The Greedy Trap and Learning From Mistakes

David Ginat
CS Group, Science Education Department
Tel-Aviv University, Israel
ginat@post.tau.ac.il

Abstract

Educators' approach towards their students' mistakes can have significant impact on the students. This paper presents a rather less considered approach of teaching by capitalizing on mistakes. In the course of teaching our students algorithm design, we noticed the phenomenon of students' "over-reliance" on intuition rather than rigor. In particular, we noticed a repeated erroneous trend of turning to intuitive, but inadequate greedy algorithmic solutions. We capitalized on the student errors for influencing their attitude and beliefs regarding intuition and rigor. The paper displays the student errors and our capitalization-on-errors approach, with colorful and novel algorithmic tasks.

Categories & Subject Descriptors

K.3.2: Computer and Information Science Education – Computer Science Education.

General Terms

Algorithms, Verification.

Keywords

Student Errors, Pedagogy.

1. Introduction

"You learn from your mistakes". We all are familiar with this relevant and very true saying. Some of us would even say: "you learn better from your mistakes". However, do we capitalize on mistakes in our teaching? Do we plan a lesson based on expected student errors, in order to "make a point", illustrate a principle, or affect student beliefs? Some of us probably do, but very little of this appears in our textbooks and educational literature.

This paper presents work on teaching with capitalization on student errors. The work is an outcome of our ongoing experience with students' tendency to turn too quickly and too decisively to erroneous greedy solutions in a variety of algorithmic problems, many of which are optimization tasks. Such tasks are very common in many CS courses, including the fundamental courses of CS1, CS2, and "Introduction to Algorithms".

In teaching algorithm design principles and techniques, at both CS1 and "Introduction to algorithms" levels, we noticed again and again that students designed greedy solutions, which they only intuitively verified. While some solutions were correct, many were erroneous. Given an algorithmic task, students tended to examine rather few and limited examples, which they regarded as "representative input examples". They derived solutions based on these examples, and argued that their solutions were correct "because they make sense, in particular when you look at the examples". Some students did not even examine particular examples, but rather had a strong intrinsic feeling of conviction, derived from "obvious observations", which seemed very natural to them.

There are several approaches one may consider for addressing such students' lack of scientific perspective. One may reiterate that intuitive argumentation and selected input examples are insufficient for verifying correctness. One may require formal argumentations of correctness. One may ask students to follow thorough testing guidelines. One may even send the students to read the famous Dijksktra et al. debate on the nature of computer science [3].

We tried the above approaches. They indeed contributed to the students' scientific perspective. However, students' "over-reliance" on intuition and common sense remained. After all, we all invoke intuition upon searching for a problem solution, and we use common sense argumentation throughout our teaching. Yet, we, the experts, are well aware of the "brittleness" of intuition and common sense. The students are not – their belief of the relevance of intuition and common sense argumentation is much beyond that of a "brittle means". They often rely on unfounded convictions.

In order to affect the student beliefs, and effect their realization of the essential role of rigor, we decided to add a new pedagogical approach to those mentioned above. Instead of only showing the students "the right way", we let them follow "the wrong way" and realize its outcome. The main theme resulting from this experience was their realization of the risks of premature convictions.

SIGCSE 2003, February 19-23, 2003 Reno, Nevada, USA.
Copyright 2003 ACM 1-58113-648-X/03/0002….$5.00

The objective of this paper is two-fold. We draw the attention to the phenomenon of repeated erroneous paths that students follow upon solving optimization tasks, and we demonstrate how these paths were utilized for obtaining learning from errors.

We call the students' erroneous optimization paths that we have noticed "**the greedy trap**", as these paths were directed by an intuitive, but misleading greedy principle, which appeared in similar variations. The general theme underlying this principle is: "maximize the gain in each computation iteration". This coincides with the notion of greed underlying the *greedy design technique*, led by "making the choice that looks best at the moment" [2].

The next section briefly mentions previous work on errors, in the domains of computer science and mathematics, and introduces our approach of capitalization on errors for learning. It also describes the actual class with which we worked. The next two sections describe our experience in class with two instances of "the greedy trap" and learning from mistakes. The algorithmic tasks in both sections are novel (developed by us for diverting from the common tasks). The last section involves some concluding remarks.

2. Learning from Mistakes

Computer science educators widely studied student errors in the last two decades. Many studies focused on novice mistakes in basic programming (e.g., [4,7,9]). Novice errors stem from various reasons. One major source of errors and misconceptions is the lack of understanding of the computer model or the programming-language constructs. Another source of errors is students' ill-planning of programs, due to lack of modularity, unsuitable data-types, or inaccurate boundary conditions. A few studies focused on novice beliefs and the gaps between novice and expert beliefs (e.g., [5]).

The above studies yielded work that focused on methods and tools to overcome difficulties (e.g., [7]). Most of these studies focused on the "right way of doing", with particular attention to elements of difficulty. No method, to the best of our knowledge, focused on capitalization on "the wrong way of doing".

In the domain of mathematics education, numerous studies were conducted on errors, and some offered capitalization on errors in teaching. One proposed method is that of teaching through conflict, or paradox (e.g., [8,10]). The theme underlying this method derives from the notion of "colliding conflicting conceptions", in order to yield realization of the right conception. Another teaching method focused on utilizing errors as springboards for mathematics inquiry [1].

Our scheme of capitalization-on-errors combines elements from the various studies mentioned above, with particular adaptation to the CS domain of problem solving:

- We identified a repeated erroneous phenomenon among CS students. Unlike previous CS error studies, the erroneous phenomenon here is not a computer-model misconception or a buggy programming construct, but an improper solution principle.
- We posed a relevant algorithmic task, and let students reach erroneous solutions and argue their justification, usually with arguments that did not capture the core problem characteristics.
- We indicated that we are not convinced by the student justification, and we would like to see more solid arguments, or disconfirmatory evidence.
- If proper argumentation was still not provided, then we repeated asking for disconfirmatory evidence. Once such evidence was found, an explanation was requested for the conflict derived from the new evidence.
- We asked to correct erroneous solutions, and devise alternative solutions. This sometimes involved the examination of different solutions offered in class.
- We reached, together with the students, the correct solution, and obtained its justification, based on rigorous, scientific arguments.
- We led a class discussion of reflection on the solution process and its lessons. Particular focus was put on the students' behavior and beliefs.

Several points should be emphasized. First, we consider a **rigorous argument** an assertion that captures the core mathematical characteristics of the problem at hand. Such an argument does not have to be formal, yet it must capture the essence.

Second, one main objective in the above scheme is to "push" students to comprehensively examine test cases and thoroughly **look for counter-evidence**. This, we believe is not less important than rigorous argumentation.

Third, upon realizing their erroneous solutions, students were encouraged to examine whether they have to perform a radical re-examination of their solution, and not just local, bug corrections. They also had the opportunity to **compare between correct and incorrect solutions**. This is different from correct-solutions comparisons, which appear for example in [6].

The final, reflection stage is of particular importance. At this stage students realize the reasons for their errors, the misguided solution process, and the difference between their initial incorrect convictions and the final correct arguments.

We applied the above scheme with a class of 33 students during the spring semester of 2002, in an "Introduction to Algorithms" course, which started with a quick review of basic, CS1 design topics. In what follows we display some of the interactions with the students, who "fell in the greedy trap" and learnt from their mistakes.

3. The Greedy Trap – Illustration-1

The following learning interaction took place in an early stage of the course. We posed the following task.

Wire Connections

N white dots and N black dots are interleaved in some arbitrary order on a straight line. The distance between every two adjacent dots on the line is 1 hop. An electrician is required to connect each white dot with a different black dot, so that the total length of the N connections will be minimal.

Develop an algorithm whose input is the interleaved order of the 2N dots, and its output is the minimal connection length.

Example: For the sequence of dots: □ □ ■ □ ■ ■ the output should be 7. When we number the dots from left to right, starting with 1, one way to obtain the minimal length is with the connections: 1-3, 2-5, 4-6. Notice that there are other ways to yield the minimum.

Over one third of the students (14 out of 33) decided that they "see" the solution rather quickly, without carefully examining examples beyond the one in the task statement. They all devised a greedy solution based on the same underlying principle: "repeatedly scan the input, and in each scan create as many minimal-length connections as possible". Thus, in the first scan, create as many 1-hop connections as possible (between adjacent dots of complementing colors). In the next scan create as many minimal-length connections as possible between the remaining dots, and so on. According to this principle, the dots in the task-statement example will be connected as follows: first 2-3 and 4-5, and then 1-6.

We asked for justification. The students confidently argued: "by repeatedly connecting as many nearest dots as possible in **each iteration**, you obtain **maximal reduction of the problem size**, with minimal connection-length". This intuitive argument may sound convincing, but it lacks rigor. We asked for more rigorous justification, or alternatively – disconfirmatory evidence.

The students offered more examples, which served as supporting evidence: the examples were a bit larger but very similar to the one in the task statement. Neither rigorous justification nor disconfirmatory evidence was provided.

We suggested that they try to characterize the examples generated so far, and generate some new pattern. One outcome was: □ □ □ □ ■ ■ ■ ■. Their solution indeed yields the minimum length for this example (with the connections: 4-5, 3-6, 2-7, 1-8), and they became even more convinced with the correctness of their solution.

One student offered the example: □ □ ■ ■ □ □ ■ ■. At first they all felt that this example only yields further support, but surprisingly it did not! Their solution yields the connections: 2-3, 4-5, 6-7, and 1-8, with a total length of 10, but the minimum total-length here is 8 (obtained for example by: 2-3, 1-4, 6-7, 5-8).

They had to explain the conflict between what they perceived as the right algorithm and the disconfirmatory evidence. Some argued that it is just a special case, which can be treated separately, without abandoning the current solution. Such reaction is often common to many students! A few tried to understand the difficulty more carefully, and realized that there is a deeper pattern.

In the last example, some students conjectured that "one way for obtaining the minimal length is by viewing the required connections as legal connections between corresponding parenthesis". These students expressed a radical change of conception. They offered to perform the computation similarly to the way "a parenthesis expression is checked for validity". Every white dot will be considered "open", and every black dot - "close", and every "open" will be connected to its corresponding "close".

Thus, the sequence □ □ ■ ■ □ □ ■ ■ will be viewed and handled as two sub-sequences of the form: □ □ ■ ■. The sequence □ □ ■ □ ■ ■ □ ■ will be handled as the sub-sequences: □ □ ■ □ ■ ■ and □ ■. Each sub-sequence of length greater than 2 will be further divided and handled according to the above principle. Notice that the students' original solution fails in these two examples.

Learning from their recent experience, the students were much more careful with the new solution's correctness. They carefully looked for disconfirmatory evidence, but did not find any. They noticed that the parenthesis-matching is based on decomposition.

Some looked at the parenthesis-matching from an "untying perspective". They claimed that the transformation of any given connection structure to the parenthesis-matching can be viewed as "untying of crossing connections", and will never increase the total connection length. The rigorous argument underlying this claim is that if you take any two white dots and two black dots that are not connected according to the parenthesis-matching, and "untie" their connection, the total connection length will not increase.

Calculation of the parenthesis-format length can be very elegantly done using a stack, with only one pass over the input. This on-the-fly solution is one way for obtaining the desired result. There are additional ways.

4. The Greedy Trap – Illustration-2

The following learning interaction took place in a rather advanced stage of the course.

Gold Collection

Golden coins are spread in different cells of an N×M matrix platform. These coins should be collected by robots, which are located in the platform's top-left cell. The robots' destination is the bottom-right cell. The robots can only move in two directions – down and right. That is, a robot step is either horizontally to the cell on the right or vertically to the cell below. Upon visiting a cell with golden coins, the robot collects the coins in that cell. The objective is to collect all the golden coins with a minimal number of robots.

Develop an algorithm whose input is the locations of the cells with golden coins, and its output is the minimal number of robots for performing the task.

Example: For the 7×8 platform, and 10 golden-coin cells: 2,6 3,2 3,8 4,3 4,5 5,2 5,5 6,8 7,5 7,6 the output should be 3. The coin layout is illustrated in the figure below.

					G		
	G						G
		G		G			
	G			G			
							G
				G	G		

There are various aspects that one may consider in solving this non-trivial task. One aspect is the **representation** of information: the matrix may be processed directly or transformed into a directed (acyclic) graph in which the nodes represent the golden coin cells. Another aspect is that of task analysis, with respect to seeking **underlying regularities**: one may try to identify an underlying mathematical pattern, as a basis for the solution. A third aspect involves the actual algorithm **design**: one may try to relate the task solution to some fundamental graph algorithm or a common design pattern. Algorithm design techniques, such as greedy computation or dynamic programming, may be relevant as well.

Although there are diverse aspects for consideration, most students quickly decided that they have a "grip on the problem at hand" and devised a solution. Twenty-one of the 33 students suggested the following greedy 'min-max' optimization that seemed very natural to them: "since the objective is to **minimize the number of robots**, we should **maximize the amount of coins collected by each robot**. That is, the first robot will visit a maximal number of coin cells. The second will visit a maximal number of the remaining coin cells, and so on …".

Most students examined very few examples, and did not thoroughly look for disconfirmatory evidence. They were confident with their 'min-max' greedy approach, and did not look for a rigorous mathematical pattern. They all felt that the 'min-max' argumentation is compelling and sufficient. The common theme with all these students was a global perspective that was not carefully checked. It seemed that a coercive process led them to their solution, with a strong feeling of certainty. The doubts we raised did not really diminish their feeling of conviction.

Directed by us, they sought disconfirmatory evidence, but did not find any. The lack of such evidence only strengthened their intrinsic conviction, as it yielded more supporting evidence. Yet, they could not provide any rigorous justification. Finally, one student who continued looking for a disconfirmatory evidence noticed that a "very fruitful collection by one robot may divide the remaining occupied cells into undesired disjoint areas". He illustrated it with the following example:

		G			G		
G	G		G			G	G
							G
			G	G			
							G

In this example, the maximal path by the first robot (through 7 coin cells) will leave 4 occupied cells that require two additional robots, implying a total of 3 robots. However, the optimal solution requires only 2 robots. This indeed shows that the suggested 'min-max' greedy scheme is incorrect.

Some students still tried to provide modifications that will solve "special cases" like the above. They were reluctant to abandon their initial conviction. Other students tried to carefully learn the characteristics of the displayed example. They realized that they need a radical change of conception, and indeed noticed that the correct solution for the displayed example can be viewed as "peeling the next leftmost strip from top to bottom". In the latter example it implies that the first robot will collect the coins from the three leftmost occupied cells in line-3, the coins in line-5, and the coin in line-6.

Justification still remained. It involved a mathematical pattern and the notion of disjoint occupied cells. Two occupied cells are considered *disjoint* if one of them is to the left and below the other. Coin collection from such two cells requires two robots. The length of the "longest chain" of disjoint occupied cells (where the first and the second cells in the chain are disjoint, the second and the third are disjoint, etc.) bounds the minimum number of robots from below. The "peeling principle" yields a decrease of 1 in the chain length by each robot, and therefore implies optimality.

5. Conclusion

We introduced a study of a common error phenomenon and an approach of learning from errors. The common phenomenon, which we called "the greedy trap", occurred in a variety of algorithmic problem-solving occasions, in particular with optimization tasks. We capitalized on this phenomenon for teaching students the "brittleness" of intuition and the essential role of rigor.

In this paper we displayed only two illustrations of "the greedy trap". Yet we noticed it repeatedly in diverse tasks. Students turned again and again to variants of the intuitive principle: "maximize the gain in each computation iteration". One may view this principle as related to the heuristic optimization technique of "Hill Climbing". The principle may yield good results, but does not always guarantee the optimum.

An example simpler than those in the illustrations is the task of returning change with minimal number of coins. The intuitive approach, influenced by our daily life, is to first take as many quarters as possible, then as many dimes as possible, etc. Thus, a change of 62 will be composed of two quarters, a dime, and two cents. However, this approach does not yield the optimum for any potential coin system (e.g., the coin system 1,6,10, and the change 12).

We mentioned students' "over-reliance" on intuition, and described our attempt to influence their beliefs regarding the importance of rigor, beyond intuition. The students gradually learnt, in repeated occasions, the risks of no-rigor. They realized it again and again during the class interactions; first, by seeking counter-evidence to their solutions; then, through radical modifications of their solutions, followed by rigorous argumentation; and finally, through reflection on the whole process. The reflection involved metacognitive elements, as the students looked back at their own (often erroneous) reasoning and decision making along the solution process.

In the end of the study (course) they explicitly indicated that their beliefs have been affected. In particular, they thoroughly looked for counter evidence and underlying, rigorous patterns. In addition, they denoted a significant point which we did not a-priori plan. They noticed a few times that the characteristics of the counter evidence they found to their initial, erroneous solutions helped them to gain valuable insight into the correct solution. This could be seen in the illustrations in the previous sections.

This study focused on teaching with capitalization on a particular, recurring error. Yet, the learning-from-errors approach presented here can be utilized with other errors, of different types. It is our hope that this study will encourage additional capitalization-on-errors studies for improving understanding and affecting beliefs.

Acknowledgement

Many thanks to the anonymous reviewers for their helpful comments.

References

[1] Borasi R., *Reconceiving Mathematics Instruction: A Focus on Errors*, Ablex Pub (1996).

[2] Cormen T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms*, MIT Press, Massachusetts, (1991).

[3] Dijkstra E.W. et al., A debate on teaching computing science, *Comm of the ACM, 32,* (1989), 1397-1414.

[4] Du Boulay B., Some difficulties of learning to program, *Journal of Educational Computing Research, 2*, (1986), 57-73.

[5] Fluery A.N., Student beliefs about Pascal programming, *Journal of Educational Computing Research, 9*, (1993), 355-371.

[6] Linn M.C. and Clancy M.J., The case for case studies of programming problems, *Comm of the ACM, 35*, (1992), 121-132.

[7] Mayer R.E. (Ed.), *Teaching and Learning Computer Programming: Multiple Research Perspectives*, Lawrence Erlbaum, (1988).

[8] Movshovitz-Hadar N. and Hadas R., Perspective education of math teachers using paradoxes, *Educational Studies in Mathematics, 21*, (1990), 265-287.

[9] Soloway E. and Sphorer J.C. (Eds.), *Studying The Novice Programmer,* Lawrence Erlbaum, (1989).

[10] Swan M., *Teaching Decimal Place Value: A Comparative Study of 'Conflict' and 'Positive Only' Approaches,* Shell Center for Mathematical Education, University of Nottingham UK, (1987).