DEVELOPMENT OF COMBINATORIAL SKILLS FOR UNIVERSITY STUDENTS IN COMPUTER SCIENCE

Galina Dimitrova Momcheva-Gardeva*
Stoyan Nedkov Kapralov**
*Department of Computer Sciences
Varna Free University 'Chernorizets Hrabar', Chaika resort, Varna 9007
**Institute of Mathematics and Informatics
Bulgarian Academy of Sciences, 8 Acad. G. Bonchev, Sofia 1113
*gmomcheva@gmail.com, **s.kapralov@gmail.com

ABSTRACT

This article aims to represent two types of instruments for building up combinatorial skills for university students in Computer Sciences: applying schemes of designing soluton to a given problem and using non-traditional tasks, such as isomorphic problems, puzzles and topological order of problem-components. Combinatorial skills are the basis of building problem solving culture and the development of combinatorial thinking is an appropriate step to the development of computational thinking for computer scientists. The article folows the ideas of Zeitz [23]that 'problem solving can be taught, and can be learned'.

INTRODUCTION

Nowadays the role of combinatorial thinking for solving business and everyday problems is quite important. The problem of existing combinatorial thinking is multidisciplinary, concerning Mathematics, Computer sciences, Biology, Chemistry, Arts, etc.

Today combinations in business are the basis of entrepreneurships. Moreover, combinatorial thinking is a regular requirement in job offers. But how to achieve it?

According to Daskalov [5], the training of managers should begin with the skill of combining. 'Employees with a combining mind (i.e. a mind easily adaptable to the situations and circumstances) climb the professional ladder faster'.

In the area of computer sciences, specialists according to the executive manager of BASSCOM (association of leading Bulgarian software development companies) Maya Marinova: 'To sustain our success, the educational system should

provide people capable of innovative thinking, with the ability to analyze problems in depth and with respect of other problems, people that could adapt quickly to various technological and multidisciplinary environments, and may work in multinational teams.' [24]

According to psychologists like Runco [21], the generative capacity of the human brain might specifically result from combinatorial processes. Conventional thinking, as for the Bono, is based on search and discovery. Parallel thinking is based on creation and construction. [2]

The emphasis in this article is on the role of combinatorial thinking for programmers and the process of development of combinatorial skills for computer science students.

In the courses on Data Structures and Algorithms (DSA) we usually aim to teach some algorithms and their usage and we usually face some problems of misunderstanding and lack of motivation.

The experiment that we have done is to explore the ability to solve combinatorial problems of 15 students studying Computer Sciences (CS) and our general conclusions are: they do not have an idea of problem solving. The only idea that is clear is to divide a problem to smaller parts. They can solve only familiar problems and cannot explain why they have done that. But they demonstrate without any doubt that they are concerned about their ability of problem solving. And finally they assert that are familiar with combinatorial objects, but cannot recognize simple examples and connect them to the term itself.

We have been studying about combinatorial objects but not about combinatorial style of thinking and the development of combinatorial skills that is of crucial importance for our students, firstly as humans and also as professionals.

In our opinion, practically there are two types of university students (according to their career planning): programmers and managers in software companies or project managers. There is also another classification of them: those how are content with the topics from school and those who are not.

For the career of students that will be programmers, it is important to know how to do things (particular programming tasks involving code generation), and for the second group it is more important how to think, how to generate alternatives. Because they will be problem solvers and decision makers.

The most important skills in learning is learning how to combine, how to think of, how to develop personal strategies for problem solving, how to make decisions and how to code.

Improving the understanding of combinatorial structures gives us the tool for checking the correctness of the given algorithm or to compare its efficiency with another one or to construct a new one and experimentally prove its correctness.

As for the students, a future development combinatorial style of thinking is important in order to develop a general heuristic thinking scheme.

COMBINATORIAL TASKS AND PROBLEMS IN MATHEMATICS AND COMPUTER SCIENCE

Piaget and Inhelder [18] defined combinatorial reasoning as ‘the capacity to determine all the possible ways in which one could link a given set of base associations with each other’.

In [23] Combinatorics is defined as the study of counting. As a matter of strategy to decide such a problem is mentioned that 'good combinatorial reasoning is largely a matter of knowing exactly when to add, multiply, subtract, or divide. '

Predominated methods for solving combinatorial problems from the International Olympiad in Mathematics, selected in [11] are mathematical methods (by induction, by contradiction, algebraic) typical for deciding combinatorial tasks for counting. So do in other books with combinatorial problems solving in math classes (in high school or in university).

The strategies and methods used in mathematical tasks could be used in CS tasks too, as careful experimentation with small numbers that is a crucial step [23] but they couldn’t be generalized.

Janačkova, Janaček [9] led an experiment for solving isomorphic problems and they recognized 11 strategies that had been used: strategy of exhausted subsets, group strategy, same number of permutations in groups, strategy of symmetry, …

Actually, a lot of combinatorial ideas persists and precedes official education in Combinatorics. Here we mean boolean logic and their usage in conditionals or using nested circles. Of course, we could not study everything in books, e.g. Lipski [14].

In [7] Grozdev and Garov suggested the following topics in Combinatorics that have been covered for competitors in informatics: generating, coding and decoding combinatorial objects like permutations, combinations and compositions (with/without repetitions) and partitions (set, number).

As for the combinatorial object that could be taught to university students, we suggest the following: permutation, combinations, compositions; partitions; Ferrer diagrams; Young tables and Gray codes (sequences). And also general operations over combinatorial objects: counting, numbering (enumerating, ranking/unranking) and generating. Some of these topics are covered in classes in Discrete Mathematics but students do not write any program - they only cover the idea.

We extended the content to the described above because of the idea that the more combinatorial objects there are, the more combinatorial ideas the student will observe. And for these topics, training observation is very important. The bigger list of combinatorial objects suggests to us more connections between different combinatorial objects and their properties. It is a natural combination between them that we can use.

According to our experience, the process of learning how to problem solve has to begin at the very beginning (of every education). Solving tasks of the same type in repetition is not a good strategy to achieve understanding by university students.

So, we can apply at every step the schemes represented in the next section of this article. For example, use some tool for exploration like Excel, Mathematica, libraries, packages or applets.

An analogy for that could be seen in the article by Abramovich [1] where he said that while making connections among geometry, number theory and combinatorics, students can experience mathematics as a whole. They explore combinatorial objects through spreadsheets. Similarly, by making connections among different representations – namely spreadsheets, manipulative and visual imagery – students can develop a richer understanding of permutations and combinations.

The education on this topic combines the content of several disciplines, such as Probability and Statistics and Discrete Mathematics. The results of this education are used in data structure and algorithms. It will be very helpful if at least labs in DM for the university students write the programs implementing described concepts or make explorations. But for most of the students (because of their poor mathematical education in high school) it is impossible. So we suggest that we let the students 'touch' the problems in some way. Using computers to do so was based on the idea of computational thinking, developing which is quite important nowadays.[3]

The general methods for solving such problems are founded on the use of combinatorial principles, solving recurrent relations, etc. But our students come to the conclusion that studying formulae is everything.

Experimental interactive computer modules are developed and used for teaching Combinatorics, Probability and Statistics at high schools [10]. The authors explore the effectiveness of experimental education in Combinatorics, probabilities in high schools in raising the activity in high school students but they do it in IT classes, not in Math classes.

There is a difference between the problems in competitions in mathematics and informatics [7]. Each mathematical problem carries information and in this sense it is knowledge. Thus, the tool acquired through it can be used for solving other mathematical problems. In CS, the solution, i.e. building an algorithm and a computer program, is an essential part. The component problems in mathematical problems, i.e. the problems whose solutions are part of the solution of the main problem, can be interpreted as algorithmic knowledge.

The difference between the mathematical and the CS approaches to formulate combinatorial tasks is grounded on the idea that if we could not find a decision at once, this is because we did not know it. We could find at least a partial solution taking in mind some other knowledge.

The next level of abstraction of these problems is where the combinatorial object is not so obvious in the text. This is the first step to create a problem situation – to hide the combinatorial objects.

SCHEMES FOR PROBLEM SOLVING OF COMBINATORIAL PROBLEMS

We know that Aristoteles noticed that reasoning corresponds to schemes. The books of Polya [19] for plausible reasoning, and the famous book 'How to solve it?' [20] (look for a pattern, draw a picture, solve a simpler problem, use a model, work backward, use a formula, be creative,…) Schemes for analysis of solving problems of Papus and Euclid are well familiar to us, too. Today the schemes are updated by Ziet [23] and many others, especially thoroughly elaborated for business purposes and a lot of positive ideas appear in different publications, too.

Ginat [6]represented an approach of 'elaborating divergent thinking in algorithm design, while capitalizing on erroneous solutions… initial faulty solution are carefully examined, and their falsifying inputs and characteristics are used for creative reasoning that yields fruitful outcomes.' It is quite a helpful approach, but we could not implement it in the teaching and learning process all the time.

The reason is that the students will get content with this approach and a repetition of similar actions usually stops thinking. But it is really helpful, so we can create a scheme from this approach to be applied in our SDA classes 'to apply consequently every design technique to the problem and verify if this works or not'. This process takes time but it deserves to apply. For some educators, this approach is their methods of teaching.

Another interesting idea is presented in [4]. They take a verbalization of actions while solving a problem for a student as an instrument in understanding students' problem solving process.

Grozdev&Garov [7] offer eight steps for solving CS problems (algorithmic solution with computer) on the system of supportive problems in the preparation for participation in informatics olympiads:

- the conditions of the problem
- constructing a model of the problem
- developing an algorithm for solving the problem
- check the correctness of the algorithm
- analyzing the algorithm and its complexity
- transferring the algorithm into a programming language with the use of software
- checking and testing the program
- writing documentation

The next scheme that we want to pay attention to is the works of Anany Levitin [12]. He defines an algorithm as a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time. So we can consider algorithms to be procedural solutions to problems.

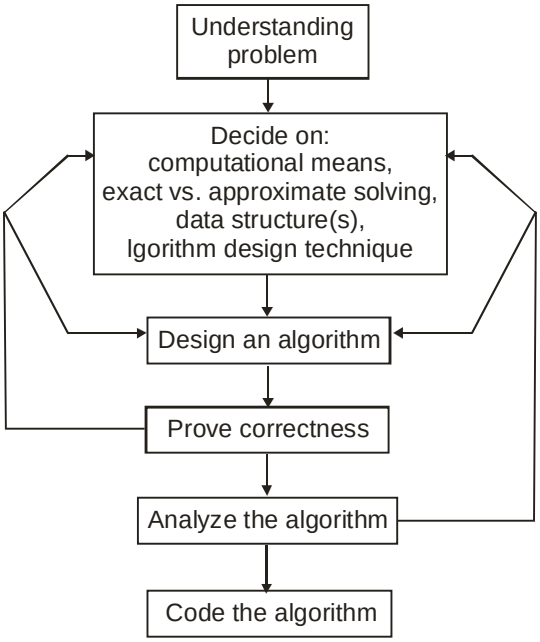

Figure 1.

We accept this scheme as a macro scheme. In order to teach students how to problem solve, we represent more partial and practical schemes for educational purposes – 'schemes instances' in applying which we could solve specific classes of problems. For the purpose of this article – combinatorial problems.

The schemes and steps mentioned above do not include finding a solution by experimenting with the use of a computer, which in our opinion is crucial in problem solution training.

The authors believe that representing schemes to the students will increase their success in problem solving because they will have new frames - models of reasoning.

The quantity of tools and strategies for general problem solving is increased, while the knowledge to solve combinatorial problems is developed.

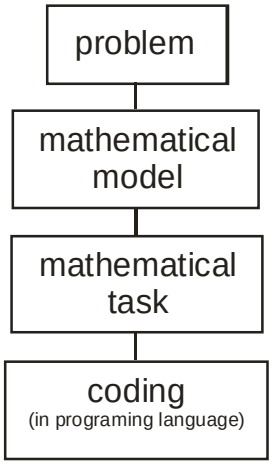

Figure 2

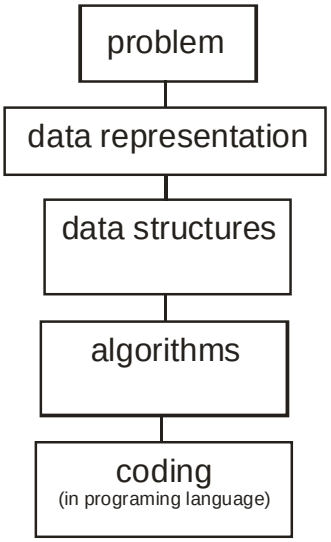

Figure 3

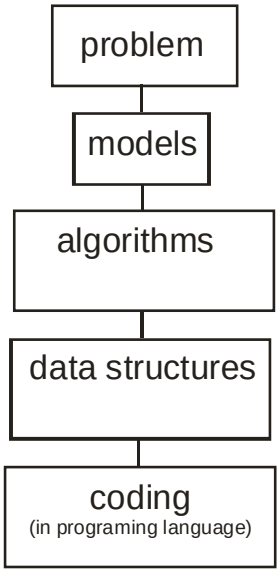

Figure 4

Figure 2 shows the so called ‘standard problem solving scheme’ where every tasks is modeled to a mathematical one and is solved though mathematical methods. We have mentioned that not every student in the university has the appropriate mathematical literacy and that is why this scheme is doomed to failure.

As for the process of combinatorial tasks, this scheme is inappropriate when we try to solve optimization combinatorial problem, or such instance of a task where there is not enough space for the program to be executed. But in many problems or for many students this scheme works.

During the writing of this article we made a little experiment with a target group of students and most of them preferred the scheme in Figue 3. Their explanations varied from intuition to finding similarity with program source.

Here representing data is not choosing data types or defining variables but preprocessing of data input. Figure 4 represents a scheme that is typical of such types of tasks for problem solving in DSA classes.

The restrictions in DSA problems are crucial. By changing the time or space limits, we could solve quite a different problem. There are two types of building problems according to their restrictions. In one of them – the restrictions made the problem easier, the problem becomes the instance of the original one. The second type comes when adding restrictions to standard problems. Such a type is the optimization one.

Developing problem solving by adding restrictions to ‘standard’ problems is a way to create problems and to create new problem situations. Here is a very simple one: from traversing a table (using nested cycles ‘for’), next could be checking whether a square is a magic one, and then complicating with finding the one

missing number in a magic square, finding two, .. more. Generating magic squares or Sudoku.

The last of the represented schemes here is Figure 5, used where we have to find optimal solution. The idea is in a constructive manner to create every possible instance of problem domain and then to explore: e.g. to exclude the odd instances, to find out a dependence, to find out a recurrent relation and reduce it and so on till a satisfying solution has been obtained.

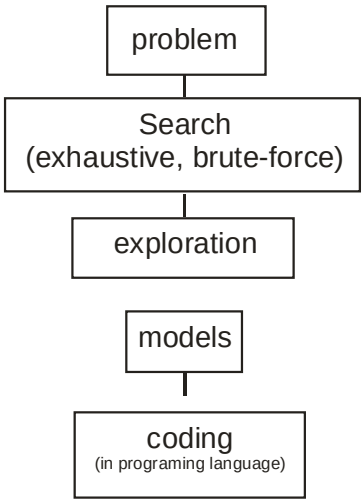

Figure 5

When the problems come to optimal putting some limits in space and time, we could change some schemes by iterating these elements (blocks and branches) that lead us to take new decision. Something like backtracking in the scheme itself.
In the experiment, we suggested to students schemes connected to algorithm design, but they were not content with them.

The similarity between problems for competitors in informatics and these for university students in CS (not being CS competitors) is that neither of them have set up strategies for solving problems, cannot apply even penultimate step Zeit [29].

In competitive problems for high school student there is a chance to receive points for partial solutions, for competitions for university students they have a chance to send a problem decision to the server several times (taking penalty for that). Even good enough students are provoked to have a chance.

And then, what about not very good students. This is a data structure and algorithmic discipline that could provoke them to be thinkers not cheaters. But how to do that?

The answer is to allow them to use the computer as a tool – if they are not good programmers – then they could use libraries, or specialized software [17]

because the main point is to learn to observe, to read patterns, and not to try to program.
So, the next two schemes are connected with the process of learning, or we could say the process of solving the problem how to organize teaching and learning.

This is an optimal scheme according to mathematical reasoning, but we go back to recursive or exhaustive searching and reasoning to apply the strategy of dirty hands in trying to solve the problem with more than best in programming students.

So, instead of using the scheme in Figure 6, where we reach the mathematical model, and then to program our solution in Figure 7.

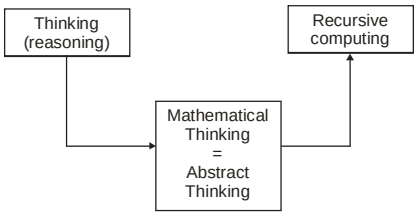

Figure 6

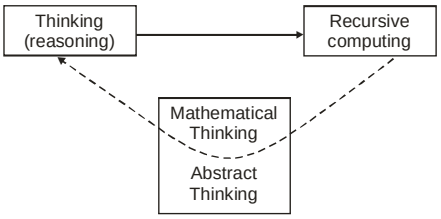

Figure 7

And after the success of the solving process we could do the reverse process to go backward to abstractions. This is the way of representing the students abstractions, too.

The generating idea (at the means of innovation) is ‘to receive a solution’ and then to optimize the solution. So the process of generating combinatorial objects is of very big importance. Then the optimizing process depends on two skills: combinatorial thinking, and analyzing data. Teaching how to do this with algorithms shows a path how to do in thinking.

[3] The synthesis strategies of parallel algorithms are basically different from the traditional procedures of algorithmic synthesis for conventional computers. Designers have three options: to parallelize an existing consecutive algorithm, to create a new parallel algorithm or to make an effective combination of existing consecutive and/or parallel algorithms for solving the problem.

But we can try to parallel programs in Java because of threads. For example, a combinatorial problem of generating permutations for bigger n is possible to implement by parallelize the process. It is possible to use generations through the algorithms of numbering. So, every thread could generate different ranges from the output domain.

NEW AND NOT SO TYPICAL TYPE OF TASKS IN DSA CLASSES

As we know, in managing DSA education we could try to give a lot of time for experimenting: by hand; using spreadsheets; using specialized software and

also to apply schemes for finding paths: drawing; using formulas; using combinatorial reasoning; using moreover seeing patterns.

In our experimental group, some of these schemes were followed, for example some people drew, others enumerated all possibilities for a given problem. We have to teach people how to transform tasks, the technique [12] 'transform and conquer'. And we could not remember because of different style of learning one prefer numbers, while another one figures.

In addition to schemes of solving combinatorial problems and general schemes, we recommend the following:

- using in education isomorphic problems
- solving topological ordered sequences of problems-components
- using puzzles

Manuel [15] represents four groups of equivalent combinatorial counting problems that can be used in the classroom and according to us could be used in generating databases for creating tests in web. He called isomorphism problem equivalence or structural identity.

In our experiment, to see the input level of our students responding to the problem, solving in data structure and algorithms, we see that students finding out the isomorphism but they solve tasks and compare the results. They cannot recognize the model. This is a straight proof of our suggestion that in every data structure and algorithms classes we have to do tasks in which recognize models.

We used in the test 3 of this four isomorphic combinatorial problems [9] 'The town problem', 'The Ice Hockey Problem', 'The Pigeonhole Problem' and 'The Line Problem'.

The students tested in our small experiment could not solve the tasks for finding the ordering of problems. The idea of solving tasks in groups is another innovative idea that is waiting for us. The next step in learning about isomorphic tasks is to insist on students to create such one. Involving in this process, increases the ability to think and also increases combinatorial reasoning.

Grozdev, Chehlarova [8]. The game develops the skills required for processing parallel information, managing different systems and switching between them, as well as skills required for handling the unknown.

The role of using puzzles in CS education is discussed [13], [22], tasks for chess, or on chess board – they developed spatial thinking, technique for noticing two things at once and organize movements on the chess table.

The learning path for problem solving of university students in CS comes through combinatorial compositions-decompositions. Learning Combinatorics gives the learner the ability of combinatorial reasoning and thinking. Taking this in advance, we have the appropriate tools for making compositions of algorithm (e.g. they are quite different in mathematics or in art). This is the ability used in problem solving because it makes easier the problem of decomposition if you are familiar to compositions. And most of the problems waiting to be solved need the ability to make adequate decompositions (quite useful today in pattern recognitions).

CONCLUSION

The combinatorial thinking is important for generating alternatives – important in heuristic thinking, testing programs, verifying experimentally new algorithms. Especially important for programmers and CS specialists is the exhaustive search (combinatorial search) which is in the basis of every search.

An interesting comparison between exhaustive and reverse search is done by J.Nievergelt [16]. He uses the research of Avis and Takuda where a set of conditions is presented that enable graph traversal without auxiliary data structures, such as stacks, queues, or node marks. The amount of memory used for book-keeping is constant, i. e. independent of the size of the graph. Their reverse search is a depth-first search (DFS) that requires neither stack nor node markers to be stored explicitly – all necessary information could be recomputed on the fly. The illustration of reverse search by Nievergelt is done by an empirical study of enumerating the k shortest Euclidean spanning trees.

A good education in Combinatorics and combinatorial thinking is a solid base for developing heuristic schemes of solving problems.

The continuation of our work is to prepare the scheme for combining schemes we mean as the composition of basic schemes and develop heuristic schemes for complicated optimization combinatorial problems.

The combination between strategic schemes and design techniques and programming languages features is still poorly explored.

The next branch of future development is networking. These schemes are personal, but they could be applied to a pair in pair programming, or to a team in project work. This is also a matter of considerable interest for authors. Once developed, such schemes could be easily virtualized.

Achieving the first one depends on the personal characteristics, that is why they are suitable for learning.

REFERENCES

[1] Abramovich, S., A. Pieper, ‘Fostering Recursive Thinking in Combinatorics through the Use of Manipulatives and Computing Technology’, The mathematical Educator, 2000

[2] Bono, E., ‘Parallel Thinking’, Penguin Books Ltd, 1990

[3] Borovska, Pl., M. Lazarova, ‘Parallel informatiion processing’, Ciela, 2007

[4] Chinn, D., C. Spencer, K. Martin, Problem solving and students Performance in Data Structures and Algorithms, ACM SIGSE, vol 39, no3, 2007

[5] Daskalov, N., ‘The combinations in business’, Abagar, Veliko Tyrnovo, 1997

[6] Ginat., D., Learning from Wrong and Creative Algorithm Design, ACM SIGCE, vl.40, no 1, 2008

[7] Grozdev, S., K. Garov, ‘On the system of supportive problems in the preparation for participation in informatics olympiads. Combinatorial objects and

algorithms' (306), Mathematics and education in mathematics, 2008, Proceedings of the Thirty Seventh Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, April 2-6, 2008
[8] Grozdev, S., T. Chehlarova, 'Cubes and Consructions', Association for the development of Education, Sofia, 2007
[9] Janačkova, Janaček, 'A classification of strategies employed by high school students in isomorphic combinatorial problems', TMME, Vol 3, pp 128-145, 2006
[10] Karashtranova, E., E. Stoimenova, 'Study the effects of teaching the theeory of combinations, the calculus of probability and statistics in secondary schools.', Mathematics and education in mathematics, 2008, Proceedings of the Thirty Seventh Spring Conference of the Union of Bulgarian Mathematicians, Varna, April 2-6, 2007
[11] Kolev, E. 'Combinatorial problems', Union of Bulgarian Mathematitions-Varna, 2003
[12] Levitin, A., 'Introduction of The Design & Analysis of Algorithms', Pearson Education, 2007
[13] Levitin, A., M. Papalaskari, 'Using puzzles in teaching algorithms', ACM, SIGCSE 33, 2002
[14] Lipski W, 'Combinatorics for programmers', Mir, 1988
[15]Manuel R., 'An introduction to problem equivalence in Combinatorics', ACM SIGSE, vol 40, num 3, 2008
[16] Nievergelt J., Exhaustive Search, 'Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power', SOFSEM 2000, LNCS 1963, pp. 18–35, Springer-Verlag, Berlin 2000
[17] Pemmaraju, S., S. Skiena, 'Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica', Cambridge University Press, 2003
[18] Piaget, J. & Inhelder, B., 'La genèse de l'idée de hasard chez l'enfant' [The origin of the idea of chance in children], Paris, 1951
[19] Polya, G., How to solve?, Narodna prosveta, 1972
[20] Polya, G., Mathematics and plausible reasoning, Narodna prosveta, 1976
[21] Runco, M., Creativity, Elsevier Academic Press, 2007
[22] Shasha, D., Puzzles for Programmers and Pros, Wrox Press, 2007
[23] Zeitz, P., 'The Art and Craft of Problem Solving', John Wiley & Sons, 1999
[24] http://basscom.org/ accessed on march 13th, 2009