Dynamic programming – structure, difficulties and teaching

Emma Enström
Schools of {Computer Science and Communication, Education and Communication in Engineering Science}
Royal Institute of Technology
Lindstedtsvägen 3
SE-100 44 Stockholm
Email: emmaen@csc.kth.se

***Abstract*—In our third year Algorithms, Data structures and Complexity course, students have considered dynamic programming hard in comparison to the other topics. We would like to know which difficulties the students encountered, where they gained their knowledge, and which tasks they were most certain that they could perform after the course. We identified subtasks that could be taught separately, and adapted the lectures to Pattern Oriented Instruction in order to help students cope with the complexity of solving problems using dynamic programming. Lectures were modified to support this strategy, and clicker questions, visualisations and a lab assignment were prepared. We constructed self-efficacy items on the course goals for dynamic programming, and administered them before and after the teaching and learning activities. Among the self-efficacy items, determining the evaluation order and solving a problem with dynamic programming with no hints had the lowest score after the course. As for the activities, arguing correctness of a solution was something many students claimed that they did not learn anywhere. Students considered the lab exercise most useful, but they also learned a lot from the other activities.**

I. Introduction

The course in which the experiments of this paper took place is a third year course in the 5-year Master of Computer Science and Engineering program at KTH in Stockholm, dealing with algorithms, data structures and complexity: ADC. Previous experience indicates that dynamic programming (below abbreviated dynprog) is considered difficult among the students. This work uses an action research approach.

A. Research questions

The questions that are addressed in this work are:

- Q1 What is perceived as difficult with dynamic programming?
- Q2 Which abilities do students judge they have after the course?
- Q3 Which of our activities do they believe to have helped them achieve those abilities?

We will not make an exhaustive examination of these questions, and the students are not interviewed on these topics. Rather, we ask them how well they believe they can perform tasks which we have defined, and where they have learned that. If they do not know things listed in the course goals after the course ends, we consider these subparts difficult. Items that students are less confident they can manage are also counted as difficult compared to the rest of the course contents.

B. Contribution

The contribution of this work is split into several areas. The answer to *What is difficult?* is in itself interesting to teachers. We also compare our guesses and assumptions with students' answers. The self-efficacy instrument that we have used is not final, nor tested for internal reliability and validity, but it is still a partial result. The experience was also a small scale experiment with Pattern Oriented Instruction.

II. Background

A. The course

The part of ADC that is relevant to this paper is the first 18 lectures, 7 tutorials and 3 computer lab sessions on algorithm construction methods, and especially lectures 9 and 10 and tutorials 3 and 4 plus one computer lab exercise, all of which deal with dynamic programming. Each lecture is 45 minutes, and each lab or tutorial session is 2 x 45 min. The computer lab exercise is compulsory, performed in pairs, automatically checked for correctness and verbally presented to a teaching assistant (TA) in lab. ADC uses continuous assessment. The lab exercise, together with an individual, written home assignment which the students afterwards present personally to one teacher or TA, constitute the major part of the assessed student work on the topic. There are also supplementary tasks for those who are unhappy with their grades. The course is graded on a scale from A to E, or F for fail, and there are course goals and criteria connected with each grade. These are presented in a matrix and a flow chart on the course home page, and addressed on lectures and during peer review. Towards the end of the course, there is a written exam for lower grade content and an oral exam for those who aspire to higher grades. In the end, how well your work meets the criteria decides your grade, so *what* work you have fulfilled, and how well you performed, matters – not just the percentage of work. The author has been one of the TAs on this course.

B. Our guesses on what was difficult, or why

Previous years' experiences suggest that the difficult parts of dynamic programming are finding a recurrence relation based

on some structure of the desired solution, and tackling the complexity of solving a problem completely from scratch, without hints. Recursion, as a well known "difficult" task, could possibly cause some trouble since it is part of the problem solving strategy of dynprog. Another necessary skill, needed for more complex problems, is coping with many dimensions – problems where a simple two-dimensional matrix is not sufficient to hold every subproblem that matters to the solution. Also, whenever "argue correctness" or "prove correctness" is part of the instructions, students struggle, or avoid the task entirely.

C. Recursion and dynamic programming

The subject of recursion has attracted educational researchers' interest for a long time. In the 1980s Kahney and Eisenstadt [1] contributed by studying students' and other programmers' answers to problems involving recursion. Their results are further described by Kahney in [2], and describe a set of "mental models" of recursion that students had, out of which one was capable of capturing the things instructors want students to know about recursion, and some were not only incomplete, but misleading. Around the same time, Ford [3] concludes that iteration is really a special case of recursion, and that recursion is a generalised control structure in programs. Similar arguments are also used by others: recursion is an example of the paradigm "Divide, Conquer and Glue", and when using iteration, the glue step is missing and students get erroneous pre-images of the paradigm [4]. Scholtz et al. [5] claim that the difficulties with recursion are connected to understanding the passive flow, whereas other authors are investigating misunderstandings that can occur around base cases [6]. They also note that recursion *is* a difficult topic for students. Many authors dwell on the topic of where in the first course recursion should appear, or how recursion is related to iteration, the computational model, or similar issues.

Ginat et al [7] suggest that less focus on the computational model, and more focus on the abstract level and the algorithm in theory can help students not to mistrust recursion as a method or their own abilities on recursion. As an algorithm construction method, dynprog follows that recommendation. It does not deal with passive flow, and the algorithm construction task indeed treats recursion as an abstract phenomenon.

Dynamic programming can be seen as an alternative method for constructing algorithms where recursion can be used. Sometimes, keeping a table of all hitherto calculated values in a recursive algorithm speeds up calculations significantly, for instance with simple sequences like the Fibonacci numbers. This technique is called *memoisation*. Instead of doing memoisation, there is also the option of changing the algorithm so that calculations are made in "reversed" order - starting with the base cases and building increasingly complex subproblem solutions, which is referred to as *dynamic programming*. In this case, the recursive calls are not needed. Hence, dynprog is a technique that alters recursion into iteration. If students are more comfortable with iteration, this should only help. On the other hand, for someone who wants to think of iterations, the concept of "finding an evaluation order" is not all too clear. Also, it can be easily imagined that difficulties in discerning base cases for a recurrence relation makes the dynprog approach uncomfortable.

D. Pattern Oriented Instruction

Since coping with many aspects of a problem at the same time might be difficult, and since teaching everything at once might lead students to focus entirely on the algorithm that is constructed, and less on the construction process, we tried using Pattern Oriented Instruction(POI). POI is based on cognitive psychology theories on construction and organisation of knowledge in schemata. A schema is a connected chunk of "information", that has been constructed by repeated experiences which share some common ingredients. When, for instance, solving a complex problem, we are processing several different types of information at the same time. This is called cognitive load, and if the cognitive load gets too heavy, our problem solving skills are heavily reduced. When having seen the same type of problem many times before, we are able to process many interconnected pieces of information as one unit, a "chunk", which reduces cognitive load. A pattern is either distilled from several different experiences with certain ingredients in common, or generalised from some specialised example of a phenomenon. It can be employed in teaching to enhance the learners' ability to create new schemata. POI deals with structuring the teaching and the content in a way that facilitates the creation of schemata, which can later be processed as chunks.

The identification, selection, progression and comparison of problems and patterns are in focus. The students get to meet with increasingly complex problems, encompassing different patterns already familiar. The comparison of problem characteristics is central, and patters are first introduced and later revisited in a different setting. The effect of POI on students' abilities in problem solving, abstraction and analogical reasoning in particular, are investigated by Muller and Haberman [8], [9], [10] and the effect on problem decomposition and solution construction abilities by Muller, Haberman and Ginat in [11].

E. Clickers and response cards

The use of clickers (or Audience Response Systems, or "voting" systems) for pedagogical or administrative purposes is widespread in the world today [12]. Among the reasons for using them is "anonymity", but since the systems are often built in a way that admits excessive data collection and even scoring on an individual level, it is worth noting that this anonymity is only towards peers, not towards the teacher. In other words, if the student is afraid of speaking up and answering questions on lectures because it could expose him or her to *the other students*, the argument is valid. If students don't want to expose possible lacks of knowledge to teachers, the system can actually be a greater threat, as temporary misunderstandings might be recorded and later graded. However, there are desirable benefits to clicker systems: they provide

means for student activity; everyone gets to think and answer the questions, not just the fastest responder; the results of the small polls allows the teacher to address misunderstandings directly; and they provide formative feedback to the students. They also incentivise the teacher to plan good questions that are suitable for this type of exercise. The use of response cards, predecessors of clickers of sorts, has been studied academically since the 1960s, and have proven to have effect on student grades and student participation [13]. We have gradually included response cards in the course during the two past years, and it has been much appreciated by the students. We have employed cards of different colours which are administered to the students in the beginning of lectures. The votes are still made simultaneously, and the feeling of being exposed seems not to be present among our students, according to their evaluation responses.

F. Self-efficacy

In contrast with self esteem or confidence, self-efficacy is described as an individual's confidence in his or her own ability to, at a given moment, perform actions in order to achieve some desired outcome. The term was introduced by Albert Bandura in the 1970s and is further developed by him in [14]. The phrasing of the self-efficacy items should be direct and not involve guesses about the future or some sort of inherent capabilities of the subject to the study. Self-efficacy beliefs are not static - on the contrary, they change with the individual's experience. It is self confidence of a sort, but situated and very localised in time and subject area contents. These characteristics have given self-efficacy a role in education. The score of a self-efficacy test is known to be an important predictor of success [15]. There are studies in how self-efficacy correlates with performance [16], how self-efficacy changes during studies[17], and how it correlates with other factors around the individual [18].

We knew of no established instruments measuring self-efficacy for theoretic computer science. In mathematics, self-efficacy instruments have been developed [16], [15] and also in programming [17], [18], which is another related area.

III. Method

A. Plan the new course activities

Previous years, other aspects of this course have been reworked, and after that, students complained that dynamic programming seemed unreasonably difficult compared to other content. For this experiment, the short part of the course that dealt with dynamic programming had to be planned with a new perspective. Visualisations of dynamic programming, a completely new lab exercise and clicker questions for the lectures were prepared. TAs were instructed on the plans and the goals of the new partitioning of the contents, and new tutorial session material was prepared. The new lab assignment was to modify and speed up a program that calculated the edit distance between two words. The system Kattis, described in [19] was used for automated online testing, and theory questions which would prepare the students for the optimisations needed were prepared. The visualisations shown in the introductory lecture exemplified calculation of the "degenerate" case of dynamic programming, the Fibonacci numbers, recursively, with memoisation and with dynamic programming.

For this "simplest" version of dynamic programming problems, number sequences, the order in which to calculate the answers seems natural, and coincides with the order you would calculate different subproblems when using memoisation - save each value the first time you need it, but use the recurrence as basis for your algorithm.

B. New structure the course content

We started by identifying what difficulties we could encounter, and what different dimensions we could allow to vary and needed examples of. The task of solving a dynamic programming problem can be thought of as built from three different tasks: 1) Finding a structure for the solution, 2) express a recurrence relation, and 3) defining and proving an evaluation order with the properties that we will always have solved "smaller" subproblems whenever we solve a "larger" problem, and that the evaluation order renders the same result as the recurrence relation would produce.

To help comparing and distinguishing various characteristics of problems, we identified what could vary between different dynprog problems: whether the history needed to be saved during computations or not; whether the evaluation order was intuitive or more intricate; whether new values depended on the indices of the elements to be calculated, or on some input, or both; whether the previous subproblems to be used for each calculation were the most recently calculated subproblems, or if some special method was needed to find relevant subproblems (for instance, "jump", skip cells, in a matrix of previously calculated values); the number of dimensions for the subproblems (do they fit in a sequence, matrix, higher order); that there could be several different recurrence relations for the same problem and that the same recurrence relation could be the basis of several differently posed questions; the "location" of the base cases, "the location" of the answer after calculation; whether only a number, or the entire path to that number was needed for the solution (constructive solution), and different combinations and variations of these features.

These were only the dynamic programming specific variations. Another important variation is in the structure of a problem, or rather, of its solutions. If subproblems do not overlap, divide-and-conquer is generally a better method than dynamic programming, and if they do overlap, dynprog might be best. A greedy solution can sometimes be proven to be the superior option. These algorithm construction methods are also covered in the first part of the course, and assessed on the same written home assignment.

C. Prototypes and patterns in dynamic programming

For the lectures, we decided to separate two phases of constructing a dynamic programming algorithm: 1) Recognising the structure of the problem and create a recurrence relation

and 2) Finding and proving an evaluation order for a given recurrence relation. This was done to direct more attention to the evaluation order and correctness arguments. At the same time, we wanted to refer to prototype problems. Some of these had already been used in the course, and others were introduced as a complement to these. The prototype problems we chose were: Fibonacci (sequences), 2-dimensional recurrences without input (2-dimensional sequences), 2-dimensional recurrences with input, (values decided from input), Longest increasing subsequence (index and input dependent, need to save full history) Matrix chain multiplication (base cases on the diagonal, construction of the solution), Swamp walk (different questions, same "solution"), Coins (different recursions for the same problem, focus on proving correctness), Longest common substring (argue correctness of recurrence relation and algorithm, construct the solution, compare to 2D-with input that rendered the same recurrence relation) and Floyd-Warshall's algorithm for finding all shortest paths in a graph (more than two dimensions used). We do not argue that these problems are *the best possible* set of prototype problems, but they contained examples of the variations we wanted to show and most of them had appeared in one form or another on the lectures previous years, only not as explicit representatives of some technique(s) each.

D. Surveys

Several different surveys and questionnaires were administered throughout the course. Their content and goals are described in this section.

1) Self-efficacy surveys: We would like to know whether students find any of the different tasks we believe are involved in dynprog especially hard, and whether students improve their self-efficacy during the course. Based on the ADC goals, the guidelines by Bandura in [20], and supported by the spirit of knowledge taxonomies such as Bloom's taxonomy [21], we have constructed a 10 item self-efficacy scale on dynamic programming, together with an 8 item scale on complexity that is further described in [22]. The students are asked to grade their self efficacy on the course's dynamic programming contents on a scale between 0 and 100. This was done on lectures, before and after the section of the course that was part of the experiment. 110 and 79 students responded, respectively, and 68 responded to both. This survey was not anonymous, but the students were promised that it would not count towards grading, and that no one would read the material until after the course was finished. This is naturally a limitation in the usability of the responses for that year, as the material could reveal important information to teachers, and also possibly introducing errors in the form of more "polished" or "adjusted" answers from students. On the other hand, we wanted to both be able to see individual trends between the two occasions, and retain the option to later compare self-efficacy beliefs with results (both their predictive value from the first occasion, since it is known that self-efficacy affects motivation and possibly performance, but also the possible correlations between results and self-efficacy beliefs after the teaching and learning activities.)

2) Home assignment cover page: Together with the homework assignment, the students got a cover page to attach to their homework, with two types of questions. One half was a participation statement, and contained questions on what sessions or activities the student had attended in the dynamic programming part of the course, and the other half was a "Where was what learned" part, and contained a matrix where students could mark where they had learned to master different aspects of dynamic programming: on their own, on lectures, on tutorials, from visualisations, from the peer assessed theory questions before the lab assignment, from the lab assignment, or if they still did not master it.

3) Course surveys: At the peer reviewed, pseudonymous theory exam, a final survey on the course was distributed in two versions, one with pre-defined answer options, and one with free-text questions. The students were randomly assigned one of these, and the one with pre-defined answer options is the one we present here. There was also, as always at our school, an online course, where some questions were about related issues. This survey is completely anonymous, done on the students' own time, and has higher non-response rate.

E. Student results and comparisons

The written homework assignment is graded A-E, so the grades might have improved if the teaching was much improved. There is also another, similar homework assignment on complexity, which can be used for comparison. The grade on the assignment, however, is based on the performance on three algorithm construction tasks, out of which one had a good dynamic programming approach. Yet another task was possible to solve with dynprog, but as this was far less efficient than another method, and this task was meant to assess the students' ability to choose the best methods for new problems, the use of dynamic programming here did not improve the grade (unless the student proved here but not on the previous task that he or she could use dynamic programming.)

IV. Results

A. The self-efficacy instrument

One of the instructions from Bandura was that the items on a self-efficacy scale should be of increasing complexity, or level of challenge. If by this we mean that ideally, the items should be ordered so that the answer values was a non-increasing sequence, this was not achieved. As the relative scores for the different items was what we wanted to investigate, the order is not of crucial importance to us. Some items could be rephrased to make them easier to understand for someone unfamiliar with dynprog. The items were (translated from Swedish):

1) I could understand dynamic programming algorithms as presented to me by teachers or in books.
2) I could decide whether an algorithm is based on the algorithm construction method dynamic programming.
3) I could construct an algorithm that calculates a recurrence relation, using dynamic programming.

item	1	2	3	4	5	6	7	8	9	10
pretest average	76	49	70	88	34	49	71	69	47	36
posttest average	90	84	87	95	68	78	88	80	67	65
threshold = 25										
less certain	0	0	4	3	3	8	2	6	5	3
no change	45	21	35	52	22	21	40	42	34	21
more certain	23	47	29	13	43	39	26	20	29	44
threshold = 50										
less certain	0	0	1	1	2	1	0	2	3	0
no change	64	46	60	62	39	40	61	58	50	47
more certain	4	22	7	5	27	27	7	8	15	21

TABLE I
NUMBER OF STUDENTS INCREASING THEIR SELF-EFFICACY VALUES FOR ITEMS 1–10. (N=68)

4) I could explain to an average peer why dynamic programming is better than a recursive procedure when implementing an algorithm to compute the Fibonacci numbers.
5) I could choose a suitable evaluation order for dynamic programming for a recursive relation that depends on separate input (and not only on indices).
6) I could explain why dynamic programming with my evaluation order solves the same problem as does the recursion.
7) I could implement a dynamic programming algorithm that is presented as pseudo code, in some programming language.
8) I could determine from the problem statement whether a problem should be recursively solved , without it being stated anywhere, if I got some time to investigate it.
9) I could construct a dynamic programming algorithm that solves a problem presented to me in natural language, without recursion.
10) I could construct the solution to a problem if I have a dynamic programming algorithm that determines that the solution exists.

Along with the instructions, the values 0, 25, 50, 75 and 100 were interpreted in words, and the students had the instruction to mark 0 where they had never heard of some concept before. The students were also introduced to the scale and to using it by some simple examples, which had nothing to do with the course: "I could lift X", where X was allowed values like "a pen", "my laptop", "this chair" and "the lecturer".

The results of the self-efficacy surveys can be seen in Table I. Only responses from students who handed in both surveys are included. For some of the items, the students had high confidence in knowing them already at the beginning of the first lecture on the topic, especially item 4. We decided to just look for increases or decreases of at least 25, which had distinctly different interpretations according to the instructions. For comparison, we also present results where students had changed their answers with 50 or more. Items 1, 3, 7 and 8 also had rather high average values at the first lecture. Naturally, only a few students could increase their confidence a lot on these items. For all but the first two items, it also occurred that some students (at most 8) *decreased* their confidence in their ability, for all of these except from 7 and 10, also when the threshold was 50. The decrease in confidence could suggest that student actually became less skilled in tampering with certain tasks after the teaching and learning activities. This, though, would be quite an achievement of negative sorts for the teaching, and seems unlikely. Another interpretation is that students became aware of details or whole dimensions they had previously not seen, and hence became less certain that they could cope with some tasks. For instance for item 6, which is a milder version of proving correctness, some students might not have seen the need to prove this, and underestimated the difficulty of the task. On the online course survey, some of the comments about the course and the homeworks reveal that proving correctness is still considered difficult. Also, after having tried some of the tasks in the survey for real, some might have more realistic estimates of their abilities.

For items 2, 5, 6 and 10, more than half of the responding students increased their confidence, and about 30 percent increased it by at least 30 %. It seems that what students perceived that they learned, was to recognise a dynamic programming approach, to modify a recursive approach to a dynamic programming approach by the means of determining an evaluation order, and to explain why this still solved the same problem as the recursive algorithm, and also to modify dynamic programming algorithms to save more history and answer with more detailed answers. This, the tenth item, might have been a distinction that students had not considered before, and not really thought of.

The items 5, 9 and 10 had the lowest average scores at both occasions, but the similarly low scores on item 2 and 6 at the first survey had increased more the second survey. Still, as mentioned before, item 6 has special characteristics and could be investigated more. According to these surveys, out of the investigated items, the students perceive it most difficult to find a suitable evaluation order, construct a dynprog algorithm from scratch, and modifying an algorithm that answers "42" to one that answers "the best value 42 is obtained by making these decisions", and also to argue correctness, although on this item many students improved their (self estimated) abilities.

B. Course survey at exam

The results from the previous section also include what students believe they know after the part of the course that dealt with dynprog. The responses from the students who got the closed form questions at the final exam survey are presented in Table II. Students were in general very positive to the "clicker questions", but also towards the visualisations and the computer lab assignment, a little more positive in general speculation than in their particular case on whether these activities contributed to learning dynamic programming. Particularly beneficial, according to the responses, is working with the computer lab assignment. No one complained about the lack of privacy of the clicker votes, neither at the exam survey nor on the anonymous online survey.

1. Was the pedagogical purpose of the activity clear?

	yes	questionable	no
clicker questions	90%	6%	1%
visualisations	81%	10%	1%
computer lab	79%	14%	1%

2. Did you find the activity meaningful?

	yes very	yes some-what	not parti-cularly	not at all
clicker questions	59%	36%	3%	0%
visualizations	40%	40%	10%	1%
computer lab	54%	40%	3%	0%

3. Did you learn dynamic programming by working with the activity?

	yes	no
clicker questions	66%	17%
visualizations	51%	24%
computer lab	87%	9%

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

	yes	no
clicker questions	76%	9%
visualizations	73%	6%
computer lab	90%	4%

5. Did the activity add something to the course?

	yes	no
clicker questions	94%	1%
visualizations	71%	6%
computer lab	96%	1%
self-efficacy surveys	17%	39 %

TABLE II
ACTIVITY SURVEY RESULTS. (DON'T KNOW OR BLANK OMITTED.)

Where did you learn to...
1. ...tell if an algorithm is based on dynprog?
2. ...tell if a problem can be tackled by dynprog?
3. ...construct a recurrence relation for a simple problem?
4. ...choose an evaluation order given the recursion?
5. ...construct a solution given a dynprog algorithm that finds the optimal value?
6. ...motivate the correctness of a dynprog-based solution to a problem?

	already knew or learned by myself	lectures	tutorial sessions	lab theory assignments	lab	homework 1	still haven't learned this
1.	18%	57%	30%	39%	42%	19%	0%
2.	16%	50%	30%	34%	37%	24%	1%
3.	72%	24%	19%	9%	6%	16%	1%
4.	26%	49%	29%	26%	25%	22%	4%
5.	11%	37%	23%	15%	32%	22%	15%
6.	6%	43%	17%	11%	15%	29%	24%

TABLE III
WHERE DIFFERENT TASKS WERE PERCEIVED TO HAVE BEEN LEARNED. (N=148)

Activities attended	mean grade hw1	hw2
>4	1.9	2.4
<2 and ≤ 4	1.2	1.5
≤ 2	0.68	0.84

TABLE IV
COMPARISON OF STUDENT PERFORMANCES AT HOMEWORK 1 AND 2 (HW1 AND HW2) GROUPED BY THE NUMBER OF ACTIVITIES ATTENDED.

C. Home assignment cover page

A more detailed view on what students on an earlier occasion believed that they learned, and where, is presented in Table III. These are the percentages of students (148 in total) who handed in the homework assignment. On the cover page they had to complete this matrix and some questions on what activities students had attended. They were allowed to mark several options, for instance, they could have learned to tell if an algorithm was based on dynamic programming both on lectures and at home.

Judged from this, all of the activities contributed to learning, but construction of recurrence relations for simple problems is not considered as hard, or at least, not considered to require the teachers' attention. Just as in the self-efficacy surveys, modifying an algorithm to output the construction of the solution rather than only an optimal value, is still difficult to some of the students, here 15 %. A quarter of the group found it difficult to argue correctness at this point. Since the question was not posed identically on the self-efficacy survey, the answers are not totally comparable. The fourth row in Table III shows that only 4 percent judged that they still could not find an evaluation order. This suggests that students who are not too certain that they could perform this task, still do not respond with "I still haven't learned this", or that they later re-evaluated their ability to choose an evaluation order.

Concerning the homework *results*, and any impact our methods might have had, in Table IV, the mean grades (where A counts as 5, and linearly down to 0 for F) for homework 1 and 2 are shown for the groups of first time students who had attended different number of activities, where the lab assignment was counted as 1.5 if submitted early. Homework 2 was on complexity. The grades are usually similar on both homeworks, but this year many students handed in a less efficient solution (based on dynamic programming) for one of the problems, and hence did not get the highest grade for this. The most ambitious students might have read too much into our questions on dynprog. These results are mostly telling that active students get higher grades, and show no special benefit for dynprog problems.

V. DISCUSSION

A. Student activity and lots of evaluation intervention

To have non-anonymous surveys is often not recommended, but the concept of *self-assessment* in teaching is not usually associated with anonymity. Self-assessment can be used as a means of facilitating learning. There is a possibility that one of the learning activities we provided this year was the frequent surveying and self-assessing. Some students indicated this, so this perspective was included in the exam survey. Most students, however, considered it an alien thought that their completing a lot of surveys had contributed to them learning dynprog. Even so, this type of questions can make students

reflect on their own abilities, and either help (a minority of them) to learn, or reduce administrative work load on the teachers. For instance exam wrappers (to be completed upon recovery of their graded assignment) are suggested to be used for such reasons by some institutions[1].

We plan to continue distributing these surveys, to be able to monitor possible effects by altered teaching methods, and for the possible pedagogical gains. Although the criteria for each grade are specified in the course information, the questions on what students felt certain that they could do and the abilities we wanted to know if and when they had learned also contribute to communicating the teachers' aims to students. If given a list of abilities, students can see the message that these are distinct abilities that the teachers value.

VI. Conclusion and future work

The contribution of this paper was to show which abilities (out of a limited sample) students find more difficult to learn or to master, what they feel confident that they know, and where they learned the things that they know. For Q3, we can conclude that a variety of activities was good. Both mandatory activities, like the homework and the lab assignment, and optional ones like lectures, tutorial sessions and visualisations were considered helpful for learning. Regarding Q2, most students are confident that they can recognise dynprog, explain what it is, construct simple recurrence relations and implement dynprog algorithms. They are less certain that they can find a good evaluation order for a recurrence relation, solve a problem from scratch or motivate correctness, which constitute our answers to Q1. Compared to what the teachers expected, this is both similar and different. The correctness part was expected, but fewer students than expected experienced difficulties with finding a recurrence relation. Further, other, or more detailed questions can be asked to find out more about this.

Next time this course is given, the prototypes will be refined and more emphasised. Correctness and writing pseudocode (i.e., describing ones algorithm on an appropriate level of detail) will be the next topics to get special attention.

References

[1] H. Kahney and M. Eisenstadt, "Programmers' mental models of their programming tasks: The interaction of real-world knowledge and programming knowledge," in *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, vol. 4, 1982, pp. 143–145.

[2] H. Kahney, "What do novice programmers know about recursion," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '83. New York, NY, USA: ACM, 1983, pp. 235–239. [Online]. Available: http://doi.acm.org/10.1145/800045.801618

[3] G. Ford, "A framework for teaching recursion," *SIGCSE Bull.*, vol. 14, no. 2, pp. 32–39, Jun. 1982. [Online]. Available: http://doi.acm.org/10.1145/989314.989320

[4] F. Turbak, C. Royden, J. Stephan, and J. Herbst, "Teaching recursion before loops in cs1," 1999.

[5] T. L. Scholtz and I. Sanders, "Mental models of recursion: investigating students' understanding of recursion," in *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, ser. ITiCSE '10. New York, NY, USA: ACM, 2010, pp. 103–107. [Online]. Available: http://doi.acm.org/10.1145/1822090.1822120

[6] B. Haberman and H. Averbuch, "The case of base cases: why are they so difficult to recognize? student difficulties with recursion," in *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, ser. ITiCSE '02. New York, NY, USA: ACM, 2002, pp. 84–88. [Online]. Available: http://doi.acm.org/10.1145/544414.544441

[7] D. Ginat and E. Shifroni, "Teaching recursion in a procedural environment–how much should we emphasize the computing model?" in *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, ser. SIGCSE '99. New York, NY, USA: ACM, 1999, pp. 127–131. [Online]. Available: http://doi.acm.org/10.1145/299649.299718

[8] O. Muller, "Pattern oriented instruction and the enhancement of analogical reasoning," in *Proceedings of the first international workshop on Computing education research*, ser. ICER '05. New York, NY, USA: ACM, 2005, pp. 57–67. [Online]. Available: http://doi.acm.org/10.1145/1089786.1089792

[9] O. Muller and B. Haberman, "Supporting abstraction processes in problem solving through pattern-oriented instruction," *Computer Science Education*, vol. 18, no. 3, pp. 187–212, 2008. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/08993400802332548

[10] B. Haberman and O. Muller, "Teaching abstraction to novices: Pattern-based and adt-based problem-solving processes," in *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, 2008, pp. F1C–7–F1C–12.

[11] O. Muller, D. Ginat, and B. Haberman, "Pattern-oriented instruction and its influence on problem decomposition and solution construction," *SIGCSE Bull.*, vol. 39, no. 3, pp. 151–155, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1269900.1268830

[12] J. E. Caldwell, "Clickers in the large classroom: current research and best-practice tips," *CBE Life Sci Educ*, vol. 6, no. 1, pp. 9–20, 2007.

[13] J. J. Randolph, "Meta-analysis of the research on response cards: Effects on test achievement, quiz achievement, participation, and off-task behavior," *Journal of Positive Behavior Interventions*, vol. 9, no. 2, pp. 113–128, April 2007.

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

[15] F. Pajares and M. D. Miller, "Role of Self-Efficacy and Self-Concept Beliefs in Mathematical Problem Solving: A Path Analysis." *Journal of Educational Psychology*, vol. 86, no. 2, pp. 193–203, 1994. [Online]. Available: http://www.eric.ed.gov/ERICWebPortal/detail?accno=EJ490260

[16] P. Iannone and M. Inglis, "Self efficacy and mathematical proof: are undergraduate students good at assessing their own proof production ability?" in *Proceedings of the 13th Conference on Research in Undergraduate Mathematics Education*. Conference proceedings, 2010, february 2010. [Online]. Available: https://ueaeprints.uea.ac.uk/16104/

[17] V. Ramalingan and S. Wiedenbeck, "Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy." *Journal of Educational Computing Research*, vol. 19, no. 4, pp. 367–381, 1998.

[18] P. Askar and D. Davenport, "An investigation of factors related to self-efficacy for java parogramming among engineering students," *Turkish Online Journal of Educational Technology*, vol. 8, pp. 26–32, 2009.

[19] E. Enström, G. Kreitz, F. Niemelä, P. Söderman, and V. Kann, "Five years with kattis - using an automated assessment system in teaching," in *Proceedings of the41st ASEE/IEEE Frontiers in Education Conference, Rapid City, SD*, ser. Frontiers in Education, IEEE, Ed. IEEE, ISBN: 978-1-61284-467-1, 2011.

[20] A. Bandura, *Self-Efficacy Beliefs of Adolescents*, ser. Adolescence and Education. Information Age Publishing, 2006, ch. 14: Guide for constructing self-efficacy scales, pp. 307–337.

[21] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Kratwohl, *Taxonomy of educational objectives Handbook 1: cognitive domain*. London: Longman Group Ltd., 1956.

[22] P. Crescenzi, E. Enström, and V. Kann, "From theory to practice: Np-completeness for every cs student," in *ITiCSE '13: Proceedings of the eighteenth annual conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2013.

[1] http://www.duq.edu/about/centers-and-institutes/center-for-teaching-excellence/teaching-and-learning/exam-wrappers