Preliminär rapport om utveckling av komplexitetsdelen av ADK, januari 2012

Pilu Crescenzi Emma Enström Viggo Kann

ABSTRACT

NP-completeness is one of the most central concepts in computer science, which has been applied in many diverse application areas. Despite this, students have problems grasping the concept and, more specifically, applying it to new problems. In this paper, we describe the approach we used while teaching NP-completeness in our courses and we present some very promising preliminary 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. On the ground of this idea, we used tools typically applied to the teaching of 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 take stand.

1. INTRODUCTION

In his 1997 invited talk at a theoretical computer science conference [17], Christos Papadimitriou 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 very interesting and valuable reasons for this success. One of these reasons is that NP-completeness is a "valuable intermediary between the abstraction of computational models and the reality of computational models". As a consequence, it now is a widely accepted fact that NP-completeness is a fundamental concept that "any CS professional should understand and be able to apply" [14].

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 since several decades ago (see, for example, [15]) and it is still an interesting and valuable research area in computer science education (see, for example, [10]).

Motivation of the usefulness of a subject is important for learning [13]. For example, Light et al 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 be less motivated to learn.

We have seen that many students have trouble grasping the central ideas and concepts of computational complexity, whose 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 the context of a NP completeness proof, what characteristics of a problem is "transferred" or "preserved" during a reduction, is a somewhat separate set of facts and connections that needs 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. Without seeing that this is the purpose of designing a reduction in this context, many students end up constructing total search algorithms while working on their proofs. Either the students are not motivated enough, or the structure of the teaching is bad.

If the perceived hardness of reductions is not due to the lack of application, it might be that 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 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 efficient algorithms: a student enjoying this latter activity should also enjoy the former one.

If the hardness is due to the fact that there is a very specific use for reductions in this context, maybe we could pinpoint exactly that without simultaneously keeping students' attention on the algorithmic aspects. Previous research on reductions have found that when confronted with the task to construct an algorithm based on a reduction, students tend to try reducing abstraction by "opening up the black box" [3, 4] , involving themselves with the details of the problem that should be reduced instead of the properties of the reduction, which easily leads to algorithms for solving the original problem instead of reducing it.

Finally, if the hardness is due to their experience with proofs, knowing what we *always* want to prove for an NP reduction might help, but this is a larger issue that possibly needs an cross-curricular approach and is better attacked by special courses like the ones described by [16]. Also the attempt to make reductions "a habit of mind" [4] might fit into that approach. In order to implement this strategy for improving the motivation to learn NP-completeness, we decided to try to improve the teaching and the learning of complexity in our courses in two different countries, referenced as School 1 and School 2 respectively, by making the theoretical subject of computational complexity into something more concrete. We decided to make use of two typical tools for teaching the design and the analysis of computer algorithms; an automated program assessment system and an algorithm visualization system (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 two experiences we are aware of are the ones described in [5, 18]). We also supported the use of these two tools with other activities mainly devoted to highlighting the usefulness of learning computational complexity and to forcing students to think and to take stand. This paper describes these activities that we introduced in our courses and how the students responded to them.

2. THE COURSES

The experiment was performed during 2011 in the courses described below.

2.1 The School 1 course (TCS)

Theoretical Computer Science (TCS) is a third-year course of the Computer Science bachelor program at School 1, 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).

2.2 The School 2 course (ADC)

Algorithms, Data structures and Complexity (ADC) is a compulsory third-year course in the 5-year Computer Science and Engineering program at School 2. The prerequisites for this course are the same as for the School 1 course.

In 2011 ADC consisted of 32 lectures (the first three two hours and the rest one hour), given by the third author, 12 two hour tutorials in three groups, given by PhD and master students, and 12 computer lab sessions. Besides the computer labs the assessment consisted of two homeworks and a written theory exam. There was also an optional oral exam for students aspiring to get the highest grades.

> Why do you need to learn computational complexity?
>
> 1. Prerequisite to advanced courses (e.g. cryptography).
> 2. Will help you to attack hard real problems as an engineer.
> 3. Makes you more attractive to several companies.

Figure 1: Arguments for learning complexity.

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 (2 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 don't think that they will benefit anything 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 merit 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, see Figure 1. An example of attacking hard real problems (second argument) came from Gustav Grundin:

> 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.

Employability arguments were also given: at interviews at Vodaphone or Google it is safe to assume that some questions will be about complexity.

3.2 Visualizations of reductions

As stated in [6], a reduction is for all purposes an algorithm, which transforms 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 [1] 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 [12, 19]) 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'.

3.3 Implementation of a reduction

Kattis is an automated program assessment system. The typical way of using this system in teaching is described in [9]. The specific exercise that the students were presented with was a reduction task, where they could choose between inputs for two NP complete problems, and then reduce it to input for a new problem that was described in their instructions. The task is to produce source code that achieves this goal, and to be able to explain why it works. During the process of solving the task, students had access to Kattis, and could make use of it in any way preferred. 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, sometimes small hints on what could have happened, and always information on whether a "yes" instance had been transformed to a "no" instance or vice versa. This is a slight change in the feedback compared to [8], where this particular exercise is described further.

3.3.1 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.

After the first 2 hours, the TCS students were asked to submit a first very simple program, that is, a program printing the message `Hello World!`. After the first 6 hours, the students were also asked to solve the sorting problem and to submit their solution to the Kattis system.

3.3.2 The reduction computer lab 1 – Peer review

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 12 hours of the complexity part of TCS (10 hours of ADC), 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.

3.3.3 The reduction computer lab 2 – Submission

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.

3.4 Low budget clickers at lectures

The advantages with clicker questions are well-known, especially for teaching physics [7]. For example, they activate the students by forcing every student to think and take stand, they reveal misconceptions immediately both to the teacher and each student, and the result is often interesting to discuss: why did some/many students answer different than the rest? A goal is to reveal and remedy misconceptions as early as possible in the learning process.

At School 2 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 answer alternatives, 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.

In the course 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.

4. EVALUATION

We have evaluated the new activities in both courses in several ways, described below. Unfortunately we were not able to organize control groups, but in the ADC course we as a part of the post-test asked the students which of the new activities they had attended.

4.1 The pre-test

After the first 6 hours of TCS (3 hours of ADC), the students were asked to take a pre-test on the notion of polynomial-time reducibility. In TCS they were asked to prove that two problems belong to the class P by making use of the reduction technique. The students had 30 minutes to solve these two problems. In ADC the students had 20 minutes to solve one NP-reducibility assignment, see Figure 2, and to prove a given NP-reduction from Subset Sum to Partition to be correct. The assignment in Figure 2 is solely about what we know of the problems involved in a reduction, and does not involve what the reduction looks like. 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 it self. We had not used this type of question before, but it was appreciated by both students and teaching assistants.

4.2 The post-test

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:

$$\begin{tikzcd} E & A \arrow[l] \arrow[r] & B \arrow[r] \arrow[d, leftrightarrow] & C \\ & & D & \end{tikzcd}$$

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 2: Assignment 1 from the pre-test (ADC)

At the end of the course, the students were asked to take a post-test on the notion of polynomial-time reducibility. In TCS they were asked to prove that two problems are NP-complete and that one problem belongs to the class P by making use of the reduction technique. The students had 30 minutes to solve these three problems. In ADC the post-test was similar to the pre-test, 20 minutes with a first assignment of the same type as Figure 2 and a second assignment to prove a given NP-reduction from Clique to Vertex cover to be correct.

4.3 The evaluation surveys

At the end of the courses, the 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.

4.4 Comparison to 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 other variables involved.However, we have tried to do this in the School 2 experiment. Since we have at least partial information about which new activities each student attended we can compare the number of activities to the student's performance on the two homework assignments of the ADC course.

5. RESULTS

We will report the results of the evaluations separately for each course.

5.1 The School 1 experiment

The experiment took part 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.

5.1.1 The exercise results

The pre- and post-tests were each assigned grades between 0 and 6 by the teacher. The grade of the peer review (which was assigned by the students themselves) was between 0 and 3. Finally, the result for the reduction computer lab

Student	Pre-test	Peer review	Computer lab	Post-test
s_1	5	3	`Accepted`	2
s_2	5	3	`Rejected`	0
s_3	3	3	`Rejected`	3
s_4	2	2	`Accepted`	0
s_5	0	1	`Accepted`	0
s_6	1	3	`Accepted`	2
s_7	4	3	Not submitted	2
s_8	3	3	`Accepted`	3
s_9	6	2	`Accepted`	6
s_{10}	4	3	`Accepted`	3
s_{11}	5	2	`Rejected`	5
s_{12}	0	2	`Accepted`	1

Table 1: The grades obtained by the students in all the activities (School 1)

either `Accepted` or `Rejected` (in this latter case, there was also some sort of justification for the rejection). The grades obtained by the students in all the activities are summarized in Table 1. It seems that the activities performed between the pre-test and the post-test did not cause an improvement in the obtained grade.

5.1.2 The survey results

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 the following table.

	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

5.2 The School 2 experiment

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

5.2.1 The exercise results

At the (non-mandatory) tutorial sessions, students were asked to take the pre-test and the post-test. They did not have to complete it, and they could choose whether to put their names on the paper. The tests were for research purposes only and did not count to their grade. 47 students were identified as having participated in both, using their real names or pseudonyms. At the post-test, the students were asked to what extent they had been participating in the different activities investigated here. 73 students filled in the post-test, out of which the majority (67 of them) responded with their names which makes it possible to compare their results on the rest of the course with their answers.

For the non-anonymous students, the average score improved by a bit over two points on the first task, and with less than one on the second. The distribution of grades looked the same for the second task on both tests. Also, several students got the highest score on the second task of the pre-test by showing that the reduction in the second task was polynomial, whereas no one got credits for this at the post-test.

5.2.2 *The survey results*

At the final written theory exam 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 result of this survey will be analyzed in a future paper. The second survey consisted of closed questions on the meaningfulness and usefulness of the activities. All students who got this survey answered it except two students who had not taken part in any of the activities, a total of 59.

The number of students in the survey who had attended each activity was: motivational lecture: 28, clicker questions: 52, reduction visualizations: 38, reduction computer lab: 52. Each row below shows the ratio for the students participating in that activity. Each question also had a "don't know" alternative, which is not presented in the summary.

1. Was the pedagogical purpose of the activity clear?

	yes	questionable	no
motivational lecture	86%	11%	4%
clicker questions	92%	8%	
reduction visualizations	80%	16%	3%
reduction computer lab	90%	6%	4%

2. Did you find the activity meaningful?

	yes very	yes somewhat	not especially	not at all
motivational lecture	21%	61%	7%	
clicker questions	38%	44%	13%	4%
reduction visualizations	28%	38%	26%	8%
reduction computer lab	65%	35%		

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

	yes	no
clicker questions	50%	40%
reduction visualizations	45%	34%
reduction computer lab	94%	4%

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

	yes	no
clicker questions	69%	19%
reduction visualizations	95%	5%
reduction computer lab	96%	2%

5. Did the activity add something to the course?

	yes	no
motivational lecture	75%	4%
clicker questions	83%	6%
reduction visualizations	76%	13%
reduction computer lab	94%	2%

Even if almost all students thought that they had learned complexity in the NP reduction computer lab, both students who did not instead thought that they had learned complexity by both clicker questions and the NP reduction visualizations. This shows that it is useful to combine different types of activities, since students are different and learn in different ways.

Activities attended	hw1 grade >hw2 grade	hw1 grade <hw2 grade	mean grade hw1	mean grade hw2
>4	10	18	3.2	3.6
2 to 4	18	16	2.8	2.9
<2	18	1	1.7	0.6

Table 2: Number of students with different performance between homeworks 1 and 2 (hw1 and hw2) depending on the number of new activities attended (ADC)

5.2.3 *Comparison to assessments*

In the ADC course there are two homeworks, the first is on algorithm construction and the second one is on complexity. Both homeworks are graded from A to F and are counted equally for the final grade.

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. Like in School 1, we could not see that students more successfully solved their tasks with the activities than without them. 23 out of 120 students failed on the complexity homework (the remaining students did not hand it in), and also when looking at the group that attended more than two activities, around 10 % failed.

We have also studied differences in the performances of the students between the two homeworks. 113 students handed in both homeworks. 35 students received a better grade in the second homework (complexity) than in the first (algorithms), 46 students received a better grade in the first homework than in the second, and 32 students received the same grades. The two homeworks had the same mean grades 2.6, in a linear scale where A is 5 and F is 0.

Our hypothesis was that students attending many activities should improve their grades to greater extent than students not attending the activities. The results, which are supporting the hypothesis, can be found in Table 2. Since we don't have *full* information about the activity attendance from all students, a few students who should rightly be in the middle group (2 to 4 activities) might have been counted in the lowest row (<2) in the table. However, this won't affect the overall picture.

Of the 44 students who attended almost all (more than 4) of the activities there were 8 (18%) more who *improved* their grade on the complexity homework than got lower grade. Of the 20 students who attended at most one activity there were 17 (85%) more who got *lower* grade on the complexity homework than improved their grade. Of the 49 students who attended between 2 and 4 activities about the same number got higher grade, lower grade and the same grade. This is a clear indication that the activities had a positive effect.

6. DISCUSSION

Why wasn't the post-test results much better than the pre-test results? Of course it is possible that the tests were not valid for the abilities we wanted to trace, or that their respective reliability differed. The pre-test might have been easier than the post-test for both courses. For the School 1 course an explanation could be that the students were already quite well acquainted with the notion of reducibility because of the first part of the course, which was devoted to the theory of computation.

At School 2, the fact that the students' results on the post-test only improved for the question type in Figure 2 also might have another explanation: Maybe students actually were not able to make use of their knowledge in a more applied setting. If the tasks were equally difficult, the type of knowledge tested in the second assignment was not developed during the course. Proving the correctness of a given reduction might require students to have reached a higher level in Bloom's taxonomy [2] than they in fact did. Another possibility is that the task is of a practical type rather than theoretical — in order to prove reductions being concrete, the student might need to acquire practical skills, that were not practiced during the activities of the course. In either case, we need to come up with more accurate ways of teaching and examining exactly these skills as we seem to have done by separating the issue of the first assignment from its context in order for the students to be able to acknowledge that it exists. Another explanation could simply be that 20 minutes was too short time for the students to solve both problems!

When it comes to mentioning that the reduction should be polynomial, the students performed worse at the post-test. On the pre-test, students got descriptions of all conditions that a reduction must satisfy, as they had not seen them before at that time, and students had no pre-conception on what to demand of a reduction, whereas on the post-test, they got no hints. Students might during the course have formed conceptions about what is generally difficult, forgetting to check "easier" parts.

That the assignment type in Figure 2 was not considered especially difficult on the exam, shows that this was maybe not that hard after all. The thing that had been the core of many difficulties was not difficult in itself, after getting the appropriate attention in teaching. It remains to investigate if this specific difficulty actually disappears also in more complicated tasks. The disposition to reduce abstraction mentioned by [3, 4] also could not affect the students' thinking about this exercise, since they knew nothing about the problems involved.

In order to connect the three parts of reductions that students find especially hard, we believe that we both need to separate the different parts and direct attention to each independently of the others, but also that we need to show how they relate to each other and to other parts of the courses (which can be extended to other parts of the students' knowledge.) If a student has the preconception that everything in the course will be about algorithms, the parts about reductions and proofs might seem less important, and the student would also not understand why a total search was not as good as any other algorithm for the purpose of proving NP completeness. This could possibly cause the student to conclude that the complexity part is *not* related to algorithm design at all, which would be a pity. On the other hand, if a student clearly likes designing algorithms, but thinks complexity is not related to this, it ought to be easy for that student to learn what specific requirements are always posed for the algorithms, i.e. reductions, that should be constructed. By learning that there are such requirements, and learning which they are, the student could feel more comfortable in designing reductions.

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 at Scool 1 and 2. We think that these problems are universal in complexity learning, and some of the concepts could be considered as threshold concepts. We plan to investigate possible threshold concepts in computational complexity in our next study.

7. REFERENCES

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

[2] Anderson, L.W. (Ed.), Krathwohl, D.R. (Ed.), Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, J., Wittrock, M.C. (2001). A taxonomy for learning, teaching, and assessing: A revision of Bloom's Taxonomy of Educational Objectives (Complete edition). New York: Longman

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

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

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

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

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

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

[9] 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*.

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

[11] Kattis. http://kattis.csc.kth.se/.

[12] Kleinberg, J. and Tardos, E. Algorithm Design. Addison Wesley, 2006.

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

[14] 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.

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

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

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

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

[19] Sipser, M. Introduction to the Theory of Computation. Thomson, 2006.