Learning greedy strategies at secondary schools: an active approach

Violetta Lonati, Dario Malchiodi, Mattia Monga, and Anna Morpurgo

Abstract We describe an extra-curricular learning unit for students of upper secondary schools, focused on the discovery of greedy strategies. The activity, based on the constructivistic methodology, starts by analyzing the procedure *naturally* arising when we aim at minimizing the total number of bills and coins used for giving change. This procedure is used as a prototype of greedy algorithms, whose strategies are formalized and subsequently applied to a more general scheduling problem with the support of an ad hoc developed software.

Key words: CS teaching, active teaching, greedy algorithms

1 Introduction

After several dark decades during which school pupils were often forced to identify informatics with mere dexterity with ICT tools, the education systems are now starting to reconsider the importance of the core aspects of the discipline. Many initiatives [5, 16, 13, 14, 4] have worked hard to change the general perception and it is now ongoing an explicit effort to bring into play key informatic concepts such as abstraction, logic, data representation and a general attitude to '*computational thinking*' [6, 8], i.e. to think about problems and their solutions in a way suitable to automatic processing. In this context, algorithms are becoming a very important topic, and there is a need to design new activities to foster learning how to reason on non trivial algorithms. Optimization algorithms, with their compelling usefulness, are very good candidates. Thus, we started thinking on potential activities with them, and we decided to focus on greedy strategies, well suited even for pupils of non vocational secondary schools.

Università degli Studi di Milano, Dipartimento di Informatica, via Comelico 39 20135 Milano Italy
e-mail: {violetta.lonati,dario.malchiodi,mattia.monga,anna.morpurgo}@unimi.it

In fact, a greedy strategy is a very *natural* way to cope with optimization problems: its short-sighted structure is attractive even for untrained minds, often uncomfortable with more elaborate planning of computing steps. However, while intuitive to adopt, a greedy solution is not always optimal and choosing the right greedy criterion (and convincing oneself that the choice is indeed optimal) is much more difficult and "unnatural". We decided, however, that this could be an important computational thinking learning objective. Thus, by following the chapter on greedy algorithms in the book by Jon Kleinberg and Éva Tardos [9], we developed a learning unit on greedy strategies for upper secondary schools based on constructivistic approach, and in paticular using our methodology called *algomotricity* [1, 2, 3, 11, 12].

The paper is organized as follows: in Sect. 2 we sketch our methodological root, in Sect. 3 we describe the learning unit, and in Sect. 4 we draw some conclusions.

2 Constructivistic learning theory and computer science education

Constructivist learning theory, which has its origins in Piaget and Vygotsky [15, 17], states that people construct their own understanding and knowledge of the world through experiencing things and reflecting on those experiences. The learners are the creators of their own knowledge, and the learning process relies to a large extent on what they already know and understand; when faced with something new they need to reconcile it with their current mental schemes and conceptions.

Starting from this assumption, the question of what stimuli and tools, methods and strategies are more effective is fundamental. If knowledge cannot be simply *transferred* but must be *constructed*, its acquisition should be an individually tailored process, where learners are in charge of the learning process and the role of teachers is to create suitable contests and materials that favor such process, accompany the learners' discoveries, and promote a metacognitive reflection about what they are doing and how their understanding is developing.

Work in small groups has the power to foster cognitive development and thus to empower learning. The group is a place to belong to and as such it provides support and motivation; it promotes cooperation and the activation of latent cognitive potentials through the sharing of different competences and working/thinking styles; it supports co-construction of knowledge by socialisation and reciprocal negotiation of meanings. Through the socio-cognitive conflict that emerges in groups, learners have the opportunity and the need to explain, confute and defend their beliefs; new aspects and prospects can be seen; personal experiences and point of views can be downsized and put in perspective.

This approach is especially fruitful to develop competences in problem solving, which are usually difficult to acquire by means of explanations and examples only. On the contrary, the learning process can be activated when facing a problem for which one's own repertoire of known procedures is insufficient [10, 7]: with exploration and discovery activities in small working groups, learners can learn together what they need to know in order to solve the problem.

Having these premises in mind, since 2011 we are experimenting with active workshops sharing a common strategy, which we call *algomotricity*. As the name suggests (a portmanteau combining *algorithm* and *motoric*), our approach exploits kinesthetic learning activities, having the aim of informally exposing students to a specific informatics topic, followed by an abstract learning phase devoted to let students build their mental models of the topic under investigation and a final computer-based phase to close the loop with their previous acquaintance with applications. Our activities start "unplugged", but they always end with work in which students are confronted with specially conceived pieces of software in order to make clear the link (but also the intellectual hierarchy) with the computing technology.

The learning goals of the unit are summarized in Fig. 1.

Knowledge

- Definition of optimization problems.
- Outline of greedy strategies and their limits.
- Approaches to analyze the correctness of a greedy algorithm.

Skills

- Describing a greedy strategy/procedure.
- Executing a greedy algorithm.
- Establishing if a greedy algorithm finds an optimal solution on a given input of small size.
- Establishing if an instance provides a counterexample for optimality of a greedy algorithm.

Competences

- Searching for a counterexample to disprove a property, or general reasons/proofs to conclude that a property holds.

Fig. 1 Learning goals of the unit on greedy algorithm.

3 The learning unit

The learning unit is organised in two main phases. In the first one students work on an algorithm for giving change using the minimum number of available coins and bills; in the second one they are faced with a scheduling problem that can be tackled by using a simple software tool we developed *ad hoc*[1]. Most activities are carried out actively by students; only between the two phases and at the end of the unit some taught explanations are given.

[1] The scheduling software is available at http://aladdin.unimi.it/sw/scheduling/scheduling.html (in Italian).

```
1. Sort coins and bills in decreasing order of their nominal value;
2. For each bill/coin, namely $X$, taken in that order:
       if the value of $X$ is not higher than the residual change
        then use it,
        else reject it (and never consider again bills/coins with this value).
```

Fig. 2 General schema for a greedy strategy

3.1 Giving the change

The class is faced with the (simple) task of giving change using coins and bills. Some examples are computed with the whole class, in order to make evident that everybody is able to accomplish such a task easily, with no need of deep reasoning. The question then is asked whether the number of bills/coins used to give the change is as small as possible: usually students have no doubts that this is the case.

After this introduction, they are asked to work in pairs to put in written words the procedure to compose a given amount by using the smallest number of bills/coins. To avoid the use of technical jargon or constructs (what occurs for instance with students who already have some programming background) and in order to make the required level of detail clear, students are invited to "write the procedure so that it can be executed by a 10 years old child who is familiar with basic arithmetic operations". A set of play money can be used to help formalize the procedure and to test it through a step-by-step execution.

Most groups usually propose a procedure that manages to accumulate the change through subsequent selections of one coin/bill having the highest value yet not exceeding the residual change; some use a more succint approach exploiting the quotient and remainder of the division between the residual change and a coin/bill nominal value. Some explicit an initial step that sorts coins and bills according to their value, some leave such natural/obvious sorting implicit.

Working pairs are then merged into groups of 4-6 people so that pairs whose procedures share a common approach are put together. Each group is required to socialize the procedures of the pairs merged in the group, and agree upon a (possibly new) common procedure.

When all are ready, each group, in turn, reads its algorithm aloud, while the remaining ones test it. Remarks are welcomed, both on the features of the presented procedures and on their commonalities or differences. Starting from these remarks, the conductor draws and highlights the commonalities and proposes the unifying schema in Fig. 2, while ascertaining that students recognize their own procedures.

3.2 Outline of optimization problems and greedy strategies

At this point, the conductor can generalize such approach to a more abstract procedure for a generic optimization problem which builds the solution by considering a set of objects in a given order, and for each of them decides whether or not it has to be enlisted in the solution according to a validity constraint (see Fig. 3). It should be emphasized that this constitutes an example of *greedy procedure*, in that each object is considered only once for (possibly multiple) addition to the solution, without any possibility to remove objects previously added to the solution.

Optimization problem

Instance:	a set A of objects
Admissible solutions:	any subset $S \subseteq A$ satisfying the validity constraint
Optimal solution(s):	an admissible solution with minimal cost or maximal value

Greedy procedure

```
1. Sort objects according to some criterion;
2. For each object, namely $X$, taken in that order:
        if $X$ satisfies the validity constraint:
         then use it,
         else reject it (and never consider it again).
```

Fig. 3 General form of optimization problems and general outline for greedy strategies.

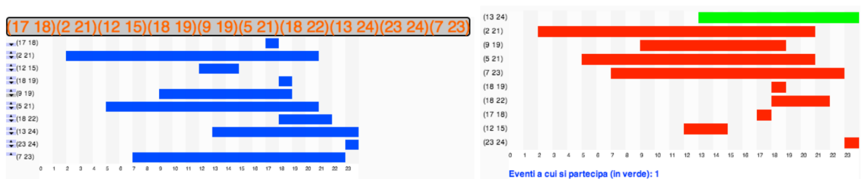
3.3 A scheduling problem

The second part of the unit consists in asking the students to apply such approach to a scheduling problem, namely that of maximizing the number of movies to be seen in a film festival whose program contains several, partially overlapping movies. First, students are guided to find the analogies between this problem, the one concerning money change and the abstract description of a greedy procedure. Table 1 highlights such analogies: for instance, students tend to easily spot that movies, as well as coins/bills, play the role of objects. Analogously, the validity constraints are that of checking whether: i) a movie does not overlap with the ones already in the solution, and ii) the considered coin/bill has a nominal value smaller than the residual change. Once the analogies have clearly emerged, the discussion can be focused on the feasibility of a greedy approach to find an optimal solution for the scheduling problem. The main effort here is to figure out how to sort the movies, what instead was obvious in the case of coins and bills. Brainstorming should be suggested in order to collect several possible sorting criteria; the following are usually spotted: starting or ending time, number of intersections with the other movies, or movies' length (either ascending or descending).

Once all alternatives are clear, students are asked to work in pairs to verify which criteria ensure the greedy procedure to maximize the number of seen movies. In particular, students are asked to find counterexamples to reject non-optimal criteria. Our piece of software supports them, by generating at random a set of movies (cfr. Fig. 4(a)) or letting them choose an initial set, rearranging it according to a chosen sorting criterion, and applying the greedy procedure (cfr. Fig. 4(b)). Thus students may experiment different sorting criteria on different instances of the scheduling problem and observe and confront the obtained solutions; they eventually realize that a counterexample is enough to discard a sorting criterion, whereas a proof is needed to accept one as optimal.

	Change problem	*Scheduling problem*
Objects	coins/bills	movies
Sorting order	nominal value, decreasing	several alternatives
Validity constraint	value not greater than the residual change	no overlapping between one movie and the ones already added

Table 1 Comparison between the scheduling and money change problems.



Eventi a cui si partecipa (in verde): 1

(a) (b)

Fig. 4 The software showing (a) a randomly generated set of movies and (b) applying the greedy procedure according to a selected sorting criterion (number of overlappings).

Consider for instance the case shown in Fig. 4(b). With movies sorted according to decreasing number of overlaps, the greedy procedure would suggest to watch only one movie (the highlighted one on the top), discarding all the remaining ones. However, it would be possible to see four different movies: the sixth (18,19), the eight (17,18), the ninth (12,15), and the tenth (23,24) from the top, and this is a counterexample proving that the criterion does not guarantee optimality.

Students in different groups tend to find counterexamples for most criteria, mainly observing the outputs produced by the tool on random instances or inventing instances to test a specific idea and detecting the suitable ones. In these cases they should be invited to devise the smallest examples as possible, in order to have insights about the reason why a criterion is not "good in general". However, usually three criteria endure the attempts of finding counterexamples, namely the one based on increasing ending time and its symmetric based on decreasing starting time, and the one considering the number of intersections. Students' attempts then start swinging between building a suitable counterexample and finding an explanation why a counterexample cannot be found for such criteria.

Actually, one can prove that the first one (and its symmetric) guarantees the optimality of the built solution, whereas for the latter counterexamples exist but they are necessarily larger than those for the other criteria. In our experience, no one succeeds in getting these results within the time devoted to the activity; however most were eager to know the correct answer and paid strong attention to the formal proof the conductor showed after a while. In several cases, some students asked to delay the final explanation to the next lesson, in order to have the possibility to think further about the problem, and indeed someone succeeded in the task with their own great satisfaction.

3.4 Final recap

In a final recap, the outcome of the previous activity is used to stress that a greedy procedure does not always lead to the optimal solution: this happens for instance if using the non-optimal criteria for the movie festival problem. Moreover, there are some optimization problems that cannot be solved by any greedy algorithm, in that no greedy algorithm is known that can guarantee to find an optimal solution. To let students address this fact on their own, they are asked to reconsider the money change problem with other sets of coins/bills: is it true that the procedure they wrote at the beginning of the workshop will always output the minimum number of coins/bills, or are there cases when this property is not guaranteed? The question usually surprises students because they had not even contemplated this issue before, but the answer now appears to be not obvious at all. Indeed, after a few attempts, the class is able to devise a suitable set of coins/bills and to make a counterexample. For instance, if we have coins with values 1, 12, 20 and we want to give the change of 24, the greedy algorithm would give 5 coins/bills, namely one piece with value 20 and four pieces with value 1, whereas a better solution would be formed by two pieces with value 12. Other examples of such money systems can be signaled, for instance the British coin system before the decimalisation (15 February 1971) or the one described in Harry Potter's books.

4 Conclusion

There is a growing interest in teaching informatics within the standard curricula even in non-vocational school, thus marking a shift with reference to the past identification of this discipline with the use of software applications. Within this trend, we proposed a learning unit rooted on the constructivism and focused on the discovery of greedy strategies in order to solve optimization problems. The learning unit has been developed and fine tuned with bachelor computer science students, and subsequently proposed as an extracurricular activity to some high-school classes between 2015 and 2017. We proposed it mainly as part of a vocational guidance effort and we did not set up any formal assessment. However, the general impression is that most students, even the less mathematically sophisticated, did in fact grasp the idea of an abstract greedy procedure. Many understood the importance to analyze the procedure to check if it works properly for the optimization task. A companion software tool helped several pupils in sensing the impact of the sorting criteria, and realizing that most of the criteria can be discarded through the use of counterexamples.

References

1. C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, and M. Torelli. Exploring the processing of formatted texts by a kynesthetic approach. In *Proceedings of 7th WiPSCE*, pages 143–144, New York, NY, USA, November 2012. ACM.
2. C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, and M. Torelli. What you see is what you have in mind: constructing mental models for formatted text processing. In *Proceedings of 6th ISSEP*, number 6 in Commentarii informaticae didacticae, pages 139–147. Universitätsverlag Potsdam, February 2013.
3. C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, M. Torelli, and L. Zecca. Extracurricular activities for improving the perception of informatics in secondary schools. In Y. Gülbahar and E. Karataş, editors, *Proc. of 7th ISSEP*, volume 8730 of *LNCS*, pages 161–172. Springer International Publishing, September 2014.
4. Computing at school: Educate, engage, encourage. http://www.computingatschool.org.uk/, 2013.
5. V. Dagiene. Information technology contests - introduction to computer science in an attractive way. *Informatics in Education*, 5(1):37–46, 2006.
6. P. B. Henderson, T. J. Cortina, and J. M. Wing. Computational thinking. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 195–196, New York, NY, USA, 2007. ACM.
7. C. E. Hmelo-Silver. Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3):235–266, 2004.
8. International Society for Technology in Education & Computer Science Teachers Association. Operational definition of computational thinking for K-12 education. https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf, 2011.
9. J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.
10. D. A. Kolb *et al.* Experiential learning theory: Previous research and new directions. *Perspectives on thinking, learning, and cognitive styles*, 1:227–247, 2001.
11. V. Lonati, D. Malchiodi, M. Monga, and A. Morpurgo. Is coding the way to go? In A. Brodnik and J. Vahrenhold, editors, *Proceedings of 8th ISSEP*, volume 9378 of *LNCS*, pages 165–174, Switzerland, September 2015. Springer International Publishing.
12. V. Lonati, D. Malchiodi, M. Monga, and A. Morpurgo. Nothing to fear but fear itself: introducing recursion in lower secondary schools. In *Proc. of the Fifth Int. Conf. on Learning and Teaching in Computing and Engineering (LATICE 2017)*, April 2017. To appear.
13. V. Lonati, M. Monga, A. Morpurgo, and M. Torelli. What's the fun in informatics? Working to capture children and teachers into the pleasure of computing. In I. Kalaš and R.T. Mittermeir, editors, *Proc. of 4th ISSEP*, volume 7013 of *LNCS*, pages 213–224. Springer-Verlag, 2011.
14. H. Partovi and M. Sahami. The hour of code is coming! *SIGCSE Bull.*, 45(4):5–5, Oct. 2013.
15. J. Piaget and B. Inhelder. *The Psychology of the Child*. Basic Books, New York, 1969.
16. M. M. Syslo and A. B. Kwiatkowska. The challenging face of informatics education in poland. In R. T. Mittermeir and M. M. Syslo, editors, *Informatics Education - Supporting Computational Thinking, Third International Conference on Informatics in Secondary Schools - Evolution and Perspectives, ISSEP 2008, Proc.*, pages 1–18. Springer, 2008.
17. L. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Cambridge: Harvard University Press, 1978.