Open and Interactive Learning Resources for Algorithmic Problem Solving

João F. Ferreira[1] and Alexandra Mendes[2,3]

[1] INESC-ID & Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal
`joao@joaoff.com`
[2] Department of Informatics, Universidade da Beira Interior, Covilhã, Portugal
`asf.mendes@gmail.com`
[3] HASLab, INESC TEC, Porto, Portugal

Abstract. *Algorithmic problem solving* is a way of approaching and solving problems by using the advances that have been made in the principles of correct-by-construction algorithm design. The approach has been taught at first-year undergraduate level since September 2003 and, since then, a substantial amount of learning materials have been developed. However, the existing materials are distributed in a conventional and static way (e.g. as a textbook and as several documents in PDF format available online), not leveraging the capabilities provided by modern collaborative and open-source platforms.
In this paper, we propose the creation of an online, *open-source* repository of *interactive* learning materials on algorithmic problem solving. We show how the existing framework Mathigon can be used to support such a repository. By being open and hosted on a platform such as GitHub, the repository enables collaboration and anyone can create and submit new material. Furthermore, by making the material interactive, we hope to encourage engagement with and a better understanding of the materials.

Keywords: Algorithmic Problem Solving · Formal Methods · Interactive Learning Materials

1 Introduction

Algorithmic problem solving is about the formulation and solution of problems where the solution involves, possibly implicitly, the principles and techniques that have been developed to assist in the construction of correct algorithms. Algorithms have been studied and developed since the beginning of civilisation, but, over the last few decades, the unprecedented scale of programming problems and the consequent demands on the reliability of computer software led to massive improvements in our algorithmic-problem-solving skills. The improvements are centred on goal-directed, calculational construction of algorithms as opposed to the traditional guess-and-verify methodology.

Algorithmic problem solving has been taught at first-year undergraduate level since September 2003 and, since then, its adoption became easier due to the publication of a textbook [1] and to the development of a substantial amount of learning materials (in particular, a collection of teaching scenarios is available for educators to use [8]). Recreational problems are often used to teach APS concepts, since they can make serious concepts more palatable to students and encourage interactivity in the classroom [10–12]. However, the materials available to support the teaching of APS are distributed in a conventional and static way (e.g. as several documents in PDF format available online), not leveraging the capabilities provided by modern collaborative and open-source platforms. In this paper, we present the first steps towards changing this current state of affairs.

We propose the creation of an online, *open-source* repository of *interactive* learning materials on algorithmic problem solving. We show how the existing framework Mathigon[4] can be used to support such a repository. By being open and hosted on a platform such as GitHub, the repository enables collaboration and anyone can create and submit new material allowing the material to evolve over time as a collaborative effort by the community. Furthermore, by making the material interactive, we hope to reach a wider audience and encourage engagement with and a better understanding of APS.

Outline. To illustrate the type of recreational problems that we use when teaching APS, we show in Section 2 an example of a river-crossing problem, which we use as a running example. We use it to introduce concepts such as state and state-transition diagram, and to illustrate principles such as the importance of avoiding unnecessary naming. In Section 3 we present an example of how APS material can be made more interactive. We conclude in Section 5, where we also present some ideas for the next steps.

2 Example: River-Crossing Puzzles

River crossing puzzles are about carrying items from one river bank to another, usually in the fewest trips possible and typically with restrictions that make a solution non-obvious. According to Wikipedia[5], the earliest known river-crossing problems occur in the manuscript *Propositiones ad Acuendos Juvenes* (English: Problems to sharpen the young), traditionally said to be written by Alcuin. The earliest copies of this manuscript date from the 9th century.

We use river-crossing problems to illustrate and introduce concepts and principles such as state-transition diagrams, symmetry, and the importance of avoiding unnecessary naming. Consider, for example, the following puzzle:

Goat, Cabbage and Wolf

A farmer wishes to ferry a goat, a cabbage and a wolf across a river. However, his boat is only large enough to take one of them at a time, making several trips across the river necessary. Also, the goat should

[4] Mathigon's website: `https://mathigon.org` (accessed 18 July 2019)

[5] Wikipedia link: `https://en.wikipedia.org/wiki/River_crossing_puzzle` (accessed 18 July 2019)

not be left alone with the cabbage (otherwise, the goat would eat the cabbage), and the wolf should not be left alone with the goat (otherwise, the wolf would eat the goat).

How can the farmer achieve the task?

We typically structure the discussion of this puzzle as follows.

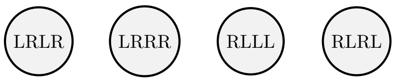
On algorithmic problems. The first discussion we have with the students is divided into two parts. First, we ensure that the problem statement is clear and that students understand the goal of the puzzle. For example, implicit in the problem statement is that initially all the four elements are at the same river bank and that the farmer has to accompany each of the other elements when crossing the river. Also, we adopt what E. W. Dijkstra called the rule of *no cancellation*, i.e. we reject any schedule in which a subsequence of successive moves results in no change [6].

Second, we discuss why this puzzle can be considered an algorithmic puzzle. We reach the conclusion that this is clearly an algorithmic problem, because the solution consists of a *sequence of instructions* (i.e. an algorithm) indicating who or what should cross. A typical instruction would be: “the farmer crosses with the wolf” or “the farmer returns alone”.

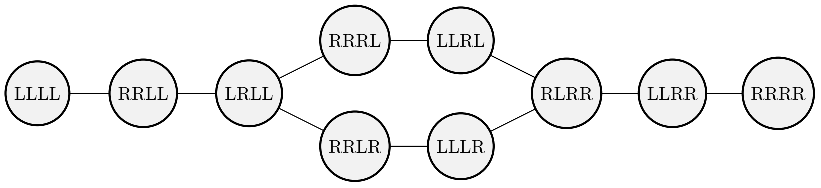
On states and state-transition diagrams. We proceed our discussion by noting that to solve this problem, it is useful to introduce a notation that identifies the position of each element. We name the two river banks as left and right and we introduce a simple notation that denotes whether each element is on the left or on the right bank. For example, we write LLRR to denote that the farmer is on the left bank, the goat is on the left bank, the cabbage is on the right bank, and the wolf is on the right bank (exactly by that order). We thus introduce the notion of *state*, representing the initial and final states of this problem as:

This discussion leads to a very natural question: *how many states exist in this problem?* Given that there are 4 elements, and each element can be in either one of two river banks, we quickly conclude that there is a total of 2^4 (i.e. 16) states. We also take the opportunity to explore how many states we would have if the number of elements were different. After only a few examples, students see that the number of states grows very quickly.

We also observe that not all of the 16 states are valid. For example, the following states are invalid:



At this point, most students already have a solution to the puzzle. After enquiring some of them about the methodology that they followed to find their solutions, we notice that they have tried multiple steps until reaching a satisfactory solution. We use this to introduce the notion of *brute-force search*. We describe the technique and introduce the concept of *state-transition diagram*. Together with the class, we build the following state-transition diagram and we observe that there are indeed two solutions to this problem:



We also take the opportunity to observe that river-crossing problems have a property that reduces the amount of effort required to solve them: they are symmetric. In other words, suppose that we have a solution that takes the elements from the left bank to the right bank; if we reverse that solution, we immediately have a solution that takes the elements from the right bank to the left bank. The state-transition diagram is an excellent device to observe this property.

On unnecessary naming. It is impossible to solve problems without introducing names. However, if we name unnecessary elements or if we make unnecessary distinctions, we add unnecessary detail and complexity to the solution.

This puzzle is a good example to illustrate that the avoidance of unnecessary naming leads to more effective and simple solutions. Recall that the problem is about taking across four individuals without violating the two conditions:

1. the goat should not be left alone with the cabbage;
2. the wolf should not be left alone with the goat.

These two conditions expose a similarity between the wolf and the cabbage: the goat cannot be left with either the wolf or the cabbage. Moreover, there are no restrictions on leaving the wolf alone with the cabbage. This clearly suggests that *both the cabbage and the wolf are playing the same role*. Why, then, are the "wolf" and the "cabbage" distinguished by giving them different names?

We restate the problem[6], this time with a naming convention that omits the unnecessary distinction between the wolf and the cabbage. In the restated problem, we call the goat an "alpha" and the cabbage and the wolf "betas":

> *A farmer wishes to ferry an alpha and two betas across a river. However, his boat is only large enough to take one of them at a time, making several trips across the river necessary. Also, an alpha should not be left alone with a beta.*
>
> *How can the farmer achieve the task?*

[6] The restatement of the problem and the subsequent two paragraphs are extracted from [1]

Now the problem becomes much easier to solve and most students find a solution immediately. Indeed, there is only one solution: take the alpha across, and then one beta across, returning with the alpha; then take the second beta across, followed by the alpha. Because there is only one solution, it is easy to discover (note that in the problem with the four individuals, we have two solutions, since we have two different choices when choosing the first beta to take across).

When elements of a problem are given individual names, it distinguishes them from other elements of the problem, and adds to the size of the state space. The process of omitting unnecessary detail, and reducing a problem to its essentials is called *abstraction*. Poor solutions to problems are ones that fail to "abstract" adequately, making the problem more complicated than it really is.

3 Interactive Learning Materials

The material presented in the previous section is taught during lectures in an interactive manner, with students being encouraged to participate actively in solving the problem individually and as a whole-class effort. For deep learning to be achieved, students need to think about what they are learning and engage with the material, rather than sit and listen — we believe that the nature of our lectures helps to achieve that. As Tyler (1949) points out (cited by Biggs and Tang [4]),

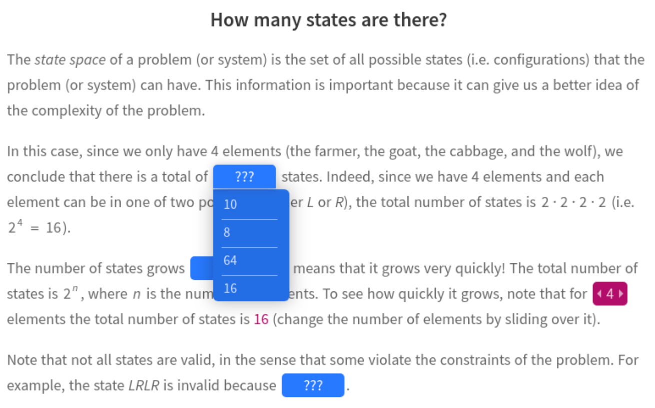
> *Learning takes place through the active behavior of the student: it is what he does that he learns, not what the teacher does.*

However, when it comes to revising the material and to further study, the interactivity disappears and students are left with only books and slides to assist them, which are static. We thus propose the creation of online, open-source, interactive material that gives students a further opportunity to take on a more active role in learning APS, even when they are alone. By making it widely available via a website, APS can potentially reach a larger audience than it would if it continued limited to lectures and static supporting material (even if available online); by being open-source, anyone can contribute with new material and improvements.

Initial prototype. To implement and experiment with some of our initial ideas, we wrote parts of the material presented in Section 2 for Mathigon, a groundbreaking new education open-source platform that enables interactive learning.

Mathigon makes it possible to present the material in a way that encourages the learner to become a participant by e.g. answering multiple choice questions, by solving interactive puzzles, and by exploring further reading (e.g. biographies and further context) presented to them via non-intrusive popups. As the learner navigates through the content and interacts with it, further information is provided to complement their learning.

For example, Figure 1 shows how multiple choice answers can be integrated in the material. The sentence that starts with *"Indeed, since we have 4 elements"* is only displayed after the student selects the correct answer. Moreover, the box with arrows is a *variable slider*, which allows inline variables to be manipulated by the student. In this example, the student is able to interactively see how the total number of states grows (e.g. if the value of the variable increases from 4 to 5, the number 16 is automatically updated to 32).



How many states are there?

The *state space* of a problem (or system) is the set of all possible states (i.e. configurations) that the problem (or system) can have. This information is important because it can give us a better idea of the complexity of the problem.

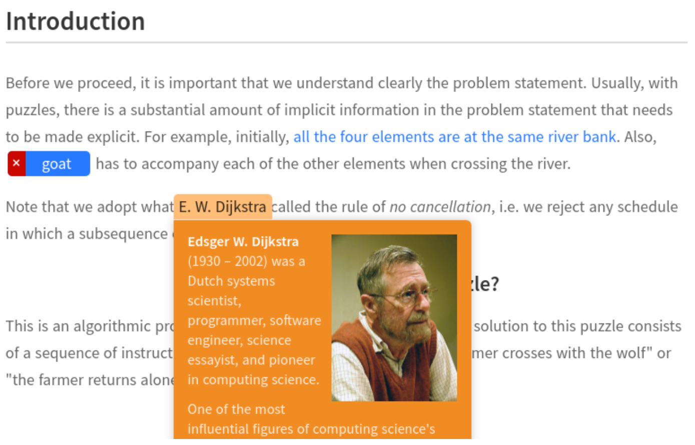
In this case, since we only have 4 elements (the farmer, the goat, the cabbage, and the wolf), we conclude that there is a total of ??? states. Indeed, since we have 4 elements and each element can be in one of two po er *L* or *R*), the total number of states is 2 · 2 · 2 · 2 (i.e. $2^4 = 16$).

The number of states grows means that it grows very quickly! The total number of states is 2^n, where *n* is the num ents. To see how quickly it grows, note that for 4 elements the total number of states is 16 (change the number of elements by sliding over it).

Note that not all states are valid, in the sense that some violate the constraints of the problem. For example, the state *LRLR* is invalid because ??? .

Fig. 1: Multiple choice and variable slider.

Figure 2 illustrates a correct choice (*"all the four elements are at the same river bank"*) and an incorrect choice, clearly marked with a cross (*"goat"*). It also shows an example of a biographical popup.



Introduction

Before we proceed, it is important that we understand clearly the problem statement. Usually, with puzzles, there is a substantial amount of implicit information in the problem statement that needs to be made explicit. For example, initially, all the four elements are at the same river bank. Also, × goat has to accompany each of the other elements when crossing the river.

Note that we adopt what E. W. Dijkstra called the rule of *no cancellation*, i.e. we reject any schedule in which a subsequence

Edsger W. Dijkstra (1930 – 2002) was a Dutch systems scientist, programmer, software engineer, science essayist, and pioneer in computing science.

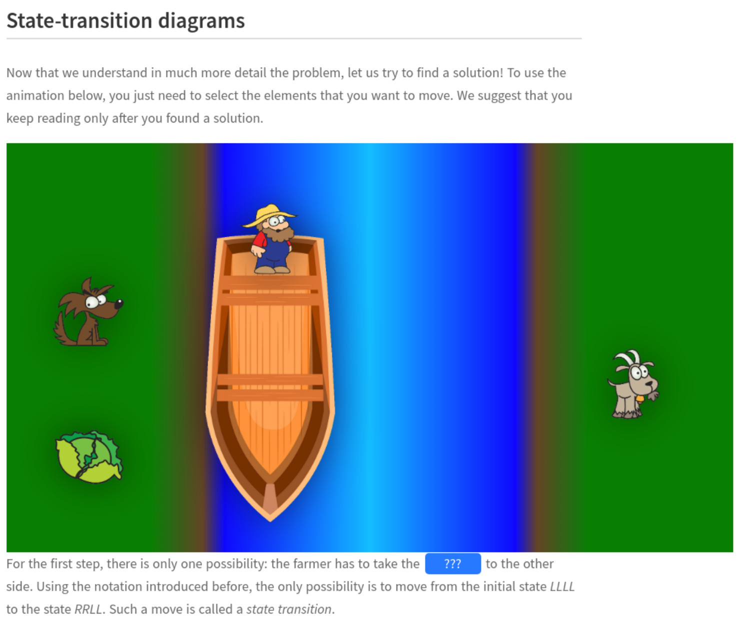
One of the most influential figures of computing science's

zle?

This is an algorithmic pr solution to this puzzle consists of a sequence of instruct mer crosses with the wolf" or "the farmer returns alon

Fig. 2: Incorrect answers are clearly marked. Biographies are shown as popups.

Given that the material is offered as a web application, we can easily incorporate interactive artefacts that are programmed in, for example, Javascript. Figure 3 shows the integration of an external Javascript implementation of the puzzle[7]. Our work-in-progress is available on GitHub[8]; everyone is welcome to contribute.



State-transition diagrams

Now that we understand in much more detail the problem, let us try to find a solution! To use the animation below, you just need to select the elements that you want to move. We suggest that you keep reading only after you found a solution.

For the first step, there is only one possibility: the farmer has to take the ??? to the other side. Using the notation introduced before, the only possibility is to move from the initial state *LLLL* to the state *RRLL*. Such a move is called a *state transition*.

Fig. 3: External artefacts can be incorporated (e.g. Javascript code)

4 Effectiveness and Students' Feedback

This paper describes a new open community project that is at a very early stage of development and that has not yet been thoroughly evaluated. The problems, puzzles, techniques, and approach have been evaluated to some extent, but the online Mathigon-based interactive material has not.

We argue that the approach is effective given that the techniques and methods taught have been used to find new solutions to non-trivial problems (e.g. new results in number theory [2,3,7] and in solitaire games [1]). However, measuring

[7] The implementation of the puzzle that we used was created by Victor Ribeiro (https://github.com/victorqribeiro/bridge)

[8] See the repository textbooks (folder content/river-crossing) in https://github.com/algprobsolving

whether the approach helps students become better problem solvers is more difficult and requires further work. So far, we performed a small-scale experiment to try to determine if a cohort of students became better problem solvers [9]. The focus was the calculational method and, generally, they adopted what was taught in the module for solving problems, but their use of the calculational style was not effective. We also tested some APS material with pre-university students [12]: after being exposed to the material, students were able to apply techniques like invariants by themselves.

Our experience is that students appreciate the approach. This is confirmed by feedback about a session that we delivered on analysing an algorithmic card trick [10]. This is further confirmed by feedback about a session delivered to pre-university students [12], which also shows that the material (or, at least, parts of the material) can even be taught at pre-university level. During some teaching sessions, we have also collected feedback from our students. In general, they appreciate the use of recreational problems and the interactivity that arises from solving those problems.

Thousands of students have been exposed to the APS material throughout the years. The material was taught for about 10 years at the University of Nottingham (it started in 2003). It was taught at Teesside University from 2011 to 2018. Backhouse's book on APS [1] is used in the course CS2104 at Virginia Tech. We believe that creating an interactive APS book will help to increase the adoption of this approach to teach APS.

5 Conclusion

The famous quote *"Learning is not a spectator sport"* [5] could not be more true for a subject like APS. We believe that the idea proposed in this paper supports this quote, by encouraging learners to become *participants*. Moreover, it is our view that Mathigon offers a sound and extensible base for the creation of interactive APS material that supports active learning. With this initiative, and by making all the artefacts open-source and available on Github, we hope to encourage the community to make a joint effort to create and use these materials.

Future work. As an immediate next step, we intend to create further interactive material to support APS. We will also explore how to create interactive material to support teaching programming methodology following an approach such as that proposed in [13]. To facilitate collaboration, we will also create documentation on how to contribute to the project. In order to incorporate in the web application all the APS material that we have been teaching in the last few years, we envisage that Mathigon will have to be extended with a few technical features. For example:

– To support the formats used in the calculational method (e.g. proof format), we might have to write extensions to AsciiMath (Mathigon is currently able to parse AsciiMath and convert it to MathML). This is important, because the calculational method is central to our approach [9, 11].

- To enable support for handwritten calculational mathematics and build on previous work [14, 15], we will explore frameworks such as MyScript[9].
- To encode some of our case studies (e.g. [10]), we might have to write specialised Javascript libraries that allow richer interactions.

Acknowledgments: This work is partially financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia through the project: UID/EEA/50014/2019.

References

1. Backhouse, R.: Algorithmic problem solving. John Wiley & Sons (2011)
2. Backhouse, R., Ferreira, J.F.: On Euclid's algorithm and elementary number theory. Science of Computer Programming **76**(3), 160 – 180 (2011). https://doi.org/https://doi.org/10.1016/j.scico.2010.05.006
3. Backhouse, R., Ferreira, J.F.: Recounting the rationals: twice! In: International Conference on Mathematics of Program Construction. pp. 79–91. Springer (2008)
4. Biggs, J., Tang, C.: Teaching for Quality Learning at University: What the Student does (Society for Research Into Higher Education). 4th ed. Open Univ. Press (2011)
5. Chickering, A.W., Gamson, Z.F.: Seven principles for good practice in undergraduate education. AAHE bulletin **3**, 7 (1987)
6. Dijkstra, E.W.: Pruning the search tree (January 1997), http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1255.PDF
7. Ferreira, J.F.: Designing an algorithmic proof of the two-squares theorem. In: International Conference on Mathematics of Program Construction. pp. 140–156. Springer (2010)
8. Ferreira, J.F.: Principles and Applications of Algorithmic Problem Solving. Ph.D. thesis, School of Computer Science, University of Nottingham (2010)
9. Ferreira, J.F., Mendes, A.: Students' feedback on teaching mathematics through the calculational method. In: 2009 39th IEEE Frontiers in Education Conference. pp. 1–6. IEEE (2009)
10. Ferreira, J.F., Mendes, A.: The magic of algorithm design and analysis: teaching algorithmic skills using magic card tricks. In: ACM ITiCSE (2014)
11. Ferreira, J.F., Mendes, A., Backhouse, R., Barbosa, L.S.: Which mathematics for the information society? In: Teaching Formal Methods, LNCS, vol. 5846. Springer (2009)
12. Ferreira, J.F., Mendes, A., Cunha, A., Baquero, C., Silva, P., Barbosa, L., Oliveira, J.: Logic training through algorithmic problem solving. In: Tools for Teaching Logic, LNCS, vol. 6680, pp. 62–69. Springer (2011), http://dx.doi.org/10.1007/978-3-642-21350-2_8
13. Hoare, T., Mendes, A., Ferreira, J.F.: Logic, algebra, and geometry at the foundation of computer science. In: Formal Methods Teaching Workshop. pp. 3–20. Springer (2019)
14. Mendes, A.: Structured Editing of Handwritten Mathematics. Ph.D. thesis, School of Computer Science, University of Nottingham, UK (2012)
15. Mendes, A., Backhouse, R., Ferreira, J.F.: Structure editing of handwritten mathematics: Improving the computer support for the calculational method. In: ACM ITS (2014), http://doi.acm.org/10.1145/2669485.2669495

[9] MyScript webpage: https://www.myscript.com