# CS Unplugged—How Is It Used, and Does It Work?

Tim Bell[1(✉)] and Jan Vahrenhold[2]

[1] University of Canterbury, Christchurch, New Zealand
tim.bell@canterbury.ac.nz
[2] Westfälische Wilhelms-Universität Münster, Münster, Germany
jan.vahrenhold@uni-muenster.de

**Abstract.** Computer Science Unplugged has been used for many years as a way to communicate concepts from computer science to audiences in a variety of settings. Despite its widespread use, there is relatively little systematic evaluation of its effectiveness. In this paper we review what (Computer Science) Unplugged means, and the many contexts in which it has been used, as it started as an outreach tool, and then found its way into other contexts such as teacher training, and more recently into the classroom to support a formal curriculum. Formal evaluations of using Computer Science Unplugged as an approach to teaching the subject of computer science are reviewed, and we also reflect on the complex considerations that lie behind the development of activities and puzzles that are simple enough for school students to use.

**Keywords:** CS education · CS Unplugged · Pedagogy

## 1 Introduction

Computer Science Unplugged (CS Unplugged) is a widely used collection of activities and ideas to engage a variety of audiences with great ideas from computer science, without having to learn programming or even use a digital device [20]. CS Unplugged is mentioned in hundreds of papers about CS education, and appears regularly in curriculum recommendations, teacher forums, and social media.

It originated as an outreach program to engage primary school students with these ideas and especially to help them understand what computer science might involve other than programming [20]. The activities published through the csunplugged.org site have since ended up being widely used for classroom and out-of-school instruction [38], and have been translated into over 20 languages. It is frequently mentioned in books on teaching computer science [26,61], and it is used as a pedagogical technique on "coding" sites such as code.org.[1] The CS Unplugged approach appears in curriculum recommendations and designs

[1] https://code.org/educate/curriculum/cs-fundamentals-unplugged.


H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 497–521, 2018.
https://doi.org/10.1007/978-3-319-98355-4_29

including example activities in the 2003 ACM K-12 recommendations [108], as part of the design of a middle-years school curriculum [95], as a component of the "Exploring Computer Science" course [53], and as a resource to support the Australian Digital Technologies curriculum [43].

But being popular does not necessarily mean that it is effective as an educational tool, and some research around its effectiveness in school settings has raised doubts about whether or not it should even be used. In this paper we review the considerable literature that has developed around using and evaluating the CS Unplugged approach.

We begin in Sect. 2 by exploring what "Unplugged" has come to mean, and then in Sect. 3 we review a variety of contexts in which it is used, including outreach, broadening participation, and teacher professional development. Section 4 reviews the literature on the effectiveness of CS Unplugged when it is used for the specific purpose of teaching topics in computer science in a general classroom setting. In Sect. 5 we then consider the broader picture of how a deeper view of the CS Unplugged topics can inform those who create examples for students, and also underpin teaching the material.

## 2 What Is CS Unplugged?

We first need to define what we mean by "Computer Science Unplugged." The term is used in several slightly different ways, and as we analyze its effectiveness, it's important to be clear how the term is being used. The term originated with a collection of activities that were shared online from the early 1990s in various formats, culminating in 1999 with a free online book co-authored by Tim Bell, Mike Fellows and Ian Witten, called "Computer Science Unplugged: Off-line activities and games for all ages" [13]. Some formal studies refer to this original book (the "classic" edition), which was intended for academics involved in outreach; a "Teachers' edition" was published in 2000, and revised in 2002 [14]; this version was re-written in a way that classroom teachers could use it, although it was still intended as enrichment and extension exercises, and did not assume that computer science would be part of the curriculum. The material can be adapted for classroom use, but this requires consideration of issues such as linking the activities to curriculum objectives and broader issues in computing, and assessing student progress [19]. Recently the material has been developed into lesson plans for use within curricula, and this includes links to "plugging it in" activities, which are ideas for related programming exercises. Unlike the other versions, this latest version is published as a website rather than a book, although all of the versions mentioned above are available through the main CS Unplugged site.[2]

The CS Unplugged website articulates a number of principles that it is based on, including no computers required, real computer science, learning by doing, fun, and resilient activities. The "Unplugged" material is not intended to be

[2] https://csunplugged.org.

used as a curriculum or program of study, but as a form of pedagogy that has several potential benefits:

- the barrier of learning programming, which can be seen as an insurmountable hurdle by some, is removed as a prerequisite to learning about great ideas in computer science [16];
- students can engage in a meaningful way with the broader and lasting issues tackled by computer science [67], and avoid the misconception that the subject is only about programming [91], which supports a spiral curriculum where students are able to maintain an overview of where their learning is heading, rather than getting bogged down in the details of one aspect of the subject;
- it can be used in situations where computers are not available, or if they are, they can have other issues such as distracting students or causing technical issues as software must be installed and deployed in the classroom situation (e.g., teaching where access to devices is limited [73]);
- if a short time slot is available and/or there is a large audience (e.g., a 15-min outreach talk to a school or a show in a science center [11]), it is easier to engage students with a CS Unplugged exercise rather than trying to have students do programming in that setting.

New activities have been developed by other educators; many have been shared on the CS Unplugged website, but many also exist independently, including books by other authors that pick up the "Unplugged" theme [24,110].

This means that the CS Unplugged approach has become more than a particular collection of activities (which themselves are evolving anyway). The general idea of using an "Unplugged" approach for making a discipline approachable is discussed in [18]. Some specific guidelines on how CS Unplugged activities can be designed are given in [88], who identify some key principles that underpin the CS Unplugged approach, particularly:

- avoiding using computers and programming,
- a sense of play or challenge for the student to explore,
- being highly kinesthetic,
- a constructivist approach,
- short and simple explanations, and
- a sense of story.

With the variety of material appearing, the term "Unplugged" has come to be used for computer science teaching activities that do not involve programming, and so the term has come to refer to a general pedagogical approach. Due to the subtle differences between each of these forms of "Unplugged," it is important to be aware of which context is being referred to when a single activity or an event is evaluated.

## 3 Survey of Applications of CS Unplugged

As noted above, CS Unplugged was initially intended for use as an outreach tool, to communicate what the subject involves to those not engaged in computer science and help them become engaged with it. This has been important as new curricula are being implemented internationally that include computer science, and those implementing the new material, e.g., officials, school management, and teachers, may not have experience of it from their own education.

In this section, we review how this has led to "Unplugged" approaches being used in a variety of situations *other than* directly in classroom teaching; a review of its use in the classroom is given in the next section. This review presents a sample of reports in the literature of how CS Unplugged has been used. These papers do not usually evaluate CS Unplugged *per se*, although they often evaluate an overall program in which is was used. Thus we cannot rely on this as evidence of the effectiveness of the "Unplugged" approach, although it does expose a variety of situations where teachers and presenters have adopted the approach, which implies that their professional view was that it held promise in that situation.

*CS Unplugged for Outreach.* CS Unplugged was originally intended for outreach, and there are many reports in the literature of it being used for camps, special events, after-school programs, and extra-curricula activities. An "Unplugged" approach can be used in a variety of outreach situations, including contests, shows, and magazine articles [16]. A common scenario is using it as part of an event for school-age children. An "Unplugged" approach gives the opportunity to cover topics of some substance in relatively little time, which is often needed in an outreach situation; while simple steps in programming can also be covered quickly, getting to the point where a student has access to the big ideas of programming takes some time. Lonati et al. [79] raise the concern that as an initial experience, programming can be negative for students, particularly if covered in a short time—they point out that the "hour of code" risks giving an "incorrect first impression," and instead propose an "Unplugged" approach. For example, Alamer et al. [5] use CS Unplugged in a camp where there is limited time, to introduce programming concepts, Khoja et al. [71] use CS Unplugged in the introductory part of a camp for girls, and Sysło and Kwiatkowska [103] use it as part of a "children's university." The "Abenteuer Informatik" project [50] has even been used as an interactive display in shopping malls, where passers-by might only spend a matter of minutes engaged with the display. Overdorf and Lang [89] report that a group of undergraduate women designing an outreach program chose CS Unplugged activities over teaching a programming language such as Scratch or Logo, Henderson [63] reports using CS Unplugged activities successfully for enrichment courses, and Rosamond et al. [94] suggest CS Unplugged as a possible element of presenting scientific results in the media, where the reader may only spend limited time reading the article.

*CS Unplugged and Magic.* A common misconception is that young people know a lot about "computer science." However, it contains many unexpected elements [15], and this can be used to present magic tricks based on ideas that students do not know, yet are right in front of them in their digital world. A number of tricks based around an "Unplugged" approach have been reported [31,54]. The use of magic shows to teach CS principles is another form of Unplugged outreach [46,51]; Curzon and McOwan [30] use magic tricks to explain algorithms and even introduce algebraic proofs for verification. One of the first CS Unplugged activities developed was the "Parity" magic trick, which uses error correction to make the audience think that the presenter is a mind-reader; Hromkovič et al. [65] discuss how this can be extended into a detailed discussion of fundamental underpinnings of Computer Science, while Greenberg [54] analyzes this quintessential Unplugged magic trick in considerable detail, and comes up with variations.

*CS Unplugged and Broadening Participation in CS.* There is considerable interest in attracting and retaining those who are under-represented in computer science, including girls, special-needs students, ethnic minorities and otherwise under-served K-12 Students [29]. Lau and Yuen [76] suggest CS Unplugged as one of three approaches that is promising for "nurturing a gender and style sensitive classroom." Cohoon et al. [27] use CS Unplugged as part of a workshop for teachers, which resulted in teachers becoming very active in recruiting girls and minority students (although of course this effect cannot be linked to CS Unplugged alone). Stewart-Gardiner et al. [100] and Sullivan [102] report on using Unplugged-based activities as the basis of games for an after-school program for girls. CS Unplugged activities can help students with disabilities engage quickly with the subject, too; for example, activities have been adapted for visually impaired students [23,69,70]. Manabe et al. [80] report on using CS Unplugged ideas for students with disabilities (for example, the sorting network is a challenge for students in wheelchairs), while Marghitu et al. [81,82] use CS Unplugged activities as part of a camp also attended by special-needs children.

*"Unplugged" Approaches to Measuring Achievement.* Assessing student achievement becomes important as the idea of computational thinking (CT) enters the curriculum. Assessing CT using programming implies that students must first develop that skill, and the concern remains that the assessment will be affected by other issues such as access to devices and detailed knowledge of syntax. Several authors have proposed an "Unplugged" approach to assessing CT. One of the most widely used forms of this is the "Bebras" challenge,[3] which is a test of computational problem solving that doesn't require the use of a programming language [34,57]. Hubwieser and Mühling [68] explore the idea of using the Bebras challenge as an internationally comparable measure in the style of PISA surveys. Bell et al. [12] report student explanations in high school assessment based on an Unplugged approach, which personalizes the work (for example, a

[3] http://bebras.org.

student explaining a comparison of sorting algorithms using their own examples based on the balance scale activity from CS Unplugged). Dwyer et al. [40] use CS Unplugged to measure students' ability to work with step-by-step instructions in algorithms, and Voigt et al. [111] consider the use of CS Unplugged activities as the basis of programming competitions.

*CS Unplugged Helping with Integration with Other Subjects.* Since CS Unplugged activities are fast to deploy and do not require expensive specialist equipment, they have also been used as the basis of connecting computer science with other disciplines. At primary school level the connections with maths are easy to see, but teachers have also connected the activities to topics as diverse as physical education, literacy, creative writing, and art [17]. Dwyer et al. [41] used an "Unplugged" approach to connect computational thinking to physics, Carruthers et al. [25] integrated algorithms with biology, and CS Unplugged ideas have also been used to integrate computational thinking and music [9,18].

*Using "Unplugged" Approaches to Make Specialised CS Topics Accessible.* Sometimes traditional introductions to topics have been difficult to access for those with a weak background, particularly in programming or mathematics. A number of authors have adapted or created CS Unplugged activities to overcome these barriers. For example, Sivilotti and Pike [98] consider CS Unplugged activities in the context of distributed algorithms, and point out the value of kinesthetic activities for "increasing student engagement and promoting open classroom discussion." Heintz et al. [62] developed a course that includes the use of a CS Unplugged activity to teach HCI, and Ford et al. [47] report positive results from using an "Unplugged" approach for teaching students about cybersecurity, to make it more accessible to those with a limited technical background. Hara et al. [60] use an "Unplugged" approach to demonstrate Page Rank (as well as teaching HTML) for a writing-intensive liberal arts course.

*CS Unplugged for Teacher Professional Development.* Computational thinking and computer science have been introduced in various forms to curricula in many countries, and this has meant that teachers have needed professional development to be able to deliver the new subject, particularly in primary schools, where teachers are generalists and are required to cover most subjects [32]. It is not uncommon that those delivering the subject did not encounter it as part of their own study [66], and this creates a significant challenge. Being able to help teachers understand key ideas in the subject and allow them to have early success in their classroom is important. If teachers find the subject hard to become enthused about, then having it in the curriculum can backfire since students might receive a negative message about the subject in school [59]. Since learning to teach programming can be daunting for teachers with no experience in this area, a CS Unplugged introduction can help to break down defenses and overcome teachers' fears. Using an "Unplugged" approach means that relatively deep ideas can be covered fairly quickly in a playful and empowering way, using familiar materials (paper, string, chalk and so on), rather than having to install and

learn to use complex software. This provides quick success and a positive experience for teachers who might otherwise find it difficult to become engaged with the subject. For example, Curzon et al. [32] used an "Unplugged" approach combined with story-telling for teacher professional development, and reported that it was "inspiring, confidence building and gave [the teachers] a greater understanding of the concepts involved." Cutts et al. [33] used an "Unplugged" approach with new material that they developed for "Enthusing and informing potential computer science students and their teachers," while Smith et al. [99] found that Master teachers (who were helping other teachers) included CS Unplugged in what they used to provide professional development for colleagues. The "MyCS" program for middle-schools, which was designed for teachers "without prior CS training" [36] drew on CS Unplugged ideas; Liu et al. [78] report CS Unplugged activities being used for teacher training in three Universities (Purdue, Carnegie Mellon, and Marquette), Pokorny and White [90] used it for a teacher workshop, and Gutiérrez and Sanders [59] suggest it as a way to break down teachers' concerns. Duncan et al. [39] found that teachers responded positively to CS Unplugged activities, and also saw curriculum integration opportunities that helped them to become engaged, but also revealed that misconceptions can arise, and that it is important to have mechanisms in place to help non-specialist teachers avoid these. Bellettini et al. [21] report that using an activity "inspired by" CS Unplugged received positive feedback from teachers, and Smith et al. [99] found that teachers valued sharing alternative ways of teaching, including CS Unplugged. Morreale and Joiner [87] surveyed which tools teachers used after attending their workshop, and found that CS Unplugged was one of the most widely adopted, particularly the lessons on binary number representations; Morreale et al. [86] included CS Unplugged in the resources that they introduced to teachers, and highlighted the importance of ideas that teachers "were able to immediately use on their return to the classroom." Sentance and Csizmadia [96] also found that when they followed up on what teachers took away from a workshop, "a significant proportion of teachers mentioned, unprompted, that they try to support students' understanding by using physical, or unplugged-style activities in the classroom."

The examples above show that an "Unplugged" approach is popular in outreach and recommended for teacher professional development, and that teachers report using it when they have found out about it. But this raises the question of whether or not it is effective in the classroom, and this question is explored in the following section.

## 4 Effectiveness of CS Unplugged for Teaching

The kinesthetic and playful nature of CS Unplugged activities makes them obvious candidates for outreach settings such as camps or short training events, which is reflected in the large number of experience reports regarding the effectiveness of CS Unplugged for these situations. However, despite the fact that CS Unplugged activities were suggested in the CSTA Standards for K-12 computer science [108], and that teachers report using CS Unplugged activities

in class [96,107,112], surprisingly few empirical studies about the use of CS Unplugged activities in a regular classroom setting have been conducted. In this section we review the studies that exist, and reflect on how an "Unplugged" approach can be used in the classroom.

Taub et al. [104] investigated the use of CS Unplugged activities (which were designed for primary school children) in middle-school classrooms and after-class activities. Using a mixed-methods approach, they collected data regarding views, attitudes, and intentions of computer science, in particular as a field of work. The main insights from this classroom-based study was that CS Unplugged activities indeed changed the students' view of computer science towards a more mathematical thinking. However, at the same time, these students still considered computers essential to computer science and, alarmingly, were found to become less attracted to the field.

Taub et al. conjectured that these effects were due to the CS Unplugged activities being only loosely related to "central concepts" in computer science and not explicitly linked to students' prior knowledge. The latter, in fact, is a direct consequence of the activities having been developed for very young children who—by definition—cannot be expected to have much prior knowledge in computer science or mathematics to build upon. Taub et al. suggested that one should revise CS Unplugged activities to be used with older children to include the following four processes described by Linn and Eylon [77, p. 523f.]: eliciting the students' existing ideas, introducing new ideas, developing criteria for evaluating new ideas acquired, and sorting out ideas. For example, the CS Unplugged sorting activity is interesting in itself, but some effort is needed to draw out the idea that sorting algorithms are used in real life, and that the time taken by algorithms is not usually linearly proportional to the amount of data, which has a significant impact on real users when the wrong algorithm is used in a program.

The study by Feaster et al. [44] seems to confirm the above observations: in their evaluation of a one-semester course using CS Unplugged activities for even older students, i.e., students in high-school, they found that CS Unplugged materials were not suited to improving content understanding, possibly because the students viewed themselves as already expert in the subject, and also were less interested in kinesthetic activities. However, they reported "a number of experimental factors" including "adjusting for preexisting differences in student attitudes and content understanding, [and] adjusting for instructor-induced impacts" [44, p. 252] to be outside the researchers' control.

Thies and Vahrenhold [105] examined the CS Unplugged activities with respect to learning objectives for students in lower secondary education. They found that the union of the derived learning objectives did not fully cover all dimensions of Bloom's revised taxonomy [6]. However, their classification also formally showed that CS Unplugged activities indeed address objectives well suited for outreach *and* for introducing new topics in class, thus showing the applicability of CS Unplugged beyond reasons of playfulness or creating intrigue. Building upon these results, Thies and Vahrenhold investigated the effectiveness

of using CS Unplugged for introducing new topics in class, comparing three activities (Binary Numbers, Binary Search, Sorting Networks) with conventional instructional methods. In a controlled experimental setting that measured short- and mid-term understanding of the concepts taught, no statistically significant differences between the two groups could be found [106]. The experiment was revalidated across different instructors, schools, and age groups; also the Treasure Hunt activity (covering finite state automata) was examined in a high-school course. Again, no statistically significant differences could be found between CS Unplugged and conventional methods [107].

Thies and Vahrenhold [107] also surveyed teachers who had attended a professional development workshop on CS Unplugged. Based upon their classroom experiences, the teachers that had used CS Unplugged in their classroom after the workshop agreed, at times very strongly, that CS Unplugged material made students curious to learn more about computer science, did not subchallenge students, and could be used across a variety of ages. Those who had not used the material in class stated that they felt uncomfortable teaching kinesthetically, feared that preparing CS Unplugged lessons would take too much time, and also that their classrooms did not allow for teaching kinesthetically; only a few explicitly mentioned that their students were either too young or too old for CS Unplugged.

Rodriguez et al. [93] followed up on the classification of the learning objectives in CS Unplugged activities [105]. They investigated which activities could be used to foster computational thinking as expressed in the following aspects: (1) Data Representation, (2) Decomposition, (3) Abstraction, (4) Algorithmic Thinking, and (5) Pattern Recognition. These activities were then reworked, for example, by adding worksheets or introducing new supplementary activities, to explicitly address computational thinking. Among other insights, they found priming activities that address naive or pre-existing ideas to be very helpful in fostering understanding, thus confirming Taub et al.'s [104] suggestion to explicitly elicit students' existing ideas. Rodriguez et al. [92] report on a project in which they adjusted not only the CS Unplugged activities but also assessment instruments to measure aspects of computational thinking. In their discussion of the results obtained, however, the authors caution against "forcing every assessment of computing skills to fit neatly within CT" [92, p. 506].

The study by Wohl et al. [114] employed a rather different assessment approach. In a comparative study on Cubelets, CS Unplugged, and Scratch in three primary schools, pupils received roughly two hours of instruction time using the respective methodology or tool. They then were asked to draw figures and to build paper models of the procedures developed. The authors found that these "could also be used as a proxy for understanding" [114, p. 57]. Combining these findings with the results from an interview study, the authors conclude that "[CS Unplugged] appeared to generate the highest level of understanding of the concepts of algorithms, logical predictions and debugging; while Cubelets proved one of the most engaging methods; and Scratch generated the most 'tool'-based questions" [114, p. 60].

Some of the negative results about the use of CS Unplugged in the classroom have been based on using it as a self-contained curriculum that completely rejects the use of computers. However, while acknowledging that sometimes it is necessary to teach regular classes without computers,[4] in a contemporary classroom setting the CS Unplugged activities are intended to be used in conjunction with conventional programming lessons.

This duality has led to discussions regarding whether or not CS Unplugged activities should be part of programming lessons. Faber et al. [42, p. 22] report that "the unplugged aspect of the lesson materials seems to elicit positive reactions from both teachers and students" and thus, even in the absence of statistical analyzes, suggest CS Unplugged activities "as a valuable alternative to regular, online programming lessons." In contrast, Grover and Pea point out that we do need to be careful with the tools that we use, and that an "Unplugged" approach "may be keeping learners away from the crucial computational experiences [such as programming] involved in CT's common practice" [56, p. 40]. A recent study, however, by Hermans and Aivaloglou [64] used a controlled-study design between two groups of students, both learning Scratch programming, but with one group spending some of the time using Unplugged material as well. They compared the groups' mastering of programming concepts, the used vocabulary of Scratch blocks, and self-efficacy. After eight weeks of instruction, the two groups of 35 elementary students in total were found to have progressed in the same way regarding mastery of programming concepts. However, the group taught using CS Unplugged material showed higher self-efficacy and used a wider vocabulary of Scratch blocks. Similarly, Gaio [49] concludes from his study of roughly 250 pupils in grades 3 and 4 that "obsessing over using just one [approach, i.e., Scratch or CS Unplugged] is not the correct decision" [49, p. 7]. This conclusion is based on the observation that CS Unplugged activities seem to come more naturally to children and lead to more "Aha! Moments," whereas connections "from [CS Unplugged] real-world-oriented tasks to computer science" seem to be difficult to establish at times.

Summing up the above discussion about whether, and if so, how, CS Unplugged can be used to teach computer science, we recall a formative statement in a paper by Fellows and Parberry at the time that Mike Fellows was developing CS Unplugged activities: "Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes" [45].[5] This reflects a concern at the time that the device became the object of a cargo cult mentality, but to continue the analogy, presumably one would not advocate that astronomers should not use telescopes, but simply that they treat it as an essential part of the work, but not an end in itself. More recently, as computers have become ubiquitous and inexpensive,

[4] CS Unplugged has been used in schools where computers or Internet access are not available, or where there is limited time, and at the time the activities were developed it was unusual for programming to be part of the junior curriculum [20].

[5] Versions of this quote are often attributed to Dijkstra [20].

things have moved to another extreme, where students do not understand the great ideas in the software that makes digital devices so popular [15].

In fact, while teaching computer science without using a machine is possible, the reality of modern classrooms is that good physical computational tools may well be available, and that the physical device is key to digital technology becoming such a significant part of daily life, even though it is the algorithms and data on the device that bring it to life [28]. Reflecting this new environment, the CS Unplugged material is being converted to lesson plans, which in turn are being augmented with "plugging it in" exercises to provide a way to use it with programming classes, and this is supported by the research reviewed above. The "programming vs. Unplugged" discussion should not be an "either-or" debate, but working out how to combine the two to bring about valuable learning, where each approach is used where it is most effective.

Taking this a step further, there are numerous reports on using an "Unplugged" approach as a way to teach programming, where an Unplugged explanation or walk-through is used before getting on the computer to help students understand the design of a computer program or language elements [5,37,48,97]. This is a promising pedagogical approach, as it gives students the opportunity to design their program away from the computer, rather than launching into writing code before thinking through the requirements of the whole task. It has also been used for exploring particular programming concepts; for example, Gunion et al. [58] use an "Unplugged" approach to successfully teach recursion to middle-school students.

We conclude this section with a quote from a recent literature review on the pedagogy of teaching computer science in schools: "Despite mixed evidence of the impact of unplugged activities on student learning, the approach appears to be popular with teachers as a pedagogical approach. Much more research is needed to determine how best to use it effectively" [112, p. 31].

## 5 Beyond Playfulness and K-12 Computer Science

Many CS Unplugged activities are appealing to young children as they can connect to the material as a Kindergarten activity, such as coloring with crayons. However, in line with the suggestion by Taub et al. [104], the activities should be embedded into the larger context of computer science. In this section, we exemplify how two topics touched upon in CS Unplugged activities can be linked to non-trivial concepts of computer science beyond the topic that is presented to the students. These relationships, in particular when used in teacher training or certification courses, can be built upon to relate K-12 computer science to more advanced concepts, thus presenting an intellectual counterpart to the perceived playfulness of the activities. Teachers can be aware of the depth of the subject they are teaching, even if the material has been simplified to a format that young students can work with.

### 5.1 Graph Coloring

When developing any activity or task for learners, educators need to ensure that the problems presented actually have a well-defined solution. For example, unless exploratory learning is employed, mathematics teachers in primary school need to make sure that exercises involving addition do not result in numbers outside the range students are familiar with; similarly, quadratic equations in high-school usually are required to have real-valued roots. It is easy to generate questions for students in these domains; it requires broader understanding to generate problems that are pedagogically valuable!

The playful nature of CS Unplugged activities often hides the reality that developing or extending these activities may require non-basic concepts of computer science and discrete mathematics to ensure that the examples for the children are simple or illustrate the key ideas suitably. The example we discuss in this subsection is the "Poor Cartographer" activity in which colorings of graphs are reasoned about using ideas well beyond a Kindergarten student, yet result in activities suitable for a young child. The setting of the activity is to help a cartographer color the regions of a map (planar graph) using as few colors as possible while making sure that no two regions sharing a border receive the same color.

The activity starts out by presenting maps that can be colored using exactly two colors and then touches upon the fact that any map can be colored using at most four colors. This is an obvious, non-trivial connection to advanced topics in Discrete Mathematics: in 1977, Appel, Haken, and Koch [7,8] showed by exhaustive enumeration that every planar map is four-colorable. In fact, their proof is said to be the first exhaustive proof done by a computer.[6]

However, even when starting to work with the "Poor Cartographer" activity in class, teachers need to prepare maps (graphs) that can be colored with two colors such that no two regions (faces) that share a border (edge) have the same color. The following problem in graph theory naturally arises:

> How can we construct a graph whose faces can be colored using only two colors such that no two faces that share an edge have the same color?

Surprisingly, there is a construction simple enough to be taught in class; this construction relies on the following lemma:

**Lemma 1.** *Any planar graph that is induced by a collection of closed curves is two-colorable.*

As a consequence of this lemma, teachers (and students) can construct graphs that are two-colorable by simply drawing overlapping circles, rectangles, or other closed curves. Thus, students can take ownership of the process by creating additional "challenges" for their peers or family at home.

[6] The field of graph coloring is still very active; indeed, the recent handbook by Gross, Yellen, and Zhang [55] contains two extensive chapters on this topic.

The proof of this lemma can be presented easily in teacher training courses. It starts by constructing the dual graph of the graph induced by the collection of closed curves. It then shows that each face of this graph has an even number of edges and completes the proof by induction on the number of faces. This proof can be used in teacher training courses to connect the formal concepts taught in undergraduate computer science courses for prospective teachers to a topic that can be taught in school using CS Unplugged.

One possible extension of this activity would be to ask whether the vertices of a given graph can be colored using a certain number of colors. Exploring this in class would also directly lead into the "Tourist Town" activity in which the focus is not on faces but on (dominating sets of) vertices. By duality, Lemma 1 carries over to the coloring of vertices of planar graphs, i.e., to color the vertices of a graph using two colors. If teachers want to give visually more challenging problems, non-planar graphs could be an option. The following is known about two-colorable graphs:

**Theorem 1 (Kocay and Kreher [74] and König [75]).** *A graph is two-colorable if and only if every cycle has an even number of edges.*

As there may be an exponential number of graph cycles [3,113], this theorem lends itself to nice connections from CS Unplugged to combinatorics, but enumerating and checking all cycles is infeasible in practice. However, a simple greedy algorithm based on depth-first search [2] is sufficient to check whether any given graph is two-colorable. This algorithm is exactly the greedy algorithm suggested in the CS Unplugged activity for the students to perform when coloring a graph.

For constructing a two-colorable graph, neither the above theorem nor the algorithm are helpful. However, there is a simple observation which can be used to construct a graph that is guaranteed to be two-colorable.

**Observation 1.** *Any bipartite graph is two-colorable.*

To construct a two-colorable graph, we thus simply take any connected bipartite graph $G = (V_1 + V_2, E)$ in which both $V_1$ and $V_2$ are non-empty, embed the graph in any way we wish, and uncolor the vertices (see Fig. 1). If the bipartite graph is drawn using a graph editor or vector graphics program, it is easy to move the vertices so that the two subsets of vertices aren't obvious visually. As $K_{3,3}$ is known to be non-planar, this method is not restricted to creating planar graphs; Fig. 1 shows a graph which contains $K_{3,3}$ and, hence, is non-planar. Incidentally, $K_{3,3}$ is also one of the few graphs for which the Ahrens' upper bound [3] on the number of cycles is tight [83].

As seen above, two-colorability can be verified (relatively) easily, and four-colorability is guaranteed for planar graphs [7,8]. It thus seems natural to also consider three-colorability. However, recognizing three-colorable graphs, i.e., ensuring that the vertices of a graph to be handed out to pupils can be colored using at most three colors, is substantially harder:

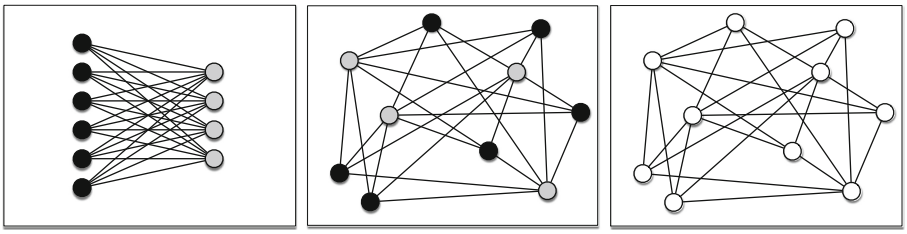

**Fig. 1.** Constructing a graph whose vertices are two-colorable.

**Theorem 2 (Stockmeyer [101]).** *Deciding whether a given graph is three-colorable, is $\mathcal{NP}$-complete.*

Fortunately, we can again use a simple construction. We start with an $n$-vertex simple polygon $P$, i.e., with an $n$-vertex polygon whose interior is connected and does not contain holes. We then triangulate the polygon by adding $n-3$ diagonals that partition the interior of $P$ into disjoint triangles, which is done by connecting vertices in a way such that no diagonals intersect (see Fig. 2). This process, while very involved if one aims at an efficient solution [109], can be done easily with a greedy algorithm that iteratively cuts off "ears" from the polygon [85].

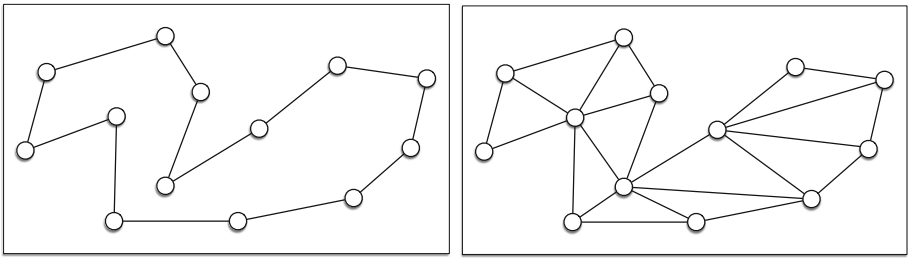

**Fig. 2.** A simple polygon and its triangulation.

The textbook by de Berg et al. [35] derives the following lemma from Meister's *two-ear* theorem [85]:

**Lemma 2.** *Any planar graph that is induced by a triangulation of a simple polygon is three-colorable.*

The idea behind the proof is to show that the dual graph of the triangulation (excluding the outer face) is a tree. Based upon this, the correctness of the following procedure can be established [35]: Start at any triangle of the triangulation and color its vertices using exactly three colors. Perform a depth-first exploration of the tree rooted at the vertex dual to this face. When visiting a new

vertex of this tree, i.e., a triangle of the triangulation, color the one uncolored vertex using the one color not used for coloring the vertices of the triangulation edge dual to the tree edge just traversed.

To construct visually "challenging" graphs whose vertices are guaranteed to be three-colorable, teachers can use the algorithm implied by the above proof to construct a three-coloring of a polygon's triangulation. They then can add arbitrary edges connecting vertices of different colors to obfuscate the original polygonal structure and then uncolor all vertices. By construction, the resulting graph is three-colorable. An example is shown in Fig. 3.

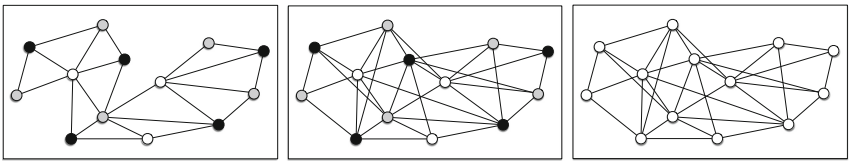

**Fig. 3.** Three-coloring and augmenting the triangulation of a simple polygon.

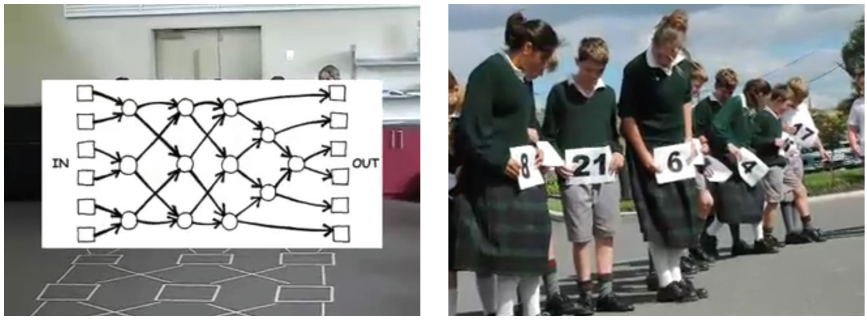


**Fig. 4.** The "Beat the Clock" activity in a classroom (left) and outdoors (right).

In conclusion, the "Poor Cartographer" activity, while seemingly playful, requires a relatively deep understanding of formal concepts if teachers want to generate interesting maps, and even slightly expand the topics presented on the worksheet. If they chose to do so, however, they can open a window into much richer topics in computer science and discrete mathematics than can be covered in K-12 computer science, thus relating the activity to tertiary education and research.

### 5.2 Sorting Networks

The second example we discuss is the "Beat the Clock" activity. This activity allows children to explore small sorting networks by enacting a sorting algorithm

using a sorting network drawn on the floor. The students' excitement when performing this activity has been captured prominently in the CS Unplugged video clips on the main website, showing the activity both in a classroom and outdoors.

The sorting network has become a popular activity with teachers, as it combines many key elements including simplicity of the explanation, physical movement, competition to complete it quickly, and intrigue for the students. It has been used for many purposes other than illustrating parallel algorithms, as it sets up a structure where students are motivated to compare values repeatedly. It can be used for any binary relation that is a total order, including musical notes (both written and sounded), text such as names, calculated numeric values such as fractions or arithmetic combinations of numbers, or sequences in nature (such as an egg transforming to a butterfly, or growth of a seed into a plant). It can even be used to memorize sequences; for example, we have seen it used to memorize a whakataukī (Māori proverb) by breaking the proverb into six elements, and having the group recite the entire whakataukī, with the students making the pairwise comparisons, noting in which order their components come in during the recitation.

Interestingly, the "Beat the Clock" activity is the only activity of those investigated by Thies and Vahrenhold [106,107] for which no statistically significant difference between traditional and unplugged teaching could be found, neither short-term nor long-term. Thies and Vahrenhold attribute this to the visual representation used in the assessments serving as a sub-conscious reminder of the procedure. A closer look, however, reveals that an additional ceiling effect might have also played a role, i.e., that the principles behind using a sorting network were both easy to grasp and easy to remember. Thus the value of this activity for integrated learning seems to be high, including the physical exercise that students get on a large network, and it is also useful as an outreach tool for quickly helping students to understand that computer science explores concepts that they may not be aware of, even if the fundamental concept can be grasped without the physical activity.

The relevance of this activity for connecting K-12 computer science and teacher training to computer science research is three-fold. The obvious connection is to the construction of sorting networks. While it is possible to construct an optimal sorting network, i.e., a sorting network that is guaranteed to use no more than $\mathcal{O}(\log n)$ parallel steps to sort $n$ numbers [4], this algorithm is almost exclusively used as a black box to prove optimality of certain other constructions, e.g., $\varepsilon$-nets [84], or protection against learning a software program from its execution [52], and (conference and journal versions combined) it has been cited over 1,500 times according to Google Scholar. The construction and analysis of sorting networks is still an active research area—see, for example, Bundala et al. [22] and the references therein.

In recent years sorting networks have become immensely relevant in practice. As sorting networks are data-oblivious in the sense that the sequence of steps taken does not depend on the input data, they are used as building blocks of

sorting algorithms for general-purpose graphic-processing units (GPUs). These chips have multiple processors (cores) accessing a shared memory in parallel; for reasons of efficiency, these parallel accesses should avoid simultaneously working on the same memory bank. To obtain such *bank-conflict free* access patterns, it is most helpful to know that an algorithm is guaranteed to perform a certain step at a certain time, and data-oblivious algorithms such as those induced by sorting networks have exactly this property. Batcher's *even-odd mergesort* [10] laid the foundation for fast sorting algorithms used on current general-purpose GPUs [1]. In fact, Batcher already envisioned back in 1968 the use of his algorithm on "a switching network with buffering, a multiaccess memory, [and] a multiaccess content-addressable memory" [10, p. 307]. His paper gives an inductive construction which may be used as a source for additional examples in class or as an example for formal reasoning about the correctness of sorting networks in teacher training. Hence, by discussing the "Beat the Clock" activity, teachers can relate to both theoretically and practically relevant aspects of computer science far beyond the classroom.

The final connection that can be made, at least in teacher training, is to the verification of sorting networks. In principle, one would need to verify the correctness of any given sorting network for $n$ input lines by checking against all $n! \in \Theta(n^n)$ permutations of a set of $n$ distinct values. However, the following Zero-One Principle shows that it is sufficient to check against all $2^n$ inputs consisting of zeros and ones only.

**Theorem 3 (Knuth [72, Theorem Z, p. 223]).** *If a network with $n$ input lines sorts all $2^n$ sequences of 0s and 1s into nondecreasing order, it will sort any arbitrary sequence of $n$ numbers into nondecreasing order.*

The proof of this theorem can be used, for example, in teacher training, to show an elegant use of contradiction: Assuming that the output contains an inversion $x_i > x_{i+1}$, one can construct a $\{0,1\}$-valued indicator function $f$ signaling whether a given input value is smaller than $x_i$. As $f$ is monotonic, this leads to the existence of a $\{0,1\}$-valued input that is not sorted correctly and, thus, to the desired contradiction.

While we might not use this proof with school students, they can reason about simpler issues, such as whether the minimum value is guaranteed to find its way to the left-hand end (and vice versa for the maximum) for the six-way sorting network (Fig. 4) by exhaustively checking the six possible starting positions.

Most examples in this section are too involved to be discussed in a standard classroom. However, they show that much more theory hides behind the playfulness of CS Unplugged than is visible at first glance, especially if teachers choose for didactic reasons to provide additional exercises or extensions. Among other implications, this prominently underlines the value of a thorough formal training of prospective teachers not only in educational and didactic matters but also in the scientific foundations of the discipline.

## 6 Conclusions

The CS Unplugged approach has become a pedagogical method that has been used for many purposes. The value of CS Unplugged for students is that it engages them with lasting ideas in our field, although the research shows that it needs to be linked to current technology and should not just be used in isolation. This also makes sense in modern curricula where programming is increasingly expected to be taught in the classroom, and the CS Unplugged approach gives opportunities to have a spiral approach rather than waiting until students have the ability to, say, program a GPU before exploring and implementing parallel sorting algorithms. Some of the literature treats the question of using CS Unplugged as an "either/or" issue, but experience is showing that if it is integrated with teaching programming then it can assist with both the motivation to explore the field as well as helping students with their "plugged-in" learning.

Focusing exclusively on whether CS Unplugged is good for students also misses an important point: it seems that it is good for teachers. It is widely used in teacher professional development, and teachers seem to find it helpful to get engaged with this topic that they are not otherwise familiar with. When computer science was not part of school curricula it was generally delivered in extra-curricula events by enthusiasts, but now that it is becoming ubiquitous, especially in primary schools, the fact that CS Unplugged is widely reported as being embraced by teachers is significant, given the influence of having an enthusiastic teacher on students' attitude towards a subject. As recent results suggest that some parts of the curriculum can be taught "Unplugged" without detrimental effects on the thoroughness of understanding, having a more enthusiastic teacher is an added benefit that can make a real difference.

Furthermore, the ideas that can be explored in an "Unplugged" mode have considerable depth, and teachers can delve into them well beyond what they might use in the classroom with students; and those developing CS Unplugged activities may find that they need to address deeper questions about the topic in order to generate examples that come across as simple explanations. For example, being sure of the chromatic number of a graph given as an example to students involves considerably more understanding than the rules that the students are given to understand the problem.

The CS Unplugged approach is used in many settings and like all approaches, if used inappropriately, it can be ineffective or even cause harm. This review, however, has identified many reports on ways that it can be used to achieve positive results, helping both students and teachers to explore computer science in a meaningful and engaging way.

## References


1. Afshani, P., Sitchinava, N.: Sorting and permuting without bank conflicts on GPUs. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 13–24. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_2
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
3. Ahrens, W.: Ueber das Gleichungssystem einer Kirchhoff'schen galvanischen Stromverzweigung. Math. Ann. **49**(2), 311–324 (1897)
4. Ajtai, M., Komlós, J., Szemerédi, E.: Sorting in $c \log n$ parallel steps. Combinatorica **3**(1), 1–19 (1983)
5. Alamer, R.A., Al-Doweesh, W.A., Al-Khalifa, H.S., Al-Razgan, M.S.: Programming unplugged: bridging CS unplugged activities gap for learning key programming concepts. In: Proceedings - 2015 5th International Conference on e-Learning, ECONF 2015, pp. 97–103 (2016)
6. Anderson, L.W., et al.: A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged edn. Addison Wesley Longman, Boston (2001)
7. Appel, K.I., Haken, W.: Every planar map is four colorable. I. Disch. Ill. J. Math. **21**, 429–490 (1977)
8. Appel, K.I., Haken, W., Koch, J.: Every planar map is four colorable. II. Reduc. Ill. J. Math. **21**, 491–567 (1977)
9. Baratè, A., Ludovico, L.A., Malchiodi, D.: Fostering Computational Thinking in Primary School through a LEGO®-based Music Notation. Procedia Comput. Sci. **112**, 1334–1344 (2017)
10. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the Spring Joint Computer Conference. American Federation of Information Processing Societies, vol. 32, pp. 307–314 (1968)
11. Bell, T.: A low-cost high-impact computer science show for family audiences. In: Proceedings - 23rd Australasian Computer Science Conference, ACSC 2000, pp. 10–16 (2000)
12. Bell, T., Newton, H., Andreae, P., Robins, A.: The introduction of computer science to NZ high schools: an analysis of student work. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education - WiPSCE 2012, pp. 5–15 (2012)
13. Bell, T., Witten, I.H., Fellows, M.: Computer Science Unplugged: Off-Line Activities and Games for All Ages (Original Book) (1999). http://csunplugged.org
14. Bell, T., Witten, I.H., Fellows, M., McKenzie, J., Adams, R.: Computer Science Unplugged: An Enrichment and Extension Programme for Primary-Aged Children (2002). http://csunplugged.org
15. Bell, T.: Surprising computer science. In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 1–11. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_1
16. Bell, T., Curzon, P., Cutts, Q., Dagiene, V., Haberman, B.: Overcoming obstacles to CS education by using non-programming outreach programmes. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 71–81. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_7
17. Bell, T., Duncan, C., Atlas, J.: Teacher feedback on delivering computational thinking in primary school. In: Proceedings of the 11th Workshop in Primary and Secondary Computing Education - WiPSCE 2016, pp. 100–101 (2016)

18. Bell, T., Fellows, M., Rosamond, F., Bell, J., Marghitu, D.: Unplugging education: removing barriers to engaging with new disciplines. In: Proceedings SDPS Transdisciplinary Conference on Integrated Sytems, Design, and Process Science, 10–14 June, Berlin, Germany, SDPS, pp. 168-1–168-8 (2012)
19. Bell, T., Newton, H.: Unplugging computer science. In: Angeli, C., Kadijevich, D., Schulte, C. (eds.) Improving Computer Science Education. Routledge, New York (2013)
20. Bell, T., Rosamond, F., Casey, N.: Computer science unplugged and related projects in math and computer science popularization. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 398–456. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_18
21. Bellettini, C., et al.: Informatics education in Italian secondary schools. ACM Trans. Comput. Educ. **14**(2), 1–6 (2014)
22. Bundala, D., Codish, M., Cruz-Filipe, L., Schneider-Kamp, P., Závodný, J.: Optimal sorting networks. J. Comput. Syst. Sci. **84**(C), 185–204 (2017)
23. Burgstahler, S.E., Ladner, R.E., Bellman, S.: Strategies for increasing the participation in computing of students with disabilities. ACM Inroads **3**(4), 42–48 (2012)
24. Caldwell, H., Smith, N.: Teaching Computing Unplugged in Primary Schools. SAGE Publications, Thousand Oaks (2016)
25. Carruthers, S., Gunion, K., Stege, U.: Computational biology unplugged! In: Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE 2009, p. 126 (2009)
26. Clarke, B.: Computer Science Teacher. British Computer Society, Swindon (2017)
27. Cohoon, J., Cohoon, J., Soffa, M.: Focusing high school teachers on attracting diverse students to computer science and engineering. In: ASEE/IEEE Frontiers in Education Conference, pp. 1–5 (2011)
28. Connor, R., Cutts, Q., Robertson, J.: Keeping the machinery in computing education. Commun. ACM **60**(11), 26–28 (2017)
29. Cortina, T.J.: Reaching a broader population of students through "unplugged" activities. Commun. ACM **58**(3), 25–27 (2015)
30. Curzon, P., McOwan, P.: Teaching formal methods using magic tricks. In: Fun with Formal Methods: Workshop at the 25th International Conference on Computer Aided Verification, number 122 (2013)
31. Curzon, P., McOwan, P.W.: Engaging with computer science through magic shows. In: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2008, pp. 179–183. ACM, New York (2008)
32. Curzon, P., McOwan, P.W., Plant, N., Meagher, L.R.: Introducing teachers to computational thinking using unplugged storytelling. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education - WiPSCE 2014, pp. 89–92 (2014)
33. Cutts, Q., Brown, M., Kemp, L.: Enthusing and informing potential computer science students and their teachers. ACM SIGCSE Bull. **39**, 196–200 (2007)
34. Dagienė, V., Stupurienė, G., Vinikienė, L.: Promoting inclusive informatics education through the Bebras Challenge to all K-12 students. In: Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 - CompSysTech 2016, pp. 407–414. ACM (2016)

35. de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-77974-2
36. Dodds, Z., Erlinger, M.: MyCS: building a middle-years CS curriculum. In: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2013, p. 330 (2013)
37. Dorling, M., White, D.: Scratch: a way to logo and python. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 191–196 (2015)
38. Duncan, C., Bell, T.: A pilot computer science and programming course for primary school students. In: WIPSCE 2015, pp. 39–48 (2015)
39. Duncan, C., Bell, T., Atlas, J.: What do the teachers think? Introducing computational thinking in the primary school curriculum. In: Proceedings of the Nineteenth Australasian Computing Education Conference, pp. 65–74 (2017)
40. Dwyer, H., Hill, C., Carpenter, S., Harlow, D., Franklin, D.: Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE 2014, pp. 511–516 (2014)
41. Dwyer, H.A., Boe, B., Hill, C., Franklin, D., Harlow, D.: Computational thinking for physics: programming models of physics phenomenon in elementary school. In: 2013 Physics Education Research Conference Proceedings, pp. 133–136 (2013)
42. Faber, H.H., Wierdsma, M.D.M., Doornbos, R.P., Van Der Ven, J.S., De Vette, K.: Teaching computational thinking to primary school students via unplugged programming lessons. J. Eur. Teach. Educ. Netw. **12**, 13–24 (2017)
43. Falkner, K., Vivian, R.: A review of computer science resources for learning and teaching with K-12 computing curricula: an Australian case study. Comput. Sci. Educ. **25**(4), 390–429 (2015)
44. Feaster, Y., Segars, L., Wahba, S.K., Hallstrom, J.O.: Teaching CS unplugged in the high school (with limited success). In: Rößling, G., Naps, T.L., Spannagel, C. (eds.) Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011, Darmstadt, Germany, 27–29 June 2011, pp. 248–252. ACM (2011)
45. Fellows, M., Parberry, I.: SIGACT trying to get children excited about CS. Comput. Res. News, 7 (1993)
46. Ferreira, J.F., Mendes, A.: The magic of algorithm design and analysis: teaching algorithmic skills using magic card tricks. In: Proceedings of the 19th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2014, pp. 75–80 (2014)
47. Ford, V., Siraj, A., Haynes, A., Brown, E.: Capture the flag unplugged. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE 2017, pp. 225–230 (2017)
48. Fowler, A.: Engaging young learners in making games. In: Proceedings of the International Conference on the Foundations of Digital Games - FDG 2017, pp. 1–5 (2017)
49. Gaio, A.: Programming for 3rd graders, scratch-based or unplugged? In: 10th Congress of European Research in Mathematics Education (CERME), TWG 22 (2017)
50. Gallenbacher, J.: Abenteuer informatik: hands-on exhibits for learning about computational thinking. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education - WiPSCE 2012, p. 149 (2012)

51. Garcia, D.D., Ginat, D.: Demystifying computing with magic, part III. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE 2016, pp. 158–159 (2016)
52. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
53. Goode, J., Margolis, J.: Exploring computer science. ACM Trans. Comput. Educ. **11**(2), 1–16 (2011)
54. Greenberg, R.I.: Educational magic tricks based on error-detection schemes. In: Proceedings of the 22nd ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2017, pp. 170–175 (2017)
55. Gross, J.L., Yellen, J., Zhang, P.: Handbook of Graph Theory, 2nd edn. Chapman and Hall/CRC Press, Boca Raton (2018)
56. Grover, S., Pea, R.: Computational thinking in K-12: a review of the state of the field. Educ. Res. **42**(1), 38–43 (2013)
57. Gujberova, M., Kalas, I.: Designing productive gradations of tasks in primary programming education. In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education, pp. 108–117. ACM (2013)
58. Gunion, K., Milford, T., Stege, U.: The paradigm recursion: is it more accessible when introduced in middle school? J. Probl. Solving **2**(2), 142–172 (2009)
59. Gutiérrez, J.M., Sanders, I.D.: Computer science education in Perú: a new kind of monster? ACM SIGCSE Bull. **41**(2), 86–89 (2009)
60. Hara, K.J.O., Burke, K., Ruggiero, D., Anderson, S.: Linking language and thinking with code: computing within a writing-intensive introduction to the liberal arts. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 269–274. ACM (2017)
61. Hazzan, O., Lapidot, T., Ragonis, N.: Guide to Teaching Computer Science: An Activity-Based Approach. Springer, London (2011). https://doi.org/10.1007/978-0-85729-443-2
62. Heintz, F., Mannila, L., Färnqvist, T.: A review of models for introducing computational thinking, computer science and computing in K-12 education. In: Proceedings - Frontiers in Education Conference (FIE), pp. 1–9 (2016)
63. Henderson, P.B.: Computing unplugged enrichment. ACM Inroads **2**(3), 24–25 (2011)
64. Hermans, F., Aivaloglou, E.: To scratch or not to scratch?: a controlled experiment comparing plugged first and unplugged first programming lessons. In: Proceedings of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE 2017, pp. 49–56. ACM, New York (2017)
65. Hromkovič, J., Keller, L., Komm, D., Serafini, G., Steffen, B.: Fehlerkorrigierende Codes - Ein Unterrichtsbeispiel zum gelenkten entdeckenden Lernen. LOG IN: Informatik Sch. Ausbildung **168**, 50–55 (2011)
66. Hromkovič, J., Lacher, R.: How to convince teachers to teach computer science even if informatics was never a part of their own studies. Bull. Eur. Assoc. Theor. Comput. Sci. (BEATCS) **123**, 120–125 (2017)
67. Hromkovič, J., Lacher, R.: The computer science way of thinking in human history and consequences for the design of computer science curricula. In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 3–11. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_1
68. Hubwieser, P., Mühling, A.: Playing PISA with bebras. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education, WiPSCE 2014, pp. 128–129. ACM, New York (2014)

69. Jašková, L.: Blind pupils begin to solve algorithmic problems. In: Diethelm, I., Mittermeir, R.T. (eds.) ISSEP 2013. LNCS, vol. 7780, pp. 68–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36617-8_6
70. Jašková, L., Kaliaková, M.: Programming microworlds for visually impaired pupils. In: Proceedings of the 3rd International Constructionism Conference, pp. 1–10 (2014)
71. Khoja, S., Wainwright, C., Brosing, J., Barlow, J.: Changing girls' attitudes towards computer science. J. Comput. Sci. Coll. **28**(1), 210–216 (2012)
72. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1998)
73. Koblitz, N.: Crypto galore! In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 39–50. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_3
74. Kocay, W., Kreher, D.L.: Graphs, Algorithms, and Optimization. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, Boca Raton (2005)
75. König, D.: Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe. Akademische Verlagsgesellschaft, Leipzig (1936)
76. Lau, W.W.F., Yuen, A.H.K.: Gender differences in learning styles: nurturing a gender and style sensitive computer science classroom. Australas. J. Educ. Technol. **26**(7), 1090–1103 (2010)
77. Linn, M.C., Eylon, B.S.: Science education: integrating views of learning and instruction. In: Alexander, P.A., Winne, P.H. (eds.) Handbook of Educational Psychology, pp. 511–544. Routledge, New York (2006)
78. Liu, J., Hasson, E.P., Barnett, Z.D., Zhang, P.: A survey on computer science K-12 outreach: teacher training programs. In: Proceedings - Frontiers in Education Conference, FIE, pp. 11–16 (2011)
79. Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A.: Is coding the way to go? In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 165–174. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_15
80. Manabe, H., Kanemune, S., Namiki, M., Nakano, Y.: CS unplugged assisted by digital materials for handicapped people at schools. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 82–93. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_8
81. Marghitu, D., et al.: Auburn university robo camp K12 inclusive outreach program: a three-step model of effective introducing middle school students to computer programming and robotics. In: Society for Information Technology and Teacher Education International Conference, Association for the Advancement of Computing in Education (AACE), pp. 58–63 (2013)
82. Marghitu, D., Zylla-Jones, E.: Educational and technological computer literacy program for typical and special needs children: an auburn university case study. In: Crawford, C.M., Carlsen, R., McFerrin, K., Price, J., Weber, R., Willis, D.A. (eds.) Proceedings of Society for Information Technology and Teacher Education International Conference 2006, Orlando, Florida, USA, Association for the Advancement of Computing in Education (AACE), pp. 2326–2331, March 2006
83. Mateti, P., Deo, N.: On algorithms for enumerating all circuits of a graph. SIAM J. Comput. **5**(1), 90–99 (1976)
84. Matoušek, J.: Construction of $\varepsilon$-nets. Discrete Comput. Geom. **5**, 427–448 (1990)
85. Meisters, G.H.: Polygons have ears. Am. Math. Mon. **82**(6), 648–651 (1975)

86. Morreale, P., Jimenez, L., Goski, C., Stewart-Gardiner, C.: Measuring the impact of computational thinking workshops on high school teachers. J. Comput. Sci. Coll. **27**(6), 151–157 (2012)
87. Morreale, P., Joiner, D.: Reaching future computer scientists. Commun. ACM **54**(4), 121 (2011)
88. Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., Kuno, Y.: A CS unplugged design pattern. In: Fitzgerald, S., Guzdial, M., Lewandowski, G., Wolfman, S.A. (eds.) Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009, Chattanooga, TN, USA, pp. 231–235. ACM (2009)
89. Overdorf, R., Lang, M.: Reaching out to aid in retention. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE 2011, p. 583 (2011)
90. Pokorny, K.L., White, N.: Computational thinking outreach: reaching across the K-12 curriculum. J. Comput. Sci. Coll. **27**(5), 234–242 (2012)
91. Prieto-Rodriguez, E., Berretta, R.: Digtial technology teachers' perceptions of computer science: it is not all about programming. In: Frontiers in Education Conference (FIE), pp. 1–5 (2014)
92. Rodriguez, B., Kennicutt, S., Rader, C., Camp, T.: Assessing computational thinking in CS unplugged activities. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2017, pp. 501–506. ACM, New York (2017)
93. Rodriguez, B., Rader, C., Camp, T.: Using student performance to assess CS unplugged activities in a classroom environment. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, pp. 95–100. ACM, New York (2016)
94. Rosamond, F., et al.: Reaching out to the media. Commun. ACM **54**(3), 113 (2011)
95. Schofield, E., Erlinger, M., Dodds, Z.: MyCS: a CS curriculum for middle-years students. J. Comput. Sci. Coll. **29**(4), 145–155 (2014)
96. Sentance, S., Csizmadia, A.: Computing in the curriculum: challenges and strategies from a teacher's perspective. Educ. Inf. Technol. **22**(2), 469–495 (2017)
97. Settle, A., et al.: Infusing computational thinking into the middle- and high-school curriculum. In: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE 2012, p. 22 (2012)
98. Sivilotti, P.A.G., Pike, S.M.: A collection of kinesthetic learning activities for a course on distributed computing. SIGACT News **38**(2), 57–74 (2007)
99. Smith, N., Allsop, Y., Caldwell, H., Hill, D., Dimitriadi, Y., Csizmadia, A.: Master teachers in computing: what have we achieved? In: Proceedings of the Workshop in Primary and Secondary Computing Education, pp. 21–24 (2015)
100. Stewart-Gardiner, C., Carmichael, G., Gee, E., Hopping, L.: Girls learning computer science principles with after school games. In: Proceedings of the Third Conference on GenderIT, pp. 62–63. ACM (2015)
101. Stockmeyer, L.: Planar 3-colorability is polynomial complete. ACM SIGACT News **5**(3), 19–25 (1973)
102. Sullivan, K., Byrne, J.R., Bresnihan, N., O'Sullivan, K., Tangney, B.: CodePlus - designing an after school computing programme for girls. In: 2015 IEEE Frontiers in Education Conference (FIE), pp. 1–5 (2015)
103. Sysło, M.M., Kwiatkowska, A.B.: Playing with computing at a children's university. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education - WiPSCE 2014, pp. 104–107. ACM Press, New York, November 2014

104. Taub, R., Armoni, M., Ben-Ari, M.: CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. Trans. Comput. Educ. **12**(2), 8:1–8:29 (2012)
105. Thies, R., Vahrenhold, J.: Reflections on outreach programs in CS classes: learning objectives for "unplugged" activities. In: SIGCSE 2012: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, pp. 487–492 (2012)
106. Thies, R., Vahrenhold, J.: On plugging "unplugged" into CS classes. In: SIGCSE 2013: Proceedings of the 44th ACM Technical Symposium on Computer Science Education, pp. 365–370 (2013)
107. Thies, R., Vahrenhold, J.: Back to school: computer science unplugged in the wild. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, pp. 118–123. ACM Press, New York (2016)
108. Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., Verno, A.: A model curriculum for K-12 computer science: final report of the ACM K-12 task force curriculum committee. ACM, New York (2003)
109. Vahrenhold, J.: Polygon triangulation. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, 2nd edn. Springer, New York (2015)
110. Vöcking, B., et al. (eds.): Algorithms Unplugged. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-15328-0
111. Voigt, J., Bell, T., Aspvall, B.: Competition-style programming problems for Computer Science Unplugged activities. In: Verdu, E., Lorenzo, R., Revilla, M., Regueras, L. (eds.) A New Learning Paradigm: Competition Supported by Technology, pp. 207–234. CEDETEL, Boecillo (2009)
112. Waite, J.: Pedagogy in teaching computer science in schools: a literature review, London (2017)
113. Weisstein, E.W.: Graph cycle. From MathWorld-A Wolfram Web Resource. http://mathworld.wolfram.com/GraphCycle.html
114. Wohl, B., Porter, B., Clinch, S.: Teaching computer science to 5–7 year-olds: an initial study with scratch, cubelets and unplugged computing. In: Proceedings of the Workshop in Primary and Secondary Computing Education, pp. 55–60 (2015)