on a contextualized graph theory task. *The Journal of Mathematical Behavior*, *55*, 100704. https://doi.org/10.1016/j.jmathb.2019.04.003

Morrow, L. J., & Kenney, M. J. (1998). *The teaching and learning of algorithms in school mathematics. 1998 yearbook*. NCTM.

Peressini, D., & Knuth, E. (1998). The importance of algorithms in performance-based assessments. In L. J. Morrow & M. J. Kenney (Eds), *The teaching and learning of algorithms in school mathematics* (pp. 56–68). NCTM.

Pinto, A., & Cooper, J. (2023). "This cannot be"—refutation feedback and its potential affordances for proof comprehension. *Educational Studies in Mathematics*, *113,* 287-306. https://doi.org/10.1007/s10649-022-10190-0

Rasmussen, C., Zandieh, M., King, K., & Teppo, A. (2005). Advancing mathematical activity: A practice oriented view of advanced mathematical thinking. *Mathematical Thinking and Learning, 7*(1), 51–73. https://doi.org/10.1207/s15327833mtl0701_4

Ross, K. A. (1998). Doing and proving: The place of algorithms and proofs in school mathematics. *The American Mathematical Monthly*, *105*(3), 252-255. https://doi.org/10.1080/00029890.1998.12004875

Selden, A., & Selden, J. (1998). *The role of examples in learning mathematics.* The Mathematical Association of America Online. https://www.maa.org/programs/faculty-and-departments/curriculum-department-guidelines-recommendations/teaching-and-learning/examples-in-learning-mathematics

Sierpinska, A. (1994). *Understanding in mathematics*. Falmer Press. https://doi.org/10.4324/9780203454183

Sinclair, N., Watson, A., Zazkis, R., & Mason, J. (2011). The structuring of personal example spaces. *The Journal of Mathematical Behavior*, *30*(4), 291-303. https://doi.org/10.1016/j.jmathb.2011.04.001

Son, J. (2016). Moving beyond a traditional algorithm in whole number subtraction: Preservice teachers' responses to a student's invented strategy. *Educational Studies in Mathematics*, *93*(1), 105-129. https://doi.org/10.1007/s10649-016-9693-8

Stephens, M., & Kadijevich, D. M. (2019). Computational/algorithmic thinking. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 483–451). Springer. https://doi.org/10.1007/978-3-319-77487-9_100044-1

Tupouniua, J. G. (2019). *Exploring mechanisms by which student-invented algorithms in mathematics emerge* [Doctoral Thesis]. The University of Auckland.

Tupouniua, J. G. (2020a). Explicating how students revise their algorithms in response to counterexamples: building on small nuanced gains. *International Journal of Mathematical Education in Science and Technology*, *53*(7), 1711-1732. https://doi.org/10.1080/0020739X.2020.1837402

Tupouniua, J. G. (2020b). Finding correct solution(s) ⇏ creating correct algorithm(s): Shedding more light on how and why. *The Mathematician Educator, 1*(2), 102-121.

Tupouniua, J. G. (2023). Differentiating between counterexamples for supporting students' algorithmic thinking. *Asian Journal for Mathematics Education*, *1*(4), 475–493. https://doi.org/10.1177/27527263221139869

Van Dooren, W., De Bock, D., Depaepe, F., Janssens, D., & Verschaffel, L. (2003). The illusion of linearity: Expanding the evidence towards probabilistic reasoning. *Educational Studies in Mathematics*, *53*(2), 113-138. https://doi.org/10.1023/A:1025516816886

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

Wu, W. R., & Yang, K. L. (2022). The relationships between computational and mathematical thinking: A review study on tasks. *Cogent Education*, *9*(1), 2098929. https://doi.org/10.1080/2331186X.2022.2098929

Zazkis, R., & Chernoff, E. (2008). What makes a counterexample exemplary? *Educational Studies in Mathematics, 68*(3), 195–208. https://doi.org/10.1007/s10649-007-9110-4