The Mathematician Educator
2020, Vol 1, No. 2, 102-121

Finding Correct Solution(s) ⇏ Creating Correct Algorithm(s): Shedding More Light on How and Why

John Griffith Tupouniua
Massey University
New Zealand

The growing global emphasis on computational thinking in school curricula requires enhancing students' competencies in algorithmatizing—i.e., creating, testing, and revising an algorithm. Building on a small body of relevant literature, the present study seeks to shed more light on a particular phenomenon pertaining to students' algorithmatizing activity—how some students can easily find the correct solutions to the problem(s) at hand, yet the algorithms that these students create are not always ones that would, when implemented, produce the correct solution(s) for the same problems. I investigate this phenomenon within the work of two Year 9 secondary school students working on a contextualized algorithmatizing task. In their work, the two students constantly solved specific instances correctly, but created algorithms that were incorrect, both for specific cases and for the general case. From my analysis of their work, I put forth plausible hypotheses for the phenomenon of interest and propose some suggestions for future research and practice.

Keywords: student-invented algorithms, procedures, algorithmatizing, computational thinking.

Introduction

Over recent years, an emphasis on computational thinking has become more and more evident in school curricula around the world (see Bocconi et al., 2016; Hart & Sandefur, 2018). For instance, in 2018 New Zealand (where the present study took place) revised its core curriculum with the goal of promoting computational thinking specifically through its new Digital Technologies Curriculum Years 1-13. This emphasis reflects a common understanding that being a competent computational thinker is necessary for coping with our modern technological society. As Hipkins (2017) asserts, "the digital curriculum is about teaching children how to design their own digital solutions and become creators of, not just users of, digital technologies, to prepare them for the modern workforce."

Agreed-upon aspects of computational thinking are: decomposition, abstraction, pattern recognition, and algorithmic thinking (Hoyles & Noss, 2015; Schute et al., 2017). In the present study, I focus on one of these aspects – algorithmic thinking, which I refer to here as *algorithmatizing* to mark an epistemological shift, from *thinking* to *doing* (see Tupouniua, 2020). The iterative process of algorithmatizing involves creating, testing, and revising an *algorithm* – a set of instructions for solving a problem (Maurer & Ralston, 1991; Thomas, 2014). Efforts to enhance students' algorithmatizing skills tend to revolve around students engaging with tasks—*algorithmatizing tasks*—that require the explicit creation and articulation

of an algorithm – a *student-invented algorithm* – for finding a solution to a given problem (e.g., see Cai et al., 1998; Carroll, 2000; Marrongelle, 2007; Rasmussen et al., 2005).

Researchers have found it particularly useful to observe students' activity on algorithmatizing tasks in order to identify challenges that students may encounter, and then devise ways that may support students overcome these challenges. Among such challenges, researchers have noted that though some students can easily find the correct solutions to the problem(s) at hand, the algorithms that these students create are not always ones that would, when implemented, produce the correct solution(s) to the same problem(s) (e.g., Cai et al., 1998; Moala, 2019; Moala et al., 2019). That is, there is a discrepancy between how the students solve a problem, and the algorithm that they create to solve the problem. This discrepancy is the phenomenon of interest of the present study.

Using Cai et al.'s (1998) distinction between procedure and algorithm, the present study explores the phenomenon of interest within the work of two secondary school students working on a contextualized algorithmatizing task. According to Cai et al. (1998) a procedure refers to the process by which one finds a solution for a particular problem at hand, while an algorithm represents a set of instructions that one devises and puts forth as a way of solving the problem, often utilising recurring steps and patterns in the procedure. Within the data presented in this paper, the two students constantly solve specific instances correctly, but create and put forth algorithms that are incorrect. I analyze their work in order to generate plausible reasons for the phenomenon of interest, building on and extending the small body of literature that have explored the phenomenon. Based on the findings, I propose some suggestions for future research and practice.

Background Literature

Although limited research exists on the phenomenon of interest, some explanations can be inferred from past studies. Analyzing the work of middle-school students on two algorithmatising tasks, Cai et al. (1998) observed that students implemented correct procedures for solving the problems they were given but struggled to create correct algorithms. Cai et al. ascribed this to the fact that the students barely reflected on and noticed any recurring patterns in their procedures when they were constructing their algorithms.

It is important to note, for the purpose of clarity, that Cai et al.'s (1998) usage of *algorithm* differs slightly from how I use the term in the present study. Cai et al. referred to a (correct) algorithm as a set of instructions for solving (all) problems within a general class of problems (e.g., a set of instructions of adding any two fractions, as opposed to a subset of fractions, say, ones with common denominators). It is with respect to this usage of algorithm that Cai et al. claim that in order for students to create an algorithm, they need to reflect and notice repeating patterns in their procedures. Thus, from the perspective of Cai et al., the issue was that students were able to solve specific instances and articulate how they can solve those specific instances, but were unable to articulate a set of instructions for correctly solving the general class of problems they were expected to solve.

In contrast to Cai et al. (1998), I view an algorithm as a set of instructions (explicitly communicated in written or verbal form) for finding the solution to a problem, and this *problem* can either represent a single problem instance or a class of problem instances. I employ this

usage of *algorithm* to account for the fact that the researcher's interpretation of the general class of problems that students are expected to solve does not always align with students' interpretation of the general class of problems that they think they are expected to solve (e.g., see Tupouniua, 2020). Further still, I have observed (see Moala[1], 2019) that sometimes students can easily find the correct solutions to a single instance of the general class of problems, and yet create an algorithm that (when implemented) does not yield the correct solution(s) for this single problem-instance.

Other explanations for the phenomenon interest can be inferred from past studies. For example, the request to create an algorithm is quite novel and thus confusing for some students, who are well-accustomed to tasks in which the foremost goal is to find solutions to the given problems (e.g., finding the solution to $17890 \div 15$ and other specific division problems) as opposed to creating an algorithm for solving the given problems (see Morrow & Kenney, 1998 for numerous examples). Furthermore, Maurer and Ralston (1991) alluded to the fact that algorithmatizing involves two distinct acts: finding a solution and creating an algorithm that can be used to find a solution. The boundaries of these two acts are clear to expert algorithmatizers, but often blurred to novices. Moreover, Threlfall (2002) provides insights into the inherent difficulty of algorithmatizing, by noting that the solution path one takes (i.e., the procedure) emerges in the moment, in a very non-linear and disorderly fashion, making it hard for one to reflect on and recall the steps of the solution process, when one is invited to create an algorithm. In this way, the request to create an algorithm may be construed as an invitation to linearize and structure a non-linear and complex process.

Researching the phenomenon of interest within the collaborative work of two groups of students on a contextualized graph theory task, Moala (2019) introduced the mechanism of *accounting for features of the solution*. Via this mechanism, students notice particular features of the solution they found, then create specific rules (instructions) within their algorithm, which guarantee that the algorithm outputs an object that possesses the noticed features. Moala proposed that the correctness of the student-invented algorithm can be dependent on the features (of their solutions) that students notice and account for.

Despite the inherent difficulties associated with creating an algorithm, studies show that some students, especially at higher educational levels, can indeed successfully create their own algorithms even for problems that are relatively novel. For example, aligning with Cai et al.'s (1998) aforementioned hypothesis, Rasmussen et al. (2005) explained how a class of undergraduate students were able to develop (re-invent) the Euler algorithm, a general algorithm that solves any first-order differential equation, by reflecting on both the solution(s) they found for specific problems (e.g., rate of change of a fish population) and the procedures by which those solutions were found. Furthermore, Marrongelle (2007) discussed how particular gestures (e.g., rise over run gesture, positive direction pen-slope gesture) and graphs (e.g., slope field graphs, tangent vector field graphs) that the students employed while predicting solution(s) to the specific equation $\frac{dP}{dt} = P$ (explained above) were summoned to support the students' creation of the Euler algorithm. More generally, Marrongelle's study showed how other aspects of students' procedures (for finding solutions to specific problems) assisted students in creating their generalized algorithms. In Tupouniua (2015), I examined the collaborative work of a group of three third-year undergraduate mathematics majors on an

[1] The author has previously published under Moala, J and Moala, J.G.

algorithmatizing task in graph theory. The task invited the group to create an algorithm for constructing a *separating set* in any *simple graph*. The students were not given any particular problem instances to solve (e.g., specific graphs in which it is relatively easy to find a separating set) but were asked to create an algorithm for constructing a separating set on any graph. One of the first things the group did, was figure out how to find the separating set in smaller cases (graphs). While solving these smaller cases, the students recognized steps that they were repeating in each case, which they built on to develop a generalized algorithm.

All in all, past research shows that students create their algorithms via various idiosyncratic mechanisms. Sometimes these mechanisms lead to a correct algorithm, and sometimes they do not. Nonetheless, paying attention to *how* students create their algorithms provides educators with necessary and valuable information for the purpose of supporting students with the algorithmatizing process. The students in the aforementioned studies (Marrongelle, 2007; Rasmussen et al., 2005; Tupouniua, 2015) were all undergraduate students who were well-acquainted with the practice of algorithmatizing. In addition, these students seemed aware about the usefulness of reflecting and building on their procedures (for solving specific problem instances) when creating an algorithm. As such, these students seemed to carry out the act of reflecting and building on their procedures naturally without external support. However, less experienced algorithmatizers, such as most students at the primary and secondary level may need more external support. The present study aims to provide some insight into the challenges that less-experienced algorithmatizers face.

Method

Participants and research setting

At the time of data collection, the participants of our study, Ata and Ben (pseudonyms) were both in Year 9 (approaching the end of Year 9) at a high-decile secondary school in New Zealand and knew each other as friends. Both students had high achievement results in mathematics and had positive dispositions toward mathematics. Both students were planning to attend university— to study civil engineering (Ata) and computer science (Ben). Ben and Ata were recruited as part of a larger project aimed at exploring students' engagement with the algorithmatizing process, building on the author's doctoral studies (see Tupouniua, 2019). The two students worked together on *The shuttle-relay task* (Tupua-Siliva & Tupouniua, 2020) in a one-hour session, which took place outside of class time and was video-recorded. The group worked in the presence of a researcher (the author), who answered clarification questions but avoided providing mathematical hints.

The shuttle-relay task: a contextualised algorithmatising graph theory task

Similar to other problems for Year 9-10 students on the New Zealand mathematics curriculum[2], the shuttle relay task is a variant of the problem of adding the first *n* counting numbers. As an added layer of difference in light of our interest in students' algorithmatizing experiences, the shuttle relay task explicitly requires students to create an algorithm. The task was pilot tested on four secondary school students (three from Year 9, and one from Year 10) in order to test the extent to which the task elicited students' algorithmatizing activity. The pilot participants underwent several iterations of creating, testing, and revising their algorithm. One of the biggest challenges they faced was the fact that the required algorithm was quadratic. Although the students had previously engaged with the quadratic equations, they had little to no

[2] See https://nzmaths.co.nz/rich-learning-activities

experience in generating a quadratic equation/algorithm. Based on the pilot study, I expected that Ata and Ben would experience similar challenges.

The task begins with a warm-up period in which the students familiarize themselves with terms such as *algorithm* and *shuttle*-relays. After completing the warm-up questions, the students are given the following scenario (Tupua-Siliva & Tupouniua, 2020, p. 1-2):

> 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 above scenario, the students and the researcher discussed the problem to ensure that the students understood how the shuttle relay worked and what they needed to create. Then, the students were asked to work together and were told that they must agree on everything that they include in their algorithm.

Data analysis

The overall nature of my analysis was to generate new theoretical perspectives on existing data, rather than to conduct detailed coding to provide generalizable empirical findings (see Clement, 2000). Additionally, my analysis is influenced by John Mason's methodology of noticing, and studies that have employed such a method of inquiry (e.g., Mason, 2002; Zazkis & Chernoff, 2008; Zazkis et al., 2008). Accordingly, I analyze Ata and Ben's engagement with *The shuttle relay task* in the hope that readers may recognize and reflect on similar instances from their own experiences. I do not intend "to capture or cover the experience of readers… [but] rather to make contact with that experience, perhaps to challenge usual interpretations, perhaps to point to some features not previously

noticed but worth discerning...validity in this methodology [of noticing] is personal, [depending] on whether you [the reader] find your actions being informed in the future, [rather than] in what I conjecture or in what I have found for myself " (Mason, 2006, p. 42-43).

More specifically, I used a five-stage process to conduct a qualitative analysis of the data. At the first stage, I read the entire transcript and identified all the algorithms and procedures that the students created and implemented, labeling each algorithm/procedure as normatively correct/incorrect. At the second stage, I marked the parts of the transcript that involved the creation and implementation of the identified procedures and algorithms from Stage 1. At the third stage, I revisited the marked excerpts from stage 2 and examined the steps that the group took to create each algorithm. I also paid attention to how the procedures were used (if at all) in the creation of the algorithms. At the fourth stage, I examined the excerpts in which the procedure was correct, but the algorithm was incorrect, checking for recurring themes in the steps by which the students created these algorithms. I then repeated this process for the instances in which the procedure was correct and the algorithm was correct, noting similarities and differences. In the final stage of the analysis, I created narratives about how and why the two students constantly solved specific instances of the problem, yet created incorrect correct algorithms.

Results

In this section, I present three chronological episodes from the group's work on *The shuttle-relay task*. Each episode begins with chosen excerpts from the group's activity followed by an analysis in which we offer explanations for the group's activity, focusing on the process by which the students propose, test, and revise their algorithm.

Episode 1

After the two students read the task instructions, the researcher asks:

1. Researcher: Can I ask you to tell me what you need to do?
2. Atas: We need to find an algorithm that solves the problem.
3. Researcher: Good. And what is that problem?
4. Ben: Finding the distance ran [sic] after a number of sets.
5. Researcher: Very good! And remember that you need an algorithm that will work regardless of what number set it is...OK? So if I give you any number, *1, 2, 3, ..., 95*, you will be able to input that number into your algorithm and find the total distance run.
6. Ben: 95 shuttles is gonna be hard [boys laugh]....
7. Ata: If we do one set...how many is that? [Ben writes down *25* x *1 + 25* x *2 + 25* x *3 + 25* x *4*].
8. Ben: If we work it out...it'll be...25 times one [*inputting numbers into the calculator]* 250.
9. Ata: That was easy! So it's just...what will our algorithm be? [*a long pause ensues*]
10. Ben: We do 250 divided by 25.
11. Ata: Why?
12. Ben: Because it's 25 [meters] per run.
13. Ata: What? Oh, duh...my bad! So 10...So the algorithm is [writes down *25* x *10*]. Yeah that's how we got it...we just timesed [sic] 25 and 10.
14. Ben: Done! Did we do it right?
15. Ata: Yeah we needed an algorithm, and that's it [Ben nods, and they turn to the researcher]

16. Researcher: OK, so your algorithm should work on any value for the number of set [sic], remember? Can you test your algorithm on, if there were two sets?
17. Ben: I knew we hadn't solved it! [laughs].
    [*Rougly two minutes pass...both students reread the problem*]
18. Ata: [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*] It's just double!
19. Ben: Double what?
20. Ata: Double the first set... so our first algorithm [points at *25* x *10*] times two.
21. Ben: Ummm.... Oh yeah... it is doubled, because first one goes up to four, and the second one goes up to eight! So second set is just double of the first set.
22. Ata: Yep.
23. Ben: And...the one for any number of sets will be...If we need to find it for any set number... would it just be 25 times 10 times whatever set it is?
24. Ata: Yeah yeah...like the fourth one goes up to sixteen so it will be four times the first one.
25. Ben: Sweet! [Writes down *25* x *10* x *number of sets*].

Table 1.
Procedures and algorithms evident in Episode 1

Item	**Procedure or Algorithm**	**Problem solved or intended to solve**
[Pro 1.1] *25* x *1 + 25* x *2 + 25* x *3 + 25* x *4*	Procedure	Used to find the distance for one set
[Alg 1. 1] 2*5* x *10*	Algorithm	Intended to find the distance for one set
[Pro 1.2] *25* x *1 + 25* x *2 + 25* x *3 + 25* x *4 + 25* x *5 + 25* x *6 + 25* x *7 + 25* x *8*	Procedure	Used to find the distance for two sets
[Alg 1.2] *25* x *10* x *2*	Algorithm	Intended to find the distance for two sets
[Alg 1.G] *25* x *10* x *number of sets*	Algorithm	Intended to find the distance for any number of sets

Analysis

Despite the emphasis placed on finding an algorithm that would work for any number of sets (see lines 1-5), the group created an algorithm for finding the distance run over one set (*Alg 1.1*). And, while this algorithm was correct, one notices a discrepancy between the group's procedure for finding the distance run in one set (*Pro 1.1*) and the process by which they create their algorithm. In lines 7-8, the group implemented the *Pro 1.1—25* x *1 + 25* x *2 + 25* x *3 + 25* x *4*—which yielded 250 meters. However, they obtained their algorithm (*Alg 1.1*) via *250/25* (see line 10), rather than noticing explicilty that *Pro 1.1* could be rewritten as *25* x (*1 + 2 + 3 + 4*) = *25* x *10*.

As such, one might infer that the students only reflected on the result of their procedure (250 meters), and created *Alg 1.1* by some sort of reverse engineering. That is, they started with the result (of the procedure) 250 meters, and then (reverse) engineered an algorithm, by proposing a process by which the same result, 250 meters, could be obtained. They knew that each shuttle run was 25 meters long, so they divided 250 by 25. Despite *Alg 1.1* being correct, the aforementioned discrepancy is fascinating. One wonders why the students reverse engineered a new algorithm, rather than put forth their procedure as their algorithm for finding the distance

over one set. Based on this observation, one might infer a potential reason why a correct procedure might not necessarily lead to a correct algorithm—rather than reflecting on how they actually solved the problem, some students might instead hypothesize a different way of solving the problem based on the solution they found. Needless to say, an unjustified hypothesis leaves room for error.

In line 16, noticing that *Alg 1.1* was specifically intended to find the distance for only one set, the researcher reminded the students that they needed an algorithm that would find the total distance for any number of sets. Immediately, the students implemented a correct procedure for finding the distance for two sets, *Pro 1.2* (*25* x *1 + 25* x *2 + 25* x *3 + 25* x *4 + 25* x *5 + 25* x *6 + 25* x *7 + 25* x *8*). However, the students proposed an incorrect algorithm for two sets (*Alg 1.2*), which they created by extending ("doubling") *Alg 1.1* (*25* x *10*). Then, assuming *Alg 1.2* to be correct, they proposed an incorrect algorithm (*Alg 1.G*) for the general case: *25* x 1*0* x *number of sets* (see lines 23-25).

Again, one notices a discrepancy between the correct procedure (*Pro 1.2*) that the group implemented to find the total distance for two sets, and the incorrect algorithm they proposed for the same problem. The students created *Alg 1.2* by noticing a relationship between their two procedures (*Pro 1.1* and *Pro 1.2*) [see line 21]—i.e., first one goes to four and the second one goes to eight—and then using this relationship to conjecture that the algorithm for two sets can be obtained by doubling *Alg 1.1*. Unlike the process by which they created *Alg 1.1*, in this process the students reflected on their procedure (rather than just the outcome). However, one might say that an assumption of linearity (i.e., the distance run over a number of sets increases steadily) distorted what the students noticed — one goes to four, and two goes to eight— and led to the faulty conclusion that the second algorithm was merely "the double" of the first. Moreover, one notices that students did not check whether *Alg 1.2* produced the same result as *Pro 1.2*. Instead, the validity of Alg 1.2 seemed to rest entirely on the foregoing relationship and assumption of linearity.

Episode 2

After Ben finishes writing [line 25 above], the researcher says:

26. Researcher: OK, so if we look at the second set...where's the second set? [points at the task instructions]. If the number of sets is 2 [points at their algorithm, line 25] what does that give you?
27. Ben: 25 times 10 times...set... 25 times 10 times 2.
28. Researcher: Good, and what do you get for that?
29. Ata: 250 times 20 is 500.
30. Ben: No, it's [inputting numbers into the calculator]...oh yeah, my bad... It is 500 meters.
31. Researcher: That's good, 500. I saw you working out that there [points at *25* x *1 + 25* x *2 + ... + 25* x *8*]...What was that?
32. Ben: Oh...that was working out the total run meters for two sets [Ata nodding].
33. Researcher: Great...and what was the result of that?
34. Ata: I think we also got 500, right? We should get 500 [Ben is inputting numbers on the calculator].
35. Ben: It's 900! 900?
36. Ata: What? You probably put in more numbers! 900? But this [pointing at *25* x *10* x *2*] is 500! [checks the calculator and inputs numbers]
37. Ben: So that's [pointing at *25* x *10* x *2*] is wrong.

38. Ata: [Writes *900/25 = 36*]...So for two sets it's actually... the correct one is *25* x *36*. It doesn't make sense because look [points at Figure 1]. 25 times 4 is double 25 times 2; 25 times 6 is double 25 times 3; 25 times 8 is double 25 times 4; and 25 times 2 is double 25 times 1.

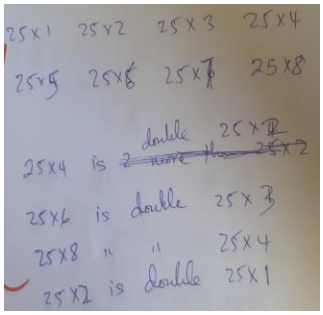

Figure 1.

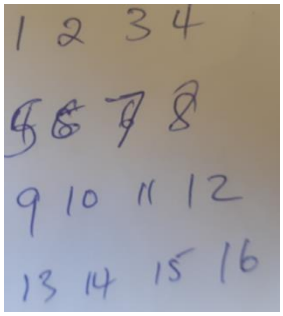

Figure 2.

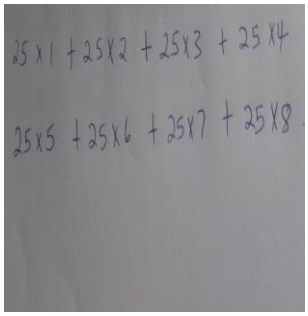

Figure 3.

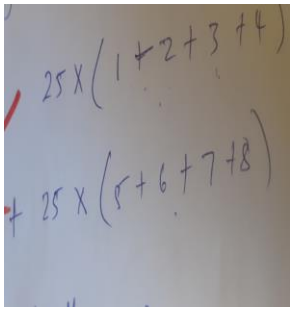

Figure 4.

After a brief period of silence, Ben writes down [see Figure 2] and discusses it with Ata. The researcher notices what Ben has written down and says:

39. Researcher: What are those numbers?
40. Ata: Number of shuttles...number of shuttles in a set.
41. Researcher: And would you use that to find the total distance?
42. Ben: We multiple [sic] them by 25...He points to [Figure 3]
43. Ata: We can write [points to Figure 3] it like [see Figure 4]
44. Ben: So first set is 25 times 10, second set is *25* x *26*.
45. Researcher: How about the third set?
46. Ata: [Writes *25* x (*9 + 10 + 11 + 12*) = *25* x *42*] 25 times 42.
47. Researcher: And...the fourth set?
48. Ben: You would just add 8... no 16, add 16... [Writes *25* x (*13 + 14 + 15 + 16*) = *25* x *58*] 58 times 25.
49. Ata: And then the fifth set would be 25 times 74...because you're just adding 16.
50. Researcher: Great, can you use that to come up with an algorithm?
    [Short period of silence ensues]
51. Ata: So for set one it's [writes *25* x *10*] and then it grows by 16... so for two sets it's [writes *25* x *10 + 25* x *16*].
52. Ben: So for any set it would be [writes *25* x *10 + 25* x *16* x *any number of sets*]...Is that it?
53. Ata: Yeah... that's it. I think we finally got it.

Table 2.
Procedures and algorithms evident in Episode 2[3]

Item name	**Procedure or Algorithm**	**Problem solved or intended to solve**
[Alg 1.2] *25* x *10* x *2*	Algorithm	Used to find distance for two sets
[Pro 1.2] *25* x *1 + 25* x *2 + 25* x *3 + 25* x *4 + 25* x *5 + 25* x *6 + 25* x *7 + 25* x *8*	Procedure	Used to find distance for two sets
[Alg 2.2] *25* x *36*	Algorithm	Intended to find distance for two sets
[Alg 2_1] *25* x *10*	Algorithm	Intended to find distance for first set

[3] Note some of these procedures and algorithms appeared in Episode 1, in which case the same lables are used.

[Alg 2_2] *25* x *26*	Algorithm	Intended to find distance for second set
[Alg 2_3] *25* x *42*	Algorithm	Intended to find distance for third set
[Alg 2_4] *25* x *58*	Algorithm	Intended to find distance for fourth set
[Alg 2_5] *25* x *74*	Algorithm	Intended to find distance for fifth set
[Alg 2.2b] *25* x *10* + *25* x *16*	Algorithm	Intended to find the distance for two sets
[Alg 2.G] *25* x *10* + *25* x *16* x *any number of sets*	Algorithm	Intended to find the distance for any number of sets

Analysis

As was evident in Episode 1, there was a discrepancy between the procedure that the students used to solve a particular problem, and the algorithm which they created, and believed to be equivalent to the procedure. Further still, at the end of the analysis for Episode 1, I alluded to how the students did not check whether *Alg 1.2* and *Pro 1.2* yielded the same result. In lines 26-37, the students came to realize that *Alg 1.2* (i.e., *25* x *10* x *2*) was incorrect, when the researcher prompted them to implement both *Alg 1.2* and *Pro 1.2*. In so doing, the students were confronted by the reality that their procedure and their algorithm were not necessarily equivalent (because their outputs were not the same).

In line 38, Ata proposed an algorithm (*Alg 2.2*) for finding the distance over two sets, by reverse engineering (exactly as they had created *Alg 1.1* at the start of Episode 1). The fact that *Alg 2.2* was not equal to *Alg 1.2* confused Ata as he reflected on *Pro 1.1* and *Pro 1.2*, and provided a justification that was partially correct (i.e., *Pro 1.2* contained "doubles" of some of the elements of *Pro 1.1*). In this way, one again notices the assumption of linearity that was evident in Episode 1. Additionally, one might say that the students selected (consciously or otherwise) a subset of available information to justify their conjecture and the validity of their algorithm (see line 38).

In lines 39-44, the students reflected on *Pro 1.1* and *Pro 1.2*, and established correct algorithms for finding the distance for the first (*Alg* 1_1) and second set (*Alg* 1_2). Subsequently, after establishing a correct algorithm for the third set (*Alg* 1_3), the group noticed a relationship between their three algorithms (i.e., "it increases by 16"). They then used this relationship to predict and validate correct algorithms for the fourth and fifth set (*Alg* 1_4 and *Alg* 1_5 respectively). However, despite creating correct algorithms, and noticing correct relationships among the algorithms, for finding the distance in each of the first five individual sets (i.e., first set, second set etc.) the students created incorrect algorithms for two sets (*Alg 2.2b*) and also (by extending *Alg 2.2b*) for any number of sets (*Alg 2.G*).

Though *Alg 2.G* was incorrect, one notices that it would yield the correct solution for invidual sets such as the first set, second set, third set and so forth (albeit with a minor tweak so that "any number of sets" "any number of sets number minus one"— *25* x *10* + *25* x *16* x (*any number of sets*-1)). Accordingly, one might say that the challenge that the students faced in this episode was combining the procedures for individual sets together, coupled with the realization that a number of sets comprises a collection of individual sets (e.g., two sets comprises the first and second set, 3 sets comprises first set, second set, third set).

Episode 3

The researcher points at the algorithm in line 52 (*Alg 2.G*), and says:

54. Researcher: Can you verify that?
55. Ben: What do you mean?
56. Researcher: Can you test that algorithm...your algorithm on...test it on two sets...the total distance for two sets?
57. Ata: [Writes out *25* x *1 + 25* x *2* + ... + *25* x *8*...while Ben is inputting it on the calculator].
58. Ben: We got 900 for that.
59. Researcher: OK, good. But you created an algorithm [points to Alg 2.G—*25* x *10 + 25* x *16* x *any set number*] over here. Can I ask you, what is that algorithm for?
60. Ata: That's like the method...the way for [sic] finding any....[Ben is nodding].
61. Ben: Yeah that's the easy way of finding the distance.
62. Ata: Like if we do a lot of sets...
63. Ben: Yeah, like 88 sets [laughs]. It will take us ages to find the distance...Maybe not ages, but too long, I can't even run for that long [everyone laughs].
64. Researcher: OK great. What I would like you to do, is to use the algorithm you just created [pointing at Alg 2.G] and try it on two sets.
65. Ata: Oh sure... so it's [points at *25* x *10 + 25* x *16* x *any set number*].
66. Ben: 25 times 10 plus 25 times 16 ... so 25 times 26. So that's 650.
67. Ata: [inputting numbers on the calculator] Yeah that's 650.
68. Researcher: So that's the total distance for how many sets?
69. Ben: Two sets... you said try it for two [pauses... they are both staring at what they wrote in line 57].
70. Ata: Why did we have 25 here [pointing at *25* x *26* from line 66]?
71. Ben: Because that's our algorithm, 25 times 10 plus 25 times 16.
72. Ata: But it's [pointing at *25* x *1 + 25* x *2* + ... + *25* x *8*]...that's *25* x *36*. So our algorithm should be *25* x *36*. [A long pause ensues]
73. Ben: So if it's 25 times 36, then it should be 25 times 10 plus 25 times 26 not 16...so maybe it's 25 times 10, plus 25 times 26 for two things... [he pauses and writes down... *25* x *10 + 16* x *set number + 10* x *25*].
74. Ata: Let's test it on two sets.
75. Ben: So...25 times...read it out and I'll write it.
76. Ata: 25 times 10 plus 25 times brackets ten plus 16 times the number of sets...close brackets [writes ... *25* x *10 + 25* x (*10 + 16* x *2*)]
77. Ben: [inputting numbers into calculator] That equals 1300. Noooooo!
78. Ata: I think that's wrong because that doesn't equal 25 times 36 which is 900.
79. Ben: I think that's a dead end.
    [Long pause...students stare at the problem and what they've done]
80. Ben: [Pointing at Figure 5 below] the sets go up in fours...So one, two, three, four... and then five, six, seven, eight...Then nine, ten, eleven, twelve. So if we have two sets ... it's one to eight. Four sets...one to...four times four...sixteen.

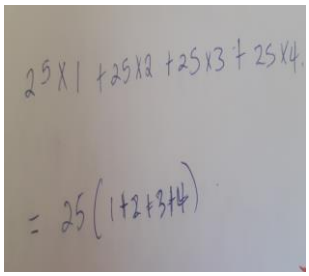

Figure 5.

81. Ata: And three sets is one to twelve... One plus two plus three and four is ten. One plus two, three all the way to eight is [inputting numbers on calculator] is thirty six. One plus two all the way to twelve is [inputting numbers on calculator] is seven[ty] eight.
82. Ben: And we're adding all of them...multiplied with 25. I don't know how to say it.
83. Ata: Yeah, that's what we did... add all of them and multiply by 25.
84. Ben: So we're adding all the numbers up to the number of things we are running...the set number, and then multiply by 5...I mean 25. [Ata is nodding]
85. Researcher: Can you try that on one set?
86. Ben: So you add one to four [Ata writes down *1 + 2 + 3 + 4*].
87. Researcher: I see that you're adding 1 to 4. But your algorithm was, add the numbers up to the number of sets...like the number of sets.
88. Ben: Oh yeah... it's not the set number it's... one four, two eight, three twelve... How do I say that?
89. Ata: It's the 4 timestable aye?
90. Ben: Yes yes yes...ummm.... set number plus...times 4....I think. What do you think?
91. Ata: Yeah that's it...because it's the number...one, two, three... multipled by four each time.
92. Ben: So our thing can be...add up all the numbers starting from one to the number of sets we're gonna run, times 4...and then multiply all of it with 25. That's a long way of explaining it!

Table 3.
Procedures and algorithms evident in Episode 3

Item name	**Procedure or Algorithm**	**Problem solved or intended to solve**
[Pro 1.2] *25* x *1* + *25* x *2* + *25* x *3* + *25* x *4* + *25* x *5* + *25* x *6* + *25* x *7* + *25* x *8*	Procedure	Used to find distance for two sets
[Alg 2.G] *25* x *10* + *25* x *16* x *any number of sets*	Algorithm	Intended to find distance for any number of sets
[Alg 2.2b] *25* x *10* + *25* x *16*	Algorithm	Used to find the distance for two sets
[Alg 2.2] *25* x *36*	Algorithm	Intended and used to find the distance for two sets
[Alg 2.2c] *25* x *10* + *25* x *26*	Algorithm	Intended to find the distance for two sets
[Alg 3.G] *25* x *10* + *25* x (*10* + *16* x number of sets)	Algorithm	Intended to find the distance for any number of sets
[Alg 3.2] *25* x *10* + *25* x (*10* + *16* x 2)	Algorithm	Used to find the distance for two sets

[Alg 3.Gb] Add up all the numbers starting from one to, the number of sets times four, and then multiplied by twenty five[4]	Algorithm	Intended to find the distance for any number of sets

Analysis

When the researcher prompted the students (lines 54-56) to test their algorithm (*Alg 2.G*) on two sets, the students immediately applied their procedure (*Pro 1.2*) rather than their algorithm (see lines 57). Of course, the researcher did not explicitly point to *Alg 2.G* at first request (line 56). This may suggest that the students perceived their algorithm as equivalent to their procedure (i.e., that it would produce the same result), albeit the algorithm is "an easy way of finding the distance...if we do a lot of sets" (lines 61-63). More explicit instruction from the researcher (line 64) prompted the students to implement *Alg 2.G* on two sets, eventually leading to the students recognizing that *Alg 2.G* was incorrect.

However, it is interesting to note that when the students set out to implement their algorithm (*Alg 2.G*) on two sets, they set the value of "*any number of sets*" to one rather than two, thus effectively implementing *Alg 2.2b—25* x *10 + 25* x *16* as opposed to *25* x *10 + 25* x *16* x *2*. Here, one notices a discrepancy between the algorithm that the students intended to use, and the algorithm that they actually used. However, despite incorrectly implementing their algorithm, the disparity in the outcomes of the procedure (*Pro 1.2*) and the implemented algorithm, seemed to motivate the students to reconsider *Alg 2.G* (lines 70-72).

Additionally, Ata realized a discrepancy between *Alg 2.2b* and *Pro 1.2* (line 72) which led him to suggest a new algorithm for finding the distance for two sets (*Alg 2.2*) by compacting the elements of *Pro 1.2* (i.e., *25* x *1 + 25* x *2 + ... + 25* x *8 = 25* x *36*). Ben then built on Alg 2.2, and proposed Alg 2.2c (25 x 10 + 25 x 26), presenting it in a similar manner to the previous incorrect algorithm of *25* x *10 + 25* x *16*). Then, Ben generalized *Alg 2.2c* to yield *Alg 3.G*. Examining *Alg 3.G* closely, one might hypothesize that Ben attempted to apply the same idea they had used in creating *Alg 2.G* (*25* x *10 + 25* x *16* x *any set number*)—the sum of the individuual sets increases by 16—in conjunction with the fact that for two sets the correct algorithm was *25* x *10 + 25* x (*10 + 16*). Subsequently, the students tested *Alg 3.G* on two sets (specifically implementing *Alg 3.2*) and noted that the outcome differed from that which they got when they implemented *Alg 2.2*, and they immediately decided that it was a "dead end."

Reflecting on the students' work, what might have been helpful for students was for them to compare the elements of the algorithm they were implementing (for a particular case) with the elements of the procedure so as to ascertain whether the two were in fact equivalent. For example, the students may have realized that *Alg 3.2* (*25* x *10 + 25* x (*10 + 16* x *2*)) was incorrect because it was equivalent to *25* x *10 + 25* x 42 which in turn was not equal to *Alg 2.2* which they had determined was correct (i.e., *25* x *10 + 25* x *36*). Perhaps the challenge here for the students was representing *Alg 2.2* in a way that made it clear which part applied to the first set, and which part applied to the second set. More generally, the challenge evident in this episode is one of coordinating the elements of an algorithm/procedure with those of problem at hand, so as to determine how a particular algorithm might change in relation to changes in the problem (e.g., moving from two sets to three sets).

[4] This algorithm was only expressed verbally.

Toward the end of the session the students made a breakthrough (line 80), building on the observation that one set comprises set one, two sets comprise set one and set two, three sets comprise set one, set two and set three, and so forth. Despite not being able to articulate a concise algorithm based on this observation, the observation was fundamentally different in their acknowledgement that *n* sets would require adding up all the sets from *1* to *4n* (lines 88-90). Furthermore, note, in both of the previous episodes the students immediately interpreted the observation that "the first set goes to four, then the second set goes to eight" to mean that the second set/algorithm was "double the first set/algorithm"; thus, incorrectly linearizing the growth of the sets. In this episode, however, an assumption of linearity is not evident. One plausible reason for this is because in line 81, Ata observes that the result of adding the second set (i.e., summing 1 to 8, which is 36) is not twice as much as the adding the first set (i.e., summing 1 to 4, which is 10). Moreover, one might argue that the representation in Figure 5 differed from previous representations (e.g., *25* x *1 + 25* x *2 + 25* x *3 + 25* x *4*) because the fact that the numbers *1* to *4n* were being added in each iteration was more readily noticeable in the former.

Summary and Discussion

The present study sought to shed more light on why some students can easily find the correct solutions to the problem(s) at hand, but the algorithms that these students create are not always ones that would, when implemented, produce the correct solution(s) for the same problems. More specifically, I investigated the foregoing phenomenon within the work of two Year 9 secondary school students working on *The shuttle relay task* (Tupua-Siliva & Tupouniua, 2020). In their work, the two students constantly solved specific instances correctly, but created algorithms that were incorrect, both for specific cases and for the general case.

For the remainder of the paper, I discuss the main findings that emerged from the analysis in relation to relevant literature, and make some suggestions for practice and future research:

Procedure versus algorithm: challenging assumptions of equivalence

Several times during their work, the students seemed to assume that the algorithm they had created was equivalent to their procedure. In some instances they assumed that their algorithm was exactly (line for line) what they had used to solve the problem. And, in other instances, they acknowledged that their algorithm was not equivalent (line for line) to their procedure, but they assumed that the former was a refined version of the latter that would produce the same result. In either case, the students did not verify whether or not their algorithm was correct. Accordingly, these assumptions of equivalence must be exposed. Instructors must support students move beyond merely assuming that their algorithm is correct, and to verify and validate whether or not it is correct.

Czocher et al., (2018) warns researchers and educators alike that supporting students' verification and validation processes requires more than merely asking whether or not the students' have validated or verified. Though Czocher et al.'s concerns were established within the context of modeling, they are pertinent to the context of algorithmatizing:

> Viewing validation and verification as dichotomous judgments that have a normatively correct answer ignores psychological and experiential aspects of the student as a rational actor. That is, because the dichotomous view emphasizes the end product of modelling over the model construction process, the locus of control for determining whether a model is adequate is

> external to the student. Shifting the view of verification and validation from a dichotomous judgment about a process or action to a richer description of students' ongoing activity means encouraging learners and teachers to attend to more than success in resolving the modelling problem. It encourages attending to activities and skills that support learning how to carry out validating activities (pp. 255-256).

Rephrasing Czocher et al.'s words above in relation to algorithamatizing, I propose that there is a need for both teachers and researchers to go beyond questions such as: Is the algorithm correct? Has the student validated his/her algorithm? Did the student revise his/her algorithm? These questions focus on the student's end-product and whether or not it has some desired quality. On the other hand, questions that should be asked are those such as: How has the student created his/her initial algorithm? How has the student validated his/her algorithm? What aspects of the student's algorithm has he/she validated?

One strategy that was useful during Ben and Ata's work was implementing both the procedure and the algorithm on specific cases, because it uncovered the fact that they do not (necessarily) yield the same result. Reflecting on the question of, "W*hy did the students not implement both their algorithm and procedure more frequently to check whether they yielded the same result?*", I hypothesize that the students' conceptions of algorithm may be an obstacle at times. In Episode 3, one gets a glimpse at what the students consider an algorithm to be: *an easy way of finding the distance, for many sets*. This may explain why several times during their work, when prompted to implement their algorithm on a specific case, the students implemented their procedure. As such, the students perhaps viewed the procedure as something to use for the small cases (e.g., one set, two sets) while an algorithm was reserved for the large cases (e.g., 88 sets, as Ben remarked in Episode 3). Consequently, students might not validate the algorithm because as it is reserved for these large(r) cases. Validating it would mean having to apply the procedure for these large cases (a task that would indeed require a lot of time and effort).

Issues arising from students' perceptions of an algorithm have been reported in past studies. For instance, Moala et al. (2019), observed how a group of students rejected a correct algorithm (i.e., they did not consider it to be an algorithm) because they believed an algorithm was something that should be able to identify the solution very quickly. Any strategy that did not meet this criterion, was perceived to be not an algorithm. Furthermore, other studies have shown that students' (mis)conceptions of what an algorithm is, can be more problematic when they are justified by beliefs, habits and the norms that support them. For instance, even when some students observe that an algorithm given to them by their teacher produces an error, they reject the error because they think that the algorithm has to be correct since it came from the teacher (e.g., see De Bock et al., 2002; Verschaffel et al., 2000; Wyndhamn & Saljo, 1997). Finding effective ways to shift students' perceptions of an algorithm towards more holistic definitions and considerations is a direction for future research.

Of course, as evident in the episodes, it is possible that students may believe that the algorithm is exactly what they used to solve the problem. Thus, the prompt to implement both, may not be effective because students merely implement one of them twice. In such cases, it may be useful for instructors to have students record their procedure as they are solving the problem, and to acknowledge it as how they actually solved the problem. Then, after they have proposed an algorithm, they can implement their algorithm and their procedure, and compare the outcome. In addition to comparing the outcomes of the procedure and the algorithm, another way to expose the nonequivalence between the procedure and the algorithm is for students to

compare the two (the parts thereof) in more detail, examining whether the two are performing the same actions/operations. This suggestion might complement comparing the outcomes, because as was evident a couple of times in Ben and Ata's work, having different outcomes may not be sufficient for getting students to acknowledge that their algorithm differs from their procedure.

Building on the procedure/algorithm of smaller cases

One of the primary struggles that the students faced in their work was extending a correct procedure/algorithm that they had established for a relatively small case to an algorithm for larger cases (e.g., De Bock et al., 2002; Lappan & Bouck, 1998; Mack, 1990). Past studies (e.g., Cai et al., 1998) noted that some students do not reflect on their procedures when they create their algorithm. Thus, these students do not notice and build on the recurring steps in their procedures. This was apparent multiple times in the episodes—rather than reflecting on the steps within their procedure, the students reverse engineered an algorithm, starting with the solution and then hypothesizing a way in which the solution could be found.

The two students in the present study did not always reflect on the procedures while creating their algorithms, however when they did they failed to convert those reflections to correct algorithms. For example, in Episode 1 the students reflected on their correct procedures for the first two cases (one set and two sets) and noticed a relationship between the two (i.e., the first set goes to four and the second set goes to eight), but then created an algorithm that was based on an assumption of linearity. Furthermore, the students validated this algorithm by selectively noticing how particular aspects of their procedure for one set were "doubled" in two sets. This selective noticing is reminiscent of what Moala et al., (2019) referred to as localized considerations, which explains how students sometimes consider it sufficient to substantiate the appropriateness of an algorithm by highlighting aspects that are correct, and ignoring (intentionally or otherwise) aspects that may not be correct. Despite the incorrectness of the algorithms that resulted from such reflections, it is clear that the issue is not whether students do or do not reflect on their procedures, but rather an issue of *how* the students reflect on their procedures.

Past studies (e.g., Marrongelle, 2007; Rasmussen et al., 2005; Tupouniua, 2015) illustrated how students can successfully create correct algorithms by reflecting on different aspects of their procedures. However, it is useful to note that the participants in these studies were all undergraduate students who were relatively familiar with the practice of algorithmatizing. Moreover, these students were familiar with the usefulness of reflecting and building on their procedures (for solving specific problem instances) when creating an algorithm. As such, these students seemed to carry out the act of reflecting and building on their procedures naturally without external motivation. Students with less experience in algorithmatizing, such as those in our study, may need more external support to conduct systematic reflections on procedures in order to notice, coordinate, and validate recurring steps in their procedures.

Explicit and systematic reflection on the procedure is of critical importance, because it increases the likelihood that students will be able to, "externalize the underlying algorithmic process...details and assumptions pertaining that might otherwise [remain] below the surface of awareness" (Tupouniua, 2015, p. 94). Reflecting on the procedures for solving small cases for the purpose of generating a recurring (and thus a general) set of steps, is similar to what

Mason (2001) refers to as specializing—*seeing the particular in the general and seeing the general through the particular*:

> Whenever I encounter a generality, I find myself testing it against particular cases. If I am trying to decide *whether* the general assertion is true, or *when* it is true, I consider special, often extreme cases. The purpose of trying out particular cases is not just to seek a counterexample, but to attend to *how* the calculations are done, with an eye to seeing if they generalise... Specialising is an act I can perform in order to make sense of what *always* happens in order to appreciate and reconstruct generality (Mason, 2001, p. 108).

Building on the foregoing ideas, it may be useful for students to record their procedures for solving multiple specific problem instances. Then, the instructor can invite the students to compare their procedures, exploring what is different among, and what is the same. Subsequently, students can be prompted to notice actions that were carried out in all of their procedures; and steps within the specific procedures that were done repeatedly (loops). Finally, the students can be invited to anticipate which of these recurring steps would occur (or not perform) in other variations of the problem and why. A direction for future research stemming from these considerations can be to explore how different frameworks for generalization in, say, algebra may be relevant to the algorithmatizing.

Aligning the algorithm and the problem

It is important to note that the above suggestion regarding explicit and systematic reflection on the procedures, should not be interpreted as a sign that the students in our study did not reflect on their procedures for smaller cases when they attempted to create a general algorithm. There were multiple times when the two students reflected on their procedures for smaller cases, but generated algorithms that were normatively incorrect. I reiterate, however, that the challenges pertaining to students' algorithmatizing activity goes beyond dichotomous considerations of whether or not the students did (or did not) perform a particular strategy (e.g., validate their algorithm, reflect on their procedure). As alluded to above, focusing on *how* students perform a particular strategy, and the corresponding problematics that arise, yields more valuable information for supporting students develop their algorithmatizing competencies.

As previously mentioned, one of the things that might have hindered the students' development of a correct algorithm when building on their procedure was an assumption of linearity. This assumption of linearity seems to align with what past studies have noted about how certain notions and algorithms are entrenched in students' intuitive knowledge (Fischbein, 1987) making them almost inaccessible for reflection, correct without a need for any further justification, and persistent in the face of conflicting evidence (see De Bock et al., 2002). As a result, some students consciously apply certain algorithms deliberately to situations wherein they are not applicable (e.g., applying algorithms for adding and subtracting natural numbers to fractions; or using linear algorithms for non-linear problems).

On the one hand, one might argue that the assumption of linearity is perhaps unsurprising given the fact that the students had little experience in creating a non-linear algorithm. On the other hand, given the students' previous experience with similar problems that are often given to Year 9-10 students in New Zealand (e.g., finding a formula for the sum of the first *n* counting numbers, finding a formula for the sum of the first n odd numbers)[5] one could argue that

[5] See Level 4 and 5 problems on https://nzmaths.co.nz/rich-learning-activities

generating an algorithm for the problem at hand was well within the students' grasp. In fact, the very last algorithm that the students generated—*add up all the numbers starting from one to the number of sets times four and then multiplied by twenty five*—may have (if given more time) led to the students' recalling a formula for summing the first *n* numbers.

All in all, the above suggests a need to support students with carefully coordinating the elements of an algorithm/procedure with those of the problem at hand, so as to determine how a particular algorithm might change in relation to changes in the problem. Even though in earlier stages of their work, the students said, "the first [set] goes to four" and "the second [set] goes to eight" in the earlier stages of their work, it was not until the latter stage of their work when the students articulated that the first set comprises the sum of one to four, the second set comprises the sum of one to eight, and so forth. Moreover, while they did not explicitly make reference to the fact that the sums were not growing at a constant rate, by adding up the numbers they had generated information that they did not have before (which clearly showed how the sets were not increasing steadily). One thing that seemed useful for the group, in Episode 3, in helping them coordinate the problem and the algorithm was their representation of the problem. I argued that the representation they used in Episode 3 (25 x (1 + 2 + 3 + 4)) made the fact that the numbers *1* to *4n* were being added in each iteration more readily noticeable as opposed to previous representations (e.g., *25* x *1* + *25* x *2* + *25* x *3* + *25* x *4*). More generally, some representations might be more effective in revealing the (non-linear) nature of the problem. Encouraging students to try different representations for the problem, the procedure, or the algorithm may be useful for them being able to coordinate how an algorithm changes in relation to the problem.

Final Remarks

Giving students more opportunities to create algorithms is necessary for enhancing their algorithmatizing competencies (Hart & Sandefur, 2018; Maurer & Ralston, 1991; Morrow & Kenney, 1998). However, merely giving students more opportunities to create their own algorithms does not necessarily mean that the students will actually engage in the process of creating algorithms. In other words, not all opportunities will provide the same level of student engagement. Much effort is required from both researchers and teachers to design rich tasks that encourage students to engage with the algorithmatizing approach. Furthermore, when students engage in these algorithmatizing tasks, educators need to pay attention to *how* the students create their algorithms, asking questions such as: *How are the students validating their algorithm? How are the students responding to prompts they are given by the teacher*? *What aspects of their algorithm do students revise according to certain prompts?* Answering (some of) these questions would likely enable instructors to give more informed and effective feedback to students while they are creating their algorithms. Attaining a richer view of students' algorithmatizing mechanisms is necessary in order to align teaching and assessment with the activities that support students in becoming competent algorithmatizers.

References

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education*. European Union Commission, Joint Research Centre.

Cai, J., Moyer, J. C., & Laughlin, C. (1998). Algorithms for solving non-routine mathematical problems. In L.J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics, 1998 yearbook* (pp. 218–229). NCTM.

Carroll, W. M. (2000). Invented computational procedures of students in a standards-based curriculum. *The Journal of Mathematical Behavior, 18*(*2*), 111–121.

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). Mahwah, NJ: Lawrence Erlbaum Associates.

Czocher, J., Stillman, G., & Brown, J. (2018). Verification and validation: What do we mean? In J. Hunter, P. Perger, & L. Darragh (Eds.), *Making waves, opening spaces: Proceedings of the 41st Annual Conference of the Mathematics Education Research Group of Australasia* (pp. 250–257). MERGA.

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.

Fischbein, E. (1987). *Intuition in science and mathematics.* Dordrecht: D. Reidel. Hart, E. W., & Sandefur, J. (Eds.). (2018). *Teaching and learning discrete mathematics worldwide: Curriculum and research*. Springer.

Hart, E. W., & Sandefur, J. (Eds.). (2018). *Teaching and learning discrete mathematics worldwide: Curriculum and research*. Cham: Springer.

Hipkins, C. (2017). New digital technologies for schools and kura. Press Release. https://www.beehive.govt.nz/release/new-digital-technologies-schools-and-kura.

Hoyles, C., & Noss, R. (2015, June 24). *Revisiting programming to enhance mathematics learning* [Paper presentation]. Math+coding symposium. Western University, London.

Lappan, G., & Bouck, M. K. (1998). Developing algorithms for adding and subtracting fractions. In L. J. Morrow, & M. J. Kenny (Eds.). *The teaching and learning of algorithms in school mathematics* (pp. 183–197). NCTM.

Mack, N. K. (1990). Learning fractions with understanding: Building on informal knowledge. *Journal for Research in Mathematics Education, 21,* 16-32.

Marrongelle, K. (2007). The function of graphs and gestures in algorithmatization. *The Journal of Mathematical Behavior, 26*(*3*), 211–229.

Mason, J. (2001). Aspects of generalisation and algebra: exploiting children's power. In L. Haggarty (Ed.), *Aspects of Teaching Secondary Mathematics: Perspectives on Practice* (pp. 105-120). London, UK: RoutledgeFalmer.

Mason, J. (2002). *Researching your own practice: The discipline of noticing*. RoutledgeFalmer.

Mason, J. (2006). What makes an example exemplary: Pedagogical and didactical issues in appreciating multiplicative structures. In R. Zazkis & S. R. Campbell (Eds.), *Number theory in mathematics education: Perspectives and prospects* (pp. 41–68). Lawrence Erlbaum Press.

Maurer, S. B., & Ralston, A. (1991). Algorithms: You cannot do discrete mathematics without them. In M. J. Kenney & C. R. Hirsch (Eds.), *Discrete mathematics across the curriculum, K-12. 1991 yearbook* (pp. 195–206). NCTM

Moala, J. G. (2019). Creating algorithms by accounting for features of the solution: the case of pursuing maximum happiness. *Mathematics Education Research Journal*, 1-22.

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.

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

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.

Schute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Thomas, M. O. J. (2014). Algorithms. In S. Lerman (Ed.). *Encyclopedia of mathematics education* (pp. 36–38). Netherlands: Springer.

Threlfall, J. (2002). Flexible mental calculation. *Educational Studies in Mathematics*, *50*, 29- 47.

Tupouniua, J.G. (2015). *On being stuck: a preliminary investigation into the essence of mathematical problem solving*. Master's Thesis, University of Auckland.

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

Tupouniua, J. G. (2020). Explicating how students revise their algorithms in response to counterexamples: building on small nuanced gains. *International Journal of Mathematical Education in Science and Technology*, 1-22.

Tupua-Siliva & Tupouniua, J.G (2020). *The shuttle relay task*. Unpublished manuscript.

Verschaffel, L., Greer, B., & De Corte, E. (2000). *Making sense of word problems.* The Netherlands: Lisse.

Wyndhamn, J., & Saljo, R. (1997). Word problems and mathematical reasoning: A study of children's mastery of reference and meaning in textual realities. *Learning and Instruction, 7*(*4*), 361–382.

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.

Zazkis, R., Liljedahl, P., & Chernoff, E. J. (2008). The role of examples in forming and refuting generalizations. *ZDM*, *40*(1), 131–141. https://doi.org/10.1007/s11858-007-0065-9.

Author

John Griffith Tupouniua (e-mail: J.G.Tupouniua@massey.ac.nz), Institute of Education, Massey University, Albany, Auckland, New Zealand.