Teaching Graph Algorithms To Children Of All Ages

J. Paul Gibson
Département LOR (SAMOVAR UMR 5157)
Telecom Sud Paris, France
paul.gibson@it-sudparis.eu

ABSTRACT

We report on our experiences in teaching graph theory and algorithms to school children, aged 5 to 17. Our objectives were to demonstrate that children can discover quite complex mathematical concepts, and are able to work with abstractions and use computation reasoning from quite an early age. We provide details of our incremental approach, which can be used with students of a wide range of abilities. Also, we comment on the importance of problem based learning where the algorithms are presented as possible solutions to games or puzzles. Finally, we conclude with a number of important observations with regard to the introduction of computer science into schools.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education; G.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms

Human Factors, Experimentation

Keywords

Computational thinking, K-12 education, Problem based learning, Modelling, Abstractions

1. INTRODUCTION

There has been much recent debate concerning the teaching of computer science in schools. Within the computer science community (acdemia and industry) there is general agreement that computer science is not about how to use a computer through the applications that it can execute, but it is about knowing how these applications work. For example, in the UK the *NextGen 2011 report on CS education* criticises "a school curriculum that focuses in ICT on office skills rather than the more rigorous computer science and programming skills which high-tech industries [...] need". Similar stories and reports have appeared in the media around the world, with politicians and policy-makers finally listening to the message. (For example, Michael Bloomberg, the mayor of New York, has publically endorsed a course on learning to program that he is taking, and which is targetted at school children.) We welcome these recent events, even if we strongly believe that computer science in schools needs to be broader than just learning to program[9]; it needs to be about computational thinking[23], algorithmic thinking[8] and algorithmic learning[12]. Programming in schools is important, but teaching about abstraction may be even more so[18].

This paper supports the move towards this new type of computer science education. However, we note that in making such an educational change, one should not forget about the teachers who are expected to teach this "new" material. It is critical that teachers understand computer science if they are to teach it well. Such an observation has already been made by Ball[1] with respect to mathematics education, where she notes (page 36):

> "Making the judgments about which student suggestions to pursue, developing the tasks that encourage certain kinds of exploration, and conducting fruitful class discussions — all these tasks depend heavily on the teacher's subject matter knowledge"

In the same report (page 37), a teacher acknowledges the importance of learning while teaching:

> "When I decided to be a teacher, I knew there were a lot of things I had to learn about teaching, but I felt I knew everything there was to teach my students. [...] I found I was as much a learner of subject matter as I was a learner of the art of teaching. My education in the future will not be limited to 'how to teach', but also what it is I'm teaching."

We propose that the best way to prepare teachers of computer science is for them to actually teach it. Thus, experimental computer science sessions should be introduced into the classroom in order to teach the teachers, as well as to introduce computer science to the students, and to help develop or improve the computer science curriculum. This would address many of the issues raised by Guzdial[13] when he states:

> "We need more education research that is informed by understanding CS — how it's taught,

what the current practices are, and what's important to keep as we change practice."

Our research was motivated by the Bruner's famous educational challenge from 1960[4](page 33):

> "We begin with the hypothesis that any subject can be taught effectively in some intellectually honest form to any child at any stage of development. It is a bold hypothesis and an essential one in thinking about the nature of a curriculum"

We wished to validate this hypothesis within the domain of computer science, and chose the subject of graph algorithms as our test case. This subject is foundational to all computer science and it provides an excellent example of the importance of mathematics[15] and the mathematical nature of algorithms and programming[20].

The structure of the remainder of the paper is as follows. In section 2 we provide a brief review of recent related work. Section 3 provides details concerning some of the sessions that we have run in schools. During the sessions, a number of important observations were made: section 4 reports on these. In section 5 we conclude the paper and make suggestions for future work.

2. RELATED WORK

Previous work that has most unfluenced our research falls under the following themes —
1. Teaching computer science without using a computer: The CS Unplugged programme as first introduced by Bell[2] inspired and motivated our teaching in the classroom, where the computer is no longer a distraction to learning about computing.
2. Preparing school teachers to teach computer science: Hazzan et al.[14] propose a model for high school computer science education, where our paper makes a contribution to 3 of the 4 key elements that they identify: teacher preparation, pedagogic research and curriculum development.
3. Teaching computer science to young children: Work with very young children has been reported by Gibson, where children of all ages (starting from 5) have been introduced to formal methods through reasoning about algorithms and proofs[11]. Many different programming languages have been developed specifically for teaching children. Feurzig et al., as early as 1969, proposed using programming to teach about mathematics[7] and demonstrate how the programming language LOGO can be used by young children. More recently researchers have reported success in schools with languages oriented towards children (and beginners) such as Alice[17]. Further, evidence suggests that young children are also able to program using standard (real world) languages such as Java[10]. Teaching young children about computers and computer architecture has been the subject of research by Bianco and Tinzanni[3], where they use interactive stories to introduce the main concepts. This approach could be complementary to our use of stories when describing the problems to be solved.
4. Teaching (graph) algorithms: Researchers have also suggested teaching algorithms during mathematics classes in schools. A typical example is from daRosa[6], where sorting and searching algorithms are used to teach about algorithm analysis. There has been much research into innovative ways of teaching graph algorithms, with most results focussing on animation and simulation[5]. However, such tools are aimed at University level students rather than for use in schools.
5. Teaching using puzzles and games: Levitin and Papalaskari propose using games and puzzles in the teaching of algorithms[19] and this approach has been successsful in other computing disciplines[21], including topics such as operating systems[16]. More formal resoning about algorithms has also been linked to puzzles and games[12].

It is beyond the scope of this paper to review all relevant publications in the teaching of computing at schools. We choose to conclude this section by referencing a paper, by Pollard and Duvall[22], that considers the knowledge transfer in the other direction: namely from schools to university. It is worthy of note that they comment on the how teaching CS at University could be improved and informed by experiences with teaching at kindergarten.

3. THE INCREMENTAL PBL APPROACH

Our teaching approach is founded on a sequence of sessions where we (aided by the teachers) work through each session at a speed that is dictated by the age and ability of the class. No session in the sequence should be skipped, but all sessions are open-ended so that more advanced students can advance at their own pace. Due to lack of space, we report only on 4 sessions that we run (each of these sessions can take between 30 minutes and 4 hours, depending on the students). Each session is divided into classes whose duration is also dictated by the level of the students. The youngest students may have 30 minute classes whilst older students may manage 90 minute classes. Our approach is to be as flexible as possible, and to follow the advice of the teacher (who is always present). All schools involved were non-fee paying and could be considered typical, with pupils' abilities following an unexceptional normal distribution.

At this experimental stage, we have no fixed learning objectives: the goal is to observe the students and to assess the impact of the classes on their general school behaviour and performance. We believe that improving their computational thinking — with emphasis on abstraction — will have a positive impact on all other classes that they study. We return to this claim in section 4.

3.1 The first session: graph concepts

This session was one of the first that we tried when we first started visiting schools more than ten years ago. It has been run twelve times in five different schools (in Ireland and France) and with five different age groups — the youngest group aged 5-6, the oldest group aged 14-15. The class sizes varied from 12 to 32 children. In the first few years in which we ran the sessions, we focused on the oldest pupils at primary school (aged 10-11). Then we extended the session with more difficult puzzles in order to use it with the oldest group of children (from secondary school) who were participating in a summer science camp. Finally, in the last year, we have adapted the session for very young children and have used it in two different classes.

Working with the youngest school children, one cannot assume that they are able to read or write. Consequently, with our first lessons we are obliged to use less abstract forms of 'language' for describing graphs. Rather than inventing such languages, we try to adapt the skills of the children to the creation of their own syntax and semantics.

Experience has shown that the youngest children are particularly motivated by drawing. Thus, we introduce a drawing game, whereby we restrict them to using only circles and lines. The game is to construct drawings — using paper and pencils — that have to follow rules that are specified using spoken natural language. For example, we start by asking them to draw "two circles connected by a single line". Typically, we see a variety of drawings, such as seen in figure 1. (Note that we have coloured the drawings to clarify the illustration, but that — at this stage in the session — the children do not yet use different colours in their drawings.)

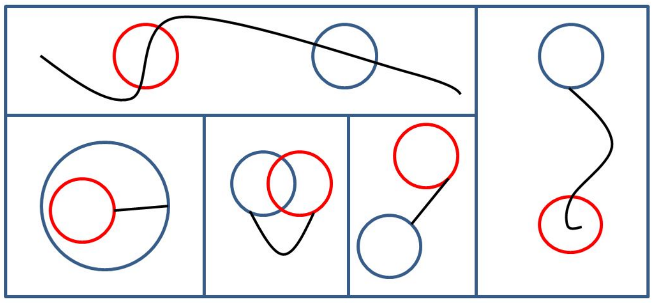
Figure 1: Two circles connected by a single line

The children than get to vote on which drawings follow the rule, and comment on whether the rule was easy to understand. At this stage, they are unsure about which drawings are right and which are wrong — so we tell them that all (within reason) are right but some are better than others. They will get to judge on which are best as they are asked to carry out more drawings.

After a few more examples we introduce colours for the circles. A typical puzzle will be for them to draw "a red circle connected to a blue circle and a green circle connected to a red circle". The two most interesting variations of drawings proposed for this problem are shown in figure 2)

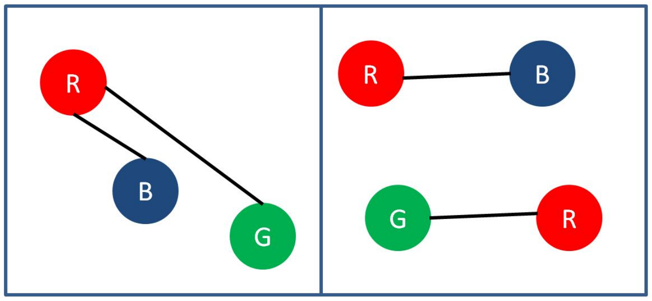

Figure 2: Red-Blue and Green-Red

At this stage, the students openly discuss about how to decide whether 2 drawings are the same, posing questions such as:

Does it matter if the lines are straight or curved?
Can the circles touch?
Can one circle be inside another?
Where can the lines start/stop?
Does it matter if the circles are different sizes?
Can the lines cross?
Do the circles have to be aligned vertically or horizontally?
Does the distance between the circles matter?

It is important to let the students decide on the answers to these questions, and very important that they understand that they can change their answers as the session progresses.

The two graphs in figure 2 usually lead to clear disagreement in the class, with 3 different opinions coming to the fore:

- The 2 drawings (we start to call them graphs in the next session) are correct and so they can be considered to be the same (or equal or equally good)
- The 2 drawings are correct but they are different
- Only 1 of the drawings is correct.

The final case usually involves heated argument concerning the meaning of the word 'and', as the students who agree that only one drawing is correct cannot agree which one.

We next introduce a description game, where the teacher no longer describes what is to be drawn (using only circles and lines) but the children have to do a drawing and to describe it to a partner, who has to reproduce the drawing without seeing their partner's original creation. The teams (in pairs) are awarded points when the rest of the class agrees that the two drawings are the same. Now, due to the competitive nature of the game, students identify the need to agree on rules for 'sameness'. Interestingly, when they re-assess the drawings in figure 2 the students almost always state that they are not the same because the number of circles is different. Asked who is to blame for their initial confusion, they usually state that the description from the teacher should have been better.

The next step is to introduce the concept of a property of a drawing that can be checked even if it is not mentioned in the description. We start with the idea of connectivity. Students have no problem in identifying connectivity in the left side of figure 2, and absence of connectivity in the right side. More advance students will even start to define connectivity in terms of paths.

For younger students, who cannot read or write, we now use the drawings as building plans for constructing models of coloured balls connected by string. These children are quite comfortable following such plans (from games such as LEGO) and so find no difficulty in making the constructions. The advantage with this approach is that the constructions are flexible and even when they change shape the students consider them not to have changed. This is the beginning of them starting to reason about semantics in terms of equivalence classes, and they return to the rules concerning sameness in the description game.

For students who are able to read and write, we introduce another challenge: to write down a textual description of the drawings using as few letters as possible. In the beginning they just write the natural language description, but the game environment motivates them to find a more concise, unambiguous syntax. Typically, the students will describe the drawings in figure 2 in the following sequence of improvements: "Red to Blue and Green to Red", "Red-Blue,Green-Red","RBGR". At this stage, we ask about the equivalence of two textual descriptions. For example, is "RBGR" the same as “GRRB” or “RRBG”? Further we ask about the meaning of strings such as “RBGRR” where there is redundant information in the textual description. In later sessions we return to this type of issue when we look at using graphs to describe languages or strings in a password game.

Finally, with the oldest students, we ask them to reason about paths. We ask if there is a path from red to blue and a path from red to green then is there a path from blue to green. This leads to interesting questions concerning whether paths are uni-directional or bi-directional, and whether different circles can have the same colour. To further the discussion, we ask if it would make any difference if the circles were locations (towns, for example) instead of colours. The question of which drawing is correct is then addressed when we talk to them about merging 2 drawings (in order to merge paths). In the merge game, we divide the textual description into two parts and give a single part to each member of a pair. We then ask them to merge their individual drawings into one. Most pairs perform a merge as seen in the right of figure 2. However, when asked “if there is a path from town1 to town2 and a path from town2 to town3 then must there be a path from town1 to town3?”, students often spend some time rethinking the merging puzzle. They start to see that merging as in the drawing on the left of figure 2 may be more useful when thinking about paths.

To complete this first session we ask the oldest students to reason about the drawings by considering only the textual representation. For example, we introduce the concept of a circuit (where lines are uni-directional) and ask them to read the textual description of a drawing (following their own syntax) and decide whether there is a circuit or not. With the visual representation the students spot circuits immediately but find it very difficult to explain how they do it. With the textual representation there is a real struggle to check for a circuit (initially) but when they explain how they do it (or try to do it) we see the start of an algorithmic reasoning, and abstract modelling.

3.2 A simple session: algorithmic concepts

Not all of the schools participated in this follow-up session: the 2 largest classes did not continue (but feedback from the teachers led us to believe that the size of the class had no influence on their decision to stop).

It has been run eight times in three different schools (in Ireland and France) and with four different age groups — the youngest group aged 5-6, the oldest group aged 14-15. The class sizes varied from 12 to 28 children. The subgraph problem was our first attempt to observe algorithmic reasoning.

To start, we explain the concept and terminology of graphs and subgraphs through the operation of erasing lines and circles in drawings (or removing balls and strings in the physical constructions). We then provide 2 drawings and ask whether one can be made simply through erasing or removing elements of the other.

It is interesting to note that nearly all students count the circles (balls) because they intuitively know that the subgraph cannot have more balls than its supergraph. However, very few count the number of lines (pieces of string).

In figure 3, we show a typical subgraph problem that students of all ages are able to reason about: are any 2 of the 3 graphs related by the subclassing relation?

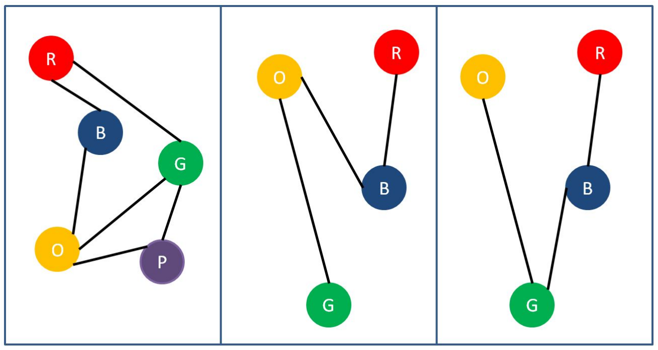

Figure 3: Typical Subgraph Problem

We then give them the same problem, but with all the balls being the same colour. This is quite challenging for the younger children, but the older children enjoy the puzzle for graphs of large numbers of circles (balls). What is interesting — for us — is to see how the older students rise to the challenge of checking for a subgraph when they have only the textual representations to play with and manipulate.

3.3 An intermediate session: shortest paths

This session has been run five times in two different schools (in Ireland and France) and with two different age groups — the youngest group aged 8, the oldest group aged 14-15. The class sizes varied from 20 to 28 children. We note that students younger than 8 were not asked to participat: speaking to the teachers we realized that the younger children did not have enough understanding of length (or size) to be able to reason about shortest paths.

To begin, we introduce the notion of different types of line. First we start with colours (and coloured string). Then we cut the coloured string into fixed constant integer lengths (in cms). The younger students can then measure paths in a relative fashion by joining up strings and comparing their lengths. The older children quickly tire with measuring strings and ask if they can just write the numbers beside the coloured lines. As a final step, they forget about colouring the lines and just write the numbers. At this point we introduce the puzzle of finding shortest paths in graphs.

We start with a very small example, and build it up in a number of steps, as illustrated in figure 4.

With the more advanced students, we allow them to collectively construct their own graphs — on the black/white board at the front of the class — to see who can find the shortest path between a start and end colour, in a limited amount of time. The students often wish to make things as complicated as possible so they add lines with enormous values (usually integers with 10+ digits), they also “sneakily” add lines with zero values, and in some instances negative values. A typical student constructed shortest path problem will look like the graph in figure 5.

After playing with a number of such puzzles, the students are invited to make comments, ask questions, propose answers to questions and to try out new instances of the problem to demonstrate the issues at hand. Interestingly, different classes often address the same issues:

Does it matter if you flip the start and end points?

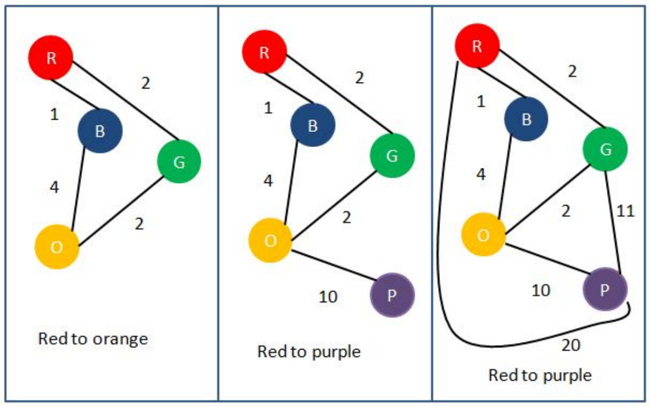

Figure 4: Shortest Path Problems — Increasing in difficulty

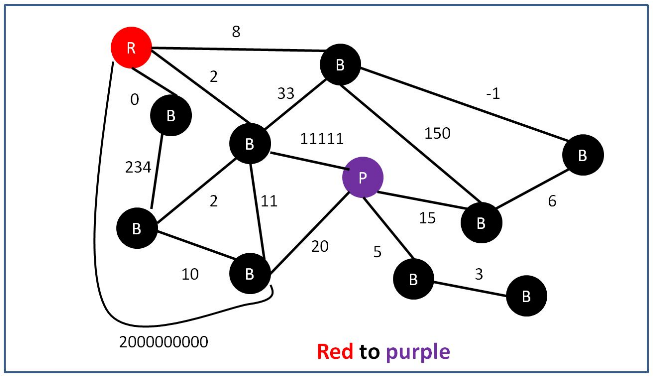

Figure 5: A typical student-generated shortest path problem

Can you ignore parts of the graph (as in links with values 5 and 3 in figure 5)?
What happens if a negative value is in a loop?
Is it always best to take the path with the least value from your current position in the graph?
Is it always best to avoid the path with the biggest value from your current position in the graph?
Is the shortest path (in terms of the sum of the values) always the shortest in terms of the number of connections?
Is it more difficult to find the longest or the shortest path?

The role of the teacher is not to answer such questions but to encourage the students to think about them amongst themselves. It is surprising how many times they find the right answer through collective discussion and creation of counter-examples.

3.4 Formal reasoning

Once the students have developed a thirst for reasoning about problems like the shortest path, we attempt to get them to reason using what they know about integer mathematics, together with their intuitive understanding of properties of graphs.

A good step for this is to introduce a problem, often as a story that involves a puzzle. For example, with the shortest path, we can phrase the problem in terms of a magician who wishes to get to a certain town and the links are distances to travel between towns. In some cases, towns are not directly connected because of physical or magical barriers.

We state that the magician already knows the shortest path between any two towns, through the large number of trips that he has made in the last few years. However, he has now been given a magic transporter wand which he can use once only for every trip in order to travel along an existing road without incurring any cost (distance or time). The question is, how does he know where he should use the wand when making the trip.

Most children, without hesitation, choose to use the wand on the connection with the highest value (on the known shortest path). We ask them to explain why. It is usually only through trying to "prove" that they are right, that they see that they are wrong. At this point the next most common suggestion is to just use the wand on the connection with the largest value (anywhere in the graph). Again, it does not take them long to see that this reasoning is also wrong. After multiple similar attempts, we explain to them that this is not a simple problem and that finding the solution is a good example of computer science in action. At this point we introduce the notion of an algorithm and proving that an algorithm is correct.

4. IMPORTANT OBSERVATIONS

Our experience shows that these sessions work only if there is an enthusiastic teacher involved. It is not critical that the teacher has each session explained to them beforehand — they can participate and interact with the children, learning with them rather than teaching them. However, it is also sometimes useful that the teacher is prepared for a particular session in advance — in order to better observe the childen learning instead of being wrapped up in the learning process themselves. When possible, it is best to have 2 teachers involved — an observer and a participant. These teachers can swap roles in order to get a better understanding of the learning process and the computer science.

Children who lacked confidence in fundamental skills such as literacy and numeracy often gained confidence from playing with the graphs and the graph problems. They were delighted to see that sometimes there was not just a right answer and a wrong answer — and that every answer (even their own) had good points, as well as bad. As a result, many of these classes have had an indirect positive impact on students' peformance throughout their schooling; and even though some of the teachers involved were sceptical at first, their feedback became more positive as the sessions progressed.

It is possible to base a whole semester of work around these graph sessions (for students ranging from 5 to 17 years old). The key is for each session to naturally progress into the next, and at the same time for the teacher be ready to add ever more difficult challenges to any particular session.

For the older students, we saw that they quite naturally started to work on algorithmic solutions to some problems by looking at the textual descriptions (using their own abstract syntax). The better students even asked about choosing the best syntax to help solve the problems. Thus, they discovered the link between data structures and algorithms.

5. CONCLUSIONS AND FUTURE WORK

This research is purely observational and we have not collected hard data in order to test any hypotheses that we may have about this teaching approach. An empirical study across many different classes could lead to interesting results. However, this paper is more about convincing teachers and CS educators to try out this type of educational process in schools near to them.

A long term research goal is a curriculum for teaching computer science to children as soon as they go to school — that is primary school, not secondary school. Children aged from 5-11 have so much potential for learning about algorithms and computation that it would be a shame to wait until they are teenagers before we teach them the foundations. We believe that our work shows that you can start teaching computer science before students even know how to read and write.

We have risen to the challenge of Bruner — as stated in the introduction to this paper — and we have shown, through our teaching of graph algorithms, that this small, but important, part of computer science can be taught effectively to most children of school age. The challenge is to extend this to cover as much of our discipline as possible.

6. REFERENCES

[1] D. Ball, U. S. O. of Educational Research, and Improvement. *Research on teaching mathematics: Making subject matter knowledge part of the equation.* National Center for Research on Teacher Education, Michigan State University, 1988.

[2] T. Bell. A low-cost high-impact computer science show for family audiences. *23rd Australasian Computer Science Conference*, 00:10–16, 2000.

[3] G. M. Bianco and S. Tinazzi. One step further the ACM K-12 final report: a proposal for level 1: computer organization for K-8. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 207–211, New York, NY, USA, 2006. ACM.

[4] J. Bruner. *The process of education.* Harvard University Press, 1960.

[5] M. D. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Comput. Educ.*, 33:253–278, December 1999.

[6] S. da Rosa. Designing algorithms in high school mathematics. In C. N. Dean and R. T. Boute, editors, *TFM*, volume 3294 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2004.

[7] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4:13–17, April 1970.

[8] G. Futschek. Algorithmic thinking: The key for understanding computer science. In R. Mittermeir, editor, *ISSEP*, volume 4226 of *Lecture Notes in Computer Science*, pages 159–168. Springer, 2006.

[9] J. Gal-Ezer, C. Beeri, D. Harel, and A. Yehudai. A high school program in computer science. *IEEE Computer*, 28(10):73–80, 1995.

[10] J. P. Gibson. A noughts and crosses Java applet to teach programming to primary school children. In J. F. Power and J. Waldron, editors, *Proceedings of the 2nd International Symposium on Principles and Practice of Programming in Java (PPPJ 2003)*, volume 42 of *ACM International Conference Proceeding Series*, pages 85–88, Kilkenny City, Ireland, 2003. ACM.

[11] J. P. Gibson. Formal methods — never too young to start. In Z. Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 151–160, Budapest, Hungary, Mar. 2008.

[12] J. P. Gibson and J. O'Kelly. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[13] M. Guzdial. Learning how to prepare computer science high school teachers. *Computer*, 44:95–97, 2011.

[14] O. Hazzan, J. Gal-Ezer, and L. Blum. A model for high school computer science education: the four key elements that make it! In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, SIGCSE '08, pages 281–285, New York, NY, USA, 2008. ACM.

[15] O. Hazzan and I. Hadar. Reducing abstraction when learning graph theory. *Journal of Computers in Mathematics and Science Teaching*, 24(3):255, 2005.

[16] J. M. D. Hill, C. K. Ray, J. R. S. Blair, and C. A. Carver, Jr. Puzzles and games: addressing different learning styles in teaching operating systems concepts. *SIGCSE Bull.*, 35:182–186, January 2003.

[17] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 1455–1464, New York, NY, USA, 2007. ACM.

[18] J. Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, 2007.

[19] A. Levitin and M.-A. Papalaskari. Using puzzles in teaching algorithms. *SIGCSE Bull.*, 34:292–296, February 2002.

[20] P. Martin-Löf. Constructive mathematics and computer programming. *Logic, Methodology and Philosophy of Science VI*, pages 153–175, 1982.

[21] B. Parhami. Puzzling problems in computer engineering. *Computer*, 42(3):26–29, 2009.

[22] S. Pollard and R. C. Duvall. Everything I needed to know about teaching I learned in kindergarten: bringing elementary education techniques to undergraduate computer science classes. *SIGCSE Bull.*, 38:224–228, March 2006.

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