Remember to run sudo texconfig pdftex paper letter if you want to produce letter size instead of a4 size... And switching back again...

From Theory to Practice: NP-completeness for Every CS Student

Pilu Crescenzi
Università degli Studi di Firenze
Dipartimento di Sistemi e Informatica
Viale Morgagni 65, 50134, Firenze, Italy

Emma Enström Viggo Kann
KTH Royal Institute of Technology
Schools of Computer Science and
Communication, Education and Communication
in Engineering Science, Stockholm, Sweden

ABSTRACT

NP-completeness is one of the most central concepts in computer science, and has been extensively applied in many diverse application areas. Despite this, students have problems grasping the concept and, more specifically, applying it to new problems. Independently, we have identified these problems at our universities in different countries and cultures. In an action research approach we have modified our courses and studied the effects. We here present some promising results. Our approach is mainly based on the idea of making more evident the fact that proving a new NP-completeness result is not at all different from designing a new algorithm. Based on this idea, we used tools typically used to teach algorithms (such as automatic program assessment and algorithm visualization systems), accompanied by other activities mainly devoted to augmenting the motivation to study computational complexity and forcing students to think and adopt a standpoint.

Categories and Subject Descriptors

F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Reducibility and completeness*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Algorithms, experimentation, theory

Keywords

Algorithm visualization, assessment, NP-completeness, self-efficacy

1. INTRODUCTION

In his 1997 invited talk at a theoretical computer science conference Christos Papadimitriou [20] observed how the notion of NP-completeness has become a pervasive and influential concept in many diverse disciplines, ranging from "statistics and artificial life to automatic control and nuclear engineering", and gave some reasons for this success. One of these is that NP-completeness is a "valuable intermediary between the abstraction of computational models and the reality of computational models." As a consequence, it is now a widely accepted fact that NP-completeness is a fundamental concept that "any CS professional should understand and be able to apply" [16].

On the other hand, it is also quite well-known that computational complexity is not easy to teach or to learn and, in general, the problem of presenting theoretical foundations of computer science in an integrated and motivating way has been studied for decades (e.g., [17]) and is still a rich research area for computer science education (e.g., [11]).

Motivation of the usefulness of a subject is important for learning. For example, Light et al. [15] write:

> In professional courses, the match between a student's understanding of what it means to be an engineer or a doctor and what the course seems to be providing can be crucial both for motivation and intellectual development.

So if the students have trouble seeing the usefulness of subjects like computational complexity they might have less incentive to learn.

We have seen many students who have trouble grasping the central ideas and concepts of computational complexity, where the most central concept is the reduction between problems — the transformation (in polynomial time) of the input of one problem into the input of another problem. In our experience, students have trouble with reductions in computational complexity on three levels: to come up with the idea for a reduction, to prove that the reduction is correct, and to describe what the implications of the existence of a reduction are. For instance, getting the direction of the reduction right in a proof is considered hard. Now, getting an idea for a reduction is very similar to getting an idea for any algorithm that we want to design. Proving it to be correct is connected to mathematical skills and knowledge of proof techniques. The implications of a reduction in a NP completeness proof, which characteristics of a problem are "transferred" or "preserved" during a reduction, is a separate set of facts and connections that need separate attention. The reduction is supposed to be used in a proof, but the motivation for doing so is the same for all NP completeness proofs. Many students do not see the purpose of designing a reduction in this context and end up constructing exhaustive search algorithms while working on their proofs.

Besides possible failures to expose the rich applications of reductions, one missing ingredient while teaching these concepts is that the concepts themselves are just another way of usefully applying the algorithmic way of thinking which

ITiCSE'13, July 1–3, 2013, Canterbury, GB.
ACM XXX-X-XXXXX-XXX-X/XX/XX.

is usually taught to all computer science students. In other words, a further motivation to learn NP-completeness is that designing reductions can be, in a certain sense, exactly the same as designing algorithms: a student enjoying this latter activity should also enjoy the former one.

If the very specific use of reductions in this context is causing the trouble, maybe we could pinpoint exactly that without simultaneously keeping students' attention on the algorithmic aspects. Previous research on reductions has found that when confronted with the task of constructing an algorithm based on a reduction, students tend to try reducing abstraction by "opening up the black box" [1, 2], which leads to algorithms for solving the original problem instead of reducing it.

Finally, if the hardness is due to lack of experience with proofs, this is a larger issue that possibly needs a cross-curricular approach and is better attacked by special courses like the ones described by [18]. Also the attempt to make reductions "a habit of mind" [2] might fit into that approach.

We are tackling these issues in our courses at our universities in Florence and in Stockholm respectively, by making the theoretical subject of computational complexity into something more concrete, inspired by action research. We decided to make use of two typical tools for teaching the design and the analysis of computer algorithms; an automated program assessment system (Kattis) and an algorithm visualization system (AlViE[1]). It is worth observing that even though NP-completeness is one of the concepts students typically struggle with, as far as we know this concept is not well covered by educational software: the only other experiences we are aware of are the ones described in [6, 21]. We also supported the use of the two tools with other activities mainly devoted to highlighting the usefulness of learning computational complexity and to forcing students to think and to adopt a standpoint. This paper describes the activities that we introduced in our courses and how the students responded to them.

2. THE COURSES

The experiment was performed during 2011 and 2012 as described below.

2.1 The Florentine course (TCS)

Theoretical Computer Science (TCS) is a third-year course of the Computer Science bachelor program at the University of Florence, , given by the first author. The prerequisites for this course are *Algorithms and Data Structures*, *Computer Architecture*, *Discrete Mathematics and Logic*, and *Programming*. The course is formed by three parts: a part devoted to the theory of computation (approximately, 36 hours), one devoted to the theory of formal languages (approximately, 24 hours), and the last one devoted to the theory of computational complexity (approximately, 12 hours). The topics covered during this latter part were the following: time complexity and the class P (2 hours), 2-satisfiability, polynomial-time reducibility, and 2-colorability (2 h), the maximum bipartite matching and the tower problem (2 h), the class NP, NP-completeness, and 3-satisfiability (2 h), 3-colorability, Hamiltonian path, and Subset sum (2 h), the Cook-Levin theorem (2 h), and the class EXP, the class PSPACE, and the Savitch theorem (2 h).

[1]http://alvie.algoritmica.org/

2.2 The Stockholm course (ADC)

Algorithms, Data structures and Complexity (ADC) is a compulsory third-year course in the 5-year Computer Science and Engineering program at KTH in Stockholm. The prerequisites for this course are the same as for the Florentine course.

ADC consists of 32 lectures (the first three two hours and the rest one hour each), given by the third author, 12 two hour tutorials in three groups, given by PhD and master students, and 4 compulsory computer labs. The assessment consists of two homeworks and a written theory exam. There is also an oral exam for students aspiring to get the highest grades.

The first 18 lectures, 7 tutorials and 3 computer labs covered construction and analysis of algorithms and data structures. The following lectures covered reductions (1 h), introduction to complexity (1 h), Turing machines and undecidability (2 h), Cook-Levin theorem (1 h), NP-reductions and NP-completeness (3 h), approximation algorithms and heuristics (3 h), other complexity classes (1 h). The tutorials in parallel covered reductions and undecidability (2 h), NP-reductions and NP-completeness (4 h), approximation algorithms (2 h), solution to the complexity homework (1 h), and complexity classes (1 h).

3. THE ACTIVITIES

We have developed several activities meant to improve the learning of computational complexity. Some of them were used before in one of the courses but not in both.

3.1 The usefulness of learning complexity

The main motivational problem could be that the students do not think that they will benefit from learning complexity, outside of the course. In order to get proof of the industrial usefulness of computational complexity, we sent the following questions to some former students now working in industry:

1. Describe a case where knowledge of computational complexity has helped you in your work.

2. Do you regard algorithmic knowledge and knowledge of complexity as a qualification when recruiting computer scientists?

We got several positive answers, leading to a 5 minute motivational part of the lecture where complexity was introduced in ADC. Three arguments were presented to the students: future courses, engineering and employability – at interviews at Vodaphone or Google, it is safe to assume that some questions will be about complexity. The following example of attacking an NP-hard real problem was given:

> At Racasse we developed the price comparison service RedElvis. It was to find the cheapest way to order books, music and video on the web, considering delivery terms, discounts etc. We showed that the complexity of the problem is too high. Therefore we implemented some heuristics instead of an optimal algorithm, and found that simulated annealing gave solutions which in every test were equal to or better than the best solution a human could obtain. [12]

3.2 Implementation of a reduction

Kattis is an automated programming assessment system. The typical way of using this system in teaching is described in [10]. The specific exercise that the students were presented with was a reduction task, where they could choose between two NP complete problems, and then produce code that reduces input for that problem to input for a new problem that was described in their instructions. During the process of solving the task, students had access to Kattis 24/7. Source code is submitted to Kattis, and the system runs secret test cases and interprets the output of the submitted code, and then reports the results to the students via email and/or a web interface. The feedback from Kattis for this problem consists of information on whether the solution was working or not, and in case it was not, whether a "yes" instance had been transformed to a "no" instance or vice versa and occasionally other hints on what could have happened. This is a slight change in the feedback compared to [9], where this particular exercise is described further.

3.2.1 Introduction to Kattis

The ADC students had already used Kattis in two labs when starting to work on the NP reduction lab. The TCS students had to be introduced in two steps in order to become acquainted with the system itself and with the way it specifies the input data. This was done by them first submitting the tutorial problem `Hello World!`, and then a solution to the sorting problem, at the very beginning of the complexity section of the course.

3.2.2 The reduction computer lab

The students were given the text of the laboratory exercise. The text also contained six theory questions that the students were asked to answer before solving the exercise.

After the first 10-12 hours of the complexity parts of each course, the students were asked to perform a peer review of the answers given to the theory questions included in the laboratory exercise text. This activity took 15 minutes.

During the next week, the students had to submit to the Kattis system their solution of the laboratory exercise. In the ADC course, the students also had to discuss their solution with a TA during the scheduled computer lab hours.

3.3 Visualizations of reductions

As stated in [7], a reduction is, for all purposes, an algorithm transforming instances of a starting problem into instances of a target problem: it is then natural to use algorithm visualization techniques while teaching reductions. In both courses we used the AlViE system to present the visualizations of two reductions in the following way.

Two of the three reductions from 3-satisfiability to Subset sum, 3-colorability and Vertex cover (inspired by [14, 23]) were selected and presented theoretically in a lecture. During the reduction explanation, the teacher made use of the visualizations, which were successively made available on the AlViE web site, so that the students could experiment themselves. The two visualizations not only show how the starting instance x is transformed into the target instance x', but they also show how a solution of x can be transformed into a solution of x'.

3.4 Low budget clickers at lectures

The advantages with clicker questions are well-known, especially for teaching physics [8]. For example, they activate the students by forcing every student to think and adopt a standpoint, they reveal misconceptions immediately both to the teacher and to each student, and the result is often interesting to discuss: why did some/many students answer differently than the rest? A goal is to reveal and remedy misconceptions as early as possible in the learning process.

At KTH, there is no clicker system, so we constructed our own low-budget system. We divided coloured (yellow and blue) A4 sheets into four cards and distributed one yellow and one blue card to each student at the beginning of the lecture. During the lecture, questions with two choices, marked yellow and blue, were presented. Every student was supposed to answer by showing either the yellow or blue card. Then the teacher summarized the result in percentage and discussed it.

The first year, the clicker questions were introduced in an algorithm lecture before the complexity lectures. At the end of the lecture, we asked (using the cards) whether we should continue to ask such questions. Everyone answered yes! We then used clicker questions in most complexity lectures to reveal and remedy misconceptions on undecidability and reductions. Some questions required discussion among the students and some questions required fast response. The second year clicker questions, with an addition of red cards, were used throughout the whole course.

A, B, C, D and E are decision problems. Suppose that B is NP-complete and that there are polynomial-time Karp reductions between the problems in the following way:

$$E \leftarrow A \rightarrow B \rightarrow C$$
$$B \updownarrow D$$

What will we know about the complexity of A, C, D, E? Mark with a cross each square that corresponds to something we know for sure.

	in NP	NP-complete	NP-hard
A			
C			
D			
E			

Figure 1: The NP-reducibility assignment(ADC).

3.5 The NP-reducibility assignment

We have introduced a new type of complexity assignment, which we have used both in the instruction and assessment, see Figure 1. The assignment hides all details about the involved problems and makes it impossible to tamper with the contents of the black box. Just asking the students this type of question, that they normally are not asked, directs their attention to the fact that this is important in itself. We had not used this type of question before, but it was appreciated by both students and teaching assistants.

4. EVALUATION

We have evaluated the new activities in both courses (TCS and ADC) in several ways, described below.

4.1 The evaluation surveys

At the end of the courses, all students were asked to fill in a short questionnaire concerning the new activities and the usefulness of the teaching material used during the last part of the course.

4.2 The self-efficacy surveys

The term "self-efficacy" was introduced by Albert Bandura in the 1970s and is further described by him later, in [4]. It refers to an individual's confidence in his or her own ability to perform actions in order to achieve some desired outcome. The score on a self-efficacy test is known to be an important predictor of success [19]. However, self-efficacy beliefs are not static, but something that changes with the individual's experiences. Therefore, self-efficacy has long been used in education. There are studies in how self-efficacy correlates with performance [13], how self-efficacy changes during studies [22], and how it correlates with other factors around the individual [3]. For our purpose, we know of no established self-efficacy measuring instrument for theoretic computer science. There are, however, instruments in mathematics [13, 19] and for programming [22, 3]. With these as examples, and guided by [5], we have constructed an 8 items long self efficacy score where we have asked about the abilities that students judge themselves as having within the ADC course complexity contents. The same questions were distributed at the first lecture and after the homework on complexity 2012. The students could grade their self-efficacy for each item on a scale between 0 and 100. Our goal was to compare the change in self-efficacy scores between the two occasions between the active and less active students. The magnitude of the changes that would be considered was 25 steps, as this seems large enough to be a non-random size of a change. The explained marks along the 100 step scale were located at 25, 50, 75 and 100, respectively.

4.3 Results of the assessments

It is very hard to show that a change in a course has a positive effect by comparing assessment results, since there are so many variables involved. We do not believe in denying some of the students access to activities that we think are beneficial to them, so we cannot organize control groups. However, we know which activities the ADC students attended. Thus we can compare the number of activities attended to the student's performance on the two homework assignments.

5. RESULTS

Since both courses and evaluation methods differ between our schools, we report the results for each school separately.

5.1 The Florentine experiment

The experiment took place in May and June 2011, with twelve participating students. According to statistics over their grade on previous partial exams, the student sample was mostly formed by medium/high level students.

The survey contained four questions concerning the usefulness of the lectures, the lecture notes, the algorithm reduction visualization, and the reduction computer lab. The questions were to be answered with a score between 0 (not useful at all) and 4 (very useful). The results are summarized in Table 1.

Score	0	1	2	3	4
Lectures	0	0	1	3	8
Notes	0	2	1	4	5
Visualization	0	0	1	5	6
Computer lab	1	4	3	4	0

Table 1: Usefulness of the activity (Florence).

5.2 The Stockholm experiment

The ADC course ran from September to December 2011 and 2012, and the experiment took place in October and November each year with about 140 students in 2011 and 150 students in 2012.

5.2.1 The survey results

At the final written theory exam, both years, each student received one of two evaluation surveys. The first one was an open question survey with questions about the pedagogical purpose of each activity and whether this purpose was fulfilled. The second survey, analysed here, consisted of closed questions on the meaningfulness and usefulness of the activities. In 2011, all students who got this survey answered it except two students, a total of 59. In 2012, all but one answered the survey, a total of 70. The results are summarized in Table 2. Each question also had a "don't know" alternative, which is not presented in the summary.

The largest difference between the two years is in the answers about the motivational lecture, where more students were inclined to answer "don't know" the second year. Afterwards, many students expressed confusion over the term "motivational lecture", which might have been the reason for this. Generally, when students were less positive in year 2, they chose "don't know".

In 2012 we also asked the students where they learned what (multiple choice), see Table 3. This table does not entirely evaluate the same phenomena as the survey, and should not be used to decide what activities should be part of the course. For instance, the students did not perceive that the visualizations had contributed much to their present knowledge on the items we asked about here, yet 33% of the same students stated that they had learned complexity from the visualizations, and that these were meaningful and contributed to learning. Those answers were given later and in retrospect, while the results in Table 3 were supplied together with the homework assignment. Apart from the visualizations, all activities were considered contributing to learning.

5.2.2 Comparison to assessments

In the ADC course, there are two homeworks assessing the problem solving proficiency, the first is on algorithm construction and the second one is on computational complexity. Both homeworks are graded from A to F, each contributing a third to the final grade.

One of the main reasons for changing the course was to improve the performance ratio of the first time students at the complexity homework, that is, the share of the students passing the homework the first time it is given. The year before the project started the performance ratio was 73%. The first year of the project the ratio was the same, but the second year it was improved to 87%.

1. Was the pedagogical purpose of the activity clear?

	yes	questionable	no
motivational lecture	86/23%	11/16%	4/4%
clicker questions	92/90%	8/6%	0/1%
reduction visualizations	80/81%	16/10%	3/1%
reduction computer lab	90/81%	6/10%	4/0%

2. Did you find the activity meaningful?

	yes very	yes some-what	not parti-cularly	not at all
motivational lecture	21/10	61/17	7/10	
clicker questions	38/59	44/36	13/3	4/0
reduction visualizations	28/40	38/40	26/10	8/1
reduction computer lab	65/67	35/23	0/1	

3. Did you learn some computational complexity by working with the activity?

	yes	no
clicker questions	50/69%	40/11%
reduction visualizations	45/33%	34/34%
reduction computer lab	94/86%	4/6%

4. Do you think that activities like this one can make it easier to learn computational complexity?

	yes	no
clicker questions	69/73%	19/9%
reduction visualizations	95/69%	5/7%
reduction computer lab	96/89%	2/3%

5. Did the activity add something to the course?

	yes	no
motivational lecture	75/27%	4/13%
clicker questions	83/94%	6/1%
reduction visualizations	76/71%	13/6%
reduction computer lab	94/91%	2/1%

Table 2: Activity survey results (KTH). Answers are presented as *first year/second year* percentages.

Where did you learn to...
1. ... tell if a decision problem is in NP?
2. ... describe the principles for an NP completeness proof?
3. ... choose a suitable NP-complete problem to reduce?
4. ... construct a reduction between given problems?
5. ... prove correctness of an NP reduction?

	already knew or learned by myself	lectures	tutorial sessions	visualizations	lab theory assignments	reduction lab	homework 2	still haven't learned this
1.	20%	59%	37%	2%	43%	39%	43%	0%
2.	14%	59%	44%	1%	30%	34%	36%	0%
3.	17%	46%	36%	1%	14%	28%	51%	3%
4.	14%	41%	36%	0%	24%	57%	52%	1%
5.	16%	34%	32%	0%	13%	32%	43%	5%

Table 3: Where different tasks were perceived to have been learned. (N=148)

	Activities attended	hw1 grade <hw2 grade	mean grade hw1	mean grade hw2
year 1	>4	41%	3.2	3.6
	>2 and ≤ 4	36%	3.3	2.9
	≤ 2	20%	2.4	1.9
year 2	>4	49%	2.0	2.7
	>2 and ≤ 4	34%	1.7	2.1
	≤ 2	11%	1.7	2.0

Table 4: Comparison of student performances at homework 1 and 2 (hw1 and hw2) depending on the number of new activities attended (ADC).

We want to compare the results on the complexity homework with the number of attended activities, where the NP reduction computer lab is counted as 1.5 if submitted early. Of the students failing the complexity homework, only a fourth had attended three or more of the six activities.

We have also studied differences in the performances of the students between the two homeworks: 114 (136 in the second year) students handed in both homeworks, 34 (57, respectively) students received a better grade in the second homework (complexity) than in the first (algorithms), 46/10 students received a better grade in the first homework than in the second, and 34/69 students received the same grades. During the first year, the two homeworks had about the same mean grades, 2.8 and 2.6, in a linear scale where A is 5 and F is 0. The second year one of the assignments of the first (algorithms) homework was considered harder, so the mean grades became 1.9 and 2.5.

Our hypothesis was that students attending many activities should improve their grades to greater extent than students not attending the activities. The results, supporting the hypothesis, can be found in Table 4. For the first year we do not have *full* information about the activity attendance from all students, a few students who should rightly be in the middle group (between 2 and 4 activities) might have been counted in the lowest row (≤2) in the table. However, this will not affect the overall picture. For the second year we have full attendancy information. Also, we know for the second year that the correlation between the attendancy of the algorithm and complexity parts of the course is high.

Of the students (44 first, 86 second year) who attended almost all (more than 4) of the activities, almost half *improved* their grade on the complexity homework. In the two groups of students attending fewer activities, the number improving their grade is considerably lower. Also, the mean grade is improved most in the top group. This is an indication that the activities had a positive effect. Another explanation would be that students benefit more from teaching and activities in complexity than they do in algorithms.

As for the self-efficacy, only 35 students completed both surveys, and only 4 out of these had attended three or fewer activities. Hence, comparing their answers to the majority's would not have been feasible. The total results are presented in Table 5. The items 4, 5, 6 and 8 relate closely to the three difficulties identified by us, and the average score is lower on these. Item 2 had a high starting score, and has therefore not increased much. Most students increased their self-efficacy during the period, but there were three occurrences of decreased self-efficacy, one for the item "I could determine that

threshold\item	1	2	3	4	5	6	7	8
25	32	25	31	20	24	28	30	27
50	19	14	24	8	18	16	23	18

Table 5: Number of students increasing their self-efficacy values for items 1–8. (N=35)

a NP completeness proof is correct", and two for the item "I could determine in what direction a reduction should go in order to be able to use it positively or negatively" (that is, for problem solving purposes or for impossibility proofs respectively.) For this item, the self-efficacy still decreased when the threshold for changes was set to 50. Among the other students, 20 increased their self-efficacy on this item with at least 25 and 8 of them with at least 50. This was the item with lowest frequency of increased self-efficacy beliefs. It is worth noticing, that the teaching assistants who were grading the homework and the oral exam where this assignment was presented, reported that this year (unlike all other years) there were no occurrences of reductions in the wrong direction. The students did not increase their beliefs in their own abilities on this one as much as on the other items, but they performed better than students had done before with respect to this particular item. There has been no in-depth statistical analysis on the results of these tests, and the internal reliability of the instrument has not been evaluated. The results are only used as reported above.

6. DISCUSSION

In order to connect all three parts of reductions that students find especially hard, (coming up with an idea, prove correctness and understand the implications of a reduction), we believe that we should both teach each item separately, and show how the parts relate to each other and to other parts of the students' knowledge, for instance by showing the connections to algorithm construction. If a student has the preconception that everything in the course will be about algorithms, an exhaustive search would seem just as good as any other algorithm for any purpose. On the other hand, if a student clearly likes designing algorithms, it ought to be relatively easy to learn what specific requirements are always posed for reductions in the context of proving NP completeness. By learning that there are such requirements and by learning them, in a context free from details about the involved problems, the student could later feel more comfortable in designing reductions.

In this study we have mainly addressed the students' motivation and the implications of reductions in NP completeness proofs. The assignment type in Figure 1, which was not considered especially difficult on the exam, shows that this was maybe not that hard after all. The fact that had been at the core of many difficulties was not difficult in itself, after getting proper attention in teaching. The disposition to reduce abstraction mentioned by [1, 2] also could not affect the students' thinking here, since nothing is known about the problems involved. It is still unknown if this specific difficulty also disappears in more complicated tasks.

We found that the students liked the new activities and that most of them thought that the activities helped them to learn computational complexity. It is hard to prove that more students now will pass the exam, but the statistics indicate that the students who attended most of the activities improved their grade by doing this.

It is hard for many students to understand and show correctness of reductions, at least in Florence and Stockholm. We think that these problems are universal in complexity learning, and some of the concepts could be considered as threshold concepts. In our next study, we will address correctness proofs and pseudo code. Later we will investigate possible threshold concepts in computational complexity.

Finally, we would like to emphasize that our cooperative course development model, where we exchanged activities and discussed problems and solutions over university and country borders, was very successful and satisfying.

7. REFERENCES

[1] Armoni, M. (2008). Reductive thinking in a quantitative perspective: the case of the algorithm course. *Proc. ITiCSE '08*, 53–57.

[2] Armoni, M., Gal-Ezer, J., and Hazzan, O. (2006). Reductive thinking in undergraduate CS courses. *Proc. ITiCSE '06*, 133–137.

[3] Askar, P. and Davenport, D. (2009). An investigation of factors related to self-efficacy for Java programming among engineering students. *Turkish Online J. Educational Tech.*, 8:26–32.

[4] Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory.* Prentice-Hall series in social learning theory. Prentice-Hall, Englewood Cliffs, New Jersey.

[5] Bandura, A. (2006). *Self-Efficacy Beliefs of Adolescents*, ch. 14: Guide forconstructing self-efficacy scales, pages 307–337. Adolescence and Education. Information Age Publishing.

[6] Brändle, M.A. (2006). GraphBench: Exploring the Limits of Complexity with Educational Software. Ph.D. Thesis, ETH.

[7] Crescenzi, P. (2010). Using AVs to Explain NP-completeness. *Proc. ITiCSE'10*, 299.

[8] Duncan, D. (2006). Clickers: A new technology with exceptional promise. *Astronomy Education Review*, **5**(1), 70–88.

[9] Enström, E. and Kann, V. (2010). Computer lab work on theory. *Proc. ITiCSE '10*, 93–97.

[10] Enström, E., Kreitz, G., Niemelä, F., Söderman, P. and Kann, V. (2011). Five years with Kattis – Using an automated assessment system in teaching. *Proc. FIE '11.*

[11] Goldreich, O. (2006). On Teaching the Basics of Complexity Theory. *Essays in Memory of Shimon Even*, 348–374.

[12] Grundin, G. (2011). Personal communication.

[13] Iannone, P. and Inglis, M. (2010). Self efficacy and mathematical proof: are undergraduate students good at assessing their own proof production ability? *Proc. 13th Conf. on Research in Undergraduate Math. Education.*

[14] Kleinberg, J. and Tardos, E. (2006). Algorithm Design. Addison Wesley.

[15] Light, G., Calkins, S., and Cox, R. (2009) Learning and Teaching in Higher Education: The Reflective Professional. SAGE Publications Ltd.

[16] Lobo, A.F. and Baliga, G.R. (2006). NP-completeness for All Computer Science Undergraduates: A Novel Project-based Curriculum. *J. Comput. Small Coll.*, **21**(6), 1937–4771.

[17] Mandrioli, D. (1982). On Teaching Theoretical Foundations of Computer Science. *SIGACT News*, **14**(4), 58–69.

[18] Muller, O., Rubinstein, A. (2011). Work in Progress - Courses Dedicated to the Development of Logical and Algorithmic Reasoning. *Proc. FIE'11.*

[19] Pajares, F. and Miller, M.D. (1994). Role of Self-Efficacy and Self-Concept Beliefs in Mathematical Problem Solving: A Path Analysis. *J. Educational Psychology*, 86(2):193–203.

[20] Papadimitriou, C.H. (1997). NP-Completeness: A Retrospective. *Proc. ICALP'97*, 2–6.

[21] Pape, C. (1998). Using Interactive Visualization for Teaching the Theory of NP-completeness. *Proc. ED-MEDIA/ED-TELECOM*, pages 1070–1075.

[22] Ramalingan, V. and Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *J. Educational Computing Research*, 19(4):367–381.

[23] Sipser, M. (2006). Introduction to the Theory of Computation. Thomson.