INTERNATIONAL OLYMPIAD IN INFORMATICS
VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

OLYMPIADS IN INFORMATICS

Volume 8   2014

Selected papers of
the International Conference joint with
the XXVI International Olympiad in Informatics
Taipei, Taiwan, July 13-20, 2014

OLYMPIADS IN INFORMATICS

http://ioinformatics.org/oi_index.shtml
http://www.mii.lt/olympiads_in_informatics

Foreword

In the first years of the Olympiads in Informatics conference a specific theme was set. Since 2009 papers have been accepted on any topic that falls under our remit – along with the occasional one that falls just outside. One way of splitting these papers is into those that relate to olympiads and those that look at the wider teaching of informatics. Many of the conference's earlier papers, when not discussing contest technicalities (tasks, graders, etc…) looked towards the organisational aspects of the events. We are now seeing papers that take as their foundation national strategies for teaching, of which an informatics olympiad is (if at all) a small part.

We are fortunate to deal with a subject that adapts well to new technology. As an illustration consider the International Olympiad in Informatics which has moved from manual grading to automated grading; originally requiring evaluators typing in data by hand, moving to evaluators running semi-automated systems, to our current state of automated grading which can – on a good day – have the results of the event ready before the students have left the contest area. What is more exciting is to see these same grading systems being adapted outside of the contest environment and becoming part of the pedagogical one.

We live in a world containing teachers with a mix of abilities, knowledge and motivations. Schools where there are limits on the amount of time to teach and the resources to teach with. Students who need to work hard on the simplest materials; those who are able to rush ahead to the complex; those who want to study the esoteric. The last few years, especially through the internet, have seen an explosion of courses that are available outside of the usual teaching environment. Again – we are fortunate to deal with a subject that adapts well to new technology. One where automated systems can give accurate feedback to students. One where, without specialist equipment, students can study at their own rate.

The development of grading and teaching environments within our community, and their spread to the wider teaching community, is one of our strengths. There is now a real choice of several strong environments that we have created and made available. This volume contains several papers discussing such systems; their technical aspects but also how they can be used to teach. It is not just our top students who are benefiting from the olympiads; not just those who compete in our contests. We are making a real difference out there.

As always thanks are due to all those who have assisted with the current volume – authors, reviewers and editors. A lot of work goes, not only to the writing of the papers, but to an extended period of review and correction and, in several cases, translation.

Peer reviewing all of the papers takes a significant amount of time and work and special thanks should be given to those otherwise unsung reviewing heroes.

Last, but by no means least, particular thanks are due to the organisational committee for IOI'2014 in Taiwan without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

3

Presenting Computer Science Concepts to High School Students

Tim BELL[1], Caitlin DUNCAN[1], Sam JARMAN[1], Heidi NEWTON[2]

[1] *University of Canterbury, New Zealand*
[2] *Victoria University of Wellington, New Zealand*
e-mail: tim.bell@canterbury.ac.nz, caitlin.duncan@pg.canterbury.ac.nz,
saj77@uclive.ac.nz, heidirosenewton@gmail.com

Abstract. Computer science at high school often focuses on programming, but a broader view of other areas of computer science has key benefits for both writing programs that are more efficient and making more theoretical concepts more accessible to those who do not find programming intrinsically interesting. With the introduction of computer science at high schools, a lack of coherent resources for teachers and students prompted the development of the NZ Computer Science Field Guide, an open-source, on-line textbook.

This paper describes the design of the Field Guide, which has fourteen chapters about various topics of computer science. The design includes written text, videos, classroom activities and interactive applications. The need for a broad view of computer science is discussed, and programming exercises to go with the topics are suggested.

Keywords: computer science, high school, curriculum, constructivism, open source.

1. Introduction

Computer science at high school level often focuses on programming, but new approaches, including new curricula in the UK (Furber, 2012), Australia (Falkner *et al.*, 2014) and New Zealand (Bell *et al.*, 2010), are being developed that offer a broader view of the subject, allowing students to delve into topics such as algorithm efficiency, encryption, human computer interaction and computer vision. This broader view has two key benefits: first, it shows programmers how to write programs that are more effective, and second, for those who don't find programming intrinsically interesting, it shows the kinds of things that are done with programming, providing the motivation to learn programming.

For example, in programming competitions students are tasked with problems to solve that must run within a time limit. Writing a program to solve a problem may not be so difficult, but to do it efficiently and effectively can involve bringing to bear ideas from computer science (such as algorithmic complexity and tractability), and a good understanding of computer science principles can enable students to improve their competition code, allowing it to execute faster and fit under time limits set by the judges. Conversely, ideas from computer science (such as data compression or formal languages)

can provide rich domains for programming exercises, and give students more experience thinking about the richness of approaches available to the computer scientist, such as hashing (an idea which can be applied to areas as diverse as searching algorithms, error detection by hash totals, and password encryption by secure hashing).

Although there are literally thousands of resources available that touch on areas of computer science that might be relevant to high school level students (Murugesh *et al.*, 2010), these resources vary greatly in suitability, and tend to occur as one-off examples that can't be used as a coherent body. To address this, we have developed the "NZ Computer Science Field Guide" (referred to here as the NZ CSFG, available at `http://csfieldguide.org.nz`), an open-source, interactive, online "textbook" that introduces a wide range of topics in computer science, without necessarily expecting students to be competent programmers before tackling the range of topics covered. It is a pilot for a wider range of computer science field guides intended for international use in a variety of contexts. The index of the NZ CSFG is shown in Fig. 1, showing the range of topics covered.

The NZ CSFG has initially been developed to support the new computer science standards that became available in New Zealand high schools in 2011 (Bell *et al.*, 2010), but it is intended to be flexible enough to support curricula for other countries, and other initiatives aimed at high school aged students, such as computing clubs and programming competitions. Because it is open source, in principle educators can adapt it to suit their situation. Because it is online it can be accessed by interested students as long as they have internet access, and an offline version is planned so that it can be delivered through other media as well. For example, while the NZ CSFG was initially prepared to support teaching computer science in New Zealand high schools, it was also being used in parallel to support a pilot for a Computer Science Club (`http://computerscienceclub.org`) for students aged around 10 to 15 years old. The club is based on a badge system where students can attain different levels of badges in topics in computer science, with many of the badge topics being associated with topics in the CSFG.
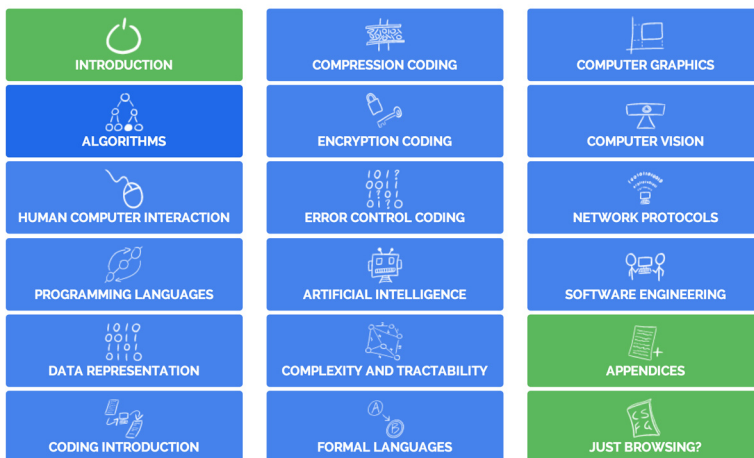

Fig. 1. The front page of the CS Field Guide.

There are other collections of information available that have goals in common with the CSFG, aimed at conveying the ideas of computer science to a high school audience or through interaction and animation. Some of these have provided inspiration and ideas for the CSFG, and others are useful as follow up for students wanting more detailed information. The "Thriving in our digital world" (`http://www.cs.utexas.edu/~engage/`) course uses a similar approach to the CSFG for its teaching material and has engaging interactive presentations, but focuses on just eight topics (four of which are general, such as "Innovations"). The Virginia tech online interactive modules (`http://courses.cs.vt.edu/csonline/`) for teaching computer science cover a range of relevant topics (Balci *et al.*, 2001), although the material again only covers a limited number of topics, and doesn't appear to have been updated since it was developed over 10 years ago. "Babbage's bag" (`http://www.i-programmer.info/babbages-bag.html`) provides a very detailed collection of technical articles on many topics in computing. It is more detailed than most high school students would need, but is valuable as a follow up on particular topics. "CS animated" (`http://www.csanimated.com/`) has interactive activities on computer science, but is more targeted at university level students. The "Computer Science For Fun" (`cs4fn.org`) project provides a very readable collection of short articles aimed at a teenage audience. It is about practical applications of topics in computer science, and has been very successful in getting students interested in computer science (Myketiak *et al.*, 2012), although it doesn't usually go into the level of detail needed to learn the topic, as it is primarily aimed at outreach. The "CS Bits & Bytes" (`http://www.nsf.gov/cise/csbytes/`) project takes a similar approach, with regular up-to-date articles about applications of computer science.

In this paper we discuss in more detail the value of a broader view of computer science for high schools students, and then describe the design of the Computer Science Field Guide, which is intended fill a gap for teaching computer science, and act as a tool to provide teachers and trainers with a rich resource for engaging students with this broad view of the subject. A case study is made using the chapter on algorithms to explain how design decisions were made, and we provide examples of how programming competition exercises could be formulated based on ideas in the CSFG.

2. The Need for a Broad View of Computer Science

It is not unusual for computer science in high school and programming competition environments to be regarded as being primarily about programming, and many on-line resources focus on "coding" (programming). This misses out on a much richer view of the field that explores how well the program might work, such as its efficiency, security, usability, scalability and reliability. In programming competitions, the areas of computational complexity and tractability are particularly important.

There are many definitions of what computer science is, and the approach we have taken covers a number of widely accepted definitions. A key benchmark is the ACM Computing Curricula document (Impagliazzo, 2006), which describes computer science as follows: "Computer science spans a wide range, from its theoretical and algorith-

mic foundations to cutting-edge developments in robotics, computer vision, intelligent systems, bioinformatics, and other exciting areas". This overview has been followed by the 2008 and 2013 Computer Science Curricula (Sahami *et al.*, 2013) which define, respectively, 14 and 18 areas of computer science that should be covered at university level, many of which correspond to chapters in the CSFG. A crowd-sourced definition of computer science can be found on Wikipedia, which (at the time of writing, in April 2014) describes it as "the scientific and practical approach to computation and its applications", and more practically, goes on to list 16 sub-topics, 8 of which correspond to chapters in the CSFG, and most of the rest are touched on at some point.

Of course, computer science can't really be broken into some finite number of disconnected topics, and it is important to emphasise links between topics (e.g. fast algorithms mean that interfaces can respond within the times recommended through HCI principles; search algorithms are required for pattern matching in compression systems; and compression in turn improves network response times which leads to better interfaces).

Many countries are now moving to increase the amount of computer science taught at high school level. This is partly driven by the dramatic shortage of computer science graduates in western countries; teaching computer science in schools can enable students to make better career choices, and a broader view of computer science beyond just programming can attract those who are interested in the bigger picture, rather than programming as an end in itself. The analogy that "Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes" (Fellows and Parberry, 1993) illustrates the value of providing a way for beginner programmers to access the big ideas in computing. Furthermore, it is very easy to write a program that is computationally inefficient (or even intractable), and beyond simple functionality, issues such as the usability and security of programs is also important. Programming competitions can accentuate the focus on barely meeting some requirements, and a wider view of computer science can encourage a healthy view of problem solving techniques, algorithms, mathematical underpinnings, and human factors.

3. Design of the Field Guide

The NZ CSFG currently has a chapter for each of 14 areas of computer science that have initially been designed to match the new New Zealand high school standards released in 2011 (Bell *et al.*, 2010). These areas correspond loosely to the 2008 ACM curriculum (which was the one available at the time that the school standards and NZ CSFG were designed). The ACM curriculum topics are considerably deeper than what is appropriate at high school level, so providing a broad overview of the topics is a challenge particularly doing it in a way that students have a meaningful experience of the topic.

Key features that have driven the design of the CSFG are as follows:

- Open source: teachers all over the world can access it freely, and improve it if they wish. It is intended to be a prototype for a broader range of CS Field Guides for other countries and contexts. The guide is licensed under a Creative Commons At-

tribution-NonCommercial-ShareAlike licence which means that users are welcome to take copies and modify them. The material is produced using the open-source Sphinx system (`http://sphinx-doc.org/`), which was originally designed for writing Python documentation, and works from plain text source files using the reStructuredText format. Much of the writing has been done by volunteers, and the more costly parts of the production have been supported by contributions from industry.

- Interactive: learning activities, games, videos and animations are embedded in the page as students read the book. The interactive components are intended to encourage direct engagement on the part of students, rather than something that is viewed passively.

- Focus on key concepts: Rather than teach a topic in depth, we establish what the key concepts are and make sure they are conveyed. For example, binary representation has some obvious conversion skills that can be learned, but the key concepts are things such as the exponential increase in descriptive power with each bit added; and "algorithms" are often published as a shopping list of many different algorithms, whereas the key concepts are more around how different algorithms can have a non-linear difference in performance, and that some problems are intractable.

- Self-Paced: there is sufficient material that students can learn independently at their own pace, but also work in an environment facilitated by a teacher.

- Teacher support: there is a semi-private version of the guide that has a lot more information for teachers, including solutions to all questions and hints for use in a classroom situation. The Sphinx system used enables conditional use of small units of text, so this makes multiple versions possible from a single source; the teacher guide and student version come from the same source text and future versions for other curricula can also be created automatically by conditionally selecting appropriate sections of text.

- Engaging: drawing on our experience with Computer Science Unplugged, it should keep students engaged. This includes the use of humour (such as fictitious scenarios and tongue-in-cheek comments), "curiosities" (which give tangential examples or stories to create interest), and the use of cartoons.

- Catering to different learning styles: the use of different ways to convey the same information provides each student with multiple experiences of the topic, and some may resonate better with one student's learning style than another.

- Short video "bumpers" that provide enticing introductions to the topic: these videos, which are generally a minute or two long, provide a somewhat humorous but sound overview of the topic raising questions and problems that are addressed in order to provoke curiosity.

- Platform independent: it should be possible to read the guide online and offline, on all operating systems (primarily in a browser), and on tablets and even smart phones. This is achieved by using the Sphinx system, which can output the material on a web site, as PDF, or as an E-book (EPUB and MOBI). The interactive activities are programmed using HTML5 and JavaScript, which will run on most web browsers and E-book readers.

- No programming required: students do not need to be competent programmers before engaging with the material. Many will be learning in parallel, but alternative ways of engaging with the concepts are provided that don't require programming ability.

Fig. 2 shows the beginning of the graphics chapter as an example. The opening video involves the presenter interacting with 3D graphics (including a slapstick attack of the ubiquitous graphics teapot) and the text begins with a "Big picture" section that conveys some of the key motivation for the detailed information that follows. The following sections cover selected specific topics in graphics (in this case transforms, and line/circle drawing algorithms) to illustrate the kind of issues that are dealt with in this topic. Each chapter concludes with a "Whole story" section, which mentions other key topics in the area of the chapter that haven't been covered but are likely to be encountered in further reading or study. Currently chapters typically only cover two or three topics, which are sufficient to illustrate the area, but could be greatly expanded in the future to present other key topics that students might want to look into.

Most chapters contain activities and projects that could be used for assessment purposes; "activities" tend to be smaller formative tasks, whereas projects provide a more in-depth task that is typically used for summative assessment.

The main topics covered were shown in Fig. 1. As discussed above, this list largely reflects the widely-used ACM computer science curriculum for universities; of course, the topics need to be presented in a way that they are approachable for students with only rudimentary programming skills and a high school math background, and so that each one can be covered meaningfully in just a few weeks of a class.

An important task has been to identify the key concepts that would give students an understanding of what issues the topic needs to address, rather than an exhaustive coverage of many sub-topics in the area.

The main concepts identified were as follows:

- Algorithms: understanding that algorithms exist independently of any programming language, and that different algorithms for the same task not only have different running times, but that the difference may not be linear.
- Programming languages: exploring the role of compilers and interpreters in enabling a human readable language to be run on a computer, and the idea that a computer language is implemented by a program itself.
- Human-computer interaction: critically assessing existing interfaces using well established principles including basic psychology, and the idea that the person who implemented the interface is not in a good position to evaluate it critically.
- Data representation: representing numbers, text, images and sound using bits, particularly the relationship between the number of bits used and the quality of the representation, and the exponential increase of range with the number of bits; hexadecimal is a shorthand for binary.
- Coding: changing the representation of data to make it smaller (compression), secure (encryption) and reliable (error control). Compression concepts include lossy vs lossless compression, and the kinds of structures in data that can be exploited to reduce file sizes. Encryption covers the concept of an attack, including approach-

Fig. 2: The beginning of the graphics chapter.

es such as brute-force and known-plaintext attacks, the cryptographic strength of keys, the key exchange problem, and areas of cryptography beyond keeping data confidential.

Error control includes the idea that error detection and correction is possible, and the ability to do this with a high probability of success can be achieved by adding relatively few bits to data.

- Formal languages: efficient ways to specify and implement programming, markup, and other languages, how formal specifications are helpful in designing and communicating languages, and how to parse and process programs or documents written in such languages.
- Network communication protocols: the techniques and algorithms applied in computer networks to ensure reliable, effective and efficient communication of data between two parts of a network in the face of different kinds of threats and failures.
- Complexity and tractability: the relationship between problems and their algorithms, and the idea that many common problems don't have tractable solutions, that brute force algorithms can result in a combinatorial explosion of the running time, and that heuristic algorithms are often the best we can do in practice.
- Artificial intelligence (AI): intelligent systems and the possibility of designing systems that exhibit aspects of human intelligence, reflecting on what intelligence is, and the practical and theoretical issues surrounding this. A significant component of Artificial Intelligence is (sadly) its limitations and understanding these can rectify popular views of AI that might be picked up from media.
- Software engineering: learning that there are systematic approaches that are applied to large software projects, typically with many team members and large amounts of program code, so that the products behave reliably and efficiently, are affordable to develop and maintain, and satisfy customer requirements.
- Computer graphics: using computers to create images and animations based on a description of a scene or collected data, including techniques such as rendering, occlusion, and transformations.
- Computer vision: processing images and recognising elements in an image, including dealing with noise, edge detection, and face detection.

We now give a more detailed description of the design of the chapter on algorithms to illustrate how the above principles are worked out in practice.

4. Case Study: Algorithms Chapter

The design of the chapter on algorithms is reviewed in this section to give an idea of the approach taken and what topics have been included in the CSFG – and just as importantly, which topics have been excluded.

The key concepts that we chose to convey through the chapter are:

- What an algorithm is and how it differs from the related concepts of programs and informal instructions.

- The concept that an algorithm has an associated cost, that this cost may be non-linear and is related to both running-time (of a program implementing the algorithm) and computational complexity.
- The concept that two algorithms may have different costs even if they solve the same problem and that this difference in costs can be non-linear.

Valuable background for the design of this chapter was provided by an in-depth analysis of reports that included work on algorithms submitted by students for assessment (Bell *et al.*, 2012). This analysis identified several key areas in the algorithms topic that had a great impact on the grades achieved by students. The majority of students chose either sorting or searching algorithms to investigate (63% sorting and 19% searching) and usually these students earned a passing grade or better for the algorithms section of the report if they explained their work satisfactorily. It was found that students who chose other algorithms or used their own programs, were much less likely to pass the algorithms section of the report. Students needed to compare the "costs" of algorithms (i.e. algorithmic complexity) to do well, and the majority of students unknowingly limited their ability to discuss the cost difference between algorithms as they choose to only compare the costs of their algorithms for relatively small input sizes, for example $n = 10, 20, 30$. Because the non-linear difference in costs for some of these algorithms only emerges when larger numbers are used, such as $n = 100$ or $n = 1000$, some students were unable to observe this relationship. About 10% of students were also unable to observe this trend as they chose to compare algorithms with the same complexity, such as Selection and Insertion sort, and so only observed a constant difference in their costs. This observation drove the selection of algorithms in the chapter; it is more important to have a small number of algorithms with different asymptotic complexities than many algorithms that had the same complexity. There were also several cases where students interpreted the "cost" of an algorithm as the length, in lines of code, of a program implementing the algorithm. This suggested students required some guidance in how to measure the cost of an algorithm.

From the study, Bubble sort and Quicksort were the most popular sorting algorithms used by students. While these have drastically different running times, which give students the opportunity to talk about the non-linear difference in their costs, it has been argued that Bubble sort has little pedagogical value and can be confusing for students (Astrachan, 2003). Selection and Insertion sort are both suitable alternatives to Bubble sort as they provide just as strong a contrast with Quicksort and are more worthwhile for students to learn. Linear search and Binary search were the most popular searching algorithms used in student work and these provided a suitable contrast for students to discuss. The choice of algorithms made a very clear difference to the quality of student reports. The algorithms which most often led to high grades were pairs of algorithms with significantly different complexities. The most successful pairs were Binary search vs Linear search (which provided a comparison of $O(\log n)$ vs $O(n)$) and Quicksort vs one of Bubble sort, Selection sort and Insertion sort (a comparison of $O(n \log n)$ vs $O(n^2)$).

Students are not required, or encouraged, to implement the algorithms themselves and use their own programs for measuring costs because this risks a bug in their program giving them the wrong impression of algorithmic performance. Therefore implementa-

tions of each example algorithm used are provided for students to download, although students can follow up by implementing their own versions. The concept of a "cost" as the number of comparisons an algorithm makes is emphasised through the interactives, and the downloadable programs measure it as both comparisons and time taken. Students are encouraged to test the algorithms with large inputs so they are able to observe the non-linear differences between algorithm costs.

Students learn better when they are given the opportunity to construct knowledge themselves through experience, rather than simply learning from definitions or complete instructions (Wadsworth, 1996). The chapter has therefore been designed so that students are given the opportunity to discover algorithms for themselves, and explore the differences in their costs, rather than simply explicitly telling students how each algorithm works and the differences between them.

Another key tool in supporting students' construction of mental models of these concepts is the use of analogies and metaphors for the use of algorithms (Forišek and Steinová, 2012). These are used throughout the chapter but especially during the introduction section (for example, searching the library) to ensure students have begun building a mental model about algorithms before they encounter the interactives.

It has been shown that learning about two algorithms in parallel and comparing them, rather than learning them separately, contributes to students gaining a greater understanding of both the algorithms and the differences between them (Patitsas *et al.*, 2013). Through each of the Searching and Sorting sections algorithms are presented and discussed in relation to each other, rather than viewing each as a separate entity.

There are several algorithm visualisation tools and interactive tutorials that were considered for use in the chapter. Visualisations of algorithms have been popular for teaching different algorithms but it has been noted that many are too complex for students to understand (Murugesh *et al.*, 2010). Furthermore, many don't require interaction from the student, and so encourage passive use of the resource. It was found when examining the visualisations and interactives available that some covered more algorithms than were necessary, and used advanced language that made them unsuitable for use by school aged students. Several contained valuable information and taught the algorithms well but unfortunately were aesthetically unappealing or repetitive, which didn't engage students. Thus, we make limited use of visualisations, and have focused on making them appealing and at a level that is meaningful to high school students.

Following the pattern for the CSFG, the chapter begins with a video and a "What's the big picture?" introductory section, each of which gives an overview of the topic of algorithms and the key concepts the chapter is going to present.

The introductory videos are not intended to teach any of the chapter content, but by giving a 'big picture' view of the main topic they give context to the lessons in the chapter. The first step in the video development process was to decide which concepts were to be conveyed. Several different combinations of concepts were reviewed, including the best and worst cases for an algorithm and the differences between an algorithm and a program, but the final key concepts chosen were the following:

- That an algorithm is a set of instructions for completing a task or solving a problem, we use them in our everyday lives, and algorithms are used to tell computers how to solve problems.

- There can be many different algorithms for solving a particular problem and some of these algorithms are better than others.
- Using a better algorithm can be better than using a faster computer.

For the video, several scenarios were considered, including algorithms for working with a collection of CDs, navigating supermarket aisles and mazes, connecting up networks, boarding planes and finding routes on a map. The concept chosen used two characters, one representing a fast computer and the other a very slow computer, and had them race each other to find a book in a library. The character representing the fast computer is much faster at looking through the books and running through the library, but uses a Linear search algorithm to try and find the book, searching the entire library book by book until they find the one they were searching for. The character representing the slow computer takes a much longer time to walk through the library and examines each book for a long time before placing it back on the shelf and moving on. This character, however, uses a Binary search algorithm, and finds the book much faster. The full video can be viewed online at `http://www.youtube.com/watch?v=FOwCCvHEfY0`, and can be viewed or downloaded at `http://vimeo.com/69609500` (all the videos in the CSFG are provided on Vimeo as well as YouTube, to make it easier for teachers to download them and play them in a classroom, as some schools limit access to online video sites.)

After the video a short introduction section reiterates and emphasises the key lessons from the video and describes the sections of the chapter. To emphasise the points that there are a number different algorithms for the same problem and that some of these algorithms are better than others, a sorting algorithm visualisation has been designed (Fig. 3). The aim of this visualisation is not to teach the algorithms, although it may be of use to refer to it again after students have learnt the sorting algorithms so they can see them in action. It is intended to be engaging, to keep students interested in the chapter content, and to show again the difference in the performance of algorithms. The visualisation we have designed shows only four algorithms to avoid overwhelming students. We have also placed a strong emphasis on the aesthetic of the visualisation as it is intended to be eye catching and engaging.

The algorithms mentioned include "bogosort", in which values are shuffled randomly until they end up in the correct order (which is very unlikely to ever happen for a large list). While this isn't a useful algorithm, it illustrates some algorithmic concepts starkly (such as relative running times, worst case time, best case time, and tractability). The other rows are sorted using Insertion sort, Selection sort and Quicksort. Once the bars are found to be in the correct order the bird beside that particular row begins a victory dance.

Teaching the details of algorithms begins by introducing Linear and Binary search. These algorithms were chosen because it is easy for students to gain a high level understanding of them and they are easy for students to perform themselves with physical objects. The non-linear difference in their complexities also makes them suitable choices for students to compare. The searching algorithms are taught using a constructivist approach through a game in which students have to search through a large number of presents in an attempt to find and collect the missing pets of two children, shown in Fig. 4. The game is based on the CS Unplugged "Battleships" activity (`http://csunplugged.org/`
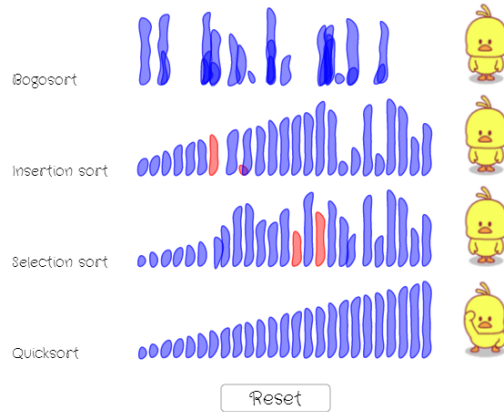
Fig. 3: The sorting algorithms comparison animation

searching-algorithms), but was changed to unwrapping presents, since blowing up battleships was likely to appeal more to male than female students.

Using the constructivist approach, students are simply told whether or not the numbers in the presents are in sorted order, and what number they are required to find. From our experience, students very quickly realise that an unsorted list is very slow to search (usually), and for a sorted list they quickly adopt a binary search related approach which enables them to find a number quickly without using too many of the small number of "lives" they are given.

There is a risk that students may try an interpolation search for the sorted list (e.g. guess that lower numbers are nearer the start); to confound this, we have adjusted the distribution of numbers to be non-uniform, so this strategy will generally not behave significantly better than a conventional binary search and will quickly discourage students from trying to guess number locations. The exact distribution of numbers was tested using simulations of likely student strategies on different number distributions, which identified patterns that would be pedagogically most valuable.

(a)

(b)

Fig. 4: Game for teaching search algorithms constructively. (a) An unsorted set of presents in which students must find the given number; a large number of lives is provided since a linear search will be required. (b) A sorted list; only a few lives are available, so students will need to use binary search to avoid looking at too many presents.

The sorting algorithms section of the chapter focuses on Selection sort, Insertion sort and Quicksort which, like the searching algorithms, were selected for their contrasting run times (for Selection or Insertion sort vs Quicksort) and the successful results achieved by student using them in past assessment (Bell *et al.* 2012). Selection and Insertion sort are very simple to explain and demonstrate with physical objects. Despite the complexity of implementing Quicksort it can also be simple to teach the basic method and demonstrate it with objects, which is all students require to understand it.

Although there are many animations of sorting algorithms available, these don't usually engage the viewer in the process. We have used a constructivist approach to explain the sorting algorithms using a balance scale that can compare objects only two at a time (simulating the data comparison step of conventional sorting algorithms). This is based on the CS Unplugged sorting activity (`http://csunplugged.org/sorting-algorithms`). Since a physical balance scale isn't always available, an online simulation was provided (shown in Fig. 5). The simulated scale has the advantage that we can enforce having only one weight on each side of the scale. Students are guided through the sorting algorithms; for example for selection sort, they are first asked to find the heaviest weight of the set, comparing just two at a time. Students soon find that this can be done in $n - 1$ comparisons, and then $n - 2$ for the second smallest, and so on. The other algorithms are also demonstrated using the approach from the CS Unplugged sorting activity. Quicksort is seeded with the idea of putting a randomly chosen weight on one side of the scale and comparing each of the others with it. Students often come up with the idea of applying the algorithm recursively to the two groups of weights.

Programs implementing all the main algorithms discussed are provided for students to download in common programming languages used in schools, so that students can test their speed, confident that the implementation is correct (since the main learning outcome desired is to observe speed differences, rather than the ability to implement well-known algorithms).

The "whole story" section mentions the range of other problems and algorithms that exist, and also the "big oh" notation that students will quickly encounter if they look at other resources on algorithms. This notation isn't needed to understand the main concepts in the chapter, and so is avoided to make it accessible to students without the necessary math background, but it is important to mention it here since it is so common in this context.

Fig. 5: The Sorting Interactive

5. Programming Exercises Based on the CS Field Guide

Creating exercises based on this kind of material has been explored previously (Voigt *et al.*, 2009), where computer science concepts based on CS Unplugged activities were adapted for a programming competition environment. For example, the parity error correction technique (which appears in the CSFG also) can be used as the basis for a task to identify which bit(s) are identified by a parity error, with a step-up obtained by going from a single row of bits to multiple rows, and from single errors to multiple errors. For this kind of activity, in principle the most challenging version would be a full implementation of an error correction protocol, which is both authentic and motivating.

Some suggestions for exercises building on the material currently in the CSFG are:

- Algorithms: implement one of the searching or sorting algorithms; solve a sorting-based problem where there is a tight time constraint that requires the use of Quicksort rather than the $O(n^2)$ algorithms.
- Programming languages: implement a simple translator or assembler based on a small language (such as MIPS, or a subset, which is used in the chapter).
- Human-computer interaction: design a progress bar (or create information to support one) that gives an accurate estimation of completion time; or implement an experiment that measures user behaviour for response time or pointing time (Fitts's law).
- Data representation: convert numbers between binary, decimal and hexadecimal; convert binary codes to the 5-bit letter system used in the chapter; perform rounding of numbers required when (say) 24-bit colour is converted to 16-bit colour.
- Coding: encode or decode run-length encoding compression; implement the longest match search required for Ziv-Lempel compression; implement a simple substitution encryption system; write a brute-force system to attack an encrypted message; detect errors in data protected with parity bits; calculate checksums for product bar codes or ISBN numbers.
- Formal languages: write a program that implements an FSA from a transition table; use regular expressions in a program to check input; implement a simple lexer; generate random text based on a formal language (grammar or FSA).
- Network communication protocols: write a program to assemble packets that arrive out of order; implement a system that can acknowledge packets and request a re-send to assemble messages reliably.
- Complexity and tractability: implement an exhaustive evaluation of a small NP-complete problem (e.g. TSP, graph colouring, vertex cover, knapsack); implement a heuristic and award points for the answer which is closest to optimal.
- Artificial intelligence: implement a pattern-matching chatbot; perform elementary data mining by statistical analysis; implement a min-max search.
- Software engineering: implement a software metric or visualisation (lines of code, comments, digraph of flow control); write a test generator to create examples to test some software that has been supplied.

- Computer graphics: implement basic transforms; write software to combine multiple transforms into one matrix operation; implement Bresenham's line-drawing algorithm.
- Computer vision: Implement a simple filter that performs a weighted sum of surrounding pixels; perform simple face recognition given measurements of facial features in pre-processed images; perform simple edge detection by finding discontinuities in an image.

The above suggestions are simply to seed ideas, and students or instructors reading the chapters may well come across ideas that could be implemented. For each topic, the difficulty of assignments ranges from a simple simulation to a full implementation of the concepts in a way that could be used in a practical situation.

6. Conclusion

The Computer Science Field Guide has taken a new approach to making concepts from computer science accessible to high school students, encouraging a very broad view of the field rather than a depth-first approach that inevitably focuses on programming. The open nature of the guide is intended to make it easy for adaptation for new situations and curricula.

The guide includes a feedback link, which provides a tight feedback loop where any user can suggest clarifications or improvements. Over 100 suggestions have been received to date. About 23% are simple typos that can be fixed very quickly; 12% are "bouquets", acknowledging the usefulness of the resource; and the remainder are a mixture of clarifications and suggestions that may take longer to implement but are being prioritised for attention.

Currently all but one of the chapters covers the key concepts that we have aimed to convey (the final one to be written, on Network Communication Protocols, is in progress). Further work is needed to add more examples to each chapter; for example, tractability is currently explained using the Travelling Salesman Problem, but other topics such as bin-packing or graph colouring could be added as other ways to illustrate tractability. Future plans include adding more topics such as Computability, Big data, Parallel computing and Databases. Other topics that are currently considered specialised may well become important as a basic part of computer science in the future (e.g. Quantum computing).

Other features that could be added to the guide include quizzes and student login and tracking. Also, multiple versions for different curricula are planned, and eventually a system to support translations would be useful.

The CSFG has been designed to be flexible to adapt to future needs of computer science education in high schools, and may end up being used in situations that can't even be imagined now. Fortunately the computer science community has a strong ethos of open systems, crowd sourcing and creativity that we hope will enable this project to adapt to future demands.

Acknowledgements

References

Astrachan, O. (2003). Bubble sort: an archaeological algorithmic analysis. *ACM SIGCSE Bulletin*, 35(1),1–5.

Balci, O., Gilley, W.S., Adams, R.J., Tunar, E., and Barnette, N.D. (2001). Animations to assist learning some key computer science topics. *ACM Journal on Educational Resources in Computing* (*JERIC*), 1(2).

Bell, T., Andreae, P., Lambert, L. (2010). Computer science in New Zealand high schools. In: Clear, T., Hamer, J. (Eds), ACE'10 : *Proceedings of the 12th Conference on Australasian Computing Education* (Australian Computer Science Communications, 32). Australian Computer Society, Brisbane, Australia, 15–22.

Bell, T., Newton, H., Andreae, P., Robins, A. (2012). The introduction of computer science to NZ high schools: an analysis of student work. In: *Proceedings of the 7th Workshop in Primary and Secondary Computing Education, WiPSCE'12*. ACM, New York, NY, USA, 5–15

Falkner, K., Vivian, R., Falkner, N. (2014). The Australian digital technologies curriculum: challenge and opportunity. In: *Proc. Sixteenth Australasian Computing Education Conference (ACE2014)*. 3–12.

Fellows, M. Parberry, I. (1993). SIGACT trying to get children excited about CS. *Computing Research News*, 7.

Forišek, M. Steinová, M. (2012). Metaphors and analogies for teaching algorithms. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, SIGCSE'12*. ACM, New York, NY, USA, 15–20.

Furber, S. (Ed.). (2012). *Shut Down or Restart? The Way Forward for Computing in UK Schools*. The Royal Society, London.

Impagliazzo, J. (2006). Computing curricula 2005. *ACM SIGCSE Bulletin*, (September).

Murugesh, S., Bell, T., McGrath, A. (2010). A review of computer science resources to support NCEA. In: Mann, S., Veerhaart, M., (Eds) *First Annual Conference of Computing and Information Technology Research Education NZ* (CITRENZ2010), 173–181.

Myketiak, C., Curzon, P., Black, J., McOwan, P.W., Meagher, L.R. (2012). cs4fn: a flexible model for computer science outreach. In: *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. ACM, 297–302.

Patitsas, E., Craig, M., Easterbrook, S. (2013). Comparing and contrasting different algorithms leads to increased student learning. In: *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research, ICER'13*. ACM, New York, NY, USA, 145–152.

Sahami, M., Roach, S., Cuadros-Vargas, E., LeBlanc, R. (2013). ACM/IEEE-CS computer science curriculum 2013: reviewing the Ironman report. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE'13*. ACM, New York, NY, USA, 13–14.

Voigt, J., Bell, T., Aspvall, B. (2009). Competition-style programming problems for Computer Science Unplugged activities. In: Verdu, E., Lorenzo, R., Revilla, M., Regueras, L. (Eds.), *A New Learning Paradigm: Competition Supported by Technology*. CEDETEL, 47151, Boecillo, Spain, 207–234.

Wadsworth, B.J. (1996). *Piaget's Theory of Cognitive and Affective Development : Foundations of Constructivism*. Longman Publishing.

T. Bell is a Professor in the Department of Computer Science and Software Engineering at the University of Canterbury. His main current research interest is computer science education, and he directs the Computer Science Unplugged project; in the past he has been also worked on computers and music and data compression. He received the ETH (Zurich) ABZ International honorary medal for fundamental contributions in Computer Science education in 2013.

C. Duncan holds a Bachelor of Science with Honours in Computer Science from the University of Canterbury, NZ, where she is currently a PhD student. Her research is focused on computer science and computational thinking education in primary and high school, and she has previously worked on the Algorithms chapter in the CSFG.

S. Jarman holds a Bachelor of Science in Computer Science from, and is now a postgraduate student at the University of Canterbury, NZ. His interests include computer science education, mobile application development, human computer interaction and software engineering. He is currently working towards completing the CSFG chapters by developing content around the topic of Network Communication Protocols.

H. Newton holds a Bachelor of Science and Postgraduate Diploma in Science in Computer Science (University of Canterbury, NZ), and is a postgraduate student in the School of Engineering and Computer Science at Victoria University of Wellington, NZ. Her research interests include computer science education and artificial intelligence. She has contributed about half of the chapters in the CSFG.

Programming Trainings and Informatics Teaching Through Online Contests

Sébastien COMBÉFIS, Jérémy WAUTELET

Computer Science and IT in Education ASBL, Belgium
e-mail: sebastien@combefis.be, jeremy.wautelet@csited.be

Abstract. Promoting computer science though programming is widespread all around the world. However, there are not always enough human resources to support trainings and teaching of programming. At the same time, online programming contests have also spread and are getting accessible to people at large. This paper is about how it is possible to use online programming contests to build trainings and to support the teaching of programming. The paper first reviews how programming contests can be classified. It then proposes classification criteria and applies them to a selection of existing online programming contests. Based on that classification criteria and review, the paper discusses how such contests can be used to build programming trainings and also to support teaching. Finally, the paper presents an online platform that allows people to create a contestant profile to compare them to other users of the platform and to discuss about the contests they took part in. All this work aims at increasing the motivation of people when learning to program and at promoting computer science among young people, with limited human resources and using online social connections between people.

Keywords: programming contest, learning programming, contestant's profile.

1. Introduction

There are many different kinds of online programming contests. The main goal of such contests is to allow contestants or teams of contestants to compete. Contests can be of many different kinds: finding the most efficient algorithm, modelling a challenging problem and writing a program to solve it, developing an artificial intelligence (AI) algorithm to be run against AIs from other contestants…

In addition to the contest itself and its main goal, participating to such contests is also a way for contestants to learn and improve their own skills. The better and more relevant the support to the contestant and the received feedback are, the more his/her learning will be of good quality. For example, providing contestants with the correct solution annotated with explanations at the end of the contest, or allowing them to participate in teams may improve their learning.

Of course, given limited human resources to train, teach and coach contestants, it is not always possible to provide individual feedbacks to all the contestants, or to provide

precise and detailed enough feedback for all the tasks that the contestants had to solve for a given contest.

This paper proposes to use existing online contests to support teaching programming skills, and more generally to promote informatics. The focus is not only on contestants that would like to improve their skills, but also to anyone who wants to start learning programming. Depending on the precise purpose and on what is about to be taught, a given contest may be more suited than another one. This work proposes a way to classify online contests according to how they can be used to support trainings and learning. This work also proposes an online platform to be used to track the performances of contestants on various online contests, in order to support their learning.

The remainder of the paper is structured as follows. The next section presents related works about classifying programming contests and about using online programming contests to support teaching and learning programming. The third section presents the proposed classification of programming contests and reviews the main existing online programming contests according to the proposed classification. The fourth section proposes a way to use online programming contests for programming trainings, based on an experience that has been set up in Belgium; and how it can be used to teach informatics. Finally, the last section presents an online platform, currently being developed, that allows anyone to have an online contestant profile to be used to support the use of online programming contests for trainings and learning.

2. Related Work

An approach to classify computer science contests has been proposed in (Pohl, 2006). In that work, the author proposes a classification scheme used to support the discussion about computer science contests. That work focuses on competitions for high school students. Six classification dimensions are proposed: scientific area, style, duration, grading, submission and divisions. Pohl (2006) also states that when participation and achievement in a contest are published, it increases the motivation and fun for the contestants. This latter statement motivates the creation of the online platform for contestant profiles proposed in this paper.

A succinct programming competitions overview is presented in (Forišek, 2013). The author highlights that most programming contests, at least those covered in his paper, are focused on the design of efficient algorithms to solve given problems. In his paper, Forišek (2013) presents tasks that have been used in past contests and that cover other areas of computer science than the one of writing efficient algorithms. The lesson of that paper is that the focus can be placed on other aspects than algorithms efficiency, and that there is consequently a place for learning about many other aspects of computer science through contests. This is in fact exactly the purpose of the Bebras contest, which aims at increasing computer fluency by secondary schools students (Futschek *et al.*, 2009).

Ragonis (2012) explores in her paper the large variety of questions that are used within the computer science (CS) discipline, and in particular she discusses in what ex-

tent those questions can be used for different teaching situations and processes, depending on the type of question. Questions proposed in the main programming contests can also be classified according to the types proposed by that author.

Programming problems similar to those used in competitions can be used to promote informatics as discussed in (Voigt *et al.*, 2010). That paper presents how competition-style programming problems can be combined with *CSUnplugged* activities (Bell *et al.*, 2009) and how they can connect together programming and computer science concepts. In their work, the authors conducted tests, which show that adding programming to the *CSUnplugged* activities deepens the understanding of the concept behind the activities among students. That result strengthens the intuition that programming helps to get a more thorough understanding of computer science concepts.

Using programming contests to increase programming skills among secondary school students has already been explored in a project presented in (Nowicki *et al.*, 2013). That paper presents programming courses that are taught using OLAT distance learning tools (OLAT, 2014). Students are monitored online through weekly programming contests. The conclusion of their work, based on a test of the project in which over 900 participants attended the activities, is that it increased programming and algorithmic skills of the participating pupils. Also, as described in (Audrito *et al.*, 2012), contests (and not only online ones) do have an effect on informatics education as it creates a movement starting in the schools.

Programming contests also help to make learning programming fun as discussed in (Garcia-Mateos *et al.*, 2009). The authors describe an e-learning experience where they used programming contests as an activity to replace the final exam for a second-year programming course for computing majors at university. The contests have been set-up with the Mooshak (Leal *et al.*, 2003) automatic judging system. The results presented in that paper showed that the approach increased self-assessment skills among the students.

Using online platform to support the teaching of programming is also becoming widespread nowadays as testified by a bunch of recent research (Combéfis *et al.*, 2012; Helminen *et al.*, 2009). Those different works have in common that they use techniques to automatically assess the code produced by the learners. Moreover, as highlighted by Combéfis *et al.*, not only automatic assessment is important, but also good quality feedbacks, such as those supported by the Pythia platform (Combéfis *et al.*, 2012).

The presented related work can be summarised with three concepts: classifying contests, promoting computer science through contests and improve learning of programming with contests. Those concepts are precisely focuses of the work presented in this paper. Another key element to remember from this related work is that using contests in combination with various activities may increase programming, algorithmic and self-assessment skills and can be fun. Finally, as discussed in (Hassinen *et al.*, 2006), the best way to learn programming is through programming and extensive practice. Programming being the language of technology (Cohen *et al.*, 2007) and in particular of informatics, and that latter one being everywhere nowadays (Verhoeff, 2013), it is even more important to work on ways to improve the teaching and learning of programming, and promoting it amongst people.

3. Classifying Online Programming Contests

This section proposes classification criteria for online programming contests. The criteria can be used to compare the contests. They can also be used to make it easier to choose whether a given contest is suited for programming trainings or teaching. The proposed criteria are inspired from those of Pohl (2006), but were extended and, additionally, a distinction has been made between criteria related to the contest in general and those related to the tasks proposed in the contest.

3.1. *Classification Criteria*

The first set of criteria is related to general information about the contest. The first criterion (**I1**) is about whether the contest is restricted to single contestants or they have to participate as a team (Table 1). The second criterion (**I2**) is about the conditions that contestants must satisfy related to their age, gender or any other constraints related to their study year. The third criterion (**I3**) is about the programming languages that are accepted. The fourth criterion (**I4**) is about the duration of the contest, or the timespan during which they are allowed to work on the tasks and submit their solutions. The fifth criterion (**I5**) is about the frequency of the contest. Note that some contests are always open and start as soon as you decided to start it. They are referred to as open contests. Finally, the last criterion (**I6**) is about how the scores of the contestants are computed for their submissions, in order to establish the ranking of all the contestants.

The second set of criteria is related to the tasks the contestants have to solve during the contest (Table 2). Of course, one given contest may mix several kinds of tasks, even if it is not generally the case, at least for the contests covered in this paper.

The first criterion (**T1**) is about the type of submission that the contestant has to provide, that is, a source code, an executable or just a text file with the output produced by his/her program. The second criterion (**T2**) is about the type of task, that is, whether the contestant has to write a function whose specification is given, to model and solve a problem, to write an artificial intelligence… The third criterion (**T3**) is about any limitation on the number of trials allowed, and also about any limitations related to the execution time or the maximal allowed memory. The fourth criterion (**T4**) is about the

Table 1

Information criteria

I1	Team	Single contestant or teams
I2	Age and gender	Ages range required to participate and accepted genders
I3	Language	Accepted programming languages
I4	Duration	Timespan during which contestants can submit solutions
I5	Frequency	Frequency with which the contest is organised or open
I6	Scoring	How the score of the contestant is computed

Table 2

Task criteria

T1	Submission	Code source, executable program, output data
T2	Type	Writing a function given specification, solving a problem, writing an artificial intelligence
T3	Limitation	The number of trials that are allowed, time and memory
T4	Feedback	The feedback produced for a submission
T5	Level	Any difficulty level or partition of the tasks

feedback that is produced when the contestant makes a submission. It can go from no feedback at all, to an indication about any compile errors, execution errors, test failed errors… Finally, the last criterion (**T5**) is about whether the tasks of the contest are partitioned, according to difficulty levels, for example.

Compared to the classification proposed in (Pohl, 2006), some of the proposed dimensions have been fixed to a single value for the contests that are relevant to the purpose of this paper:

- The scientific area of the contests is limited to those with a focus on algorithmic.
- Only contests with an automatic grading are considered.
- And finally, the submissions for the considered contests are limited to software (executable or source code) or answer value.

3.2. *Review of Existing Online Contests*

This section reviews several online programming contests, and positions them according to the classification criteria proposed in the previous section. The review is not meant to be comprehensive but covers the main online contests.

Internet Problem Solving Contest (IPSC) is a contest for teams that can contain up to three people. Contestants have to solve problems, by finding the outputs that correspond to given inputs to the problems. They can write programs to solve the problems but it is not always necessary to do so. The contest has been organised yearly since 1999 and is opened to everyone, but with a special category for teams out of secondary schools.

ACM International Collegiate Programming Contest (ACM-ICPC) is an international contest, made of several stages: local, regional and then international. The contest is opened to university students that have to participate as teams. The contestants receive problems that they have to solve, providing a program written in C, C++ or Java. Moreover, the problems of ACM-ICPC are available after the contest on the *UVa Online Judge platform*, which makes it possible to try to solve them at any time, in the same conditions and with the same grader than the one used during the ACM-ICPC contest.

IEEE also proposes a contest, namely the *IEEEXtreme Programming Competition*, which lasts 24 hours and is dedicated to teams of students. All the teams receive a set of programming problems and as for ACM-ICPC, they have to solve the greatest number of problems.

Google Code Jam is a contest put in place by Google to identify potential persons to recruit. The contest consists of a set of algorithmic problems that must be solved within a time limitation. Any programming language is accepted since the contestants just have to provide the outputs corresponding to generated inputs for each problem. The contest has several rounds, all taking place online all over the world, except the worldwide final, which is hosted in one unique location for all finalists.

A lot of countries do have online programming contests to make the selection for their national team to be sent to the *International Olympiad in Informatics* (IOI). Most of those contests are opened to anyone in the world, not to compete but to participate. Just to mention some of them: *USA Computing Olympiad* (USACO), *France-IOI*, *Croatian Open Competition in Informatics* (COCI), *French-Australian Regional Informatics Olympiad* (FARIO)… All those contests are following the same philosophy as the IOI.

ProjectEuler is a collection of challenging mathematical and computer programming problems that cannot be solved only with mathematical insight. It is not exactly a contest as the previous ones. Rather, people can connect on the website at any time and solve the different problems to increase their position in the ranking.

Finally, *CodeChef* is an online contests hosting platform. The platform proposes contests regularly (short ones and also long term contests) that are open to everyone. A very large number of programming languages are accepted. The contestants have to submit their source code that is automatically graded by the platform.

3.3. *Classification of Contests*

Table 3 summarises the review of the selected contests presented in the previous section, according to the classification criteria proposed in Section 2. Information concerning the IOI is not all true for all the IOIs that already took place. The specific information shown in the table comes from the rules of IOI 2013. In particular, criteria **I6**, **T1**, **T2** and **T3** are not true for all the IOIs.

Looking at the table testifies that there are several ways to classify online contests depending on what is the focus and goal of the comparison. Here are examples of possible classifications:

- One could be interested in contests that can help to improve teamwork skills. In that case, the only criterion to look at is **I1**.
- One can be interested in contests to support learning programming, and not being an expert in algorithm designs. Looking at **T3** provides clue about the limitations that may mean that the focus in on performance and complexity (penalties for wrong submissions and time taken) and **I6** provides the conditions to win, which is whether the focus is on correctness or efficiency. Criterion **T5** can also bring information, in particular if the contest proposes subproblems with increasing complexity/size.
- Another possible way to classify contests is with respect to the feedback that is produced for each submitted solution. The **T4** criterion contains that precise infor-

mation. It is important to have good feedback if the goal is to allow the contestants to get a better learning while diminishing the intervention of trainers.

● If contests are to be used to build a training, as explained in the next Section, it is important to have some regularity (**I5**) and to have enough accepted programming languages (**I3**) if the training is to be general, or to have the taught programming language accepted if the training is to be specific.

Another review of programming contests is proposed in (Forišek, 2013). The considered contests are not restricted to online ones. Four categories to classify the considered contests are proposed by the author:

● ACM-ICPC.
● IOI.
● Company-branded contests such as Google CodeJam.
● And finally large portals hosting contests regularly such as CodeChef.

That classification, even if very simple, summarises well the actual situation. The two first contests are old worldwide and well-established ones. They are mainly focused on efficient algorithms designs and have a limited number of accepted programming languages. They both consist in a set of problems with as goal to solve the maximal number of them in a given amount of time. Both contests require the contestant to submit a program that will be executed for automatic grading. Finally, in both cases only a very limited feedback is proposed to the contestants for a given submission.

The third category, namely company-branded contests, corresponds in fact to a tool used by those companies to help them in their recruit process. In addition to Google CodeJam, contests in this category include Facebook Hackaton, for example.

Finally, the last category corresponds to platforms that host contests. Those platforms are themselves managing and proposing contests but also allow anyone to create his/her own contest. In addition to CodeChef, contests in this category also include TopCoder, for example Table 3.

Table 3

Review of selected contests according to the proposed classification criteria

Internet Problem Solving Contest (IPSC)	**I1**	Teams of up to three people	**T1**	Output data
	I2	Open to everybody, separate ranklists for individuals and teams in the secondary school division	**T2**	10 to 20 problems to solve, provided input and output specifications, and examples
	I3	N/A	**T3**	10 submissions at most for each subproblem (unless declared otherwise)
	I4	One block of five hours	**T4**	Correct or wrong
	I5	Once every year, since 1999	**T5**	Easy and hard input data
	I6	Winner is the team with the most points received; criteria taken into account are time, number of wrong submissions and difficulty level		

ACM International Collegiate Programming Contest (ACM-ICPC)	**I1**	Team of three people	**T1**	Source code
	I2	Basically students enrolled in a degree program at the sponsoring institution at least a half-time load and having left secondary school for less than 5 years	**T2**	At least 6 for regional and at least 8 for world final problems to solve, provided input and output specifications, and examples
	I3	Java, C, C++	**T3**	20 penalty minutes per wrong submission; execution time limit for problems
	I4	One block of about five hours	**T4**	Accepted or Rejected (run-time error, time-limit exceeded or wrong answer)
	I5	Once every year, with several stages: regional contests then world final, since 1978 (the first edition being on 1974)	**T5**	-
	I6	Winner is the team with the most problems solved; criteria taken into account are earliest time of submittal of correct submission and number of wrong submissions		
IEEEXtreme Programming Competition	**I1**	Team of up to three people (max 2 graduate students)	**T1**	Program
	I2	IEEE members (student or graduate student)	**T2**	A set of programming problems
	I3	C, C++, C#, Java, Python, Ruby, Perl, PHP	**T3**	No limit
	I4	One block of 24 hours	**T4**	?
	I5	Once every year (edition 7.0 in 2013)	**T5**	Easy, Medium and Hard problems
	I6	Winner is the team with the most points; criteria taken into account are: difficulty evaluated as the number of other teams who succeeded the problem		
Google Code Jam	**I1**	Individuals	**T1**	Output data + source code
	I2	Open to everybody	**T2**	A set of problems, provided input and output specifications, and examples
	I3	N/A	**T3**	4 min for small input data and 8 min for large ones
	I4	Four online rounds and one on-site world final	**T4**	Message for malformed or over-sized submission; Correct/Failed for small input data and no feedback for large input data
	I5	Once every year (since 2003)	**T5**	Small/Large input data
	I6	Winner is the contestant with the most points; criteria taken into account are the number of correct submissions and time		

International Olympiad in Informatics (IOI)	**I1**	Individuals	**T1**	Source code
	I2	Secondary school students enrolled during the period September to December in the year before IOI'*n* and is not older than 20 on the 1st July of the year of IOI'*n*	**T2**	Two times three tasks, provided input and output specifications, and examples
	I3	C, C++, Pascal	**T3**	One submission per minute and at most 100 submissions for a task; execution time and memory limit for the tasks
	I4	Two blocks of five hours	**T4**	Solved or Not solved (Incorrect solution, Run-time error/Out of Memory or Time limit exceeded)
	I5	Once every year, since 1989	**T5**	-
	I6	Winner is the contestant with the most points; criteria taken into account are the number of subproblems solved		
ProjectEuler	**I1**	Individuals	**T1**	A number
	I2	Open to everybody	**T2**	468 problems to solve, provided a description
	I3	N/A	**T3**	N/A
	I4	N/A	**T4**	Correct or wrong
	I5	Always opened	**T5**	-
	I6	Ranking based on the number of problems solved		
CodeChef	**I1**	Individuals	**T1**	Source code
	I2	Open to everybody	**T2**	Generally between 4 to 12 problems
	I3	Many languages among which C, C++, C#, Erlang, Haskell, Java, Pascal, Perl, PHP, Python, Ruby, Scala	**T3**	Execution time limit for problems
	I4	Generally between 2 and 3 hours, and a week for long challenge	**T4**	Accepted or Rejected (Time limit exceeded, wrong answer, runtime error or compilation error)
	I5	At least one contest a month (since 2009)	**T5**	-
	I6	Ranking based on the number of problems solved		

4. Using Existing Online Contests for Trainings and Teaching

As introduced above, the main goal of the programming contests mentioned so far is to allow contestants to compete against each other, alone or with a team, in order to get the best score. This section proposes two other ways to use online programming contests, to support programming trainings or teaching of informatics. The section also presents an

online platform, currently being developed, where people can create contestant profiles to be shared and compared with others.

4.1. *Programming Trainings*

The *Belgian Olympiad of Informatics* (be-OI) is already using existing online contests during the team selection process for the International Olympiad in Informatics (IOI). The be-OI is composed of several stages: several local semi-finals, one national final and an IOI selection process (Combéfis *et al.*, 2011). More precisely, for the IOI selection process, a certain number of contestants who got the best scores for the final are invited to join a pool of contestants, from which the four Belgian representatives for the IOI are selected.

Pupils from the pool were asked to participate to various selected online programming contests, and the scores they realised were tracked and put on an online wiki so that the pupils were able to compare themselves. The scores they achieved on those contests were taken into account in the IOI team selection process. Achieving good scores for a pupil of course increases his/her chance to be selected. Since it is not possible to check whether pupils were helped or not by external people, the selection process includes an additional small contest for which all the pupils from the pool are physically in the same room at a same moment.

The experience that was set up with the be-OI was positive for the coaches as well as for the pupils from the pool. Generally speaking, several advantages can be identified for building programming trainings based on existing online contests:

1. It reduces the human resources needed to coordinate the training.
2. It suppresses the need to create programming exercises and tasks.
3. It allows pupils to code a lot and therefore to improve in some way their coding efficiency with practical exercises that they can do at home.
4. It allows pupils to compare themselves with other worldwide contestants.

Using such a programming training for the IOI team selection process is therefore relevant when the available human resources are rather low. It saves them time to take care of other interesting aspects of the training. For example, the coaches can use their time to review with the pupils some of the exercises proposed in the selected online contests, to teach them new concepts or techniques to maybe better solve those exercises. Such a consideration is the subject of the next section.

There are also disadvantages of using existing online contests for programming trainings. The main ones that can be observed are:

1. It is very difficult to motivate pupils to spare time to participate in all those contests for which the hours where it is possible to participate are not always convenient.
2. It is honour based, meaning that there is no control on whether it is actually the pupil who participated to the contest, which makes it impossible to use for a selection process.

As discussed in section 4.3, the proposed online platform for contestant profiles aims at overcoming the first disadvantage, by increasing their motivation such as described by Pohl (2006), by publishing their scores publicly or in a group of selected pupils.

Not all contests are best suited to be used for programming trainings. Since the goal is essentially for individual trainings, only contests where individuals can participate should be considered (**I1**). If the contest is to be used to rank people, it is important to carefully look at (**T6**) in order to vary the different kinds of evaluations, just for the same reason for varying the types of questions exposed in (Ragonis, 2012). The accepted languages (**I3**) are also important if the goal of the training is to improve the skills of a specific language or to prepare the learners to a specific contest.

4.2. *Teaching Informatics*

As detailed in the previous section, using online contests for programming trainings does not provide any guarantee about the skills the contestant will actually learn from the training. That latter fact is reinforced when the feedback provided to the contestants for their submissions is of poor quality. However, this does not mean that it is impossible to teach informatics with the support of online contests for some activities, as it has been highlighted in the related work section.

Limiting the activity of the pupils to just participating in the contest is not enough to get a good learning. Additional activities supervised by trainers must be proposed in addition to the contest. As explained in (Combéfis *et al.*, 2012), when it is to learn programming, the feedback that is given to learners is very important to support their learning. The additional activities that have to be proposed to teach informatics with online contests are thus centred on feedback. In case of online programming contests, the additional proposed activities or material to provide are:

1. Solutions to the problems, the more detailed and annotated, the best.
2. Feedback about the solution of the learner, that is, information about why it is not correct or which elements can be improved.

Most of the time, online contests do not take into account any qualitative consideration about the code submitted by contestants. As highlighted in (Forišek, 2013), most traditional programming contests only focuses on the design of efficient algorithms. As it can be observed in the review of the main online programming contests, the feedback that is provided for each submission is either non-existent or is very limited.

Using online programming contest to do "real-time" teaching of informatics is certainly not appropriate due to the lack of feedback. However, and as the project presented in (Nowicki *et al.*, 2013) testifies for example, it is still possible to combine programming contests with a given educational device in order to improve the quality of learning. To do that, it is important to provide feedback to the learners. The platform presented in section 4.3 can be used to attach feedbacks to problems of various online contests, so that contestants can learn from the problems after having tried to solve them.

4.3. *An Online Platform to Share Contestant Profiles*

This section describes the *My Contestant Profile* (MCP) online platform that can be used to support programming trainings and teaching of informatics by using online programming contests[*].

The purpose of the platform is to maintain a list of online programming contests, and to allow its users to have a contestant profile on it. As soon as a user participated to an online contest (or a round of a contest), he can go on his profile and add the score he performed. The main benefit of doing so is the possibility for the user to compare his score with his friends, people from the same city/region/country or any other relevant group of people that could be defined. That first feature can be used to monitor programming trainings. A specific group of people can be set up on the platform so that the coach can monitor the performance of the users belonging to the group. It could be used for the be-OI contest, for example, defining a group containing the contestants from the pool.

The second important feature of the platform is the discussion forums used to discuss about a contest that took place, or more precisely on the problems of a given contest. It is possible through the platform to attach feedback information and detailed solutions for each problem. Letting users discuss amongst themselves enriches their understanding.

The aim of the MCP online platform is to provide a tool to support the two previously uses of online contests, namely programming trainings and teaching of informatics. The first feature described is used to manage a pool of trainees and to improve their motivation. It allows trainers to monitor the contestants, and to use their performance as one indicator for a selection process. The second feature described is used to support teaching of informatics. It proposes a place where additional materials related to the tasks and problems of contests can be posted and especially discussed between trainers/teachers and contestants/learners, but also between the users of the platform.

5. Conclusion and Perspectives

This paper takes the opportunity that there are more and more freely accessible online programming contests to discuss the possibility to use them to build programming trainings and to teach informatics. The paper first proposes a short review of the main online programming contests and describes them according to a proposed classification. The classification is built around two sets of criteria. The first set contains criteria on general information about the contest such as the conditions of admission or the accepted programming languages. The second set contains criteria about the tasks/problems used in the contest such as the type of submission or time and memory limitations.

Whereas using online programming contests for programming trainings has already been experimented with in Belgium, using them in the context of teaching informatics has only been tested by some researchers as described in the related work section. This paper highlights that online programming contests can be used to build programming

---

[*] The online platform is available on: `http://mcp.csited.be`

trainings and to support teaching informatics. In order to support those two latter uses of contests, the paper presents an online platform to host contestant's profiles.

Future work includes deploying and populating the MCP platform with contests and problems, and to produce feedback information about those problems. Based on that, experiences to build trainings or to teach informatics have to be set up and monitored to assess whether they improved programming skills. Another way of investigation is to measure the increase of motivation among users of the MCP platform.

Perspectives include developing a way to measure the impact of the platform on the motivation to participate in more contests and learn programming. Experiments have to be done on several groups of people: contestants that are to be trained for a specific contest (IOI, for example), pupils at school that have a programming class and are learning to program, and finally people at large studying programming and willing to improve their skills and share their thoughts about tasks/problems found in online contests.

References

Audrito, G., Demo, G., Giovannetti, E. (2012). *Olympiads in Informatics*, 6, 3–20.

Bell, T., Alexander, J., Freeman, I., Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *Journal of Applied Computing and Information*, 13(1), 20–29.

Cohen, A., Haberman, B. (2007). Computer science: a language of technology. *SIGCSE Bulletin*, 4(39), 3–14.

Combéfis, S., Leroy D. (2011). Belgian olympiads in informatics: the story of launching a national contest. *Olympiads in Informatics*, 5, 131–139.

Combéfis, S., le Clément de Saint-Marcq, V. (2012). Teaching programming and algorithm design with pythia, a web-based learning platform. *Olympiads in Informatics*, 6, 31–43.

Forišek, M. (2013). Pushing the boundary of programming contests. *Olympiads in Informatics*, 7, 23–35.

Futschek, G., Dagiene, V. (2009). A contest on informatics and computer fluency attracts school students to learn basic technology concepts. In: *Proceedings of the 9th World Conference on Computers in Education (WCCE 2009)*.

Garcia-Mateos, G., Fernandez-Aleman, J.L. (2009). Make learning fun with programming contests. In: *Transactions on Edutainment II.* Lecture Notes in Computer Science 5660, 246–257.

Hassinen, M., Mäyrä, H. (2006). Learning programming by programming: a case study. In: *Proceedings of the 6th Baltic Sea Conference on Computing Education Research (Koli Calling 2006)*. 117–119.

Helminen, J., Malmi, L., Korhonen, A. (2009). Quick introduction to programming with an integrated code editor, automatic assessment and visual debugging tool – work in progress. In: *Proceedings of the 9th International Conference on Computing Education Research (Koli Calling 2009)*. 59–62.

Leal, J. P., Silva, F. (2003). Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6), 567–581.

Nowicki, M., Matuszak, M., Kwiatkowska, A., Sysło, M., Bała, P. (2013). Teaching secondary school students programming using distance learning: a case study. In: *Proceedings of the 10th World Conference on Computers in Education (WCCE 2013)*.

OLAT Team. (2014). *OLAT 7.8 – User Manual (1.2014 v7.8)*. http://www.olat.org

Pohl, W. (2006). Computer science contests for secondary school students: approaches to classification. *Informatics in Education*, 5(1), 125–132.

Ragonis, N. (2012). Type of questions – the case of computer science. *Olympiads in Informatics*, 6, 115–132.

Verhoeff, T. (2013). Informatics everywhere: information and computation in society, science, and technology. *Olympiads in Informatics*, 7, 140–152.

Voigt, J., Bell, T., Aspvall, B. (2010). Competition-style programming problems for computer science unplugged activities. In: E. Verdu, R. Lorenzo, M. Revilla, L. Regueras (Eds.), *A New Learning Paradigm: Competition Supported by Technology*. Boecillo: CEDETEL, 207–234.

S. Combéfis obtained his PhD in engineering in November 2013 from the Université catholique de Louvain in Belgium. He is also finishing an advanced master in pedagogy in higher education. He founded the Belgian Olympiad in Informatics (be-OI) with Damien Leroy in 2010. In 2012, he introduced the Bebras contest in Belgium and at the same time he founded the CSITEd non-profit organisation that aims at promoting computer science in secondary schools. He is interested in computer science education, and in particular on how to increase computer fluency among people at large, and also on how to build and design online programming courses that support learning as best as possible.

J. Wautelet is studying computer science at Université catholique de Louvain. He is now a first year bachelor student. He is actively involved in the Bebras Belgium project. He is also currently member of the CSIT-Ed non-profit organisation where he contributes on several projects including Pythia. Being freshly graduated from the secondary school, he brings a fresh view about how computer science is perceived there.

Learning Strategies of Informatics Contestants

Gilberto CUBA-RICARDO, P. Alberto LEYVA-FIGUEREDO,
Laura L. MENDOZA-TAULER

José de la Luz y Caballero University of Pedagogical Science of Holguín. Cuba
email: {gilberto.cuba, albertoleyva, laura}@ucp.ho.rimed.cu

Abstract. This paper shows a study about learning strategies that informatics contestants use. These are the result of the experience in the topic through systematized interviews and participant observations by coaching the contest group.

This paper also details the methodology used and describes the logic we followed in order to determine the strategies used by contestants. Among these: Competitiveness to be "First", tendencies to use "Trial and Error", "Traceability" of source code, study of existing source code along with algorithm description, and use of mnemonic systems.

Keywords: learning strategies, programming contests, informatics olympiads.

1. Introduction

Knowledge competition activities, as instituted in the Ministerial Resolution 91/2007 (Cuban Ministry of Education); are designed to eradicate reductionist conceptions; such as identifying these activities only as competitions. The goal, then, is to transform them into creative environments where students can extend and deepen their general proficiency. In particular in subjects of their interest throughout the school year.

Recent research (Campos-Maura, 2006) has shown that knowledge competition: incentivize mass participation, develops interest in the subjects of study, improves the quality of learning and morally encourages the effort and work of students and teachers.

In Cuba, this activity is carried out on several levels of Education, from primary school to senior high school. Particularly, regarding this latter level of education, the Cuban Minister of Education, (Velázquez-Cobiella, 2011), speaking at the inaugural conference of "Pedagogía 2011" Congress, highlighted that "…the work done at Vocational Senior High School Institutes of Science is strengthened by stimulating students to continue university studies in science, a necessary condition for the scientific and technical development of the country".

Without a doubt, according to this idea, senior high school is the cornerstone of student education and future professional training according to Fidel Castro's ideas when he says: "The future of our country must be necessarily a future of men of science; it must be a future of men of thought".

Taking into account the premise that this level of education (senior high school) should decide the student's future profession, delving into the actual contents of the curriculum should not be enough. An environment where all the student's potential and

capabilities can be developed must exist. Knowledge competitions provide such an environment, sorted by subjects and grades.

High school seniors can enroll in the competition's groups once they are identified and selected by the teacher acting as trainer (from now on: coach). These requirements also include: basic skills and potential related to the subject of their preference. The students selected are often known as contestants or challengers, and in the scientific literature are categorized as talented. This is manifested by their high degree of motivation to perform tasks and by their high performance and potential. This potential is evidenced in any of these fields separately or in combination: general intellectual skills, particular academic aptitude, creative or productive thinking, leadership skills, visual or performing arts' skills, and psychomotor skills (Marland, 1971).

Given that, special attention must be paid to students selected for competition (from now on contestants). We have created co-curricular environments in which the knowledge is deepened, theirs skills are prepared, and their potential developed and improved with the objective of achieving better performance and precision in the results of the competitions they carry out. All this increases the quality of their learning and prepares them for future life, thus achieving one of the aspirations expressed by the Minister of Education, this time at the "Universidad 2010" conference (Velázquez-Cobiella, 2010), when she say that is necessary: "... that students acquire basic knowledge to achieve higher levels of learning, with strength and possibilities of application, and the development of skills aimed at solving learning and social life problems".

Contestants not only prepare for their future profession, they also proudly represent their country in the knowledge and skills competitions at Latin American and international levels. In the process, contestants acquire a sense of belonging and identity for their nation. To this day, they have achieved outstanding results, including medals and the moral personal recognition they bring.

However, the decorous results obtained today do not meet the expectations of such talented students who have the intellectual abilities to solve efficiently and in a short amount of time the programming problems presented at these competitions. Therefore it is necessary to strengthen the intellectual formation of programming contestants. The ultimate objective being to integrate students into society, encouraging the scientific and technological development of the country. The reality on the current competitions is characterized by inadequate strategies and techniques to solve programming problems, which impedes the search for efficient solutions in the shortest time.

The purpose of this paper is to show contestants' learning strategies diagnosed during the competition activity. In addition, we consider the importance of each learning strategies and their correct use.

2. Informatics Contests Activity

Programming contests are defined in Cuba (Hernández-González, 2008) as curriculum activities where students develop their potential and talents, although the contents and skills to be developed in the activity are extracurricular. Algorithms and programming

languages are not taught in the senior high school, only in a very introductory way during twelfth grade.

Contestant groups are made of students with high motivation and programming skills, the activity consists of two stages: training sessions and competitions. In both stages the fundamental activity is to solve programming problems that generally have an algorithmic nature (Verhoeff *et al.*, 2006).

According to Hernández-González (2008), the contestants need to develop some specific skills to set up their learning efficiently. These are:

- Write code to solve the proposed problem using a programming language.
- Develop a program using the advantages of an Integrated Development Environment (IDE) and debugger facilities.
- Create test cases covering all variants of the problem.
- Consider the elapsed time and memory available when choosing algorithmic solutions and data structures according to its limits.

Due to the nature of the programming competitions activity and specially training sessions, the contestants acquire an academic and scientific education; mainly because they have been researching new knowledge to solve programming problems. This improves the contestants learning strategies, and develops the skills that stimulate self-study and learning. These are important elements in the formation of a contestant who wants to grow the skills to solve programming problems.

Unlike academic training, the programming competition group also educates values of modesty, honesty, cooperation, solidarity, collectivism, and especially national identity. Moreover, it should create friendship among group members, and communication ought to flow spontaneously until the objectives are met. Therefore, one of the objectives of the Cuban Education is designed to improve the position, the quantity and quality of the Cuban winnings on the International Olympiad of Informatics (IOI); where to do so, we should improve the results of the National Informatics Competitions.

The development of contestants' skills depends on the learning process during the assimilation of the contents in the training sessions, and during the activity of solving programming problems that are performed in competitions. Given that, influencing the learning of each contestant is a key goal for coaches.

3. Intervention in the Contestant's Learning Process

Thus, in an effort to modify the contestant's learning process, the following actions have proposed. These are supported in the methodological work developed before the training sessions:

1. Diagnose the particular learning strategies of the contestants.
2. Determine the relationship between learning strategies used by contestants, and the activity of solving programming problems.
3. Address individual differences considering the diagnosed learning strategies.

4. Promote the conscious use of diagnosed learning strategies so contestants are able to control them.

5. Reorient the use of learning strategies that are not suitable for certain processes.

6. Teach new learning strategies that are not used by the contestants.

Learning strategies are described by several authors (Chamot and Kupper, 1989; Oxford, 2003; Weinstein and Underwood, 1985; Weinstein *et al.*, 1988; Carrasco, 2004), which emphasize the important role in the cognitive process in general. They are mostly recognized as a set of procedures, actions and activities used by individuals to acquire, store and/or use information in order to make the learning process more effective (Chamot and Kupper, 1989; Oxford, 2003; Carrasco, 2004; Ortiz *et al.*, 2007); and as a number of different skills that have postulated themselves as necessary, or helpful, for effective learning and retention of information for its later use (Weinstein and Underwood, 1985; Nisbet and Schucksmith, 1986; Weinstein *et al.*, 1988). Furthermore, learning strategies are kind of rules that make proper decisions in a certain time of the cognitive process (Ortiz *et al.*, 2007).

Considering the characteristics listed by each of these authors, and contrasting the contents of learning strategies with them, the assumption is that it recognizes learning strategies as a set of procedures, actions, activities used by individuals to acquire, store and/or use information in order to make more effective the learning process.

Knowledge of learning strategies used by programming contestants, as diagnosed by the coach, must be used to ensure that educational activities are prepared according to its advantages and disadvantages to improve the cognitive process during training sessions. To do this, it is necessary to know how strategies have classified, this will allow determining the "when" and "how" to carry out or reorient the learning strategies.

One classification that appears in the work of many researchers is the one that considers (Chamot and Kupper, 1989; Weinstein and Mayer, 1991; Oxford, 2003): cognitive strategies, metacognitive strategies and socio affective strategies, each one these as a kind of learning strategy.

4. Learning Strategies of Informatics Contestants

Based on theoretical considerations related to learning strategies and the previously stated actions on intervening in the learning process, it is necessary to improve the skills developed by contestants to solve programming problems of an algorithmic nature.

Several of the solutions commercially available (Weinstein and Underwood, 1985), and major of those (Learning and Study Strategies Inventory, LASSI; Learning and Study Questionnaire, LSQ; Shortened Experiences of Teaching and Learning Questionnaire, SETLQ; Experiences of Teaching & Learning Questionnaire, ETLQ; Visual Aural Read/Write Kinesthetic, VARK; Approaches to Studying Inventory, ASI) are positivist with a predominant quantitative paradigm. Their purpose is predominantly the assessment of students' awareness about the use of learning and study strategies. Ferrera (2008) considers that, when students answer the tool's questions, they do not objectively assess themselves, which mean that students are not using true learning strategies. To make the

appropriate assessment of student knowledge and learning strategies, one must use the analysis of concrete execution on learning tasks.

Context

The programming competitions described here are carried out in one senior high school in the Holguín province. There, the incoming contestants are selected from their willingness to join the contests group and also according to interviews and test results. In the latter, the reasoning and informatics skills that the future contestant has are assessed. Therefore, students of 10th, 11th and 12th grade comprise the competition group.

The training takes place in extracurricular school sessions since there isn't any other time scheduled for training, which requires self-preparation of the contestant. In addition, training is carried out separately on the three grades. When a competitive event is near, training camps are created by grouping all the contestants. Training is performed mostly using Charguéraud and Hiron (2008) proposed method, hence the diagnosis of learning strategies are mainly aimed at learning programming, and not the algorithmic theory in isolation.

The timeframe of the study coincides with the years of the author's experience in the field, specifically the last three years.

The instruments and procedures are applied by the coach without the intervention of others. This avoids the introduction of extraneous variables in the system. Furthermore, the conditions created in the group are of empathy and trust, through the interaction of each training session, year after year. It also emphasizes that at no time should become explicit to the contestants that they are being evaluated as part of an experiment.

Participants

Year after year, in the programming competition group, the amount of contestants in the three grades is between 10 and 20 students (age M = 16.28, SD = 1.23; all male). This is the approximate amount of contestants that participate in the current research. In particular cases there were other samples that will be described as soon as needed.

Materials

Due to the variety of the learning strategies, various resources were used. In general, we used interviews by contestants and teachers; the participant observation keeping a written record of the contestants' behaviour; programming tasks; a digital recorder; and some applications (the C/C++ IDE, Code::Blocks, GNU C++ Compiler, "gcc" with GNU Debugger, "gdb", and a distributed revision control system, "git") configured together to monitor the contestants' actions during the problem solving process.

The interview questions are open, this makes it possible to collect as much information as possible, as well as to relate it to the results of the observation and interviews with other people in the contestant's social circle.

The most prevalent learning strategies determined in a diagnostic first step are: competitiveness to be "first", behavioural tendency to "trial and error", the "trace" of source code, reading source codes and problem solutions, and using mnemonic systems.

The following subsections explain some of these strategies, but that does not mean they are listed in order of priority or importance. In this paper, only the first step description of the previous actions is presented.

4.1. *Competitiveness to be "First"*

Training sessions are organized into classes that mostly assess the content taught in previous classes. These assessments are setting a global ranking about the contestant behaviour. Taking into account the ranking hierarchy, "first" contestants are taken into consideration to make the team that represents the school, province or country in any competition: IOI, National Informatics Contests, National Cups, etc. Hence, the motivation is to be the "first" or at least be among the "first ones".

There is a high degree of motivation triggered by the competition and the desire to represent, including: recognition from the contest group, other schoolmates or from community and family members. This stimulates the student to achieve stronger and efficient acquisition of knowledge, particularly in the activity of solving programming problems.

Revilla *et al.* (2008) mention some related ideas of the self-competitive behaviour when users participate in online judges. They recognize too, the importance of several learning strategies involved in the training process, which can be very positive for the student's formation and maybe neutralize the negative effects that many people impute to any kind of competitive learning activity (Revilla *et al.*, 2008).

To identify this learning strategy in contestants, some practical methods are used. For instance, in interviews, contestants were asked some of these questions:

- What interested you the most about joining the competition group? (Commonly known as elite group).
- What are your future aspirations about programming competitions?
- Do you like solving programming problems?
- What kind of feeling do you experiment while solving problems?
- What interest do you have in solving more tasks than others contestant?

The majority of these interviews (that we call "conversation") were digitally recorded and then analyzed for the occurrence of strategy processes and behaviours.

In addition, contestants were asked to express considerations and experiences about some of the colleagues individually and about the contests group in general. Not all interview questions are asked at the same time. Questions were asked according to the contestants' behaviour.

Some examples of the contestants' answers are: "When I am trying to solve a problem, I feel a new challenge that invites me to compete, then I begin to look for a solution and when I get it, I think that I can help other friends and demonstrate them that I can"; "I aspire to obtain a medal and join the national pre selection team", and the one which gave us the name of the current learning strategy, "I compete to be the first in the group (…)".

Besides this, the problems that the contestants tried to solve were monitored; and so were the participant observations. Some indicators were taken into considerations. For instance: what are the main subjects of conversation with their partners; what

interest do they have for the celebration of competitive activities; how are they motivated by the content; what kind of relationships they have with their partners; were they able to express their ideas openly in the group. More indicators are registered by the coach and then checked against the contestants' behaviour and their answers to interview questions.

The contestant's behaviour changes constantly and it is impossible to assure that they have "some kind of motivation" to any subject. When contestants begin competing, their young age brings on an exploration of wishes, motivations and interests, and the measuring of their knowledge.

Hence, it becomes difficult to ensure the presence of this learning strategy. An example of this is shown in the table below (Table 1), in which it has been represented the state of this strategy (LE) in the contestants (C). In all cases, checks were made in the middle of the school year. In the second year, most of the contestants who were in 10th and 11th grade from last year were repeated. In these grades, the contestants are unstable. As a consequence, they could stop joining the programming competition group with relative ease.

Table 1 also shows this learning strategy occurs more in the contestants of 11th and 12th grade. Such is the case, that sometimes they prioritize the learning of content related to programming skills to the extreme and neglect the subjects of their grade level curriculum.

In general, this marked interest manifested by contestants has been identified as a socio – affective learning strategy. Therefore, this strategy is contrasted with some cited study tools, and the result is that there are coincidences. For instance, in LASSI with the "attitude" and "motivation" scales, and in ASI with the "intrinsic motivation", "extrinsic motivation" and "achievement motivation" scales.

Although this strategy is of great importance and has considerable influence on the contestants, the coach should be careful, as it can cause the formation of habits not consistent with the objectives outlined in the competition group. The contestants should always remember that although they compete with peers to be "first" and that most tasks are performed individually, they must show solidarity, be honest and also act modestly. In the competition group must prevail the collectivism and exchange of ideas.

Generally, with this strategy comes also motivation to learn new content, its peculiarities and practical applications. This provides an efficient support for contestants when trying to solve programming problems.

Table 1

Contestants' coincidences of learning strategy "competitiveness to be first"

Grades	First Year			Second Year			Third Year			C		LE	
	C	LE	%	C	LE	%	C	LE	%	M	SD	M	SD
10th	8	4	50,00	9	3	33,33	9	5	55,56	8,67	0,58	4,00	1,00
11th	6	5	83,33	7	6	85,71	6	6	100	6,33	0,58	5,67	0,58
12th	3	3	100	4	4	100	4	4	100	3,67	0,58	3,67	0,58
Totals	**17**	**12**	**70,59**	**20**	**13**	**65,00**	**19**	**15**	**78,95**	**18,67**	**1,53**	**13,33**	**1,53**

4.2. *Behavioural Tendency to "Trial and Error"*

When the contestant solve exercises that represent a programming problem, he has to search the algorithm that coded into a programming language will correctly provide the results to all provided data sets. During this process of heuristic search in which he is creating an algorithm, the contestant, using an IDE, implements the source code solution. Here can we notice the use of the current learning strategy, when the source code is compiled into a runnable program and executed to evaluate if the solution is correct.

To diagnose this learning strategy, the C/C++ IDE, Code::Blocks, and a distributed revision control system, "git", was used. "git" was configured within the Code::Blocks, so that when contestant send a compile and run order, a "commit" is made taking the changes produced into the source code. This allows to register date and time of execution, and register also the changes made in the source code since the last execution. The "git" also tracks changes in modified lines, which gives a measure of the amount of changes made during each execution.

Once they have configured these applications on each computer, they proceeded to perform tasks. Each contestant had to solve a daily exercise for three consecutive days. Each exercise lasted 80 minutes. At two and four months, the same process was repeated. Each three days were called a "season". In the second season, one exercise from the first season is repeated with a different description and the same input and output order and file structure. In the third season, one exercise from the first season was presented, making sure it did not match the one repeated in the second season.

Regarding the lines modified in the source code, this depends of the coding style used by the contestant, the programming language used, and of the algorithm. However, despite this, we reached some notable conclusions. For instance:

- The contestants who had a partial solution to the problem, made more executions than the ones who solved it correctly.
- There is an inverse proportionality between the amount of modified lines and executions of the program made by contestant.
- The process of observing the source code building in its various states, is of much greater value to study than to observe the final program.
- Overusing this strategy does not lead to good results in the ratings of the contestants and leaves them little time to reason out the solution to the problem.

Some features that were observed in this process are:

- In the first 10 minutes, executions are not manifested, which is pretty obvious, because the contestant is performing the exercise of reading; although, in the observations made, many contestants write during this time some basic instructions that are common to any program, and are recognized as a standard or template.
- As time progresses, the number of executions increases and the modified lines decreases considerably, demonstrating the use of this trial and error learning strategy.
- Most of the contestants who showed a large number of executions in the exercises, failed the same problem and did not reach the highest score in the exercises that were repeated in the second and third seasons.

● In the lasts 10 minutes, there was a slight decrease in executions.

During experimentation, the most significant question we asked ourselves was whether "the contestants actually learn using this approach". The conclusion was that they do, and this was demonstrated on the data collected each year. While we acknowledge that the sample is not as representative as to ensure precision; it gave us a trend of behaviour and that at least "something" was obtained. This made us wonder again about, what do they learn during this process? It was obvious that in the contestants' interviews we were not going to find the answer to the question. So, in reviewing the modified source code before each execution, we found the answer.

The review of the various states of the source code during building the solution algorithm revealed that contestants using this trial and error procedure adjust the implementation of a solution algorithm, so they improve their knowledge and the internal structures that compose it.

Besides, as a direct practical study linked to the behaviour of the contestant, it was determined from the regularities which data structures or algorithms are more difficult when being encoded by the contestant.

Although it is not considered an unsuitable strategy for study sessions that have a predominantly problem-centric approach, it is necessary to be careful with its use and especially the number of times a contestant uses the strategy. The internalization of instructions of a particular programming language is a slow process and it is improved with practice. Also with the help of other learning strategies that are more effective than this; an example of which is the "trace" of source codes. However, inside source codes that are quite large and which complexity increases because the complex data structures have much processing to do, it is necessary to use the current learning strategy to understand and to adjust in a quick way the logic of coded algorithm solution.

This learning strategy should not become a habit and common practice for the contestant, because it doesn't help the reflection and interpretation of source code, and therefore it is a warning point that coaches should notice for its proper use.

4.3. *The "Trace" of Source Code*

Good programming practices recommend, in many cases, the use of tools to trace the functionality of the program's source code. Although trace functionality appeared for other purposes (debugging or looking for errors in program source codes) today, it is an essential tool in learning and internalizing functionality for the user who begins in computer programming.

This sequential process of tracing or debugging is executed step by step, and it is memorized in the subconscious of the contestant to then use it in interpreting code that is subsequently analyzed without the debugging tool. Hence the importance of its use, it helps encoding better algorithms that are created to solve programming problems.

This is evidenced by the results obtained using the same procedure described in the previous strategy. The only difference is that the Code::Block was also configured to record, each execution of the debugger "gdb" that is integrated into the IDE. This made it possible to know the lines "traced" and the variables that were observed during this process.

Following analysis of the data collected, the contestants were interviewed to refine and clarify the inferences obtained from the traces. Based on regularities and the integration of these results it was determined that:

- Novice contestants use it more as a "tracer", to understand how it was codified and what they aspire the implementation of the solution algorithm to be.
- More experienced contestants, use it as a "debugger", to find and fix the semantic errors given in algorithms have been coded in the program.
- Total average time of debugging uses by contestants of each grades in the 9 exercises (the same exercises described in the previous subsection), showed a clear predominance of 10th grade over the rest grades.
- Most of the answer regarding the use of "trace" corresponds to its use adjusting the source code to the solution algorithm found.
- In general, all contestants, in the 9 exercises, use this learning strategy.

In addition, those allowed knowing that data structures and algorithms cause major problems in the implementation. It also allowed to attend the individual differences of the contestants and to make a more detailed work regarding their use.

Similar than the previous learning strategy, when a contestant traces or debugs a source code program, he adjusts the implementation of a solution algorithm, so he improves his knowledge and the internal structures that compose it, and also, learns about his errors.

Tracing source code is peculiar in the sense that it is a learning strategy for those novices in computer programming, and then it is transformed into a tool as contestants develop knowledge based on the programming logic. It ceases to be a contestant's learning strategy to become a tool that is only used in some cases while trying to find the logic expressed in a source code of a program, or finding certain semantic errors that have been introduced during the encoding process.

Improving the use of this learning strategy is one of the pillars in programming teaching when the contestant begins his or her work on the competition group. For that, in many cases, the source code of program examples is given and contestants should be able to interpret and to deduce the solution.

4.4. *Reading Source Codes and Problem Solutions*

During the solving of programming problems, the contestants execute a program which solves the exercise and validate it against provided data sets, searching the validity and effectiveness of the response to each data set. Sometimes, the contestant does not reach the true solution, and needs impulse and stimulus oriented to enhance the knowledge already obtained. This is most likely the basis of the correct solution. However, even insisting on the coach presence to mediate in the problem solving process, the contestant does not always achieve right the solution.

In response, the contestant looks for strategies to acquire the knowledge that helps solving the problem correctly. Consequently, they rely on solutions that are part of the source code previously developed by another contestant or coach, or the description of

the algorithm (or combination algorithms) that solve the problem. Contestants do not always have the description or source code, so it is necessary that communication between group members flow freely; it is common some contestants have solved the problem before or know a possible solution.

Shown in Table 2, below, is a comparative study regarding the behaviour of the contestants (C) after solving each exercise, consultation to the description of the solution algorithm (SA) and the source code (SC) program. This was sampled over the first three years of the first season described in 4.1. For each, the correct solutions (CS), the partial solutions (PS) and incorrect (IS) were quantified.

From the analysis of these data and after reviewing the responses from the interviews with the contestants, the following regularities were found:

- When the solution is correct, the contestant rarely looks at the algorithm description, and even less at the source code.
- When the solution is partial, the contestant first reads the algorithm's description, then checks that the algorithm matches the one he used. After that, he checks the source code, trying to find necessary adjustments that this source code needs to be correct or fulfil the time and memory restrictions.

Table 2

Contestant's consultation of the algorithm's description and source code

		First Task				Second Task				Third Task			
		CS	PS	IS	T	CS	PS	IS	T	CS	PS	IS	T
10th Grade	C	2	3	3	8	0	5	3	8	3	3	2	8
	SA	0	2	1	3	0	2	3	5	0	2	1	3
	%	0	66,67	33,33	37,50	0	40	100	62,50	0	66,67	50	37,5
	SC	0	1	1	2	0	1	3	4	0	0	0	0
	%	0	33,33	33,33	25,00	0	20	100	50	0	0	0	0
11th Grade	C	1	3	2	6	0	3	3	6	1	4	1	6
	SA	0	3	2	5	0	3	3	6	0	4	1	5
	%	0	100	100	83,33	0	100	100	100	0	100	100	83,33
	SC	0	2	2	4	0	2	1	3	0	3	1	4
	%	0	66,67	100	66,67	0	66,67	33,33	50	0	75	100	66,67
12th Grade	C	1	2	0	3	0	1	2	3	0	2	1	3
	SA	1	2	0	3	0	1	2	3	0	2	1	3
	%	100	100	0	100	0	100	100	100	0	100	100	100
	SC	0	2	0	2	0	0	2	2	0	2	1	3
	%	0	100	0	66,67	0	0	100	66,67	0	100	100	100
Totals	T	4	8	5	17	0	9	8	17	4	9	4	17
	SA	1	7	3	11	0	6	8	14	0	8	3	11
	%	25	87,5	60	64,71	0	66,67	100	82,35	0	88,89	75	64,71
	SC	0	5	3	8	0	3	6	9	0	5	2	7
	%	0	62,5	60	47,06	0	33,33	75	52,94	0	55,56	50	41,18

- When solutions are partial and the description doesn't match the algorithm, the contestant experiences disappointment. In most cases, he or she does not check the source code.
- The contestants with more experience mostly consult the proposed algorithm, not the program's source code.
- Some contestants that prefer that a partner reads it and comments on the algorithm to use.

It is important to point out that the contestant is not forced to consult on the description and the source code when trying to solve the task. On the contrary, the necessary mechanisms are created for them to feel the necessity to study them as a part of self-learning. The first learning strategy described in this article influences this behaviour, along with the degree of motivation the contestant exhibits when applying previously acquired knowledge. However, it is necessary that he always consult at least the description of the proposed algorithm. This knowledge offers the tools to solve the current task as well as future problems.

The act of reading source code and algorithm description develops the independence contestants need in order to: acquire new knowledge, perfect implementation techniques, increase understanding of source code and improve preparation to solve problems of similar nature.

This learning strategy, when used properly, is remarkable for strong and accurate knowledge during cognitive process.

5. Other Learning Strategies

The technological influence in the training sessions of contests group allows other kinds of learning strategies that improve the use of autonomous learning. These strategies are within the cognitive type when reinforcing and applying the content previously received. An example of this is when the contestant makes summaries in digital documents or animations that visually show certain geometric and mathematical algorithms, among others. Also considered as metacognitive strategies are those that require planning. A set of actions and steps for reading and interpreting new algorithms, or the order in which contestants try to solve exercises from easy to more difficult.

Similarly, coaches can encourage the use of learning strategies common in other areas of knowledge, but are peculiarly absent in informatics contestants. Examples of these strategies are the graphic representation of certain data structures, and the creation of conceptual maps that summarize the content previously learned.

It may seem to the reader as if only the listed learning strategies are manifested in the informatics contestant, but this is not so. The challenge is to diagnose, discover and study them when they occur during training sessions or competitions, in order to improve learning.

Confirming Chamot and Kupper (1989) ideas, in interviews with contestants, they do not reflect and recognize the use of learning strategies. More of them express the use of strategies to solve problems and not for learning, confusing one with another.

It is necessary to note, that although there is little relation between them, learning strategies should not be confused with strategies for solving programming problems. The first have a very marked objective and addressed to what it is explained in this paper. The second are aimed to achieve in the shortest possible time, the solution to the problem, through a source code that must fulfil the requirements as expressed in the exercise.

6. Conclusions

All of the learning strategies explained in this paper are used unconsciously by contestants and are not recognized as such by them (self-learning strategies). There are other learning strategies that contestants use during competition, and with the practical methods to collect data as Code::Blocks C/C++ IDE + "git", the participant observations and interviews, more of them are identified.

Science, especially Pedagogy plays an important role in the study of this issue that is manifested in the didactic of computer science, and specifically of computer programming, since promoting these activities causes an individual impact, collective and social, that is triggered by the results obtained during national and international competitions.

Learning is not only manifested when a new programming language or a new algorithm is learned, but also when the contestant solves a task that constitutes a problem for him.

The method proposed by Charguéraud and Hiron (2008) has been applied in our informatics contests activity, it has created some learning strategies in contestants.

References

Campos-Maura, E. (2006). *Estrategia Metodológica para la Preparación de Alumnos que Participan en Concurso de Español-Literatura en Preuniversitario.* PhD thesis in Pedagogical Sciences, Villa Clara, Cuba.

Carrasco, J.B. (2004). *Estrategias de Aprendizaje para Aprender Más y Mejor.* RIALP (Eds.), Alcalá, Madrid.

Chamot, A.U., Kupper, L. (1989). Learning strategies in foreign language instruction. *Foreign Language Annals*, 22(1), 13–22.

Charguéraud, A., Hiron, M. (2008). Teaching algorithmics for informatics olympiads: the french method. *Olympiads in Informatics: International Journal.* 2, 48–63.
http://www.mii.lt/olympiads_in_informatics/htm/INFOL018.htm

CUBA. (2007). *Ministerial Resolution 91/2007: Indications Related with Application of Knowledge and Skill Contests.* Ministry of Education, La Habana, Cuba. (In Spanish)

Ferrera, R.A. (2008). *Estrategias de Aprendizaje. Construcción y Validación de un Cuestionario-Escala.* PhD thesis. Universitat de Valencia, Servei de Publicacions.

Hernández-González, F.J. (2008). *Metodología para el Entrenamiento de los Estudiantes de Preuniversitario que Participan en Concurso de Informática.* PhD thesis in Pedagogical Sciences. Villa Clara, Cuba.

Marland, S.P., Jr. (1971). *Career Education: A Report.*
http://www.eric.ed.gov/PDFS/ED080743.pdf

Nisbet, J., Schucksmith, J. (1986). *Estrategias de Aprendizaje.* Madrid: Santillana.

Ortiz, L., Salmerón, H., Rodríguez, S. (2007). La enseñanza de estrategias de aprendizaje en educación infantil. *Profesorado. Revista de Currículum y Formación del Profesorado*, 11, 2.

Oxford, R.L. (2003). Language learning styles and strategies: an overview. *Learning Styles & Strategies/ Oxford, GALA*, 1–25.

Revilla, M.A., Manzoor, S., Liu, R. (2008). Competitive learning in informatics: the uva online judge experience. *Olympiads in Informatics. International Journal.* 2, 131–148.
http://www.mii.lt/olympiads_in_informatics/htm/INFOL035.htm

Velázquez-Cobiella, E.E. (2011). El reto a la educación: el futuro de las naciones latinoamericanas y caribeñas. In: *Conferencia Inaugural, Pedagogía 2011*. La Habana. Cuba.

Velázquez-Cobiella, E.E. (2010). La Educación en Cuba y los retos del personal docente. In: *7mo Congreso Internacional de Educación Superior. Universidad 2010*. La Habana. Cuba.

Verhoeff, T., Horváth, G., Diks, K., Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 4(1), 193–216.

Weinstein, C.E., Wicker, F.W., Cubberly, W.E., Roney, L.K., Underwood, V.L. (1980). D*esign and Development of the Learning Activities Questionnaire*. (Tech. Rep. No 459.). Alexandria, Va: U.S. Army Research Institute for the Behavioural and Social Sciences.

Weinstein, C.E., Goetz, E.T., Alexander, P.A. (1988). *Learning and Study Strategies: Issues in Assessment, Instruction and Evaluation*. Nueva Cork: Academic Press.

Weinstein, C.E., Mayer, D.K. (1991). Implications of cognitive psychology for testing: contributions from work in learning strategies. In: Wittrock, M.C., Baker E.L. (Eds.), *Testing and Cognition*. Englewood Cliffs. N.J.: Prentice-Hall.

Weinstein, C.E., Underwood, V.L. (1985). Learning strategies: the how of learning. In: Segal, J.W., Chipman, S.F., Glasser, R. (Eds), *Relating Instruction to Research* (*Volume 1 of Thinking and Learning Skills*)*. London, Lawrence Erlbaum Associates.

G. Cuba-Ricardo is a doctoral student at the Curricular Doctorate of the University of Pedagogical Sciences of Holguín, Cuba. He is a researcher at the Department of Resource Development for Learning. His main research interest is the role of computer programming in educational processes and contestant training. He is a consultant and a coach of the informatics contests team in Holguín city.

P.A. Leyva-Figueredo holds a Ph.D. in Pedagogical Sciences. He is the Director of the Centre of Studies for Labor Education and Coordinator of the Doctorate Collaborative Curricular Program at the University of Pedagogical Sciences of Holguín. He has an extensive experience in labor education, doctoral training, research methodology, professional skills and professional guidance. Also, he is a member of the Provincial Scientific Committee of Science, Technology and Environment (CITMA) and a Permanent Member of the Evaluation Board to get the Scientific Degree of Doctor in Pedagogical Sciences.

L.L. Mendoza-Tauler received her Ph.D. in Pedagogical Sciences in 2001. She is the Director of the Centre of Studies in Educational Research at the University of Pedagogical Sciences of Holguín and Coordinator of the Ph.D. Program of UBV-2 Caracas, Venezuela. She teaches at the Ph.D., Masters and Qualified Program Studies in Venezuela and Perú. She is a member of the Academy of Sciences of Cuba in the Commission of Social Sciences. She is also a member of the Provincial Scientific Committee of Science, Technology and Environment (CITMA) and of the Evaluation Board to get the Scientific Degree of Doctor in Pedagogical Sciences.

International Olympiad in Informatics: Team Selection, Training, and Statistics – The Tale of Two Countries

Valentina DAGIENĖ[1], Ela ZUR[2], Tamar BENAYA[2]

[1] *Vilnius University, Institute of Mathematics and Informatics*
 Akademijos st. 4, Vilnius, LT-08663 Lithuania
[2] *The Open University of Israel, Computer Science Department*
 108 Ravutzky st. Raanana, Israel 43107
e-mail: valentina.dagiene@mii.vu.lt, {ela, tamar}@openu.ac.il

Abstract. The paper discusses the issue of supporting informatics (mainly focusing on programming) education through competitions for secondary school students. Competitions play an important role for learners as a source of inspiration, innovation, and attraction. The International Olympiad in Informatics (IOI) is the primary computer science competition for young students, up to the age of 20. The primary goals of the IOI are to stimulate challenges in Informatics among exceptionally talented young students from all over the world, and have them share scientific and cultural experiences. We describe the selection and training process in Israel and Lithuania. An overview of infrastructure and development of competitions from international and regional levels to the national one (Israeli and Lithuanian) is presented in short. In addition we provide some statistics from the years 2010 to 2014, such as: Israeli and Lithuanian medals, number of participants in the different stages of the training process, and age and gender distribution of contestants in the National contests. We conclude with a discussion comparing the IOI projects in both countries.

Keywords: learning through competition, programming contests, training, olympiads in informatics, IOI.

1. Introduction

The IOI – the International Olympiad in Informatics – is the primary computer science (CS) competition for young students, up to the age of 20. The IOI is one of several annual international youth olympiads, including: the IMO in mathematics, the IPHO in physics, the ICHO in chemistry, the IBO in biology, and the IAO in astronomy. The IOI is hosted every year by a different country. It started with 13 participating countries, in Bulgaria in 1989, and expanded to more than 80 countries today.

The primary goals of the IOI are to stimulate challenges in CS among exceptionally talented young students from all over the world, and have them share scientific and cultural experiences. Each participating country conducts a preparation process, and brings an IOI team which includes four contestants. In the IOI, the contestants compete individually in the course of two competition days, each involving three challenging algorithmic tasks, to be solved and programmed.

The task solutions require careful task analysis, insightful correctness and efficiency considerations, and skilful programming implementation. Creativity, competence in algorithmic topics (Verhoeff *et al*., 2006), and implementation accuracy are essential. The better half of the students in the two-day competition, win gold, silver, and bronze medals.

Teachers have noticed that competitions are very important for students to improve their skills in programming. Just the idea of participation in a competition is often enough to increase significantly students' motivation level to learn programming. The competition structure usually allows comparing students' work to that of their peers. These opportunities give positive evidence regarding the strength of one's own capabilities. International competitions are also very useful networking events both for students and teachers.

Different countries invest different amounts of effort and resources in preparing their IOI teams (e.g.: Diks *et al*., 2007, Casadei *et al*., 2007, Forisek, 2007, Kolstad and Piele, 2007), yet the preparation outlines seem similar. A call-for-participation engages an initial amount of interested students, from whom the top ones are chosen, through a selection and training process. In what follows, we briefly describe the selection and training process both in Israel and in Lithuania, and then display some statistics of this process and participation in the IOI.

2. The Selection and Training Process

2.1. *The Israeli Case*

In Israel, the IOI project is operated and supported by Tel-Aviv University, the Open University of Israel and the Ministry of Education. The primary objective of the project is to offer challenges in CS to motivated students, who show interest and competence in problem solving in general, and algorithmic problem solving skills in particular.

The project is composed of four stages: a regional competition; a national competition, an advanced training and team-selection stage, and the national team's preparation to the IOI. The different stages are operated by a small training team, of five to six trainers – the head coach and his deputy, a couple of high-school teachers, and a couple of former IOI contestants.

2.1.1. *Regional Competition*
The 1st stage is conducted at the beginning of the winter. It starts with a call-for-participation for the regional competition sent to high-schools and posted in the national CS teachers' website (maintained by the high-school CS inspector in the ministry of education). The interested students are referred to the project's website (The Israeli IOI Project Website), and are encouraged to prepare for the regional competition, by self-studying rather basic programming and data-structure constructs (e.g., recursion and trees) and solving previous national competition tasks. The goal of the regional competition is to offer algorithmic challenge to an audience as wide as possible; to engage CS secondary

school teachers in posing the challenge; and to identify competent students, who will advance in the project activities.

A 5-questions questionnaire was posted in the website of the CS inspector of the Ministry of Education. The questionnaire was posted at a given time, which was a-priori told to all the secondary schools in Israel. Secondary school teachers, downloaded the questionnaire, and posed it to their selected students, as a 2-hour exam. Questions during the exam, about the exam tasks, were directed in real-time (phone) by the teachers to the training team. The students wrote their answers on exam sheets, which were downloaded from the internet. All the sheets were sent to the training team for grading. A couple of days after the exam, the solutions were posted, with broader perspectives of notions that appeared in the exam questions.

The teachers' role in this activity was to encourage their better students, and have them take the exam. They supervised their students during the exam, and sent to the team the student notebooks.

As one of our goals was to expose the project to as wide an audience as possible, we posed algorithmic tasks for which the required answers were not an algorithm, but rather the outcome of an algorithmic computation. This approach offers the opportunity of reaching students who are less acquainted, or even unacquainted with programming. The exam questions focused on mathematical and algorithmic insight, on which one had to capitalize her/his computation.

We invited to the next stage all those who obtained a score of 80+, plus students who obtained a lower score but nicely answered one or more of the insightful sections in the questions. We expected students to learn from our posted solution, and from our previous national competitions, in preparing for the next stage – the national competition.

2.1.2. *National Competition*

The 2nd stage is conducted in the late winter (February). It involves the national competition, which is a three-hour exam, with pencil and paper. The students are gathered together, and are asked to solve four algorithmic tasks, and provide a written description of their solution idea and their solution code, or pseudo-code (according to their preference). The goal of the exam is to identify the students that demonstrate the highest potential, primarily in problem solving. Thus, the CS knowledge required at this stage is relatively basic.

The first task of the national competition usually requires recursion, which may be implemented with a rather simple dynamic-programming scheme. The second task involves a mathematical game, or a similar task, whose solution is based on a hidden invariant property. The third and fourth tasks are more involved, in terms of the required insight and the solution scheme. Yet, the code required for each of the tasks is rather short. The students are explicitly directed to focus on task analysis, and carefully notice correctness and efficiency considerations. The exam format and example questions are described in a previous paper (Zur *et al*., 2011). In grading their solutions, the team particularly examines their creativity, accuracy, and scientific discipline. They pay less attention to detailed programming features, as long as the criteria indicated above are met.

We select the best 30 students, plus possibly a few additional ones, in cases where there are females or students from remote schools that are close to the top 30. All these students are invited to the next stage.

2.1.3. *Advanced Training and Team-Selection Stage*

The 3rd stage is conducted in the spring. The objective in this stage is to teach the top 30 students more advanced algorithmic and problem solving features, and test them about these features. The top four students of this stage are chosen for the national team. This stage involves 5–7 practice days (one or two such days a week). It does not involve a camp (as offered in some other countries), but rather a day gathering in a computer lab, due to our limited resources.

Each practice day lasts 8–10 hours. Prior to that day, students are asked to study particular topics (e.g., basic graph algorithms). In the first part of the day, three algorithmic tasks are posed to program in five hours, which involve the indicated topics and the previous days' topics. The students are asked to both program their solutions and write on paper their solutions' underlying idea. At the end of this activity each student is interviewed about his/her solutions. The goal of the interviews is to examine their insight and extract potential errors and difficulties that arise and recur. In addition, the student programs are tested on diverse test-cases.

Following the interviews and the program evaluations, all the participants are gathered for a two–three hour discussion on the day's task solutions and their related CS topics. The discussion involves particular focus on insightful analysis, common errors, and essential efficiency considerations. The latter is particularly underlined, as many of the posed tasks may be solved in several ways, of different time and space complexities. The trainers strongly emphasize two elements: potential and recurring errors and algorithmic and problem solving features used in the day's task solutions, which are relevant beyond these tasks (e.g., particular task representations and illuminating perspectives). Some of these elements are described in papers and columns (Ginat, 2001, 2003a, 2003b, 2007).

At the end of the practice day, the students are asked to program at home alternative solutions that were discussed, and to further study the algorithmic and problem solving features that were examined. At the end of these 5–7 practice and evaluation days, four students that demonstrated the best accumulated performance, in both algorithmic problem-solving and programming, are selected to the national team. The rest of the students are encouraged to return in the following year and convince other students from their schools to join as well.

2.1.4. *National Team's Preparation to the IOI*

The 4th stage is conducted thereafter and usually lasts up to two months, until the IOI. In this stage, the team is directed to learn and practice the topics relevant for the IOI, solve previous IOI and additional olympiads' tasks, and thoroughly practice the programming features required in the IOI. The team members meet with the project trainers once every one or two weeks, practice task solutions, discuss solutions, and receive advice and tips from previous team members who competed in the IOI. A particular emphasis is put on one's selection of test-cases before submission.

2.2. *The Lithuanian Case*

The teaching of informatics has a long tradition in Lithuanian schools; a rich experience in the field has been accumulated (Dagienė, 2006). However we complain that in the last decade our schools have spent too much attention on application of information technologies. The education programme of lower secondary schools, starting with the fifth grade, includes a separate course on information technology (IT), a part of which is devoted to introduce programming using Logo or Scratch. Students have a possibility to choose an optional programming module in grades 9 and 10. After that they can continue with an advanced programming module in grades 11 and 12.

However students have possibility to obtain deeper programming skills while participating in various non-formal activities: Young Programmers' School (Dagys et al., 2006), olympiads and contests in informatics (programming). A combination of all these activities leads students to the IOI (Fig. 1).

2.2.1. *National Olympiad in Informatics*
The first Lithuanian nation-wide informatics olympiad was organized in 1990, i.e. the year after the first IOI in Bulgaria.

In the beginning the olympiad consisted of the three stages: the 1st is stage was organized in autumn in schools; the 2nd stage was conducted in December by municipalities (60 municipalities in Lithuania). The main goal of participants was to qualify for the next level competitions.

The 3rd stage, named as a national stage, was conducted in spring. Since 1993 the national stage has been split into two parts: the 3rd and 4th stages. Initially the 3rd stage was organized using e-mail, later and nowadays participants of the 3rd stage submit their solutions through a contest management system; the 4th – final-stage is an on-site con-
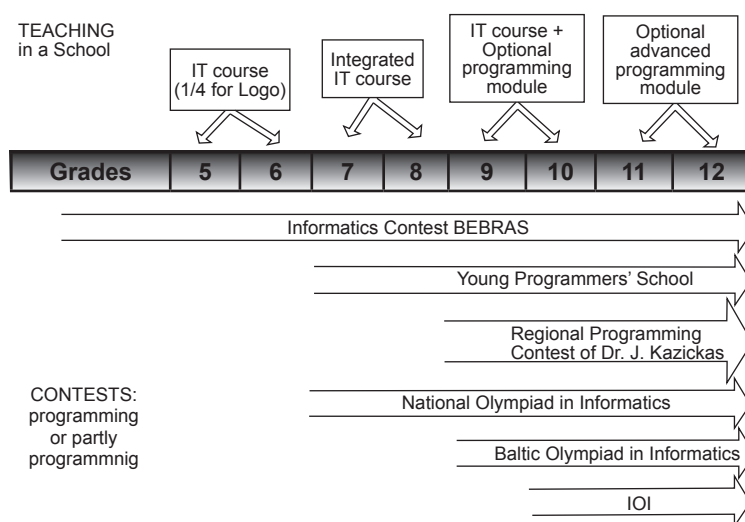
Fig. 1. Formal and non-formal ways of teaching programming in Lithuania.

test. The structure of four-stage-olympiad is more convenient and is used until now. In the each of the 1st–3rd stage students should solve three algorithmic tasks during a half of day (4–5 hours). The students are asked to provide their solution code and the description of the solution idea.

The final stage of the national olympiad is organized in a different region each year. Organizing the event in different regions not only allows the contestants to get to know the region but also gives a possibility to the teachers of local schools to look at the olympiad from inside – to observe how the final versions of tasks are being prepared, and to look closer at the contest system and the grading process.

About 50 participants from all over Lithuania are invited to compete in face-to-face exams. Students solve from 5 to 7 algorithmic tasks during two competition days (five hours each day). The competition days are combined with some leisure activities (sports, games, excursions, museums, etc.).

2.2.2. *Organizing on-Line Contests*
Twenty years ago the structure of the national olympiads in informatics was quite complicated. Each of the sixty municipalities in Lithuania designated winners of their competition for the national stage. As it was not possible to arrange an on-site competition for more than two hundred students, the 3rd stage used to be arranged in several selected municipalities at the same time.

A significant breakthrough became possible in 1993, when the computer network became available for several schools in each municipality. It was decided to organize the 3rd stage in each region using e-mail. Solutions were delivered through e-mails and afterwards graded using black-box testing for the ten years from 1993 until 2002.

The automatic contest management and grading system that allows the submission of programs via a web-interface during contest time, and checking whether they compile and comply with format requirements, has been used in Lithuanian olympiads since 2003.

All students of lower and upper secondary schools are invited to participate in informatics olympiads. Approximately 3000 students take part in 1st stage each year. The number of younger (grades 7–9) participants has significantly increased when a separate division for younger students was established and 30% of the places in the finals of the national competition were reserved for students from younger division. This motivated both younger students and their teachers.

2.2.3. *Baltic Olympiads in Informatics*
In order to ensure better preparation for the IOI and to strengthen regional relations, various regional olympiads are being organized. The Baltic Olympiads in Informatics (BOI) were established by the initiative of the three Baltic countries – Estonia, Latvia, and Lithuania – in 1995. Year by year six other Baltic countries (Denmark, Finland, Germany, Poland, Sweden and Norway) joined the BOI and now all these countries send their teams annually. The host countries still maintain the tradition of inviting guests to BOI. In 2005 Lithuania invited an Israeli team.

Compared to the IOI, the BOI is a short-term (the duration is 5 days) and inexpensive event. It has a cosy and good neighbourly atmosphere.

The organisation of BOI has changed over the years. To keep the event manageable, the number of contestants per team was decreased from 8 to 6.

Even though BOI is a mini-model of IOI it differs significantly. The organization of the scientific part of the BOIs is based on mutual trust of participating countries. The leaders of all the participating countries take part in proposing and selecting problems for the coming BOI. After draft problem formulations are presented, the problems are discussed via e-mail and each country takes part in vote for the problem set for the competition. Most of the problems are translated into native languages by the leaders before leaving for the BOI.

Each country is asked to submit at least one task proposal – with 9 participating countries there is no additional need for each country to come up with more proposals. Tasks are algorithmic in their nature:

1) Combinatorial search tasks where it is possible to go through all reasonable solutions (possibly with some optimisations) and choose the optimal solution.
2) Dynamic programming tasks where the problem can be divided into independent sub-problems.
3) Graph theory tasks where the problem can be transformed into a graph and solved by constructing a graph algorithm.
4) Mathematical tasks which include the tasks concerning arithmetic, geometry, number theory and probability.

Also unusual, innovative tasks which require an original non-traditional solution method or algorithm are very welcome. Even though all the tasks are of an algorithmic nature they represent cultural and methodical differences.

Automated contest and grading systems, mainly developed and maintained by the host country, are used to manage the contest. The neighbourly help of countries with more experience of managing contests to host countries with less or no experience makes it possible to host well organised contests in all countries.

During the competition leaders are involved in solving various problems which might occur, for example, some misrepresentation in the formulations of contest problems. This is a unique possibility for country representatives to gain experience in organizing scientific part of a small international olympiad (Poranen *et al*., 2009).

The BOI is also a form of learning for its participants. The organizers of BOIs try to follow as close as possible the newest IOI trends in problem types, compilers, platforms and contest systems. It is not always possible to do that in national contests. Many students come to the BOI to gain international experience after participating in domestic contests. The BOI can be considered as a pre-arranged international form of learning.

2.2.4. *National Team's Preparation to the IOI*

The regional BOI serves as selection of students for the IOI (Gal-Ezer *et al*., 2009). Usually Lithuania selects the best 4 students from the 6 students participating in BOI. The BOI is organized at least two months before IOI so there is still time for students to learn and practice the topics relevant for the IOI, solve tasks and practice the programming features required in the IOI.

A week long face-to-face training session is organized before each IOI, usually during summer time. In the training session not only the 4 IOI team students are invited to practise but also up to 10 best participants from the national olympiad that were not invited to the IOI team, especially the younger ones who are expected to be candidates for the IOI team in the future. Former IOI participants volunteer to work in the training session.

3. Some Statistics

This section presents some statistics regarding achievements of contestants, participation in the different stages of the IOI project, age and gender distribution of participants.

3.1. *Medal Distribution*

The main achievements of the IOI are medals. Achievements of both Israeli and Lithuanian teams are similar during years however last year was very successful for Israel – 4 medals including a gold one (Table 1). These achievements might be attributed to the increased funding that the Israeli IOI project received from the Ministry of Education in the year 2013. The funding enabled hiring additional trainers and organizing more extensive training sessions.

Table 1

Achievements of the Israeli and Lithuanian teams in the IOI

Year	Israel			Lithuania		
	Gold	Silver	Bronze	Gold	Silver	Bronze
2010			3		1	2
2011		2			1	2
2012		2	1		2	2
2013	1	2	1		1	
Total	**1**	**6**	**5**	**0**	**5**	**6**

Table 2

Participation in the Israeli IOI project

Year	Regional Competition	National Competition	Advanced Training and Team Selection
2010		251	30
2011	1442	674	22
2012	1767	359	33
2013	1131	263	32
2014	1283	386	37

3.2. *Participation in the IOI Project*

Table 2 shows the number of students who participated in the different stages of the Israeli project in the years 2010 to 2014. The regional competition began in 2011 therefore we do not have data for the number of participants in the regional competition in 2010. These students come from approximately 20 to 70 different high schools located all over the country.

The number of participants in the Israeli national competition is usually 250 to 400 except in 2011 where the number of participants was much higher (Table 2). This can be attributed to the fact that in that year we started to conduct the regional competition which exposed many students to the olympiad project.

The number of participants in the different stages of the Lithuanian Olympiads in Informatics is presented in Table 3.

3.3. *Age and Gender Distribution in the IOI Project*

The majority of the participants in the Israeli national competition are 11th and 12th grade students (17–18 years old). In all the years except 2012, 11th grade had the highest number of participants (Fig. 2). The reason that the number of participants in the 12th grade

Table 3

Participation in the Lithuanian IOI project

Year	1st stage	2nd stage	3rd stage	Final	Training
2010	~2200	~750	303	53	11
2011	2220	748	336	50	11
2012	2217	750	330	51	12
2013	2507	791	345	49	10
2014	2319	943	298		
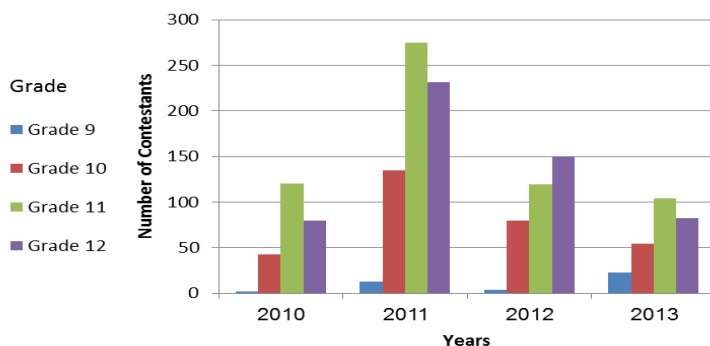
Fig. 2. Grade distribution in the Israeli national competition.

is a bit lower than the number of participants in the 11th grade is because the 12th grade students are busy at the time of the national competition with the high school matriculation exams.

Table 4 shows the percentage of female and male participants in the national competition of the Israeli IOI project in the years 2010 to 2013.

The percentage of female participants in the national competition has decreased from 30% to 13%. The percentage of female students who select CS in high school is approximately 30% (Gal-Ezer *et al*., 2009).

In spite of our efforts throughout the years to increase female participation, we found that girls are less attracted to competitions and therefore they avoid their participation in the competition. Throughout the years very few girls have been selected for the advanced training stage but the team trainers have invited the girls who achieved best results in the national competition to participate in that stage. We tried to increase teacher's motivation and involvement in the IOI project, particularly in attracting more girls to the different stages of the project (Dagienė and Skūpienė, 2004).

Fig. 3 shows the age distribution of students who participated in the 2nd stage of the Lithuanian project in the year 2011 to 2014.

In Lithuania traditionally programming is a "boys" subject. Very few girls have chosen to participate in the National Informatics Olympiad because it is a purely programming contest. A very small number of girls participate in the 1st and 2nd stages, almost no girls in the 3rd and 4th stages.

Table 4
Gender Participation in the Israeli national competition

Year	Male	Female
2010	70%	30%
2011	80%	20%
2012	77%	23%
2013	87%	13%
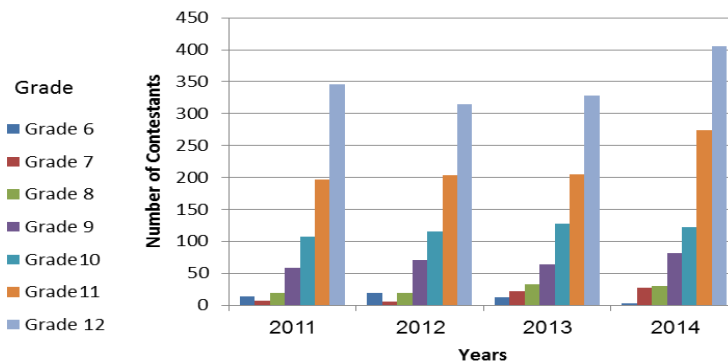
Fig. 3. Grade distribution in the Lithuanian national competition (2nd stage).

4. Summary and Discussion

The IOI project stimulates challenges in CS among exceptionally talented young students and enables sharing of scientific experiences. The tasks, with which the students are faced, require careful task analysis, insightful correctness and efficiency considerations, and skilful programming implementation. Creativity, competence in algorithmic topics, and implementation accuracy are essential. This extracurricular activity promotes talented students and benefits CS studies in participating countries.

Both countries put a lot of effort into the selection and training process. The achievements of both countries are similar. Some of the notable differences are:

Lithuania involves the teachers in the first stages of the selection process (particularly in the 1st and 2nd stages), while in Israel most of the work is done by the training team. Israel has tried in the past to involve the teachers but most of the teachers avoid involvement because they feel that they do not have enough experience with such questions and this puts the teacher in an uncomfortable position (Zur *et al*., 2012). We believe that with a proper teacher training, the teachers will feel more comfortable to collaborate with the training team. This collaboration will contribute both to the selection and training process.

The final stages of the training process in Lithuania include participation in the regional olympiad (BOI Olympiad) while in Israel the participation in regional olympiads has begun only recently. There is no doubt that this participation contributes greatly to the final training and selection process.

As we can see from the above sections, each country developed a unique selection and training program. Fig. 4 summarizes the selection and training stages in both countries.

All in all, the IOI project in Israel is rather modest. The training team's hope is to extend their resources and activities in the coming years, expand the training team, hopefully with additional IOI veterans, and attract a larger number of interested students (males and females) already in the early stages.
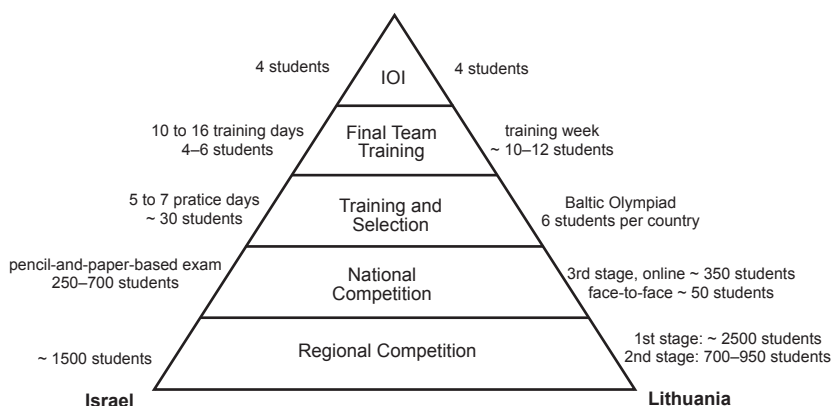
Fig. 4. Competition, selection and training stages in both countries.

References

Casadei, G., Fadini, B., Genovie De Vita, M. (2007). Italian Olympiads in informatics. *Olympiads in Informatics*, 1, 24–30.

Dagienė, V. (2006). *The Road of Informatics*. Vilnius, TEV.

Dagienė, V., Skūpienė, J. (2004). Learning by competitions: Olympiads in informatics as a tool for training high grade skills in programming. In: T. Boyle, P. Oriogun, A. Pakštas (Eds.), *2nd International Conference Information Technology: Research and Education*. London, 79–83.

Dagys, V., Dagienė, V., Grigas, G. (2006). Teaching algorithms and programming by distance: quarter century's activity in Lithuania. In: *Proc. of the 2nd Int. Conference on Informatics in Secondary Schools: Evolution and Perspectives, Vilnius, 7–11 November*. 402–412.

Diks, K., Kubica, M., Stencel, K. (2007). Polish Olympiad in informatics – 14 years of experience. *Olympiads in Informatics*, 1, 50–56.

Forisek, M. (2007). Slovak IOI 2007 team selection and preparation. *Olympiads in Informatics*, 1, 57–65.

Gal-Ezer, J., Shahak, D., Zur, E. (2009). Computer science issues in high school: gender and more… *Inroads SIGCSE Bulletin*, 41(3), 278–282.

Ginat, D. (2001). Misleading intuition in algorithmic problem solving. In: *Proc. of the 32nd ACM Computer Science Education Symposium*. SIGCSE, ACM Press, 21–25.

Ginat, D. (2003a). Board reconstruction, colorful challenges column. *SIGCSE Bulletin*, 35 (4), 25–26.

Ginat, D. (2003b). The greedy trap and learning from mistakes. In: *Proc. of the 34th ACM Computer Science Education Symposium*. SIGCSE, ACM Press, 11–15.

Ginat, D. (2007). Hasty design, futile patching and the elaboration of rigor. In: *Proc. of the 12th Conference on Innovation and Technology in Computer Science Education*. ITiCSE, ACM Press, 161–165.

Kolstad, R., Piele, D. (2007). USA computing olympiad (USACO). *Olympiads in Informatics*, 1, 105–111.

Poranen, T., Dagienė, V., Eldhuset, A. et. al. (2009). Baltic Olympiads in informatics: challenges for training together. *Olympiads in Informatics*, 3, 37–49.

The Israeli IOI Project Website. `http://www.tau.ac.il/~cstasks`

Verhoeff, T., Horvath, G., Dicks, K., Cormak, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 4, 193–216.

Zur, E., Benaya T., Becker, O., Ginat, D. (2011). IOI Israel: the regional and national competitions. *Olympiads in Informatics*, 5, 161–168.

Zur, E., Benaya, T., Becker, O., Ginat,, D. (2012). Israel – the regional competition and teacher involvement. *Olympiads in Informatics*, 6, 218–225.

V. Dagienė is a head of Informatics Methodology Department at the Institute of Mathematics and Informatics of Vilnius University. Her current research is in Computer Science Education, focusing on cognitive aspects of algorithmic thinking as well as computational thinking. She has published over 200 scientific papers and methodological works, has written more than 50 textbooks in the field of informatics and information technology for primary and secondary education. She works in various expert groups and work groups, organizing the olympiads and contests. She has participated in several EU-funded R&D projects, as well as in a number of national research studies connected with technology and education.

E. Zur is involved in the Israel IOI project since 1997, and repeatedly served as a deputy leader. She holds a PhD Degree in Computer Science Education from Tel-Aviv University. She is a faculty member of the Computer Science Department at the Open University of Israel. She designed and developed several advanced undergraduate Computer Science courses and workshops, and currently serves as a course coordinator of several courses. Her research interests include Distance Education, Collaborative Learning, Computer Science Education, Computer Science Pedagogy, Teacher Preparation and Certification and Object Oriented Programming.

T. Benaya holds a M.Sc. in Computer Science from Tel-Aviv University. She is a faculty Member of the Computer Science Department at The Open University of Israel. She designed and developed several advanced undergraduate Computer Science courses and workshops, and she serves as a course coordinator of several courses. She also supervises student projects. She is a lecturer of Computer Science courses at The Open University of Israel. Her research interests include Distance Education, Collaborative Learning, Computer Science Education, Computer Science Pedagogy and Object Oriented Programming.

Technology for the Development of Thinking of Preschool Children and Primary School Pupils

Michael DOLINSKY

Department of Mathematics, Gomel State University "Fr. Skaryna"
Sovetskaya str., 104, Gomel, 246019, Republic of Belarus
e-mail: dolinsky@gsu.by

Abstract. This article describes an elaborated technique for creating computer exercises that were used to diagnose and develop thinking and learning skills of hundreds of scholars of Gomel and Gomel Region at Distance Learning Belarus site (`http://dl.gsu.by`).

The exercises consist of basic intellective operations that can be classified into 5 groups: operations with pairs; operations with sets; operations on a set; Boolean operations; complex operations.

Keywords: children's thinking, preschool, assessment, training, computers.

1. Introduction

Early childhood development is under permanent analysis. Among the most interesting aspects of the analysis are curriculums, challenges of evaluation (Lia and Wongb 2008), thinking and mathematics as the main results of preschool education (Aubrey, Ghent, and Kanira 2012); pro and contra using of computers (Howard, Miles, and Rees-Davies 2012); preschool-home cooperation, active learning.

This article describes the theory and practices of all these aspects in the author's project. For many years the author was engaged in training Gomel's schoolchildren for programming contests. The training is supported by the computer learning site (`http://dl.gsu.by`, further DL) created by students of the mathematical faculty of the Gomel Fr.Scaryna State University under the direction the author (Dolinsky M., 2012). Over the years the training was started at increasingly early ages; from the year 2007 it has begun with first grade. Quite effective teaching system had already been developed by that time. However the author began to think, WHY does the progress vary so much among children having the same good teaching system the same motivation and identical investment of time for training? The author came to the following conclusion: the problem is that the pupils have different initial thinking ability. Basic DL exercises relay in a substantial degree on pupils' skills to compare, analyze and draw conclusions. But the pupils are so differently skilled in thinking that this is the fundamental factor in their advancement speed in the curriculum. It turns out that it is more advantageous to precede programming training with preliminary training of general thinking skills. But how to diagnose and develop thinking? What components do thinking consists of? Finding an-

swers to these questions helps to develop the exercises specially aimed at levelling each of these components. For the "components of thinking" the author uses a term "Basic intellective operation".

The author aimed to develop a practical set of basic intellective exercises that can be integrated in the DL system to provide multitude of operating advantages including effective simultaneous training of many pupils by single teacher, accessibility both from school and home, accumulation of training results and subsequent statistical analysis.

Section 2 introduces "Basic intellective operations". Section 3 describes obtained results. Section 4 contains conclusions.

2. Basic Intellective Operations

The author distinguishes the following set of base intellective operations:

- Operations with pairs: comparison, rearrangement, association.
- Operations with sets: union, intersection, subtraction.
- Operations on a set: classification, structuring, generalization.
- Boolean operations: negation, conjunction, disjunction, equivalence, implication.
- Complex operations: synthesis, memorizing, analysis, imagination, analogy, abstraction, positioning.

Then follow descriptions of the proposed base intellective operations that also allow the selection of exercises for their diagnostics, development and control. Also provided are concrete exercise examples that are based on operations with pictures and don't require ability to read. It allows the user of exercises with wide age range starting from preschool children.

Obviously, it is difficult enough to select exercises that develop or diagnose selected single base intellective operation separately. At the same time, it is possible to select exercises having one of the base intellective operations dominant. In addition, the author promotes a concentric training plan when exercises are also split by the levels of complexity. At the beginning all base intellective operations are included at the first level of complexity, then goes the second level and so on.

Operations with Pairs

Comparison of two or more objects to find differences or the same parts.
    For example:
- Select given picture.
- Put a picture on the same picture (then both disappear).
- Find the differences between the left and the right pictures.

Rearrangement of several objects (pictures, letters, words) to position them in some special order (by colour, form, size).
    For example:
- Transposition of numbers in ascending or descending order.
- Transposition of triangles in order of size growth.

Association – to specify a some kind of relation.

    For example, for some pictures:

- «Whose kid?».
- «Whose house?».
- «Professions».

Operations with Sets

Union of elements of source sets.

    For example:

- There are given some pictures presenting two sets of objects.
- There is an outlined area where it is required to copy pictures making the union of two sets.

Intersection of elements of source sets.

    For example:

- There are given some pictures presenting two sets of objects.
- There is an outlined area where it is required to copy pictures making the intersection of two sets.

Subtraction – the set of elements from the first set that are absent in the second one.

    For example:

- There are given some pictures presenting two sets of objects.
- There is an outlined area where it is required to copy pictures making the subtraction of two sets.

Operations on a Set

Classification – to split up the set of objects in subsets by accordance to some criterion.

    For example:

- The pictures of geometrical figures (triangles, squares, circles of different colours and sizes) are given.
- It is required to move figures into corresponding areas.
- Triangles into one area, squares into another area, circles into the third area.
- There are also similar exercises with classification by colours and sizes.

Structuring – to point hierarchical order of components of some system.

    For example:

    To place white and black complete sets of chess figures on a board.

Generalization.

    For example:

- A picture of a strawberry is shown in the left part of the screen and there are six pictures of mushrooms, berries and flowers in the right part of the screen.
- It is required to select the pictures with berries.

Boolean Operations

Negation – to form the object that is negation for given object.
    For example:
    To repaint black squares into white and vice versa.

Implication – to find cause-and-effect relation for several events or facts.
    For example:
    Girl watering potted plants - beautiful flowers grown in pots.

Conjunction.
    For example:
    To bring into the indicated area only red squares. (Red and Squares).

Disjunction.
    For example:
    To bring into the indicated area all red figures or squares. (Red or Squares).

Equivalence.
    For example:
    To bring into the indicated area all the red squares and also neither red nor squares (Red & Squares or Not Red & Not Squares).

Complex Operations

Synthesis – to collect unit from parts.
    For example:
    Gather a picture from separate fragments.

Memorizing – it is required to reproduce something shown before.
    For example:
    ● The picture is shown for 10 seconds and then four similar pictures are presented.
    ● It is required to choose the initial picture.

Analysis – a unit is offered, it is required to find out what parts this unit consists of.
    For example:
    ● There are pictures of a marble, a helicopter, a bicycle and a penguin in the left side, each coloured with three pencils of different colours.
    ● There are pictures with triples of colour pencils on the right side.
    ● It is required to pick up the pencils that match the colours on the picture.

Imagination – a part is offered, it is required to imagine the whole.
    For example:
    ● The picture is given with cutted part and few parts.
    ● It is necessary to choose the cut part from these few parts.

Analogy – implementation of some set of actions forming a result "by analogy".
    For example:
    Various IQ-tests-like exercises.

Abstraction.

For example:

- There are some pictures.
- Pupil needs to choose mathematical abstraction for them (point, line, polygon, curve).

Positioning – mark the defined part of picture.

For example:

- There are left and right squares ruled into cells.
- The question character appears in the some cell of the left square.
- It is needed to select the same cell of the right square.

During the elaboration of base intellective operations and the exercises there were also concurrently created convenient applets for visual design of such exercises. One of the main requirements for such tools was minimal level of computer literacy requirement for users. As a result, the exercises can be created not only by teachers and students, but even by pupils of seventh, fifth, third and even second grades.

3. Application Results

The electronic course "Learning to think", containing exercises for the basic intellective operation development, was introduced in September 2008. It was used by 156 scholars from Gomel and Gomel region in the 2008–2009 academic year. The number of users increased to 737 scholars and teachers in the 2009–2010 academic year, including all pupils of 1–3,5 grades of Gomel school 27. As a result, the course was completed by 49 pupils of the 1st grade, 67 pupils of the 2nd grade, 54 pupils of the 3rd grade and 55 pupils of the 5th grade. The minimal, maximal and average time spent on doing the exercises of the course is given below (Table 1).

Comparing minimal, maximal and average times on different grades, we can see essential difference between the first and second grades and almost no difference between the second and next grades. The conclusion is that learning in the first grade significantly increased the thinking skills of most pupils. Another important observation is the strong differentiation in the training of pupils that remain at higher grades: the strong pupils do progress three times faster than the weak ones, both in the first grade and in all subsequent grades.

Table 1

Time spent doing the exercises

	1 grade	2 grade	3 grade	5 grade
Minimal time	10:26	05:32	05:53	05:41
Maximal time	33:02	18:40	18:42	16:54
Average time	17:34	11:16	10:10	09:24
Number of pupils	49	67	54	55

4. Conclusion

The main conclusions are:

- The learning process develops thinking possibilities mostly at first grade.
- Strong differentiation in the progress of pupils, which is discovered in the first grade, doesn't disappear during transition of pupils from grade to grade.
- The results of the best and the worst students increasingly differ as exercises become more complicated.

These conclusions gave birth to strong doubts about the effectiveness of joint teaching of so differently advanced pupils. To solve the problem, author proposes to use the described exercises for computer-based testing of pupils before enrolling the first grade. In case of poor test results, infants with their parents should be recommended to attend to an additional one-year "pre-school" training with described exercises, which may be sufficient to align the skills of mental activity and highly raise chances of subsequent successful learning.

References

Aubrey, C., Ghent, K. Kanira, E. (2012). Enhancing thinking skills in early childhood. *International Journal of Early Years Education*, 20(4), 332–48.

Dolinsky, M. (2013). An approach to teach introductory-level computer programming. *Olympiads in Informatics*, 7, 14–22.

Howard, J., Miles, G., Rees-Davies, L. (2012). Computer use within a play-based early years curriculum. *International Journal of Early Years Education*, 20(2), 175–89.

Lia, H., Wongb, N. (2008). Implementing performance indicators of early learning and teaching. *A Chinese Study International Journal of Early Years Education*, 16(2), 115–31.

M. Dolinsky is a lecturer in Gomel State University "Fr. Skaryna" since 1993. Since 1999 he is leading developer of the educational site of the University dl.gsu.by. Since 1997 he is heading preparation of the scholars in Gomel to participate in programming contests and Olympiad in informatics. He was a deputy leader of the team of Belarus for IOI'2006, IOI'2007, IOI'2008 and IOI'2009. His PhD is devoted to the tools for digital system design. His current research is in teaching Computer Science and Mathematics from early age.

Components and Architectural Design of an Autograder System Family

Jordan FERNANDO, M. M. Inggriani LIEM

Data and Software Engineering Research, School of Electrical Engineering and Informatics
Institut Teknologi Bandung
e-mail: fernandojordan.92@gmail.com, inge@informatika.org

Abstract. A new automatic grading (autograder) system has been set up to support the Indonesian selection and preparation process of IOI candidates in Indonesia. As interest in programming competitions is increasing, we need an autograder system that can be prepared for a specific purpose and is scalable to be able to handle the increasing number of users. Therefore, we have redesigned a system with a new concept and presented it in this paper. The new autograder system consists of interchangeable components to fulfil all kinds of operational purposes. Instead of having a big and complex system, the proposed system will be automatically composed and deployed for specific operational needs. Design, implementation and operation of the autograding and contest management systems are supported by IOI alumni.

Keywords: auto grading, training, programming competition, components.

1. Background and Related Work

The process that Indonesian IOI participants (called "TOKI", abbreviation of Indonesian Computer Olympiad Team) go through is a step-by-step one, starting from the school level, then advancing to regency, province and finally to the national level. After national selection, the candidates are prepared and selected by a team of coaches consisting of Indonesian university faculty members and IOI alumni. Faculty members from five national universities – the University of Indonesia (UI); Bandung Institute of Technology (ITB); the Institute of Agriculture, Bogor (IPB); Gajah Mada University in Yogyakarta (UGM); and the Institut Teknologi Sepuluh Nopember Surabaya (ITS) – contribute as coaches. The preparation and selection process for narrowing down the IOI candidates consists of 4 phases, where 30 candidates must take tests to select sixteen, eight and finally four IOI participants. Whether or not they make it through the national process, they form a body of national training participant alumni and IOI alumni that play significant roles in the process in the following years. Some of them become students at national universities or study overseas.

The process of selecting and training IOI candidates is carried out using the autograder system. The first autograder system in Indonesia was developed at the Univer-

sity of Indonesia by Suryana Setiawan, an Indonesian IOI team leader. This first system was the basis of the newer development at ITB.

In 2008, the IOI Selection Committee identified the need for a better autograder system that could be used by the public. Therefore, Petra Barus, an alumnus of the National Programming Competition, developed the *Toki Learning Center (TLC)* as part of his final project in the Informatics Program at the Bandung Institute of Technology (Novandi, 2009). *TLC* was used for the Open National Olympiad in Informatics, for the online training before the national training and selection process and for local competitions. Its user communication language is Bahasa Indonesia.

In 2010, Petra Barus and Karol Danutama, an IOI alumnus, developed a new version of *TLC*, called *LX*. *LX* is open to the public at `http://www.tokilearning.org`, and it offers a new feature called Training Gate that emphasizes self-exercise. Training Gate provides problem sets that are grouped by solution types and ordered by level of difficulty. Until now *LX* has been used for the overall national training and selection process.

Driven by the need to handle tasks submission and to automate the grading process for large programming classes at ITB, and inspired by *Coursera* (`https://www.coursera.org/`) and *Marmoset* (Spacco *et al.*, 2006), Karol Danutama developed an autograder system that combines automatic grading on *LX* with a Learning Management System (Danutama and Liem, 2013) as part of his final project in Informatics Engineering Study at ITB. The autograder system is called *Oddysseus*, and it provides grading services to many clients such as *Moodle* and *Doppel-Ganger* (Chandra and Liem, 2013). *Doppel-Ganger* is an educational programming tool designed for simple PCs and mobile devices that enables students to run simple programs anywhere. *Doppel* is a dedicated source code editor that provides assessment of the coding process. *Ganger* provides for the compilation and execution processes through *Oddysseus* or a local compiler. The architecture of *Oddysseus* consists of three layers and it is scalable. It has been tested in an Object-Oriented Programming course with 164 students and in an Algorithm & Programming course with 173 students. The autograding process is more adapted to teaching than to competition, and until now has been used in courses in Informatics Engineering at ITB. We have found it very useful in reducing man-hours for grading student assignments as well as motivating the students to do more exercises. However, program execution assessment is not enough for teaching. Therefore the grader should be enriched with various types of source code assessments, which we categorize as white box grading.

The evolution of the autograder systems maintained by ITB is shown in Fig. 1. The usage of the autograder has evolved from competition to teaching programming.

In a programming competition, solutions are graded using black box grading, where the system compares the provided program output with solution output. A pair of input and solution output is called a testcase. Black box grading needs checkers when a problem solution has many possible output. An interactive problem solution needs a special grading process.

There are several ways to grade a source code. Grading types used in competition are subtask, batch, output-only, and interactive. In subtask grading, the testcases are grouped into several subtasks. Each subtask has a score and contains testcases that require certain
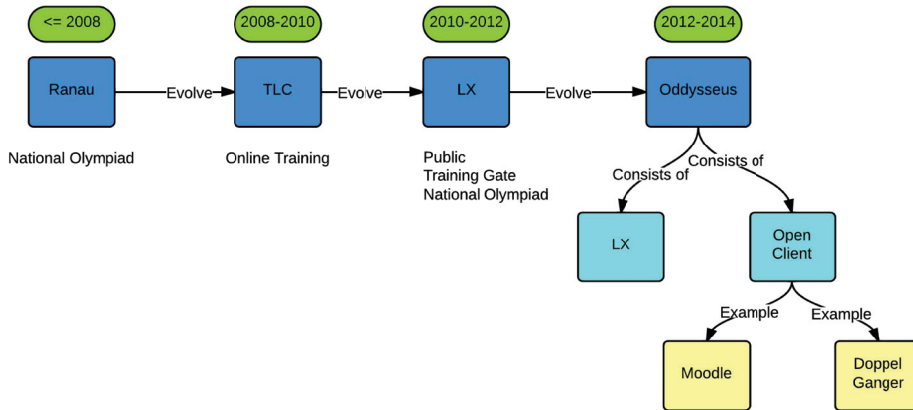
Fig. 1. Evolution of Autograder System maintained by ITB.

algorithm as a solution. In order to gain a score in a subtask, the solution must pass all testcases in the subtask. Testcases of batch grading are not grouped and can be scored partially depending on the correct number of testcases. In output-only grading, a user submits solution in the form of output file(s). In interactive grading, the solution interacts with judge programs to solve the problem.

Currently, *LX* supports subtask, batch, output-only, and Interactive grading types and C, C++, Java, and Pascal programming languages.

Based on experience in using *Oddysseus* for teaching programming, it has been found that program execution assessment is not sufficient. In a formal teaching programming context, the autograder system must be equipped with source code inspection, such as static analysis, bad smell detection, and plagiarism detection. Plagiarism detection is used to compare student source codes with optimal solutions (Kustanto and Liem, 2009).

The autograder system needs problems sets (problem description and testcases) to do the autograding process. The preparation of problems sets takes time to ensure the problems descriptions are clear and the testcases cover all cases. Because *LX* and *Oddysseus* have been used consistently over the years, each system contains many problems-sets. Unfortunately these sets of problems are not well organized and are rarely reused.

With the growing number of users in Indonesia and new problems types in IOI systems, new needs have come up, such as how to set up a competition quickly, how to scout for talent in preparation for the IOI, and how to give support to new IOI grading types as the IOI evolves and is improved. The problem sets need to be pooled in a repository, since existing problem sets are decentralized due to coaching being done at different locations, not only at ITB. Therefore, the IOI Selection Committee requires new autograder systems to meet national IOI preparation and selection objectives. With the benefits of using the autograder system in teaching, the new autograder systems are also designed to support programming courses in the university.

In this research and development, *Oddysseus* serves as the baseline for the development of the new autograder system. The authors have used reverse-engineering tech-

niques to study and improve the design of *Oddysseus* (Pressman, 2010). Other than *Oddysseus* and *LX*, there are other autograder systems such as *Marmoset*, *Open Judge System* (`https://github.com/NikolayIT/OpenJudgeSystem`), and *CMS – Contest Management System* (`https://github.com/cms-dev/cms`). The existing autograder systems only support specific purpose such as competition or learning. The new autograder system is designed to build and deploy many specific purpose autograder systems easily.

2. Problem Statement

The evolution of the autograder system shown in Fig. 1 has produced many versions of the autograder system that exhibit a lack of controls, and the system has been reassembled according to various needs. This situation raises a research question: How can we easily and quickly provide a specific-purpose autograder system and control its versions?
Referring to the needs described in the background, we identify the following specific autograder systems:

    a) Competition system.
    b) Programming learning system.
    c) Programming training system.
    d) Problem set repository.

    New types of systems are potentially needed in the near future.

3. Methodology

The methods used in the research and development of the new autograder system include: the study of related work, reverse engineering, component requirement analysis, design of the proposed system architecture, component implementation, and case implementation.

During the study of related work, the authors scrutinized the evolution of the system and characteristics of specific systems. We learned how to save varying data and added cross-cutting aspects of the system to be used in the new autograder system (Suwandi, Liem, and Akbar, 2014). The authors also adopted the techniques used in continuous integration to be used in the new autograder system (Humble and Farley, 2010).

In the reverse-engineering stage, the author implemented reverse-engineering techniques on *Oddysseus* to uncover autograder system components.

As a result of previous work and reverse engineering, we propose the architecture of the autograder system. Our main focus is on the static aspect (component design) and on dynamic behaviour or runtime systems where components are assembled into one system. The components were implemented and we performed unit tests on each of them. In addition to component testing, integration testing was also done when the components comprised a single system.

4. Component Requirement Analysis

Our analysis is driven from use cases of the new system covering the usage in all specific systems derived from components. The components are arranged in layers adopted from *Oddysseus*. Each layer of the new autograder system contains components derived from the autograder system use cases. The mapping between the autograder system use cases and the components is shown in Table 1.

The results of the analysis and reverse-engineering processes are shown in Fig. 2.

BB consists of black box grading types such as Output Only (BB1), Subtask (BB2), Batch (BB3), and Interactive (BB4). WB consists of white box grading types such as Static Analysis (WB1), Bad Smell Detection (WB2), and Plagiarism Detection (WB3). Other Black Box and White Box grading types are also included in BB or WB.

Components that have been extracted can be used for specific factory systems in a product line; some examples of build script elements using the components are given in Fig. 3.

Table 1

Mapping between autograder system use cases and components

Use cases	Components
Manage contests	Contest Management System (CMS)
Manage problems and test cases	Repository of Problems (RP) Communicator (C)
Manage results	Live Scoreboard (LS), Front-end Result (FR) Communicator (C)
Submit solutions (competition)	Competition Front-end (CF) Communicator (C), Black box grading (BB)
Manage courses and classes	Learning Management System (LMS)
Submit solutions (courses)	Learning Front-end (LF) Communicator (C), Black box grading (BB), White box grading (WB)
Manage training resources	Training Management System (TMS)
Submit solutions (training)	Training Front-end (TF) Communicator (C), Black box grading (BB), White box grading (WB)
Monitor system resources	Monitor (M), Communicator (C)
Manage users	User Management Front-end (UMF)
Autograding request	Any Front-end (F) Communicator (C), Grader (G)

$$BB \rightarrow \{BB1, BB2, BB3, BB4, \ldots\} \qquad \text{[Black Box grading]}$$

$$WB \rightarrow \{WB1, WB2, WB3, WB4, \ldots\} \qquad \text{[White Box grading]}$$

$$LX \rightarrow \{TF, CF, CMS, RP, C, BB, LS, FR, M, G\} \qquad \text{[LX System]}$$

$$Oddysseus \rightarrow \{LF, LMS, RP, C, BB, WB, FR, M, G\} \qquad \text{[Learning System]}$$

Fig. 2. Component extraction from existing autograder systems.

$$\{CF, UMF, CMS, RP, C, BB, LS, FR, M, G\} \rightarrow Competition\ System$$

$$\{TF, UMF, CMS, RP, C, BB, LS, FR, M, G\} \rightarrow Training\ System$$

$$\{LF, UMF, LMS, RP, C, BB, WB, FR, M, G\} \rightarrow Learning\ System$$

$$\{F, UMF, RP, C, BB, WB, FR, M, G\} \rightarrow Problem\ sets\ Repository\ System$$
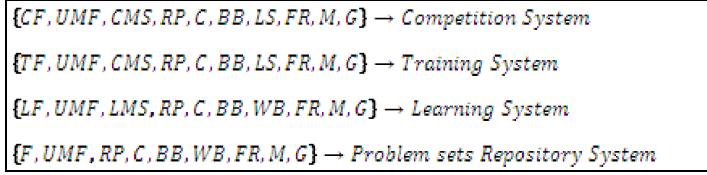
Fig. 3. Component composition into specific purposes for the autograder system.

Fig. 3 shows how to create an autograder system for competition. Build script will be generated to build and test Competition Front-end, User Management Front-end, Contest Management System, Repository of Problem, Communicator, Black box grading, Live Scoreboard, Front-end Result, Monitor, and Grader component. After building and testing components, all components will be integrated as system and integration test will be conducted. A competition autograder system is ready to be used for a competition.

By having components of autograder systems, we can recreate many types of autograder systems and simplify the deployment and testing process.

5. Proposed Solution

Our proposed solution consists of static system components and dynamic runtime environment.

5.1. *System Components*

By using reverse-engineering techniques on *Oddysseus*, we found that *Oddysseus* consists of three layers of components: front end, service, and back end. These layers will be adopted with some improvements. The service layer will be changed into a communicator layer and we will add two layers which are cross-cutting and database layers. The communicator layer provides communication methods from the autograder to all external components and provides job distribution. The cross-cutting layer contains components whose function is to take care of other aspects of the system such as security and monitoring. The database layer contains components that take care of the data management system.

Functional components of the system described in the previous section are summarized in Fig. 4.

The general autograder system layers are shown in Fig. 4:

1. Interface Layer.
   User interacts with the autograder system through this layer, using web pages. The functions provided by the interface depend on the type of system autograder. In addition to the functions that depend on the type of autograder system, there are also common functions for all types of user interfaces such as user management.
2. Communicator Layer.
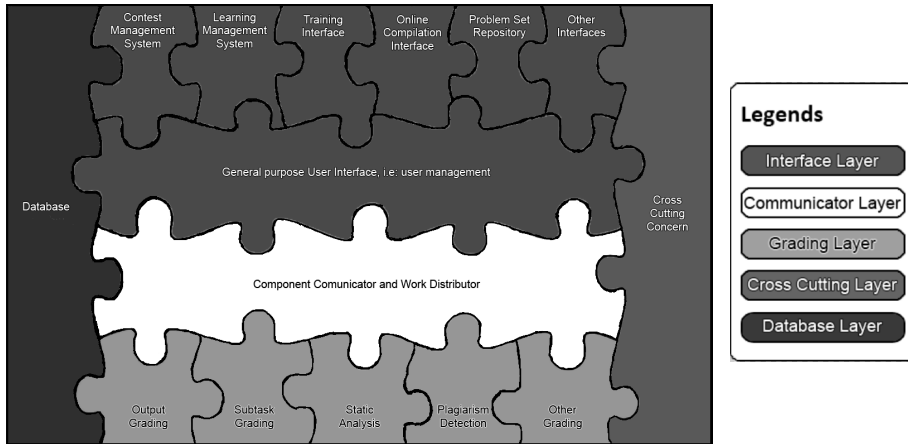
Fig. 4. Layers of multipurpose autograder systems.

In an autograder system, some components need to communicate with each other. Communication can be done through this layer. The main task of the components in this layer is to provide communication between the interface layer and the autograding layer. This layer also provides services to distribute jobs received from the interface layer to the existing components in the grading layer.

3. Grading Layer.

Automatic grading is the main function of the autograder system. This layer provides grading services to other components. Grading services are invoked by sending a request via the communicator layer. Automatic grading is done by performing a process on the files contained on the grade request such as compilation, execution, and source code analysis. In this layer, each component serves a type of grading such as subtask, outputs only, static source code analysis, and plagiarism detection.

4. Cross-cutting Layer.

Security and monitoring are cross-cutting concerns that we consider important. All of these concerns can be implemented outside of the main system through aspect-oriented analysis and design.

5. Database Layer.

The components of the interface layer, communicator layer, and grading layers may need to store specific data to support multiple functions. This function is supported by the database layer components and is implemented as a Database Management System (DBMS). The data model for each use will be designed specifically for that model.

In Fig. 4, components have not been integrated into one specific system. With separation into components, the deployment and testing process will be simplified and automated. Moreover, with the separation of the autograder system into components, component addition or replacement can be done with minimal effort. We have defined components with a higher level of abstraction so that the open-closed principle can be satisfied (Meyer, 1988).

5.2. *Runtime System*

The runtime system consists of two main parts: a front-end subsystem and a grading subsystem. Users of the autograder system interact through the front-end system. The front-end is designed to be implemented as a web-based application, whereas the grading subsystem provides services that are invoked by the front-end subsystem. The web application is being developed based on a data model and a set of available user interface patterns. A new pattern or a specific web page can be integrated as a new link in the web site. Each pattern supports a use case. The patterns are grouped in two categories:

a. User interface patterns that create, read/view, update, and delete (CRUD) data (such as problem set, users, announcements, and events). The data models that are managed through this pattern can be general data models such as users and specific data models such as contests. Data variants are managed by techniques presented in (Suwandi, Liem, and Akbar, 2014). Rules are implemented separately from CRUD.

b. Interface patterns that trigger the autograding process.

The top-level runtime architecture of the autograder system is shown in Fig. 5.

A grading subsystem works when a request of the grading service is invoked. A new grade request is inserted into the job queue through a communicator. A worker pulls the job from the job queue and starts the grading process. After a worker has finished the grading process, the worker sends the grade results back to the communicator. The front-end subsystem then pulls the grade results from the communicator.

The autograding process is the predominant function of the autograder system. In the competition-type autograder system, the grading process only needs to be done using black box with some help from checkers. The autograder system needs to be secure when doing black box grading because the system runs the solution which can be harm-
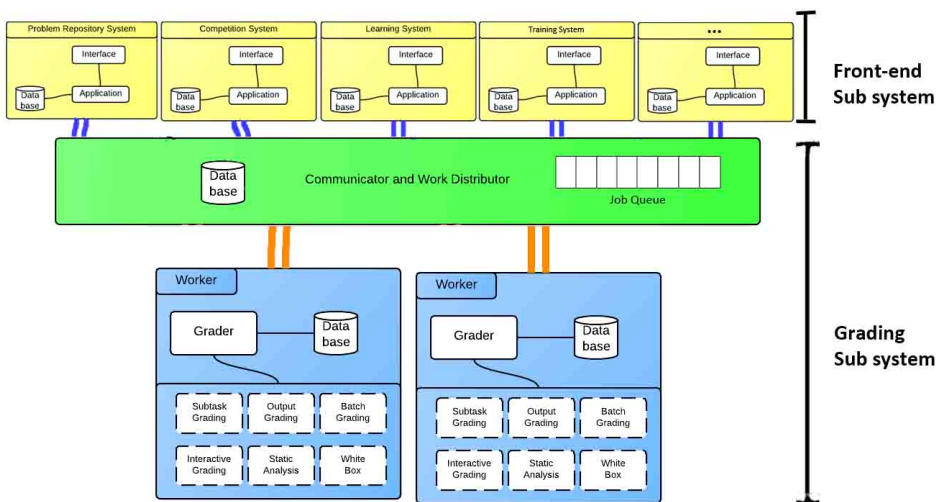
Fig. 5. Runtime architecture of the autograder system.

ful to the system. To ensure security when doing black box grading the solution is run within a sandbox.

A learning-type autograder system requires black box grading and white box grading, such as static analysis, plagiarism detection and bad smell detection. The system can use open source static analyzer tools. Other specific systems that need autograding can have a combination of components and a specific front-end.

The versioning system is set up from the beginning of component development by adopting techniques that exist in the Version Control System (VCS). An overview of the autograder system versioning can be seen in Fig. 6. By implementing these techniques, we have traceable components and also variants of the running system. Each version can be extended into a new branch in which the components can be added or subtracted as needed. Branching is necessary because each system may have a special need that is not the same as for the other systems.

The autograder system must be scalable and robust enough to support many users and a high demand of computing for complex problem solving. The grade requests can be distributed to many worker instances to ensure high performance. Maximum runtime and maximum memory usage of each worker must be set to ensure high performance.

The technology used for the building of the system is *gradle* (`http://www.gradle.org/`), which utilize a Domain-Specific Language to define the building process, and *artifactory* (`http://www.jfrog.com/home/v_artifactory_open-source_overview`) to manage the supply of autograder system components.
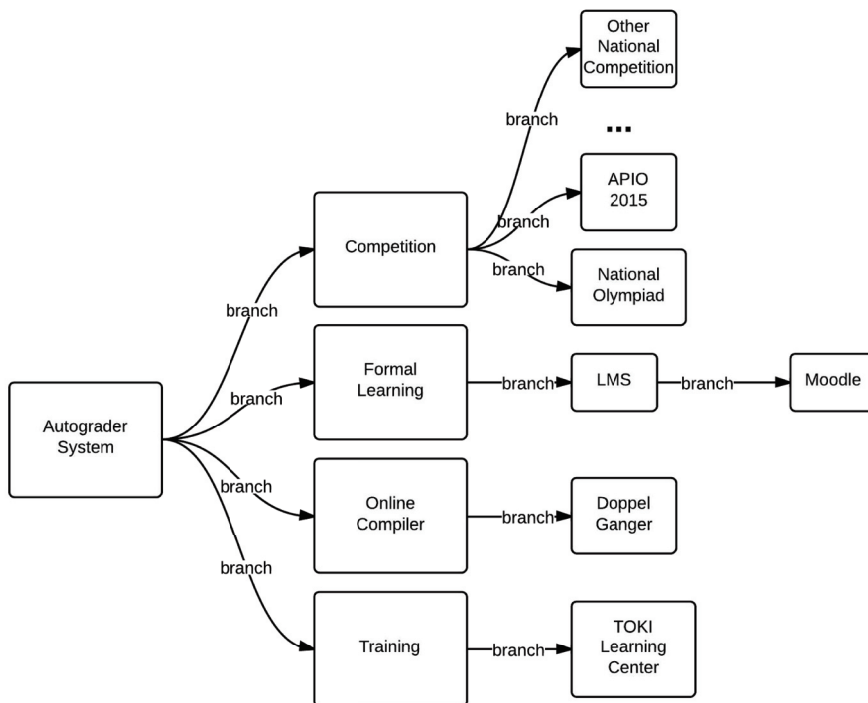
Fig. 6. Autograder system versioning and branches.

6. Case Study

The components will be used to deploy a new competition system, as a part of national preparation in May 2014. In this event, other countries will be invited to participate, since the communication will be in English. The result will be presented in IOI 2014 seminars. The new system is also targeted for use in Asia-Pacific Informatics Olympiad (APIO) 2015, where Indonesia will be the host.

7. Conclusion

The evolution and variation of the autograding system in our national programming competition and formal education is the fruit of IOI alumni and national programming contest alumni contributions. Their experience in competition has led them to be excellent researchers of autograding systems. The results of the work are useful for competition and also for teaching programming and education.

The new component-based system has been proven to make the process of versioning, deployment, testing and synchronization easier. The development of a new grading type or interface could be done with more flexibility to meet particular needs.

The main contribution of this work is a set of components and generic runtime system that can be used to build and deploy a specific-purpose autograder system. The family of specific autograder systems is deployed and tested automatically. The architecture is tested to meet the requirement of scalability, extendibility and adaptabilities.

The new autograder system runs on *Linux* platform. The components are built using Java programming language. The DBMS that is used by the components currently is *MariaDB*. The new autograder system need minimal total RAM of 2GB size, but it is recommended to be run on computer with total RAM of 4GB size. Due to its state as prototype, the source code of the new autograder system has not yet published, but the grading service is opened for public.

Acknowledgements

We would like to thank Mr. Adi Mulyanto and Mr. James Robert Holmboe for their valuable comments.

References

Chandra, T.N., Liem, I. (2013). Source code editing evaluator for learning programming. In: *ICEEI 2013*. 169–175.

Danutama, K., Liem, I. (2013). Scalable autograder and lms integration. In: *ICEEI 2013*. 387–395.

Humble, J., Farley, D. (2010). *Continuous Delivery*. Addison-Wesley, Boston.

Kustanto, C., Liem, I. (2009). Automatics source code plagiarism detector. In: *ACIS/SNPD 2009*. Daigu, Korea, 481–486.

Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice Hall.

Novandi, P.S. (2009). TOKI Learning Center – Sistem Pelatihan Kompetisi Pemrograman Komputer. *ITB Final Project*.

Pressman, R.S. (2010). *Software Engineering a Practitioner's Approach Seventh Edition*. McGraw-Hill, New York.

Spacco, J., Hovemeyer, D., Pugh, W., Fawzi, E., Hollingsworth, J.K., PaduaPerez, N. (2006). Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In: *ITICSE '06*. 13–17.

Suwandi, A., Liem, I., Akbar, S. (2014). Concern-based SaaS application architectural design. In: *ICT-EurAsia 2014*. 228–237.

J. Fernando is a student of Informatics Engineering at Institut Teknologi Bandung. He is the Indonesian technical committee team leader in the national IOI preparation in 2013 and 2014. He is doing his research in the development of autograding components and runtime systems as a part of his final project.

M.M.I. Liem is a member of Data and Software Engineering Research Group in the School of Electrical Engineering and Informatics at Institut Teknologi Bandung (ITB). She has been teaching programming at ITB since 1977. She obtained her doctoral degree in Universite Joseph Fourier in Grenoble, France in 1989, with the teaching of programming as the major topic of her dissertation. Since 2004, she has been involved as a team member in national recruitment, training and IOI preparation for the Indonesian team. She is also the ITB ACM ICPC coach and advisor.

On Inductive Progress in Algorithmic Problem Solving

David GINAT

Tel-Aviv University, Science Education Department
Ramat Aviv, Tel-Aviv, Israel 69978
e-mail: ginat@post.tau.ac.il

Abstract. Induction is known, first and foremost, to mathematics and computer science students as an essential means for proving theorems. But induction is much more than that. Induction is also a core heuristic in the process of problem solving. In algorithmics, a problem solver should seek gradual observations of patterns of the problem at hand, and then capitalize on them in devising an algorithmic solution. In this paper we elaborate on the heuristic of inductive progress during algorithmic problem solving. We demonstrate its essential role with three different examples. Such an elaboration may enhance the awareness of tutors and students to components of the gradual process of problem solving.

Keywords: induction, problem solving, problem representation.

1. Introduction

In his book *How to Solve it* (1954), George Polya says "… Induction tries to find regularity and coherence behind observations … In mathematics and the physical sciences we may use observation and induction to discover general laws ..." (Polya, 1954, p. 117).

The general laws to which Polya refers are assertional, or declarative patterns of phenomena and regularities. The specification of assertional patterns is fundamental in mathematics and science, including computer science. Yet, in computer science there is an additional facet to general laws – the facet of formulating a general, operative, computational scheme.

In computer science the utilization of induction is two-fold:

1. For recognizing and proving assertional patterns.
2. For formulating general, algorithmic schemes, and justifying their correctness.

These two components are essential in the design of algorithms. During the design process, one has to first recognize patterns of the relationships between the input and the output of a given algorithmic task, and then capitalize on these patterns in combining, or composing suitable algorithmic schemes. Pattern recognition is an essential component of problem solving (Schoenfeld, 1992), and the composition of suitable algorithmic schemes is the basic means of algorithmic design (Linn and Clancy, 1992; Astrachan *et al.*, 1998; Wing, 2006).

In this paper, we underline and illustrate the relevance of induction during the process of algorithm design. We display examples of different levels of difficulty, and illuminate different aspects of the utilization of induction – inductive design of a rather simple algorithmic scheme, inductive extension of perspective, and inductive development of a suitable problem representation. We display each of these aspects with a separate example in the following section. In each example, we present a gradual solution process, which progresses in inductive steps.

2. Inductive Progress

We display solution processes of three very different tasks. The first task involves an inductive process of algorithmic design, which starts with gradual recognition of assertional patterns and continues with capitalization on these patterns in the design of a linear algorithm. We developed this task in order to underline inductive progress, both in the design process and in the resulting algorithmic computation. The next two tasks appeared more than four decades ago in mathematics Olympiads. We display inductive solutions to these tasks, in which we elaborate on gradual extension of the perspective underlying the tasks' solutions. The second of these tasks is tied to binary representation, which is essential in algorithmics. Each task is displayed in a separate sub-section, which is titled according to its primary inductive aspect.

2.1. *Inductive Algorithmic Design*

We start with a relatively simple task. The solution process that we present below involves inductive recognition of task characteristics, combined with the gradual development of an algorithmic scheme. The justification of unfolded patterns and the resulting algorithm involves induction as well.

> **Fence Levelling.** A fence of tiles, made of N columns, should be levelled. The total number of bricks in the fence is $N \times h$, where h is the average height of a column. In one operation of brick-moving, one may transfer any number of bricks from one column to an adjacent column. Devise an algorithm whose input is h (the average height of a column), followed by a list of N positive integers, denoting the heights of the columns of the fence; and whose output is the minimal number of brick-moving operations needed for levelling the fence.
>
> For example, for the fence of five columns of heights: 1 4 11 3 6 (where h is 5), four operations of brick-moving are required (5 bricks from the 3rd column to the 2nd column, then 4 bricks from the 2nd to the 1st column, then 1 brick from the 3rd to the 4th column, and 1 brick from the 5th to the 4th column).

One way to approach the task is inductive. We may gradually consider various inputs, of increasing N, starting with N = 2, and continuing with various cases of N = 3, 4, 5, and 6. Notice that there is no need to simulate the brick-moving operations, but only output their amount.

The case of $N = 2$ is trivial, as there may be at most one brick-moving operation. The case of $N = 3$ may require up to two operations. One operation is required if exactly one of the end columns is of height *h*. Two operations are required if both ends are different from h. The characteristics of the various cases of $N = 4$ are similar in nature to those of $N = 3$, just slightly longer. One particular case to notice, of $N = 4$, is the case in which the first two columns may be levelled separately from the last two columns, for example – the case of 6 4 3 7.

We may learn a lot from the various cases of $N = 3$ and $N = 4$. First, it seems that the number of brick-moving operations may not exceed $N - 1$. And, it may be lower, if separate parts of the fence may be levelled independently (as in the case of 6 4 3 7). We may conjecture the following assertional patterns from the latter observations:

> *An N-column fence may not require more than $N - 1$ brick-moving operations.*
> *If the number of bricks in a sub-sequence of K columns is $K \times h$, then these K columns may be levelled independently.*

We may vary further cases, of fences of 5 columns and 6 columns. One case may be: 7 5 3 4 5 6. Another may be: 7 3 5 4 5 6. We may notice that not only may a sub-sequence of K columns, with $K \times h$ bricks, be levelled independently, but also: if this sub-sequence may not be broken into smaller independently-levelled sub-sequences (where the average number of bricks in each is *h*), then the minimum number of operations required to level this sub-sequence is $K - 1$. This observation may be proved by induction on K. The proof also implies our first pattern above (of at most $N - 1$ operations), and yields the following illuminating assertional pattern:

> *Let S be the maximal number of independently-levelled sub-sequences into which the N-column fence may be divided. Then, the minimal number of brick-moving operations required for levelling the fence is $N - S$.*

Up to this point, we gradually unfolded assertional patterns. Now, we may devise a computational scheme. A natural idea that comes to mind is: to first recognize the smallest leftmost sub-sequence that may be levelled independently; then recognize the next such sub-sequence to its right; and so on. This idea calls for a linear scheme, which will progress inductively over the input:

> *Read the input column-by-column and count the maximal number of independently-levelled sub-sequences, by "collecting" these sub-sequences from left to right. The "collection" of each such sub-sequence ends once the average of the "collected" columns of the sub-sequence is exactly h.*

The correctness proof of the latter scheme may be formulated by induction on the number of columns read, using a suitable invariant of the single, rather simple loop of the computation. The intuitive justification is based on the notion that the computation finds the smallest left part that can be levelled, and then continues inductively.

All in all, the process of devising the algorithmic scheme involved: an initial stage of gradually unfolding assertional patterns, while inductively examining diverse inputs; and a second stage of devising an inductive "collection" of independently-levelled sub-sequences. The justification of the recognized patterns and the devised scheme is also based on induction. We leave it to the reader, as our focus here is less on formal proofs and more on inductive unfolding and specification of patterns. A slight modification of the task, into a circular fence, makes the task more challenging, as there is no specific (left or right) end from which the computation may start.

2.2. *Inductive Extension of Perspective*

The focus of the task in this section is inductive development of a suitable perspective, which encapsulates optimization. The task requires optimal placement of elements in a given structures. Computer science involves computations with diverse structures, such as matrices, trees, and graphs. The following task involves placing elements in a 3D matrix.

> **Rooks in a cube**. Given a cube of size N (i.e., $N \times N \times N$ structure of $1 \times 1 \times 1$ unit cubes), place as few rooks as possible in the cube, so that all the N^3 unit cubes will be threatened (each by at least one rook). A rook threatens all the unit cubes that are in the X-axis, Y-axis, and Z-axis of its unit cube (including its own unit cube).
>
> For example, for N = 2, two rooks will suffice – one in the bottom-left unit cube and one in the top-right unit cube. Each rook "covers" exactly 4 separate unit cubes.

In our presentation below, we display on a 2D paper, the rook placements in a 3D structure. In order to do so, we use a 2D square in which we indicate the Z-"level" (height) of each rook with an integer in the range 1..N. In addition, we use the terminology "bottom-left" and "top-right", for both 2D and 3D cases (where actually for 3D, we mean "bottom-left-front" and "top-right-back"). Thus, the 3D solution below, of the task statement example, for N = 2, is displayed with a 2D square as follows:
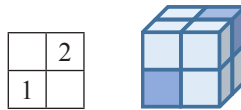
We advance, inductively to the case of N = 3. A natural attempt to extend the placement in the case of N = 2, is by placing rooks on the main diagonal, in different levels. However, this placement does not yield a "cover" of all the 27 unit cubes. The placement is illustrated in the left figure below. The right figure below shows unit cubes in the 3rd level that are not threatened. There are additional unit cubes, in the other levels, that are not threatened.

		3
	2	
1		

x		
	x	

Thus, we need to add rooks, and perhaps also change rook placements. An important characteristic that we may learn from the above placement is the following:

> *If there are K unit cubes in a level, which are not threatened by rooks*
> *of that level, then we need at least K rooks on the other levels, one in*
> *each "pillar" (Z-axis) of these unit cubes.*

Following this observation, we may first seek a placement of rooks in two levels, which will cover as many unit cubes as possible in these levels, and then add rooks in the third level. We may notice that two rooks that are placed diagonally in a level may cover 8 unite cubes in that level, as illustrated in the figure below (using the "+" sign):

+	+	
+	R	+
R	+	+

If we place two rooks in one diagonal of the bottom-left 2×2 square of the 1st level, and two additional rooks in the other diagonal of that 2×2 square, on the 2nd level, then we manage to cover 8 unit cubes in each level, plus the 2×2 bottom-left square of the 3rd level, as shown below:

	.	.
2	1	.
1	2	.

The only unit cubes not covered in the first two levels are the ones in the top-right. The unit cubes that are not yet covered in the 3rd level are those in the upper row and the right column. The covering of all these cubes may be achieved with one rook in the top-right unit cube of the 3rd level:

		3
2	1	
1	2	

So, we managed to cover the $3 \times 3 \times 3$ cube with 5 rooks. Combing the ideas used in the cases of $N = 2$ and $N = 3$, we may advance inductively to a $4 \times 4 \times 4$ cube, and notice that if we extend the top-right cube of $1 \times 1 \times 1$ into a top-right cube of $2 \times 2 \times 2$ (as we

just did with the bottom-left cube, in the transition from the case of N = 2 to the case of N = 3), then we may cover all the 64 unit cubes with 8 rooks, as follows:

		4	3
		3	4
2	1		
1	2		

It seems, from the latter cases that it may be beneficial to divide the view of the N × N × N cube into two sub-cubes of sizes as close as possible – a bottom-left cube and an upper-right one. This view encapsulates a divide-and-conquer perspective. We may do so also in the case of N = 5, using 13 rooks, if we manage to cover the 3 × 3 × 3 sub-cube in the bottom-left part of the following figure:

			5	4
			4	5
R	R	R		
R	R	R		
R	R	R		

Now we cannot use anymore only main diagonals (as in the simple case of 2 × 2 × 2 structures), yet we may still place rooks diagonally in a systematic way, on "corresponding pairs" of diagonals, so that they will cover all the unit cubes in the first three levels, except for those threatened by the rooks at the levels 4 and 5:

			5	4
			4	5
3	2	1		
2	1	3		
1	3	2		

In extending the above inductively to the case of N = 6, we may cover a 6 × 6 × 6 cube with 18 rooks, placed in two 3 × 3 × 3 sub-structures, as follows:

			6	5	4
			5	4	6
			4	6	5
3	2	1			
2	1	3			
1	3	2			

At this stage we may generalize the rook placement, for the case of even N:

> *Place the rooks in two sub-cubes of the original $N \times N \times N$ cube, such that $N^2/4$ rooks will be placed in the bottom-left $(N/2)^2 \times (N/2)^2 \times (N/2)^2$ sub-cub, in diagonals of the sub-cub's levels (as in the figure above), and $N^2/4$ rooks will be placed in the top-right $(N/2)^2 \times (N/2)^2 \times (N/2)^2$ sub-cub, in the same manner.*

The case of an even N requires at least $N^2/2$ rooks. The proof of "minimality" is as follows: Let layer L be the layer with a minimal number of rooks, among the 3N layers of (the N) X-Y planes, (the N) X-Z planes and (the N) Y-Z planes. Let L be an X-Y plane, and let its rooks dominate r rows and c columns, where $r \geq c$. There are at least $(N-r) \times (N-c)$ rooks that dominate the one-unit cubes in L that are not dominated by the rooks in L. If we now change perspective, and look at the N layers of the X-Z planes, we notice that N-r of these layers contain $(N-r) \times (N-c)$ rooks; and in each of the remaining r layers there are at least r rooks (by the choice of L). The minimum of the expression $(N-r) \times (N-c) + r \times r$ is obtained with the value $N/2$ for both r and c. The case of an odd N is similar, and requires at least $(N^2+1)/2$ rooks.

All in all, the inductive solution process involved gradual illuminations, including: the relationship between the number of rooks in a particular layer and the number of additionally required rooks; the different ways of placing rooks diagonally in a square so that they will threaten the whole square; and the construction of two rather sparse cubes of rooks, of sizes $(N/2)^2 \times (N/2)^2 \times (N/2)^2$, each threatening three more, "non-rook" sub-cubes of the same dimension. The inductive process yielded a divide-and-conquer perspective of the whole structure of size $N \times N \times N$, as a "coarse" $2 \times 2 \times 2$ structure.

2.3. *Inductive Extension of a Representation*

The last task that we present is solved by recognizing a suitable representation. The selection of a suitable representation is a key element in problem solving in general, and algorithmic problem solving in particular. It illuminates task characteristics on which an elegant solution may be based.

> **Buckets.** Given three buckets of water, the goal is to empty one of the buckets, by repeated pouring of water between the buckets. At any given time, one may pour water from one bucket to another, in an amount that doubles the water in the bucket into which the water is poured. Thus, the bucket from which water is taken must contain at least as much water as the bucket into which it is poured. Each bucket is very large, and never overflows. Devise an algorithm whose input is three integers – A, B, C, denoting the water amounts in the three buckets; and whose output is the sequence of operations for emptying one of the buckets.
>
> For example, for the initial water amounts: 10 5 3, we may first pour 3 from the 2nd bucket to the 3rd, and obtain the amounts: 10 2 6 in the buckets, then pour 6 from the 1st to the 3rd and obtain 4 2 12, then pour 2 from the 3rd to the 2nd and obtain 4 4 10, and finally pour 4 from the 2nd to the 1st bucket and obtain: 8 0 10.

It may seem at first glance that perhaps there are initial cases for which there is no solution. Apparently this is not case.

An initial examination shows that the last operation is always conducted between two buckets with equal amounts of water. For gaining further insight, we turn to induction again. We may first solve the task for the case of the amount $C = 1$, then for $C = 2$, and then for larger values of C. In our description below, we use the terms A, B, and C, to denote bucket names as well as bucket amounts.

Thus, we first solve the task for the initial amounts: A B 1. We may notice that if we keep pouring water into bucket C (i.e., the 3rd bucket), without pouring water from this bucket, then the amounts of water in this bucket will always be powers of 2. The water amount in the bucket will grow from 1 to 2, to 4, to 8, etc. If we can transform the amount in one of the other two buckets to a power of 2 as well, then we may be able to obtain two buckets with equal amounts of a power of 2.

The notion of powers of 2 is an essential notion of problem representation. We know that each integer may be represented as a sum of powers of 2. Thus, we may pour water from one of the buckets A or B, in quantities which are powers of 2, and leave in that bucket an amount that is a power of 2. For example, let $B = 57$. Then we may represent B as the sum: $57 = 32 + 16 + 8 + 1$. If we pour from B to C first 1, then 8, and then 16, we will be left with 32 in B. So, we may start by pouring 1 from B to C, bringing the amount in C to 2. In order to pour 8 from B to C, we need to have 8 in C. That is, we need to pour 2, and then 4 into C, but not from B. At this point, bucket A will help us. We will pour 2, then 4, from A to C, and then continue to pour 8 and 16 from B. Both B and C will reach 32, and we will be able to empty B (or C).

Notice that we used A as a "complementing" source of water, whenever we needed to increase C with amounts that will not be taken from B. This will always be possible if A is not smaller than B initially. In addition, the representation, or perspective, of powers of 2 corresponds to binary representation. We may summarize the above scheme as follows:

> *For the case of A B 1, where $B \leq A$, we may empty B, by: pouring into C (which starts with 1) powers of 2 amounts from B, which correspond to the 1-bits in the binary representation of B, interleaved with pouring into C powers of 2 amounts from A, which correspond to the 0-bits of the binary representation of B.*

Progressing inductively, we may now examine the case in which $C = 2$ initially. If B is even initially, then the above scheme will lead to an empty B. But, if B is initially odd, we will be left with 1 in B in the end, as the beginning of the process of pouring into C starts by pouring of 2. However, if we are left in the end of this process with 1 in B, we may apply the scheme again, this time with B in the role of the smallest bucket.

We may now proceed to the case of A B 3. Following the idea underlying the case of A B 2, we may notice that if we keep on pouring water into C then it will keep growing in amounts that are powers of 2 multiplied by 3. That is, C's value will progress from 3×2^0 to 3×2^1, to 3×2^2, to 3×2^3, and so on. Thus, we may now look at the representation of B

as: $3 \times (a\ sum\ of\ powers\ of\ 2) + remainder$, and apply the A B 1 scheme on A B 3. As with the case of C = 2, here too, we may be left with some remainder in B. But, this remainder may only be 0, 1, or 2. If it is 0, then we are done; if it is 1 or 2 then we will apply the scheme again, this time with B in the role of the (new) smallest bucket.

For example, let the initial state be: 20 17 3. The amount in B may be represented as: $17 = 3 \times (2^0 + 2^2) + 2$. Thus, we may pour 3×2^0 from B into C, then 3×2^1 from A into C, and finally 3×2^2 from B into C. This process will result with the remainder 2 left in B, which now becomes the smallest bucket. We may apply the scheme again, this time with a smaller C than in the previous iteration.

Following the analysis of A B 3 we may extend the utilization of the binary representation described above, and formulate the following scheme:

> *For the case of A B C, where $A > B > C$, represent B as: $C \times (a\ sum$*
> *of powers of 2) + remainder. Pour water from A and B in accordance*
> *with B's above representation, and the powers-of-2 strategy (above)*
> *of the case of A B 1. If in the end of this process, the remainder left*
> *in B is not 0, then solve the task again, this time with B as the new C.*
> *Repeat this computational scheme until the remainder left in B is 0.*

All in all, the inductive process led us to an initial idea of capitalizing on integer representation as a sum of powers of 2, which was later extended to a representation of: an integer multiplied by a sum of powers of 2, plus a remainder. The suitable problem representations were the underlying key for the solution. Each water-pouring iteration of leaving a remainder in B is bounded by $\log(N)$ pouring operations, where N is the largest among A, B, and C; and there may less than N iterations, as the remainder left in B at the end of each iteration is always smaller than the remainder left in B of the previous iteration. Thus, the total number of pouring operations is bounded by $N \log(N)$.

3. Discussion

The notion of induction goes much beyond proofs and correctness argumentation. Its essential nature is related to the process of seeking a solution, and discovering general laws (Polya, 1945; Holland *et al.*, 1986). Careful application of inductive search, by examining simple cases upon looking for hidden patterns, may be a key element in successful problem solving. Careful application of inductive design, upon devising an algorithmic scheme, may serve as a constructive means in algorithm design. Our objective in this paper was to underline and elaborate on these latter two elements.

The solution process of the first task in the previous section illustrated and underlined careful, gradual progress, which initially focused on the recognition of underlying patterns. Illuminated patterns then served as underlying characteristics in the design of a linear, greedy computation scheme. In a sense, this design process demonstrated Dijkstra's perspective of combining assertional and operational elements "hand-by-hand" in the design of algorithms (Dijkstra *et al.*, 1989).

The solution process of the second task unfolded gradual observations of the problem at hand. An initial picture was extended gradually, and involved accumulated notions, derived from extending the size of the problem. The final outcome yielded an elegant divide-and-conquer scheme.

The solution process of the third task involved inductive, flexible extension of an initial idea. The initial idea involved binary representation, or the representation of an integer as the sum of powers of 2. Yet, binary representation alone was insufficient for solving the general task. Inductive progression was applied, for more general cases than the basic one. A subtler solution scheme was developed, which combined binary representation with a multiple by an integer, plus a remainder.

The computation schemes reached during the three inductive processes – of greedy computation, divide-and-conquer, and extended binary representation – are essential in algorithm design (Cormen *et al.*, 1990). In examining and teaching our students, one of our objectives was to enhance their awareness of such outcomes and develop their transfer competence in "inductive" problem solving (Mayer and Wittrock, 1996).

We posed the three tasks at different stages of our national Olympiad activity. The first task was posed in one of our early national competitions, as the easier among four tasks. The second task was posed in a later stage, in order to examine students' observations and illuminations. The third was posed in an even later stage, to the better students. It is a challenging task if posed as is. It is much easier if presented in stages, according to the inductive process described above.

In our experience, students demonstrate different levels of competence with these tasks. Some offer erroneous solutions. Others offer partial solutions to some of the cases. And some solve the tasks, but with limited insight and without being able to elaborate on their observations. They phrase a solution, which they reached with relevant associations, but their "picture" of the task characteristic is vague.

A primary objective in examining our students and teaching them, during the Olympiad competitions and training, is to improve their problem solving competence, and strengthen their computational thinking perspective (Wing, 2006). An ordered inductive progress like the ones presented here may assist is attaining this objective. It illustrates, in an apprenticeship manner, an essential approach that may increase students' awareness of the process of problem solving, and enhance the link between assertional patterns, problem representation, and algorithmic schemes.

References

Astrachan, O., Berry, G., Cox, L., Mitchener, G. (1998). Design patterns: an essential component of CS curricula. In: *Proc of the 29-th SIGCSE Technical Symposium on CS Education*. ACM, 153–160.

Clancy M.J., Linn M.C. (1992). The case for case studies of programming problems. *Communications of the ACM*, 35(3), 121–132.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. (1990). *Introduction to Algorithms*. MIT Press.

Dijkstra, E.W. *et al.* (1989). A debate on teaching computing science. *Communications of the ACM*, 32(12), 1397–1414.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, R.P. (1986). *Induction*. MIT Press.

Mayer, R.E., Wittrock, M.C. (1996). Problem solving transfer. In: Berliner, D.C., Calfee, R.C. (Eds.), *Handbook of Educational Psychology*. 47–62.

Polya, G. (1954). *How to Solve it*. Princeton University Press.

Schoenfeld, A. H. (1992). Learning to think mathematically: problem solving, metacognition, and sense making in mathematics. In: Grouws D.A. (Ed.), *Handbook of Research on Mathematics Teaching and Learning*. 334–370.

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.

D. Ginat – heads the Israel IOI project since 1997. He is the head of the Computer Science Group in the Science Education Department at Tel-Aviv University. His PhD is in the Computer Science domains of distributed algorithms and amortized analysis. His current research is in Computer Science and Mathematics Education, focusing on cognitive aspects of algorithmic thinking and learning from mistakes.

Utilizing an Exercise-Based Learning Tool Effectively in Computer Science Courses

Erkki KAILA, Teemu RAJALA, Mikko-Jussi LAAKSO,
Rolf LINDÉN, Einari KURVINEN, Tapio SALAKOSKI
University of Turku
e-mail: {ertaka, temira, milaak, rolind, emakur, sala}@utu.fi

Abstract. Educational technology and learning environments are becoming more and more common in all levels of education. Still, the main focus in research seems to be on which tools to use rather than how to effectively use them. In this paper, we first discuss the aspects that should be considered when adapting an exercise-based learning environment into curriculum. Based on our earlier research on the topic, we present three rules for adapting the tools. Next, a six-year study on using a learning environment in two courses is presented. Throughout the six course instances, the adaptation and integration of the tool is gradually altered. The results seem to confirm the positive effect of changes made in adaptation. When the three rules presented earlier are revisited in correlation with the results obtained, we can state that following the rules of adaptation lead to better student performance.

Keywords: learning environments, automatic assessment, course design, tool adaptation.

1. Introduction

According to multinational study by McCracken *et al.* (2001), programming is one the most difficult skills to acquire. There are various educational tools developed to aid the process, but comprehensive research about their pedagogical usage is still quite rare. Any tool or system, no matter how proficient, can only produce real educational value if adapted and utilized properly. In this article, we consider various factors that may have an effect on the efficiency of the tool usage: tool introduction, student engagement, motivation and reward. Based on our earlier research, most of these factors have a considerable effect on learning results. Hence, it seems that in addition to considering which tool to use, it's equally important to consider how to use it.

In this article, we present three rules for adapting a learning environment, based on our earlier experiments. Though named rules, they are actually ideas to consider when designing course structure and educational technology adaptation. We also present a comprehensive 6-year study, where a learning environment was used in two courses throughout six years. In latter instances, new exercise types were introduced to try to improve motivation. Some other changes in the tool and the usage were also introduced throughout the years. The holistic idea has been to gradually improve the tool adapta-

tion to find out how the learning effects and motivation can be maximized. The student performance and grades are presented to find out whether the changes had effect in particular years.

2. Literature Review

There are various learning environments developed over the years. Crescenzi and Nocentini (2007) present a two year experiment of adapting an algorithm visualization tool into a programming course. The student feedback was mainly positive, but they don't report any changes in student performance. Laakso *et al.* (2005) adapted an algorithm visualization tool called TRAKLA 2 into two courses at separate universities at Finland. They found out that the pass rate increased significantly, and the student feedback was mainly positive. Still, the same group (Laakso et al, 2009) found out later that using the same tool in collaboration with another student has an even higher positive effect on learning. Hence, anyone adapting a tool should be encouraged to find out further whether the positive effects can be enforced.

Educational technology can of course be used in all kinds of courses. De Lange *et al.* (2002) surveyed students' opinions on adaptation of WebCT on accounting course, and found out that their satisfaction with environment is tightly associated with lecture notes, forum, on-line assessment and other tools in that environment. Paechter *et al.* (2010) suggest that the key factor in affecting students' motivation is making the learning objects transparent and providing possibility for self-assessment. Self-assessment should hence have a major role in any exercise-based learning environment. Students' attitudes should have a considerable effect on adaptation: Saunders and Klemming (2003) reported a two-year experiment where they integrated technology into traditional learning environment, and found out that though the students found the module harder to complete than others, their performance was actually better. The cognitive load for adapting new tools (see for example Chandler and Sweller, 1996) is an issue that should be considered when designing technology enhanced curriculums.

Liaw *et al.* (2007) also surveyed the attitudes of students and educators towards e-learning, and found out that the instructors' attitudes are highly positive. The analysis on students' attitudes revealed, that an effective learning environment is influenced by learner autonomy and teacher help, among other things. Hence, it is important to remember that educational technology is not something that can be added into curriculum and then forgotten. Lockyer and Patterson (2008) in fact state, that "the lecturers may have to play a considerable technical support role in helping students who are new to such technologies".

There are other studies that emphasize the instructors' satisfaction in educational technology. For example, Zuvic-Butorac *et al.* (2010) present a huge effort of implementing an e-learning environment of more than 400 courses and 15,000 students in Croatia. The teachers' attitudes were surveyed and found out to be highly positive towards the technology. Still, as O'Neill *et al.* (2004) state in their literature review about eLearning implementation, if new technology is to be integrated into learning properly, comprehensive training and support for instructors should be provided.

3. How to Adapt an Exercise Based Learning Environment

3.1. *Selecting a Suitable Environment*

The first step in adaptation is selecting a proper environment. There are various issues that should be considered when selecting the tool. First – and probably the most important issue – is that the selected tool should provide adequate benefits for both the teacher and the student. As discussed earlier, the most obvious benefit for teacher is the time saved in assessing exercises and assignments. However, to gain any real benefit in time saving, the environment either needs to come with an existing set of usable exercises or the time cost of preparing the exercises needs to be tolerable. As stated in Naps *et al.* (2001), the lack of time is the most important reason for teachers not using visualization tools; the same can probably be applied to any learning environment.

From the student's point of view, the most obvious benefits come from automatic assessment and immediate feedback. The ability to do the exercises any place and any time, and still get supportive feedback, is something that is hardly possible with traditional methods. Improved learning results (Kaila *et al.*, 2009A) are also a significant benefit both for the student and the teacher. Evaluating the learning effects outside controlled studies might be difficult as there are various factors influencing the learning outcome. Still, as shown in Kaila *et al.* (2010), and furthermore in the later sections of this paper, it is possible to significantly improve the results in CS course if a learning environment is introduced and used properly.

There are also some technical issues that need to be considered when selecting an environment. First, there is the initial cost of tool utilization and management. Though most of the common learning environments can be adapted free-of-charge, there might be hidden costs, such as upgrading server equipment and training the users. If the tool is hosted externally, these costs can however be kept in minimum. Moreover, the technical requirements for using the tool should be evaluated beforehand. Some exercises may need plugins – such as Java or Flash – installed into browser before working properly. In some physical environments installing additional components may be difficult or impossible.

3.2. *Three Rules for Adaptation*

In this section, we present three rules that should be taken into account when adapting a learning environment into a course. The rules are based on our earlier results on the topic, and are revisited when the results of this research are discussed.

3.2.1. *Rule 1: Introduce and Integrate*
The first rule is that the tool should be properly introduced and integrated into course. We have previously studied the effects of cognitive load on students when using a visualization tool (Laakso *et al.*, 2008). In the study, the students who went through a comprehensive tutorial about using the tool statistically significantly outperformed the control group. Hence, we suggest, that a separate introductory session should be arranged before the tool is adapted into actual learning. The introduction should be made from two points

of view: technical and pedagogical. The technical introduction contains issues such as logging in and user interface. The pedagogical introduction should address issues such as the order and schedule of the exercises taken, using additional materials to assist learning, and the role of exercises as a part of comprehensive learning experience.

This leads as to the second part of this rule: we suggest that the learning environment should be properly integrated into the course. This means that the exercises in the environment should substitute and supplement the existing materials, where relevant. In practice, this may mean that the course needs to be partially redesigned. In Laakso *et al.* (2014) we presented a programming course reform, where half of the lectures were replaced with interactive tutorials that emphasized active and collaborative learning. The results were remarkable, as the dropout rate decreased and the grades improved statistically significantly after the change. Moreover, the students seemed to find the active approach more motivating and enjoyable.

3.2.2. *Rule 2: Engage the Students*

Naps *et al.* (2002) presented a hypothesis of engagement taxonomy, where they divided the usage of visualization tool into passive (no-viewing and viewing) and active (responding, changing, modifying and presenting). They suggested that using a visualization tool may only produce considerable learning if the tool is used in active levels. We later confirmed the hypothesis in Kaila *et al.* (2009B). We suggest that the results gained from using a visualization tool can be generalized to any types of exercises: if the students are engaged into active learning process, the results are better. Moreover, collaboration can be used to deepen the level of engagement. In Rajala *et al.* (2009) we found out, that if exercises are done in collaboration with another student, the learning results can be significantly improved. In Laakso *et al.* (2014) we presented a programming course reform (see previous Section), where collaboration was brought to classroom exercise sessions by introducing a collaborative mode in learning platform.

3.2.3. *Rule 3: Make it Mandatory, but Reward the Students*

As a third rule, we suggest that the usage of the tool should be made mandatory, but the students should still be rewarded from doing the exercises in the environment. A typical approach is to set minimum limits that need to be reached, and reward the students from exceeding that limit. The reward can be divided into two categories: an internal reward is something gained within the tool. Typically points are awarded when a student successfully completes an exercise or assignment. An external reward is something the students gain outside the learning environment. For example, the students may be awarded with grade improvement, bonus points for exam, or other forms of compensation from completing the exercises in the environment.

In Laakso *et al.* (2014) we present a case where students were required to complete at least five out of seven tutorials during the programming course. However, no minimum score limit was set. The students however completed a remarkable amount, 91% of all points on average, though reaching this amount meant doing extra work outside tutorial sessions. The students could pass the course without a final exam by completing at least 90% of all points awarded from all course components (including lectures, tutorials and course assignments). Still, of all students that reached the 90% level, only a handful skipped the final exam.

4. ViLLE

4.1. *Background*

ViLLE is a learning environment, developed at the University of Turku, Finland. It started out as a program visualization tool in 2004, and later expanded into comprehensive collaborative exercise and course management environment. From the beginning, ViLLE has been developed based on the research done. All major features have been tested with controlled experiments, and only the useful ones have been included in the published version. For example, the engagement taxonomy hypothesis (Naps *et al.*, 2002) lead into developing interactive questions into then-passive visualization tool, and the good experiences on collaborative use (Rajala *et al.*, 2009) encouraged us to develop collaborative mode that enabled two or more students working at the tasks together.

Since the earlier version was used in the first course presented in this paper, both versions are introduced separately.

4.2. *The Early Version of ViLLE – The Visual Learning Tool*

The first version of ViLLE is a program visualization tool (see Fig. 1) that can be used to display the execution of programs one row at a time. The execution is visualized with various components: the current and previous rows are highlighted, the variable states are displayed in their own area, and each subprogram with its local variables is displayed
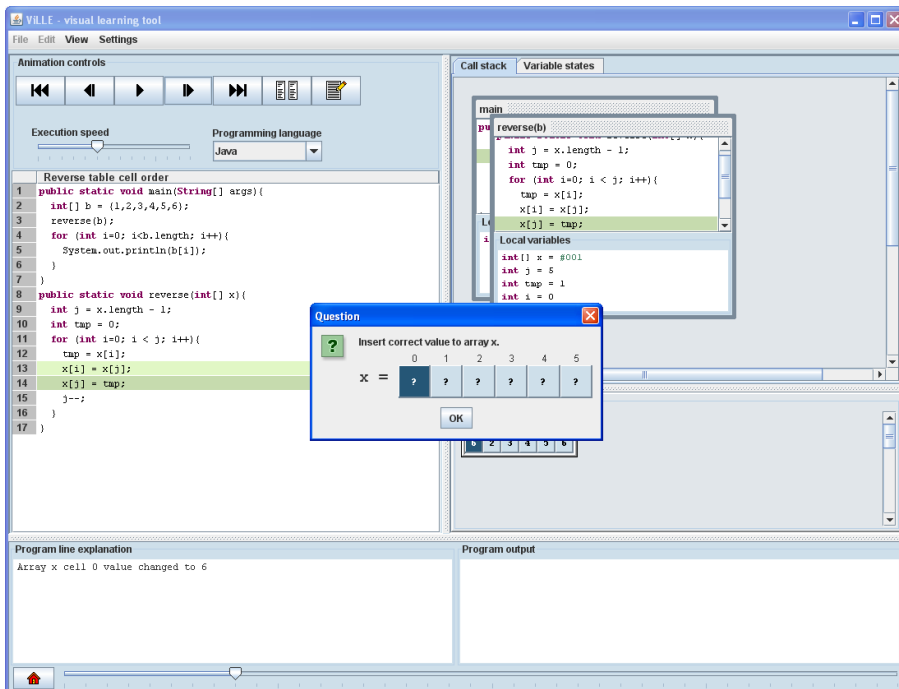
Fig. 1. ViLLE version 1: the student view displaying visualization exercise.

in a single frame in call stack and so on. Moreover, ViLLE displays a verbal explanation about the currently executed line. The tool supports a variety of imperative programming languages – including for example Java, Python and C++ – and automatically translates the programs written in Java to other supported languages. Students can view the execution in parallel view, which displays the executed program in two selectable languages at the same time.

To enhance active learning, the example programs can be accompanied with multiple choice questions or graphical array questions. The questions are inserted into desired steps in program. When a question is encountered the program execution halts until student gives an answer. ViLLE version 1 was deployed as a Java applet or Java application, but it could be connected to a TRAKLA II server (Malmi *et al.* 2004). In this case the server tracks student logins and all achieved points in different exercises. A complete description of the tool can be found in Rajala *et al.* (2007) and in Kaila *et al.* (2009A).

4.3. *ViLLE Now – a Collaborative Learning Environment*

As of 2009, ViLLE was expanded into an exercise-based collaborative learning environment. New client-server architecture was designed, with a focus on teachers' collaboration and with a support for various exercise types. In ViLLE version 2 (see Fig. 2), the teachers can use the built-in editors to create and edit virtual courses and assignments. Moreover, all content set as public can be browsed, utilized and modified by all other teachers registered in ViLLE. New exercise types for various topics were created. For programming, coding exercises, code sorting exercises and simulation exercises were designed among many others. Moreover, exercise types for mathematics, language
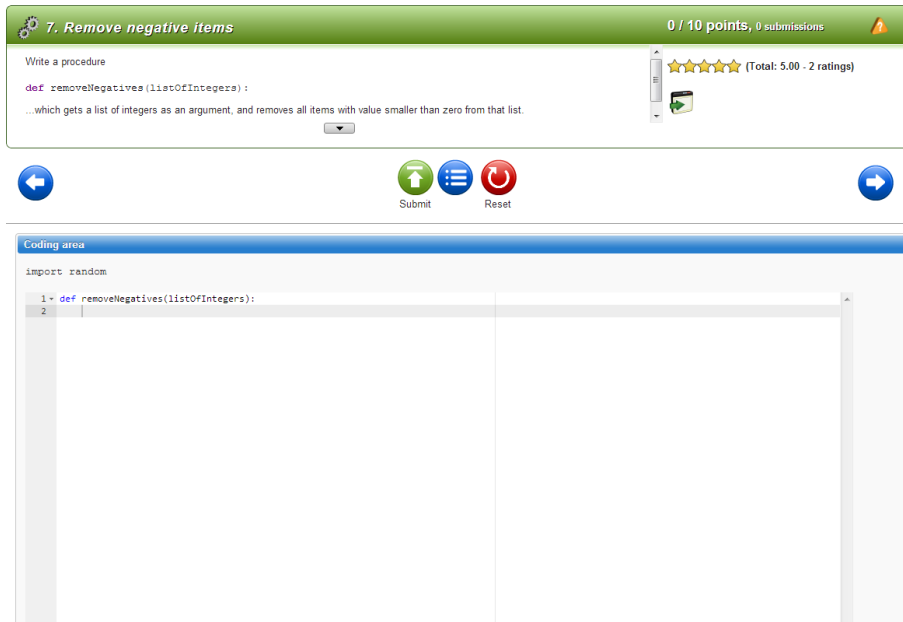
Fig. 2. Coding exercise in current version of ViLLE.

teaching and various other topics have been developed over the past few years. A comprehensive list about ViLLE exercise types can be found in Appendix A.

Since the new version introduced a dedicated ViLLE server, there was no more need to utilize the TRAKLA II server. ViLLE automatically collects a vast amount of data on student performance, including for example all achieved scores and time used to complete the exercises. Additional exercise specific data is also collected: for example, in visualization exercises ViLLE automatically saves all control usage data, including time stamps, when the student does the exercise. All data gathered can be viewed in ViLLE's statistical view by course's teachers.

The new version also supports collaborative learning where more than one student can join the same session. Besides exercises, there are various other tasks that can be used in courses: if accompanied with RFID readers, ViLLE can be used to easily record course attendances and demonstrations. It also supports study journals and course assignments, to name a few. All exercises, whether they are automatically assessed or not, can be used in electronic exams. It also has an editor for building tutorials that combine exercises with other materials, and a research project management system for research collaboration.

The complete description of the tool can be found in Laakso *et al.* (2014)*.*

5. Methodology

5.1. *Overview*

The research was carried between years 2007 and 2012. The data was collected from two separate courses: in the first course – observed in three instances between 2007 and 2009 – the version 1 of ViLLE was used, while in the second course – with three instances between 2010 and 2012 – the newer version 2 was utilized. The usage of tool varied in different instances of the course: the tool was adapted more thoroughly year by year. A gradual increase in the usage was justified by excellent results and feedback gathered from teachers and students.

5.2. *Course Instances*

The first course observed (from now on **Course 1**) was called an Introduction to Information Technology. The goal of the course is to teach computer science fundamentals as well as introductory programming concepts to CS majors at University of Turku. The course is somewhat typical introductory course in computer science, containing basic principles of algorithms and data structures, accompanied with programming fundamentals in Python. Three instances of the course were researched: in 2007, ViLLE was introduced to the course. The usage of the tool was not mandatory; instead, a link to exercises was provided in course web page. In two consecutive instances, 2008 and 2009, ViLLE was made a mandatory part of the course: if the students did not complete at least 40% of all ViLLE exercises, they failed the course. All course instances were taught by the same teacher, and no other significant chances between instances were made.

The second course observed (from now on **Course 2**) is called an Introduction to Programming. It is a mandatory course in Bioinformatics program at University of Turku, and aims at teaching basic programming concepts in Python. The course hence contains essential topics in imperative programming, such as variables, loops and functions, as well as some Python specific topics, but does not include object oriented programming. Three instances of Course 2 were also observed: at 2010 ViLLE was included – as mandatory component, but with visualization exercises only. In two latter instances various other exercise types were introduced as well. In the latest instance (2012) ViLLE was also used to keep track on lecture attendances and demonstration scores, with bonus awarded for good performance on these components. Moreover, in the last instance, ViLLE was also used as a platform for course final exam. As was the case with Course 1, all instances were taught by the same teacher, and no other substantial chances were made in course through these instances.

The usage of ViLLE throughout the course instances is displayed at Table 1**.**

5.3. *Exercises*

In Course 1, ViLLE was first introduced as an optional supplement. At two later instances the usage of the tool was made mandatory. A total of 60 exercises were divided into seven categories: variables and conditions, strings, loops, sub programs, arrays, recursion and sorting algorithms. Each exercise consisted of visualized program code and 5 to 10 questions. Each exercise was scored in scale of 0 to 10 based on the correctness of answers. All exercise rounds were open from the beginning of the course, and references to suitable exercises were made on other course materials.

In Course 2, ViLLE was mandatory in all three instances. At the first instance only visualization exercises were used – the exercise collection was roughly equivalent to the collection used in Course 1 with minor modifications. At the two latter instances other exercise types were introduced. The course was hence divided into eight exercise rounds, based on the topics in course: the first round was an introduction to ViLLE, and the latter rounds about variables and data types, strings, selection, loops, functions, lists and tuples, followed with a round of additional exercises. Each round consisted of five different types of exercises:

Table 1

Usage of ViLLE at course instances

Year	Course	ViLLE exercises	Mandatory
2007	Course 1	Visualization	No
2008	Course 1	Visualization	Yes
2009	Course 1	Visualization	Yes
2010	Course 2	Visualization	Yes
2011	Course 2	Various	Yes
2012	Course 2	Various, including other performance and course exam	Yes

- *Visualization exercises*: these were similar to exercises used in Course 1 and in the first instance of Course 2. A handful of existing visualization exercises were selected, of which some were slightly modified to suit the topics better.
- *Code sorting exercises*: exercises where code lines were shuffled into random order, and the student needed to sort them into correct order according to given task.
- *Puzzle exercises*: an exercise, where the student needs to combine for example variable types and value ranges or string operations and results.
- *Coding exercises*: an exercise where the student needs to write a program – or a missing part of the program – in Python to fulfil given task. The program written in ViLLE can be instantly translated and executed.
- *Quizzes*: ten multiple choice and open questions about the topic at hand.

In Course 2 the exercises were integrated into course curriculum more tightly. Each round of exercises was opened after the lecture about corresponding topic was given. The exercises were designed to cover all aspects of the topic at hand as thoroughly as possible. After opening, all rounds were open until the final exam.

5.4. *Method*

Since the research contains two different courses, only instances of the same course are compared. From each course instance, final grades were obtained. The experiment is a between-subject design with final exam results a dependent variable. ViLLE usage was the only significant between-subject factor (independent variable), since no other significant changes in courses during the observed period were made: the instances were taught by the same teacher, and there were no substantial changes in other course components or materials. Since Course 2 used the most recent version of ViLLE, we also had access to comprehensive exercise data on those instances; hence, statistics about ViLLE usage in Course 2 are also discussed.

6. Results

6.1. *Course 1*

All instances of Course 1 were graded on scale of one to five, five being the best. If the student did not pass the course, no grade was given. The final grade distribution in all course instances is displayed in Table 2 and visualized in Fig. 3**.**

As seen on Table 2, the pass rate and the average grade improved at latter instances, when the exercises were made mandatory. The grade distribution is visualized in Fig. 3.

As seen on Fig. 3, the amount of lesser grades (1, 2 and 3) is clearly smaller at the latter instances of the course, compared to first year when ViLLE exercises where optional. To confirm this, a chi test between course grade distributions was used to calculate the independence between all instances, using the formula

$$x^2 = \sum_{i=1}^{6} \sum_{j=1}^{2} \frac{(C1_{ij} - C2_{ij})^2}{C2_{ij}}$$

where $C1$ and $C2$ are the course instances compared. The results are displayed at Table 3.

As seen on Table 3, the grade distribution at first instance is independent, while latter instances seem to follow the same pattern more tightly.

Table 2

Grade distribution in Course 1 instances

	2007 (N=131)	2008 (N=134)	2009 (N=181)
5	34	40	46
4	17	19	27
3	9	20	33
2	21	16	25
1	25	15	23
Fail	25	24	27
Total passed	106	110	154
% of all passed	80.92 %	82.09 %	85.08 %
Grade mean	3.13	3.48	3.31
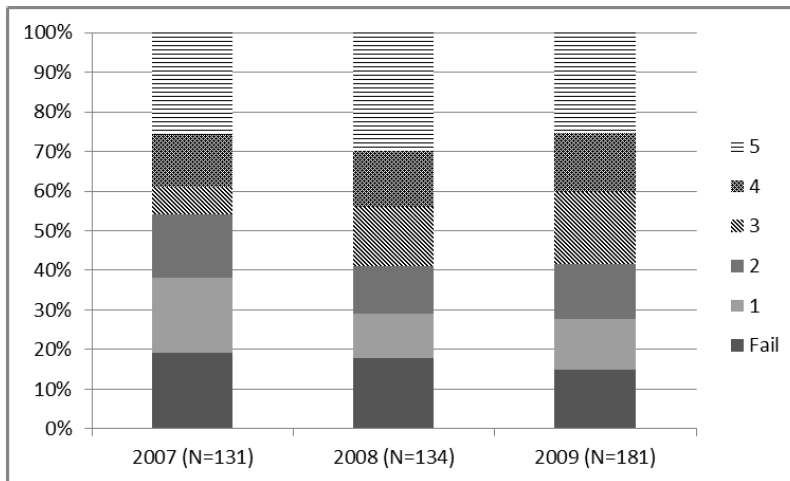
Fig. 3. Grade distribution at Course 1 visualized.

Table 3

Independence between grade distributions of Course 1 instances

Courses	2007 and 2008	2007 and 2009	2008 and 2009
x^2	0.002	<0.0001	0.022

6.2. *Course 2*

It is to be noted, that the number of students in all instances of Course 2 was rather small. Still, certain trends can be observed. Course 2 was also graded in standard scale of 1 to 5. The final grade distribution of all instances is displayed in Table 4.

The percent of students who passed the course has been extremely high in all instances. However, it seems that there is a trend to be seen on the average grades: the average is higher on the latter instances of the course, where varied types of ViLLE exercises were used. The grade distribution is visualized at Figure 4.

Since all of the instances did use ViLLE exercises, and in all instances the usage was required, course grade averages were also compared to earlier instances (<2010) of the course (see Table 5). However, since the teacher was different, and there were other minor changes in the course at 2010 as well, the data should be observed with caution.

Points gathered from ViLLE exercises in all instances of Course 2 are displayed at Table 6.

Table 4

Grade distribution in instances of Course 2

	2010 (N=23)	2011 (N=16)	2012 (N=25)
5	10	10	16
4	3	1	2
3	4	1	3
2	2	1	1
1	3	1	2
Fail	1	2	1
% of all passed	95.65 %	87.50 %	96 %
Grade mean	3.52	3.75	4.04
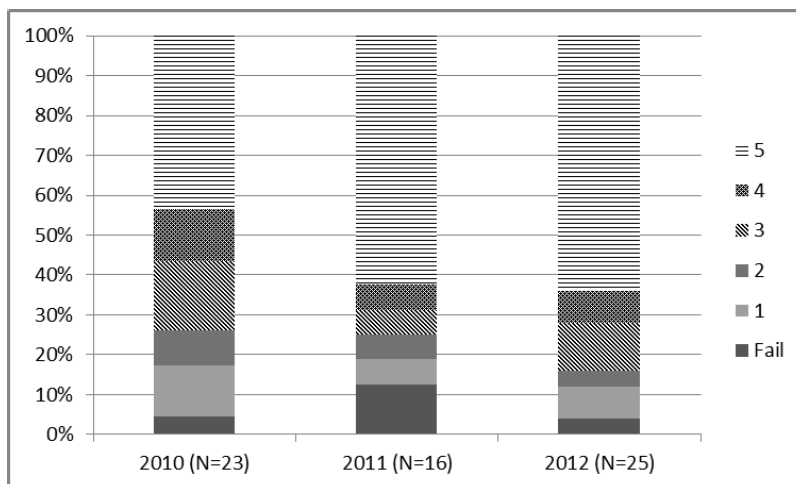
Fig. 4. Grade distribution at Course 2 visualized.

Table 5

Course 2 instances' mean grades throughout 2006…2012

Year	Grade mean
2006…2009 (N=21)	3.14
2010 (N=23)	3.52
2011 (N=16)	3.75
2012 (N=25)	4.04

Table 6

Points gathered in ViLLE in instances of Course 2

Year	Total maximum	Mean score	Std. dev.	% of maximum
2010	700	552.36	99.57	78.91 %
2011	660	567.06	128.06	85.92 %
2012	660	588.73	83.39	89.20 %

Though the differences are rather small, it seems that the students completed more exercises when visualization was accompanied with other exercise types starting from 2011.

7. Discussion

Based on the results presented, it seems that ViLLE exercises have a positive effect on learning. The average grade in both courses increased – though in Course 2 there are no significant changes (though this might be because of the low N). The pass rate in Course 1 also improved. In this section, the results for both courses are first discussed separately. Then the rules for adaption presented earlier are revisited in context of the results. Finally, as there are issues when measuring and comparing the performance at whole course level, some critical points of view are presented.

7.1. *Performance at Course 1*

Three instances of Course 1 were observed: at the first instance (2007) a link to ViLLE applet was given to students at course web page, but no points were collected and hence no minimum score limits set. In consecutive instances (2008 and 2009) ViLLE was made mandatory at course, as the minimum of 40 % of all points in ViLLE needed to be collected to pass the course. Based on the results, it seems that this had an effect on learning results. The mean average increased, and the amount of lower grades (1, 2 and 3) decreased. Also, the passing percent increased from 80.92 % to 82.09 % and 85.08 %, respectively.

It seems, that the visualization exercises combined with active learning in the form of questions has a positive effect on results. We have previously shown (see e.g. Kaila *et al.* 2009A, Laakso 2010), that visualization seems to have a highly positive effect on learning. It seems that the results gained from controlled two hour experiments can be generalized to learning at whole course. This also seems to confirm our earlier results on high school level programming course (Kaila *et al.*, 2010).

7.2. *Performance at Course 2*

There were also three instances observed in Course 2. In all of them ViLLE was made a mandatory part of the course, with minimum amount of 50 % of all points to be gathered to pass the course. The difference between instances was that at two latter instances (2011 and 2012) new exercise types were presented: only a handful of earlier visualization exercises were kept and four new exercise types were presented.

The same trend seems to exist at Course 2 results as well: when compared to earlier instances with no ViLLE (2009 and earlier) of the course, the grade mean seems to be higher when ViLLE exercises were used. Moreover, it seems that at the latter instances (2011 and 2012) of observed courses the distribution of grades seemed to focus more on the higher level of grades. No statistical differences could be found between groups, though one possible reason for this might be the low N. The trend in number of exercises completed at latter instances is still interesting: the students seemed to do more of the exercises when new types were introduced among the visualization. It is likely, that more heterogeneous set makes doing the exercises more motivating.

7.3. *The Rules of Adaptation Revisited*

The **first rule** we presented about adapting learning technology was to *introduce and integrate.* The results from Course 1 seem to underline this: when ViLLE was presented as an external tool with no connection to course otherwise, it did not seem to have a strong effect on learning. When the tool was made a mandatory part of the course, with connections drawn to other material, the grade and pass rate got higher. In Course 2 the introduction and integration was even tighter: there was a special introductory round in ViLLE where the exercise types were presented. Moreover, the exercise rounds in ViLLE were tightly integrated into course curriculum. Each round was opened after the lecture about the topic was given.

The **second rule** was to engage the students. The engagement taxonomy presented by Naps *et al.* (2002) states, that higher the level of engagement, the better the learning results. In latter instances of Course 2, new exercise types were presented. While visualization exercises lie in the engagement level of responding, most of the new types are on the higher levels of engagement. Based on the results, it seems that the students were more motivated in doing the exercises after the change, and it also seems, that the learning results were better. Though, as mentioned before, no statistically significant differences were found due to low number of students in course.

The **final rule** was to make the tool mandatory, but reward the students on using it. This rule was adapted on two final instances of Course 1 and in all instances of Course 2. In Course 1 the effect can be clearly seen: the results got better as soon as the tool was made mandatory. It is possible, that not all students find the visualization exercises motivating enough to complete them on their own. It is also likely, that at least the weaker students might not have enough patience to go through the more difficult exercises if they are not required. In Course 2 bonus points for final exam were rewarded if enough ViLLE points were gathered. This also seemed to have a motivating effect, as seen on scores obtained in ViLLE exercises: the 50 % minimum limit was clearly exceeded in all instances of the course.

7.4. *Issues in Course Long Performance Measurement*

There are some known issues when measuring the learning effects throughout the course. First, there are usually several factors that affect the learning results. In both courses, other variables were kept as steady as possible: the same teacher taught all instances of both courses and no significant changes in materials or course curriculum were made between instances. Still, isolating all factors that affect the learning is practically impossible. Also, measuring the actual learning outcome is difficult. The best we can do on course level is to compare the total grades obtained from course. As long as the components affecting the grade – and the components used to measure the grade – are kept somewhat similar, the mean grade should be reliable enough, – especially if the number of students in the course is high enough.

References

Chandler, P., Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, 10, 151–170.

Crescenzi, P. and Nocentini, C. (2007). Fully integrating algorithm visualization into a cs2 course: a two-year experience. In: *Proc. of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* : *Dundee, Scotland, June 25–27, 2007*. ITiCSE '07, ACM, New York, NY, 296–300.

DeLange, P., Suwardy, T., Mavondo, F. (2001). Integrating a virtual learning environment into an introductory accounting course: determinants of student motivation. *Accounting Education*, 12(1), 1–14.

Kaila, E., Rajala, T., Laakso, M.-J., Salakoski, T. (2009A). Effects, experiences and feedback from studies of a program visualization tool. *Informatics in Education*, 8(1), 17–34.

Kaila, E., Laakso, M.-J., Rajala, T., Salakoski, T. (2009B). Evaluation of learner engagement in program visualization. In: *12th IASTED International Conference on Computers and Advanced Technology in Education* (*CATE 2009*) : *November 22–24, 2009, St. Thomas, US Virgin Islands*.

Kaila, E., Rajala, T., Laakso, M.-J., Salakoski, T. (2010*)*. Long-term effects of program visualization. In: *12th Australasian Computing Education Conference (ACE 2010)* : *January 18–22, 2010, Brisbane, Australia*.

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A., Malmi, L. (2005). Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1), 49–68.

Laakso, M.-J., Rajala, T., Kaila, E. and Salakoski, T. (2008). The impact of prior experience in using a visualization tool on learning to program. In: *Proceedings of CELDA 2008*. Freiburg, Germany, 129–136

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A. and Malmi, L. (2005*)*. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1), 49–68.

Laakso, M.-J. (2010). *Promoting Programming Learning. Engagement, Automatic Assessment with Immediate Feedback in Visualizations*. TUCS Dissertations no 131.

Laakso M.-J., Kaila E. and Rajala T. (2014). Ville – collaborative learning environment. Sent to *Computers and Education*.

Liaw, S.-S., Huang, H..M. and Chen, G.-D. (2007). Surveying instructor and learner attitudes toward e-learning environments. *Computers & Education*, 49(4), 1066–1080.

Lockyer, L., Patterson, J. (2008). Integrating social networking technologies in education: a case study of a formal learning environment. In: *Proceedings of 8th IEEE international conference on advanced learning technologies*. Santander, Spain (2008), 529–533.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment : TRAKLA2. Informatics in Education, 3(2), 267–288

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. In: *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, 35(2), 131–152.

O'Neill, K., Singh, G., O'Donoghue, J. (2004). Implementing e-learning programmes for higher education : a review of the literature. *Journal of Information Technology Education*, 3, 312–23.

Paechter, M., Maier, B, Macher, D. (2010). Students expectations of, and experiences in e-learning : their relation to learning achievements and course satisfaction. *Computers & Education*, 54, 222–229.

Rajala, T., Kaila, E., Laakso, M.-J., Salakoski, T. (2009). Effects of collaboration in program visualization. In: *Technology Enhanced Learning Conference 2009 : TELearn 2009, October 6 to 8, 2009, Academia Sinica, Taipei, Taiwan*.

Rajala, T., Laakso, M.-J., Kaila, E., Salakoski, T. (2007). VILLE – a language-independent program visualization tool. In: Lister, R. and Simon (Eds.) *Koli Calling 2007 – Proceedings of the Seventh Baltic Sea Conference on Computing Education Research*, *Koli National Park, Finland, November 15–18, 2007*. (Conferences in Research and Practice in Information Technology, 88). Koli National Park, Finland, ACS.

Saunders, G., Kelmming, F. (2003). Integrating technology into a traditional learning environment. *Active Learning in Higher Education*, 4, 74–86.

Zuvic-Butorac, M., Nebic, Z., Nemcanin, D. (2011). Establishing an institutional framework for an e-iearning implementation –experiences from the University of Rijeka, Croatia. *Journal of Information Technology Education*, 10, 44–56.

Appendix A. ViLLE Exercise Types

Exercises for Computer Science

Visualization exercise: Combines the graphical, step-by-step execution of the example program with three types of questions: multiple choice questions, open questions and array questions.

Code sorting exercise: Commonly known as Parson's puzzle: the students need to arrange the shuffled program code lines into correct order so that given task is fulfiled.

Coding exercise: The task is to write a program – or a missing part of the program according to given specifications. ViLLE supports a variety of programming languages, including Java, C++, C# and Python.

Robot exercise: The goal of the exercise is to move number of boxes into specified target locations. The boxes are moved by writing an algorithm that controls a robot crane. Idea is to teach loops and methods in Java.

Clouds & Boxes: Reverse-visualization type exercise: the students are supposed to simulate the state of program after each step executed.

Other CS exercises: In addition, there are exercise types for testing binary calculations and conversions between hexadecimal, decimal and binary.

Exercises for Mathematics

Math exercises for elementary school level: There are several exercise types meant for teaching elementary level mathematics. In these exercises, the students for example need to find out the missing number, drag and drop numbers into number line, do long division, find out values in bar charts, calculate with fractions and so on.

Math exercises for higher levels: There are also exercise types meant for students in higher levels: for example, solving quadratic and first degree equations, doing differential coefficient calculations and writing inequality equations and sign charts.

Other Exercises

Quiz: The most basic exercise type: contains multiple choice and open questions with attachable materials. Quizzes can be utilized in any level and topic.

Survey: Can be used for course opening and closing surveys. Moreover, ViLLE surveys are typically utilized to implement assignments that are graded by teacher, for example essays.

Sorting: ViLLE contains exercise types for image puzzles, and for general sorting and pair matching of textual items.

Language exercises: There are several exercise types meant specifically for language teaching (such as fill-in, dialog, word ordering, punctuation and case, vocabulary test, compound exercise and so on). However, most of these can be utilized under other topics as well.

Image tagging: Exercise where the students need to identify areas in given or uploaded image.

E. Kaila, M.Sc., is working on his PhD about utilizing learning environments and assignments effectively. He has 25 scientific publications in peer-reviewed journals and conference proceedings. His research interests include program visualization, learning environments, automatic assessment and course design utilizing new technologies. He has been working on development of ViLLE from the beginning of the project.

T. Rajala, M. Sci., is a university teacher in Software Engineering at University of Turku, Finland. In addition to teaching, his work and research mainly focuses on developing educational software tools and studying their effectiveness in learning programming and algorithmic problem solving. Rajala finished his master's degree at University of Turku in 2007. He has 25 scientific publications in peer-reviewed journals and conference proceedings. He has also been working on ViLLE project since the beginning.

M.-J. Laakso, PhD(tech), is a lecturer and adjunct professor at Department of Information Technology at University of Turku. He has more than 35 scientific publications in internationals journals and conference proceedings. His main research interests are educational technologies, learning environments, automated assessment, visualization, immediate feedback, eAssessment and effect of collaboration in aforementioned topic. He is heading ViLLE team research group at University of Turku studying all these aspects and developing ViLLE – the collaborative education platform.

R. Lindén, M.Sc., is working on his PhD about automated student profiling and counselling. His research interests involve data mining, systems analysis and graph theory. He has three publications in peer-reviewed journals and conference proceedings. Lindén has been working on ViLLE project since the beginning of 2012.
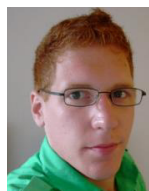
E. Kurvinen, M.A. (Education), is finishing his M.Sc. in computer science, and starting his PhD about recognizing learning disabilities in mathematics. His research interests mainly concern usage of educational technology in mathematics education in all levels. Kurvinen has been working on ViLLE project since the beginning of 2012.

T. Salakoski, PhD, is a professor of Computer Science at University of Turku. He is the Dean of Science and Technology Education at the university, and the Head of the Department of Information Technology. He has more than 200 scientific publications in international journals and conference proceedings, and has supervised more than 10 PhDs and numerous MScs. He serves in scientific editorial boards and has organized and chaired international conferences. He is heading a large research group studying machine intelligence methods and interdisciplinary applications, especially information retrieval and natural language processing in the biomedical and health care domain as well as technologies related to human learning, language, and speech.

The Approach of Early Olympiad Preparation "Olympic Lift"

Vladimir M. KIRYUKHIN[1], Marina S. TSVETKOVA[2]

[1] *Dept. of Informatics and Control Processes, National Research Nuclear University "MEPhI"*
  31 Kashirskoe Shosse, Moscow 115409, Russian Federation
[2] *Academy of Improvement of Professional Skill and Professional Retraining of Educators*
  8 Golovinskoe Shosse, Moscow 125212, Russian Federation
e-mail: vkiryukhin@nmg.ru, msvm@lianet.ru; tsvetkova@lbz.ru, msvm@lianet.ru

Abstract. An approach of involvement of 10–14 years old children in early education in informatics and their training for participation in informatics olympiads is presented in this paper.

Keywords: informatics, computer science, olympiads in informatics, IOI, preparation for informatics olympiads, methods of work with talented children, developmental teaching.

1. Introduction

The observations from the last years have shown that to win gold medal at the International Olympiad in Informatics (IOI) becomes harder and harder.

There are many reasons for this:

- **First**, the number of the countries that participates in the IOI is increasing. The number of competitive teams and students that are able to win medals is increasing too. It means that these countries adopt the world experience for training the talented in the domain of informatics children. And this process is supported by organized the IOI community conference, by placement on the Internet of a huge amount of didactic and methodical materials on training for programming contests in the different countries, and by the increasing the opportunities for school students of the world to participate in the various open programming contests on the Internet.
- **Second**, the development of information technologies increases the complexity of the IOI. Olympiad tasks become considerably complicated. There was a necessity to use automatic evaluation systems for testing the solutions of the tasks from previous IOI. Such evaluation systems (including a large archive of competitive tasks) are already available on the Internet and students actively use them in preparation for IOI. In Russia, such a service also already exists.
- **Third**, the complexity of the problems of an IOI and the technology of the contests narrowed very much the range of teachers that are ready to be coaches of

young students. This creates serious problems for school students. All teachers understand that children need to start preparing for programming contests as early as possible in order to have sufficient time to achieve high results at the IOI. But many teachers consider themselves not ready to work with the top level students or that it is very difficult for them.

Organizers of the Olympiad in Informatics in Russia have already created methods of work with talented students aged 15–17 years (Kiryukhin and Tsvetkova, 2010). These methods brought good results. But the start in 15–17 years is rather late. The school teacher could reveal the talented child and start process of her/his preparation to participate in contests earlier – at the age from 8 to 12 years. This paper describes the experience of Russia to establish in each school an environment – a so called "Olympic lift", for training of very young students to participate in programming contests.

2. A School Training Environment for Programming Contest

There are about 45000 secondary schools in Russia. We will describe the system of training talented schoolchildren starting in 5th grade (when students are 10 years old). The goal is, when these students are in the 9th grade (15 years old), to be able to compete on equal terms with the best IOI participants. This system could be applied at schools in Russia where children voluntarily participate in the National (Russian) Olympiad in Informatics since 5th grade (Kiryukhin, 2008, 2009, 2011, 2013).

Important methodical approach in development of school children talent is the choice of a syllabus for teaching informatics. But this isn't enough. To provide a successful start of the talent in any subject, it is necessary to have a teaching syllabus in the primary school, aimed at the development of the child. A number of schools in our country teach students according to programs of "early development". The main feature of these programs is including in them (for all subjects that could be chosen by school children) additional tasks, including specially selected creative tasks. This specificity of the talented children teaching is important for two reasons: first, early development of children in primary school helps schools reveal talents, and, second, it helps the children to receive a motive for development in the chosen area of teaching.

For school informatics it was required to combine different curricula into one comprehensive curriculum from the second to the eleventh grades.

This curriculum includes:

- Curriculum of a regular school course in informatics for all grades of the secondary school from 2nd to 11th.
- Curricula of additional teaching of informatics: additional courses in mathematics and informatics for 5th–7th grades (zone of nearest development).
- Individual curricula for olympiad training (horizon of talent development).

Curriculum of regular school course in informatics is created in such way that the pupil could follow the course according to an individual plan. That is why the programs and textbooks are developed for the different profiling directions: humanitarian, social

and economic, scientific, and technological. Each child has her/his "route" for studying informatics at school. For participants in the Olympiad in Informatics both the scientific profile and the technological profile are dedicated. The general course of informatics for 5th–6th grades is provided as a preparation to enter into a profile. The schools, which are actively working with pupils and are training them to participate in the Olympiad in Informatics, certainly include the general course in the current curriculum of the school. Further they can choose curricula and textbooks of different level of complexity. At primary school different possibilities of studying informatics are provided for children too: beginning from the 2nd or 3rd grade at the choice of school.

Curricula of additional teaching on informatics are curriculums for studying the basics of informatics: algorithms, programming languages, elements of mathematical logic, sets theory, introduction to counting, probability theory, graph theory, number theory, geometry. Schoolchildren can study such topics not only after lessons at school, but also in programming clubs at school or near the home, as well as at schools for distant education attached to the best universities in the country. For such schoolchildren in different regions of the country, winter and summer schools of informatics are organized where teaching is combined with the rest.

Individual curricula for olympiad training are purposeful curricula for self-study work of schoolchildren preparing for the different stages of National Olympiad in Informatics. It defines the horizon of her/his talent development. For the studying of such curriculum the children are suggested to participate in remote training contests and in the open internet contests in informatics in the country and in the international internet contests in informatics. And more, school children studying such curriculum receive a plan of work for every half-year, set of tasks to solve during the year and a consultant – an experienced coach to help (through the internet) the olympiad winners in the region.

The usage of three different curricula for students helps the talented children to reach increasingly higher levels of personal achievements annually. We will call such growth of achievements "Olympic lift". As a result, this method of talented children development forms super intellectuals who successfully implement the social lift in professional activity, becoming IT professionals in their country and in the world.

The analysis of the results of Russian students from the Olympiad in Informatics over the last 7 years showed that the most successful are those students who began their olympic lift at the age of 8-10 years. These children were revealed at early age and had the possibility for equitable and sustainable development in the school, in partnership with their coaches and the academic community.

The three curricula are included into the methodical system of the National Olympiad in Informatics which includes:

- Methods of teaching school courses in informatics.
- Methods of development of the talent (formation of a zone of the nearest development of the talent, choosing the content of an advanced course in informatics and mathematics, studying of additional topics on informatics in advance).
- Methods of individual teaching (development the content of Olympiad in Informatics, achievement of the development horizons of the talent at the different stages of the National Olympiad in Informatics).

It is important to note that formation of the nearest development zone and development of school students' motivation in the field of informatics just begins at the primary school. The school teachers of informatics together with the primary school teachers involve the students in the subject and reveal the children interested in informatics. It is very important to give the chance to kids to participate in competitions on development of algorithms. An example of such a competition for students of primary schools is offered with free access on the website with virtual labs in informatics (System of Virtual Labs in Informatics "Book of Problems 2–6", 2008).

For pupils of 5th–6th grades the curriculum of school course of informatics and the curriculum of additional classes in informatics and mathematics (as a zone of the nearest development of the talent) are dominating, and the horizon of development is an additional group training (according to an individual plan) for school or the municipal stage of the National Olympiad in Informatics, requested by students. It is very important that this training doesn't demand involvement of the special coach and is carried out by the school teachers of informatics with the tasks of school and municipal stages of the National Olympiad in Informatics. Achievement of the child (the development horizon) in this case is the diploma of winner of a school stage of the National Olympiad in Informatics. The school stage of the olympiad in Russia is organized at each school that has children intending to participate, starting from 5th grade (10 years old). At a school stage of the National Olympiad in Informatics it is very important for the teachers of informatics to reveal the talented students interested in informatics and to involve them as soon as possible in groups for profound studying of informatics (following an additional curriculum) at their schools or in programming club at schools where such classes are given. Such careful on time attention to the talented children from the 5th grade increase the opportunities to fully develop their talents in the future.

Russia's experience has shown that every school teacher of informatics should know about the whole curricula in informatics in order to be able to choose the appropriate road for school students with different motivation, to motivate talented children to work on an individual plan and understand their capabilities and the capabilities of the potential pedagogical partners, who work in the system of additional education or universities.

Teachers and coaches are often surprised that their efforts in advanced classes do not bring good results at the Olympiad in Informatics to their students. The reason is that the olympiad is a competition and has its specificity. It only fixes the level of growth of the talent. For increasing the achieved level additional training for the olympiad is also required within an individual curriculum. Moreover, it should be decided – by the teacher, the coach and the pupil – to what kind of competition the student is preparing. This will determine what kind of competition tasks (or which tasks book) has to be used in the training process.

The individual curriculum of olympiad training can't be accomplished without regular (daily!) work of the student. Since 10–12 years old children do not have the experience to plan their work yet and have not yet formed strong-willed qualities, they certainly need an adult helper – a mentor or tutor. Any talented child studies at school. He is there almost every day, so his school teacher can act as her/his mentor. This is the most important mission of the school teachers in informatics which could: monitor individual

self-preparation of the students (especially of these aged from 10 to 14 years); contact the coach of the child in case of need; help the child to enrol in a district programmers club or distant school at an university; trace participation of the child in the selected for her/him competitions; help the child to co-ordinate his absence from school during the olympiad; explain to parents the child's problems; and others. This particular cooperation of child and mentor forms a real willingness of a schoolchild to manifest her/his talent and sustained successful results in all stages of the National Olympiad in Informatics – from school stage to the final stage (the "Olympic lift").

The basis of self preparation for different stages of the Olympiad in Informatics and constructing an individual trajectory of such preparation (individual schoolchild plan) consists of the following methodological and didactic materials on the Olympiad in Informatics:

- The sample curriculum of the Olympiad in Informatics (offered to schools by the Central Methodical Commission of the Russian Olympiad in Informatics – ROI) which is used by the teachers in training for school and municipal stages of ROI, by coaches of the student for the regional and the final stage of the ROI and the coaches that train the national team for the IOI.
- Materials for theoretical preparation – printed and published in electronic form on sites, including video lectures.
- Collections of olympiad tasks of all levels of complexity and all topics of olympiad preparation with brief methodical guidelines for their solution.
- Websites with collections of olympiad tasks and possibility of automated testing of tasks solutions.
- Websites providing regular online competitions in Informatics and programming.
- The sample curriculum of the ROI has three levels of complexity: "initial" – for 5th–6th grades (10–12 years), "basic" – for 7th–8th grades (12–15 years) and "advanced" – for 9th–11th grades (15–17 years). The third level provides a special part dedicated to training the national team for the IOI. This curriculum is a basis for development of programs of individual olympic preparation of the students for the stages of the ROI.

The curriculum for olympiad preparation in informatics for student aged between 10 and 12 years is fixed in the individual plan of each student. It is created on the basis of her/his achievement of studying both the basic curriculum of a school course in informatics and the curriculum of additional profound preparation. I.e. the individual curriculum for olympiad preparation has to be formed from both the school informatics teacher and the coach of the student. It is clear that if the student studies in 5th grade and her/his horizon of development is participation in the IOI in the 8th grade then her/his individual curriculum of olympiad preparation has to be mastered for and completed during the corresponding period (3 years in our example).

An important component in self-preparation for the Olympiad in Informatics is participation of the students in the internet programming contests which are organized regularly in many countries. It is important to choose for each student a subset of such contests, without overloading her/him, and then to ask her/him to participate in the chosen contests. It will allow the student to gain solid experience in participating in program-

ming contests. And moreover, in such a way school students learn to self-assess because they periodically compare results of their performance with the results of coevals from other schools, regions and even from other countries. It gives the chance to the teacher-mentor to trace constantly deficiencies in teaching of the student, to correct the individual plan of her/his self-preparation curriculum, and also to make recommendations to his coach in the programming club or university.

Besides, the teacher can recommend to the student to complete the school course of informatics in advance, according to her/his individual plan. For example, in 5th grade the student could study the course for 7th grade, in 6th–for 8th, and during the 7th–9th grades to study the course for 9th, 10th and 11th grades. Then during the training at 10th–11th grades she/he will be able to choose an advanced course in informatics at the most difficult level. The coach, in turn, could introduce amendments in the additional curriculum of profound teaching of this student.

The possibility to participate in online programming contest for each student, irrespectively of in what school she/he is enrolled in and where they live, helps her/him for self-preparation, but most importantly is that it helps the best regional teachers, mentors and coaches from the programming clubs and universities to pay attention to this student and to include her/him in the activities of additional profound teaching of informatics and to create for her/him an individual curriculum of olympiad preparation.

The helpful international websites for individual training of Russian students for programming contests were identified as:

```
http://www.topcoder.com/tc
http://www.hsin.hr/coci/
http://acm.uva.es, http://train.usaco.org/usacogate
http://www.acsl.org, http://www.inf.bme.hu/contests/tasks
http://www.mii.lt/olympiads_in_informatics
```

3. The Teacher's Role in Olympic Preparation of School Students

Many teachers ask us how they can prepare the pupils from the 5th grade for successful participation in all stages of the ROI. The fact is that the olympiad preparation of such students must be situated after the lessons and on specific curricula (Kiryukhin and Tsvetkova, 2011). In different years children from 3rd, 4th, 5th, and 6th grades participated even in the final stages of the ROI. For example, 245 school students participated in the final stage of the ROI in 2014 and among them were one 9 years old student, one 11 years old and five students of 12–13 years old.

Taking into account the above mentioned, the work of the teacher with gifted children in domain of informatics, in the contemporary world, has to be organized as follows:

- **First**, the search of talented children has to be done in the primary school. All necessary conditions for this are available and the school students after the 4th grade are already ready to study in 5th–6th grades more difficult algorithms and the basic capabilities of programming environments. And here we are discussing about co-operation of teachers in primary and secondary school that will permit the smooth passing of students who have demonstrated outstanding ability in primary school

under the concerns of the secondary school teachers. The role of the primary school teachers is just this – to reveal kids that are enthusiastic for studying informatics.

- **Second**, the informatics teacher has to organize additional classes with children for olympiad preparation at school. The role of the teacher of the secondary school is to attract children from the 5th–6th grades to the school stage of the Olympiad in Informatics, to inform them about programming clubs in the area or at the school, to provide information on Internet resources and textbooks, to invite them to visit additional classes of group for initial olympiad preparation. It is desirable that this group includes school students of different age, for example, 10–12 years old. It would be very useful if some classes dedicated to analysis of olympiad tasks for these children was carried out by older pupils, aged 12–14. Children perfectly understand each other.

  It is important that winners of the regional or final stages of the ROI, as well as other awarded students, were involved in training of group of children from the basic levels of olympiad preparation, and winners and awardees of the municipal stage participated in preparation and holding of school stages of the ROI for children of 5th–6th grades. Teachers have to keep contact with olympiad winners that graduate of school, becoming university students. They could be further coaches of groups for profiled olympiad preparation and on-line consultants for children of groups for basic olympiad preparation.

- **Third**, it is not possible to demand all informatics teachers work with talented children, to ask from them strong management of all necessary theory and practice for solving of difficult olympiad tasks. In this situation a task of the teacher is to contact with those who can help him, and the student, to gain necessary knowledge and skills. Such persons could be professors of the closest universities, the former olympiad winners who are university students and even high school students who already achieved certain success in the Olympiad in Informatics. That is, the role of the teacher is to know the olympiad community of the region or municipality, to be in contact with this community and to include the gifted school student in it in time.

- **Fourth**, the work with each talented student has to be based on an individual trajectory of teaching according to the individual plan (Individual preparation plan to the IOI, 2010). The plan shall be for a fixed period and includes a list of planned to explore topics theoretical preparation, a list of related training tasks that are settled, the list of necessary resources to implement the plan and the results of the plan and periodic self-test.

  In this plan the self-training of the student plays an important role. To help students make such a plan and to control its fulfilling is also an important task of the teachers who work with talented school students. With this plan the teachers can monitor the dynamics of student achievements, completeness of olympiad tasks solutions, speed and quality of the student etc.

- **Fifth**, the work with gifted students should be done with the use of modern information technologies (e-mail, chat, distance learning systems, internet systems of remote video presence) and on a regular basis. It is important that during the inde-

pendent work the student has the opportunity to quickly ask for help, not only from
the teacher, but also from the coach and the members of the olympiad community.
Besides, it is necessary to provide the students with access to automatic evaluation
system in order to receive complete and accurate information about the results of
checking the correctness of task solutions, to identify its deficits and to adjust in
time his plan for the future work on the solution. At the same time, planning the
work of teacher on olympiad preparation should be based on well-known collec-
tions of tasks from past Olympiads in Informatics, including IOI tasks. These tasks
are a methodical collection of the teacher.

4. Schoolchildren Olympic Lift

When the teacher trains talented schoolchildren they should not approach the stages of
their olympiad preparation the same way as it is done in a school. Stages of training of a
talented child should be defined in accordance with his high development potential and
taking into account the natural stages of his growing up. This means that the teacher can-
not overload the children physically, but cannot also stay away from their high demand-
ing, intellectual capacity and motivation for learning and development. This is similar to
the movement of the lift which moves steadily upwards from floor to floor (horizons of
development) without shocking jumps and overloads, but with a guaranteed result of the
upward movement, not from school year to school year, but from one development floor
to another such floor, that could be ahead of traditional school grades.

    These olympiad lift floors are: elementary school (talent identification), 5th–6th grade
(10–12 years old), 7th–8th grade (12–14 years old) and 9th–11th grade (15–17 years old).
Correspondence with various forms of work with talented schoolchildren for each olym-
pic lift floor is shown in Table 1. Olympic lift floors reflect also the stages of growing
of children, taking into account changes in the nature of their behaviour (changing of
priorities in behaviour, as psychologists say) and physical capabilities. Thus, a sample
of teaching route of talented child (lift movement) must fit into his school life and to be
correlated with his physical and psychological changes of growing up.

    In the movement of the olympic lift it is possible to identify three thresholds of
growth:

- **The first threshold** is the child's transition from primary school to basic school. If
teachers were engaged in the identifying of the talent, this transition will be smooth
for the child, and she/he will be included in a natural way in the olympiad. This
threshold is also the *entrance in the olympic lift*.
- **The second threshold** of growth is between 6th and 7th grade at school. This is the
stage of the threshold of the determination of the talented student – in what area
he will develop. This is his *individual profiling choice*. After it a planned and very
hard work comes, following an individual program of olympiad preparation.
- **The third threshold** comes between 9 and 10 grades (when the student is 15-16
years old). Talented children in this moment of development (in the case of regular
work with them) have already made the choice for themselves and determined their

future specialty. And this choice helps them to make next move of the olympic lift. This allows talented students to develop steadily in Olympiad in Informatics and to organize their profiled preparation very intelligently and independently.

For every talented schoolchildren it can be shown the sample route of movement on the olympic lift which is presented in Table 1. It is necessary that teachers understand well what the olympic lift is for pupils. Only then they will be able to help talented pupils throughout their preparation as mentors.

Table 1
Forms of work with talented schoolchildren for each olympic lift floor

Stages of schooling	Olympiad lift floors in promoting schoolchild in the Olympiad in Informatics
Primary school, 1st–4th grades.	Informatics club in primary school with informatics teacher.
Basic school, 5th–6th grades.	• Additional teaching and informatics club at school. • School stage of the Olympiad in Informatics. Goal – to become the winner. • Municipal stage of the Olympiad in Informatics. Goal – to become a winner or a medallist.
Basic school, 7th–8th grades (an individual teaching plan for the 7th grade, an individual teaching plan for the 8th grade).	• Lyceum program on informatics and mathematics. • Participation in on-line programming school at the university. • Programming club, network community of the olympiad participants, summer and winter schools in the region. • Participation in the regional stage of the Olympiad in Informatics. Goal – to become the winner. • Participation in the final stage of the National Olympiad in Informatics. Goal – to become a winner or a medallist. • Participation in a training camp of candidates for the National team.
Going to high school, 9th grade. High School, 10th–11th grades (individual plan of pro-filing tea-ching for 9th–11th grades).	• Individual programs of profiled teaching. • Participation in the final stage of the Olympiad in Informatics. Goal – to become the winner. • Regular training camps, including using distance environment of olympic preparation. • Participation in the International Olympiad in Informatics.

5. School Resources for Advancing of Schoolchildren with Olympic Lift

Every school in cooperation with the organizations for additional education, the associations "school-university" and the system for distant education in the region creates the conditions of ensuring the advancement of gifted pupils with the olympic lift. Among these resources are the following:

- Full-time training camps with the program of the Olympiad in Informatics for up to 30 days per year.
- Open internet collection of tasks from school, municipal, regional and final stages of the Olympiad in Informatics from the last 5–7 years.
- An environment for on-line communication with regional coaches working with gifted children and receiving consultation from them.
- On-line training contests, carried out on a regular basis according to the level of olympiad preparation.
- Open collection of video lectures on the basic topics of olympiad preparation.
- Internet environment for self-training and open olympiad e-library.
- Computing equipment for organizing of all kinds of activities with gifted students, satisfying the requirements of the Olympiad in Informatics.
- Team of coaches (on-line moderators of training sessions).
- Medallists of regional and final stages of the ROI and university students, participants of the world collegiate olympiad (ICPC).
- Website for training sessions, with forum for leaders and training teachers.
- Regular on-line training contests for students included in the long list of the National team for participation in the IOI (at least once every two weeks) using tasks of past years with appropriate level of difficulty.
- Statistics of results and coaches analysis of the individual deficits in preparation of the gifted students.
- Individual plan for independent work of each student, based on the analysis of her/his common failures in solving the olympiad tasks and the task from on-line contests.

6. Conclusion

In conclusion, it should be noted that in modern conditions the work on development of gifted students should take place in close cooperation of primary school teachers (revealing young talents), basic school teachers (involving children in the olympiad community and developing their talent), high school teachers together with universities teachers and top students (achieving success in national and international competitions). The coordinating role in this interaction must belong to informatics teachers who are directly involved in the education of gifted children in school and a very important educational role – the role of mentors.

   It is possible to formulate an important conclusion: programming contests allow every talented child, who decides to participate, a guaranteed formation of creative personality, exhibiting high availability of creative evolution in the future professional activities. It is very humane and highly significant quality of the creative personality, which could be formed in any country where their Olympiad in Informatics is organized, on the basis of systematic teaching of schoolchildren in accordance with the programs for development potential of the child at different age levels. And the technique of working with talented children called the "olympiad lift" is quite effective for the purpose.

References

Kiryukhin, V. (2008). *Informatics. Russian Olympiads*. Issue 1, Prosveschenie, Moscow. (In Russian, *Информатика. Всероссийские олимпиады*. Выпуск 1).

Kiryukhin, V. (2009). *Informatics. Russian Olympiads*. Issue 2, Prosveschenie, Moscow. (In Russian, *Информатика. Всероссийские олимпиады*. Выпуск 2).

Kiryukhin, V. (2011). *Informatics. Russian Olympiads*. Issue 3, Prosveschenie, Moscow. (In Russian, *Информатика. Всероссийские олимпиады*. Выпуск 3).

Kiryukhin, V. (2013). *Informatics. Russian Olympiads*. Issue 4, Prosveschenie, Moscow. (In Russian, *Информатика. Всероссийские олимпиады*. Выпуск 4).

Kiryukhin, V., Tsvetkova, M. (2010). Strategy for ICT skills teachers and Informatics olympiad coaches development. *Olympiads in Informatics*, 4, 30–51.

Kiryukhin, V., Tsvetkova, M. (2011). Preparing for the IOI trough developmental teaching. *Olympiads in Informatics*, 5, 44–57.

System of Virtual Labs in Informatics "Book of problems 2–6". (2008). (In Russian, *Система виртуальных лабораторий по информатике «Задачник 2-6»*.
    `http://lbz.ru/files/5799)` `http://www.lbz.ru/books/264/5211/`

Individual preparation plan to the IOI. (2010).
    `http://metodist.lbz.ru/lections/6/files/ind-plan.xls`

V.M. Kiryukhin is professor of the Russian Academy of Natural Sciences. He is Chairman of the Federal methodical commission on Informatics which is responsible in Russia for carrying out the National Olympiad in Informatics. He is author of many papers and books in Russia on development of Olympiad n Informatics and preparations for competitions in Informatics. He is the exclusive representative who took part at all IOI from 1989 as a member of the IOI International Committee (1989–1992, 1999–2002, 2013–2014) and the Russian team leader. He received the IOI Distinguished Service Award at IOI 2003, the IOI Distinguished Service Award at IOI 2008 as one of the founders of the IOI making his long term distinguished service to the IOI from 1989 to 2008 and the medal "20 Years since the First International Olympiad in Informatics" at the IOI 2009.

M.S. Tsvetkova is professor of the Russian Academy of Natural Sciences, PhD in pedagogic science, prize-winner of competition "The Teacher of Year of Moscow" (1998), main expert of state projects of school education informatization in the Ministry of Education of the Russian Federation (2001–2005), the expert of the World bank project "Informatization of Education System". Since 2002 she is a member of the Central methodical commission of the Russian Olympiad in Informatics, the pedagogic coach of the Russian team for IOI. She is the author of many papers and books in Russia on the informatization of education and methods of development of talented students.

CMS: a Growing Grading System

Stefano MAGGIOLO[1], Giovanni MASCELLANI[2],
Luca WEHRSTEDT[3]

[1] *London, U.K.*

[2] *Faculty of Sciences, Scuola Normale Superiore*
   Piazza dei Cavalieri 7, 56126 Pisa, Italy

[3] *Department of Mathematics, University of Bologna*
   Piazza di Porta San Donato 5, 40126 Bologna, Italy
e-mail: s.maggiolo@gmail.com, giovanni.mascellani@sns.it,
luca.wehrstedt@gmail.com

Abstract. We give an update on CMS, the free and open source grading system used in IOI 2012, 2013 and 2014. In particular, we focus on the new features and development practices; on what we learned by running dozens of contests with CMS; on the community of users and developers that has started to grow around it.

Keywords: CMS, contest management system, grading system, IOI, IOI-like competitions.

1. Introduction

CMS (*Contest Management System*) is a free and open source grading system to run the IOI and similar programming contests[1]. Since our first presentation in (Maggiolo and Mascellani, 2012) the project saw a lot of activity: new features were added, some parts were redesigned, many bugs were fixed. CMS has been used in two IOI editions (and will be used in 2014) as well as in dozens of other contests all around the world, both on-line and on-site, from small local contests to international ones. It has received suggestions, bug reports and code contributions from various enthusiastic developers in many different countries.

We thus believe that it is time for us to give a new public update to the IOI community about the state of the project, summarizing what has happened since the first presentation of CMS and briefly covering where the CMS development is headed.

We will not go again over the motivations, design principles and general structure of CMS: most of what was described in (Maggiolo and Mascellani, 2012) is still valid. Instead, we focus on what we learned from working on a more mature code base, with wider adoption, larger feedback from users and more contributions external to the core team.

---

[1] CMS's home page is `http://cms-dev.github.io/`

2. New Development

2.1. Development History

When (Maggiolo and Mascellani, 2012) was being written, CMS was about one year and a half old, and it was a project led and developed almost exclusively by three core developers involved in the Italian Olympiads in Informatics and later in the organization of IOI 2012, held in Italy.

At the time, obviously, CMS development was very tied to the IOI schedule: the CMS development group was a subset of the IOI's Host Scientific Committee, and all efforts were directed to be ready for IOI 2012. Therefore, we used a simple development model, without formal releases: IOI 2012 was essentially the first public appearance of CMS, and we planned to release CMS's first official version soon after. As hosts know, the IOI week is a hectic time when all sorts of previously overlooked small bugs start causing lots of problems, and at the same time unorganized fixes accumulate. With our post-IOI release, we implemented proper solutions substituting the fast fixes and we identified specific areas of improvement for future releases.

Indeed, a very important criterion for a grading system used at the IOI is the ability to easily merge upstream the changes introduced during the IOI, as this guarantees that known problems do not propagate to the following IOIs, and that new features (for example, to support new rules) are not implemented several times by different hosts.

We released CMS 0.9 in November 2012[2]. Its structure is essentially the same as that described in (Maggiolo and Mascellani, 2012). Apart from many small improvements and fixes, we implemented user tests (in the sense of the IOI rules): the possibility for contestants to execute their source code against their own input files in the same environment where their solution will be evaluated.

In March 2013 we released CMS 1.0[3]. This was intended to be an evolutionary release that continued the post-IOI work. Its highlights were a vastly improved documentation[4] and full support for the translation of the contestant interface.

The version used at IOI 2013 was cut from the post-1.0 development branch two months later, and it included two major additional features: the new sandbox, *isolate* (Mareš and Blackham, 2012), and task versioning.

We continued the development of CMS 1.1, which is going to be released before this article is published. The main additions have been the transition to the new event loop library, a new service taking care of the communication with *RankingWebServer*, support for additional programming languages, easier to write importers, new translations, improved testing.

We also started improving our development practices: we began reviewing all new code entering the repository; we focused in improving our tests, increasing their coverage; and we set up a continuous integration system[5].

---

[2] Release notes at `https://github.com/cms-dev/cms/wiki/CMS-0.9.0-RELEASE-NOTES`
[3] Release notes at `https://github.com/cms-dev/cms/wiki/CMS-1.0.0-RELEASE-NOTES`
[4] The documentation is available at `https://cms.readthedocs.org/`
[5] The continuous integration web interface is reachable at `http://cms.di.unipi.it/jenkins/`

During the two years that brought us here, we had the pleasure to appoint two new core developers (from Italy and Australia) and to receive contributions from other sixteen people around the world[6]. Many contributions came from future IOI hosts that decided or are considering using CMS as their grading system. Nonetheless, as the number of national teams using CMS for training and selection rises, we have seen also a growing number of contributions from people not involved in IOI hosting.

2.2. *New Features*

We list in this section the main differences between CMS pre-0.9 and CMS 1.1.

User tests. As per IOI rules, contestants can test their solutions against an input they propose, and the execution will be performed in the same environment as the evaluation against the official testcases.

Improved contestants interface. We implemented a new web UI for contestants, based on Bootstrap (Twitter, Inc., 2010), much nicer to the eye and easier to understand. The interface has also been made completely translatable and contestants can change the language.

Translations. At the moment of writing, CMS has been translated in nine languages: Bosnian, Dutch, English, French, Italian, Japanese, Lithuanian, Russian and Traditional Chinese. We welcome contributions to extend the list further.

Task versioning. More often than one would want, during a contest it is realized that some testcases are wrong. With CMS, administrators can create new sets of testcases, evaluate all submissions against them and find out how many contestants were affected by the problem; all of this in "background", without taking down the task or the scores for the initial set of inputs. When the new testcases are validated, administrators can switch to them and notify only the affected contestants, without any downtime and without most contestants even noticing.

Task versioning is not limited to input files: it can also be used to test new time and memory limits, or different libraries, graders, or scoring functions.

New sandbox. The previous sandbox, *mo-box* (Mareš and Gavenčiak, 2001), was based on system calls filtering; maintaining the list of allowed calls for compilations and evaluations was often difficult, as it depended on the architecture, the operating system and the programming language. The new sandbox, *isolate* (Mareš and Blackham, 2012), was again co-developed by Martin Mareš and is based on the new namespace features of the Linux kernel. It requires a reasonably recent version of the kernel (at least 3.8) and the *isolate* executable must be run as root (which is accomplished in CMS using the *suid* flag), but it does not require special configuration and in particular architecture-dependent ones. Moreover, it enforces limitations directly on the resources, instead than on the calls used to obtain them. It also causes much less computational overhead.

---

[6] A complete list is at `https://github.com/cms-dev/cms/blob/af11e8d6/AUTHORS.txt`

New event loop library. Up to CMS 1.0 our custom-made RPC system was based upon Python's *asyncore* framework, which now exists for "backward compatibility only" and is eventually going to be removed from the standard Python libraries. Our HTTP servers were built on top of *Tornado* (Facebook, Inc., 2009), which had its own event loop: we had therefore to have them both running simultaneously, interleaving their steps. The awkwardness of this design and the serious performance issues indirectly caused by it that came up at IOI 2012 (see section 3.1 for more details) prompted us to switch to *gevent* (Bilenko, 2014), a coroutine-oriented Python library based on the low-level *libev* (Lehmann and Giaquinta, 2014) event loop.

New RPC system. Our RPC library, called AsyncLibrary, was based on *asyncore* and was hence dropped after the transition to *gevent*. We wrote a new one that fully benefits from the new paradigm. That has been a good chance to make it more modular and safer (for example by catching and logging all exceptions in callbacks).

We also improved performance by avoiding opening more than two connections (one in each direction) between any pair of services.

ScoringService. The IOI 2013 experienced issues with slow scoring (Blackham, 2013): after fixing a testcase, the rescore took so long that they could not determine the affected contestants before the end of the contest. The slow rescoring was introduced on purpose to return the control to the event loop regularly (as the service would have otherwise appeared stuck to the rest of CMS). The problem was fixed by porting the service to *gevent*: that made the regular pauses unnecessary as the event loop could take back control in any time during the execution of I/O.

ProxyService. The philosophy of CMS has always been to use many small services that have only a small number of duties, possibly just one; this helps keeping most of the functionalities up in case something goes wrong in a specific part of CMS. In the previous design, *ScoringService* had two duties: to compute the score of each submission and to send these scores to the ranking server. Therefore, we moved the latter to a new service, called *ProxyService*.

Importers and loaders. In CMS 1.0 we had a utility, called *YamlImporter*, to easily load into CMS contests and tasks prepared using the file system format of the Italian Olympiads. A companion utility, *YamlReimporter*, was used to "reimport" an already-existing contest, i.e., updating its data without losing the submissions already sent by the users.

We always stressed that CMS should not force a specific file system format to the administrators, but the complexity of *YamlImporter* and *YamlReimporter* made it difficult to write similar utilities for other formats. Therefore, we split them into two format-independent parts (*Importer* and *Reimporter*) and a loader, which is specific to our format. This way, the support for another format can be added by just implementing a new loader, which only has to create the appropriate objects from an external source, usually a file system representation. We received some externally contributed loaders over the last months.

Programming languages support. We added support for new competition languages: Java (through gcj), Python, PHP, in addition to the classical C, C++ and Pascal. It is now

trivial to add support for other compiled languages and for some interpreted ones. Note that this is not an endorsement for allowing such new languages in the IOI; in particular, individual languages can be allowed or not for each contest.

Extended documentation. CMS has now rather comprehensive user documentation, covering the whole process of setting up CMS to run a contest. From the developer side, we have about two lines of comments every three lines of code, thanks especially to our commitment to write docstrings for every function. As CMS becomes a larger project, some shortcoming of Python's duck typing system started to become apparent, and we reacted increasing the documentation of the types of arguments and return values of functions. Moreover, a tool was developed to ensure that CMS was actually respecting the indications written in the docstrings (Maggiolo, 2013).

3. CMS Usage

3.1. *IOI*

CMS was used for running two IOI editions, in 2012 (Sirmione and Montichiari, Italy) and 2013 (Brisbane, Australia); it will also be used in IOI 2014 (Taipei, Taiwan). In both past cases CMS performed mostly well; while during the two contests there were some technical problems, most of them did not depend on CMS misbehaviour, but on mistakes in the data provided to it (e.g., wrong testcases or graders) or on other faults in the network environment. For a detailed discussion of what happened at IOI 2013, please see (Blackham, 2013).

There were, though, some issues that were CMS bugs. Probably the most important single issue was the inefficiency in the networking framework on which CMS was based. The RPC and HTTP servers were built on top of *asyncore* and *Tornado*, and took advantage of their non-blocking, callback-based APIs. Unfortunately, connections opened outside the scope of these frameworks did not benefit from it and any read or write operation on them was blocking for the whole application. Such instances were, in particular, the connections to the database (handled by SQLAlchemy) and the HTTP requests to *RankingWebServer* (handled by *httplib*).

Both of these caused serious performance bottlenecks at IOI 2012. Some services (like *ContestWebServer* and *AdminWebServer*) usually spend most of their time doing database queries: being unable to handle other requests while waiting for the results of a query made them unresponsive, especially during periods of high load or when performing large queries. At the IOI this resulted in *ContestWebServer* not being able to handle the request burst at the beginning of each day and appearing to be down for minutes. *AdminWebServer* did also hang often, but this did not cause problems to contestants.

On the other hand the internet connection at the IOI 2012 site was very poor and there was a lag of a few seconds on all outgoing requests. That caused the rate at which data was sent to *RankingWebServer* to be much less that the rate at which new data was coming in: *ScoringService* was spending all its time waiting, neglecting its duty to score

submissions and building up large queues. This issue was somewhat relieved by grouping all queued data into a single HTTP request. Yet, it was not enough and we ended up using two threads for the two distinct operations. That contributed to induce us to split off *ProxyService*.

Using for example *Tornado*'s *HTTPClient* (instead of *httplib*) to handle the HTTP connections may have resolved this issue, but we could not find viable alternatives to SQLAlchemy: the few that existed seemed to be less powerful and mature. In the end we decided to switch to *gevent*. Its execution model is based on having many execution units called "coroutines", that are a lightweight form of cooperative threads: each of them runs code that performs reads and writes using a synchronous blocking API, but I/O operations are transparently translated to non-blocking calls and, while waiting, control is returned to the event loop that allows other coroutines to resume their work. Within CMS, SQLAlchemy uses Psycopg as backend towards the PostgreSQL server, which is easily made compatible with coroutines, as detailed in (Varrazzo, 2010). Other libraries, that were not originally designed to be cooperative, can be added support to by using *gevent*'s monkey-patching capabilities. Although the *gevent* support was already written, it was not used at IOI 2013, because it was still young and not well tested.

3.2. *National and Local Contests*

After its presentation at IOI 2012, CMS was used in many different countries for contests with sizes ranging from local to international. We are aware of contests organized in Argentina, Australia, Belgium, Chile, Croatia, India, Italy, Japan, Latvia, Lithuania, Serbia, Slovenia, Taiwan and Tunisia[7]. It was used both for on-line and on-site contests, from a dozen to around a hundred contestants; some contests run with CMS were also hosted on public cloud computing services, such as the well-known Amazon EC2 engine.

CMS is also used to run permanent online instances, which do not serve specific contests, but allow users to continuously submit solutions to the set of offered tasks. Such instances are used as tools for the training of national IOI teams[8] or for collecting the homework assigned to students during university courses and make the students able to receive a direct feedback on their work[9].

Some forks were devised from CMS for handling more specific situations or contest types. For instance, William Di Luigi and Luca Versari added some social features like the possibility for users to interact with a forum[10]; Masaki Hara runs a CMS instance which serves contests for the Japanese Olympiad[11] which has support for login via Twitter or Facebook authentication.

---

[7] See a more complete list at `http://cms-dev.github.io/testimonials.html`

[8] For example, there is an instance for the training of the Italian team at `http://cms.di.unipi.it/`

[9] For example, `http://judge.science.unitn.it/`, handling exercises for the Algorithms and Data Structure class at the University of Trento.

[10] This is the case of the already mentioned instance `http://cms.di.unipi.it/`, which is run by code at `https://github.com/veluca93/oii-web`

[11] Code at `https://github.com/qnighy/cms`, public instance at `http://cms.ioi-jp.org/`

4. Future Plans

Our main goal for the future, as members of the core development team, is to make us less central in the development of CMS: to do so, we need more people to send us contributions. Translations, bug reports and fixes are always welcome; for people intending to become more stable contributors, we set up a page[12] with some ideas for interesting, self-contained projects that offers a gentle introduction to the development side of CMS. We are open to offer help and tutoring during the implementation of these ideas.

An obvious area for improvement for us is to learn to release more often. CMS 1.1 took too much time to be released and this created problems as the features introduced in the development version started to justify using it despite being, for obvious reasons, less stable than CMS 1.0. Smaller, more frequent releases will allow us to deliver new features much sooner, and we intend to get better at that. The new testing and continuous integration infrastructure will help us with this goal. Therefore, increasing the coverage of our tests, and hence the trust on them, is another main goal.

The IOI is by far CMS's main client, therefore we will continue supporting any IOI rule change and any new task format. In our experience of these past years, we realized that national competitions often have different requirements. We tried to do our best to serve the community while keeping our focus on the IOI, and we will certainly continue working with the interested national teams to support as many use cases as possible.

In terms of new features, we have at least two big changes coming ahead. The first is a reorganization of how files associated to a task or to a submission are specified in the task configuration; this will make it easier to configure tasks and possibly write new task types. The second is a redesign from the ground up of *AdminWebServer*, that will expose a simpler and more informative interface for contest administrators and will realign it to the UI of *ContestWebServer*.

5. Conclusion

We have described what has changed in CMS in the last two years, the *status quo* and where we plan to direct our development effort. After three years of work, we believe CMS to be a valid and proved contest system and we invite the whole IOI community (and, more generally, all those who are interested in organizing programming contests) to try it, evaluate its suitability for hosting the types of contests they are interested into and let us know their impressions and suggestions. In our development decisions we welcome and consider the feedback received from our users.

As pointed out above, we are looking forward to receive contributions. Beside code development, another way of contributing is by providing translations: it is our commitment to offer an easy to use interface for all contestants, also in cases where English is not necessarily the *lingua franca* (for instance, for local or national contests). Potential

---

[12] http://cms-dev.github.io/contribute.html

contributors are welcome to read the relevant pages in the documentation[13] and get in touch with the CMS development team to have their translations accepted in the main repository.

Acknowledgments

References

Bilenko, D. (2014). *gevent*. http://www.gevent.org/
Blackham, B. (2013). "*Did that Really Just Happen?*" – *A Behind the Scenes Look at IOI 2013*.
    http://goo.gl/f63r78
Facebook, Inc. (2009). *Tornado Web Server*. http://www.tornadoweb.org/
Lehmann, M., Giaquinta, E. (2014). *libev*. http://software.schmorp.de/pkg/libev.html
Maggiolo, S. (2013). *Pydoc Checker*. https://github.com/stefano-maggiolo/pydocchecker
Maggiolo, S., Mascellani, G. (2012). Introducing CMS: a contest management system. *Olympiads in Informatics*, 6, 86–99.
Mareš, M., Blackham B. (2012). A new contest sandbox. *Olympiads in Informatics*, 6, 100–109.
Mareš, M., Gavenčiak, T. (2001). *The Moe Contest Environment*. http://www.ucw.cz/moe/
Twitter, Inc. (2010). *Bootstrap*. http://getbootstrap.com/
Varrazzo, D. (2010). *Coroutine Support for Psycopg*.
    https://bitbucket.org/dvarrazzo/psycogreen-hg/

---

[13] http://cms.readthedocs.org/en/latest/Localization.html
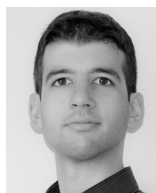
S. Maggiolo is a software engineer at Google and holds a Ph.D. in Geometry from SISSA/ISAS, Trieste. He participated in IOI 2002, winning a bronze medal and in IOI 2003. Since 2006 he collaborates with the training and selection process for the Italian team at IOI, and has been Observer in IOI 2009, Deputy Leader of the Italian team in IOI 2011 and a HSC member for IOI 2012. He is one of the main authors of CMS.

G. Mascellani has been a contestant in IOI 2007 and 2008, winning a silver and a bronze medal. He is a Ph.D. student in Geometric Analysis at the Scuola Normale Superiore (Pisa, Italy) and collaborates in training the Italian team for IOI. He was a member of the HSC for IOI 2012 and is one of the main authors of CMS. He is a Debian Developer.

L. Wehrstedt participated twice in the IOI, in 2010 and 2011, winning a bronze medal. He is attending a Bachelor's degree in Mathematics at the University of Bologna and he is a student at the Collegio Superiore di Bologna. He has been involved in the training of the Italian IOI team for the last three years, helped organizing two Italian Olympiads in Informatics and was in the HSC of the IOI 2012. He is one of the core developers of CMS.

On Maximal Level Minimal Path Vectors
of a Two-Terminal Network

Marija MIHOVA[1], Natasha STOJKOVIKJ[2],
Mile JOVANOV[1], Emil STANKOV[1]

[1] *Faculty of Computer Science and Engineering, University Ss.Cyril and Methodius*
 Rugjer Boshkovikj 16, Skopje, Republic of Macedonia
[2] *Faculty of Computer Science, University Goce Delchev*
 Krste Misirkov bb, Shtip, Republic of Macedonia
e-mail: {marija.mihova, mile.jovanov, emil.stankov}@finki.ukim.mk
 natasa.maksimova@ugd.edu.mk

Abstract. The reliability of a two-terminal flow network with a discrete set of possible capacities for its arcs is usually computed in terms of minimal path or minimal cut vectors. This paper analyzes the connection between minimal path vectors and flow functions, which supports the development of an efficient algorithm that solves the problem of finding the set of all such vectors.

Keywords: two-terminal flow network, minimal path vector, minimal cut vector.

1. Introduction

Many real-life industrial systems, such as telecommunication, electric power generation and transmission, transportation and manufacturing systems may be viewed as networks whose arcs have discrete set of possible capacities. Such systems can be regarded as multi-state systems with multi-state components, where the arcs are the system's components, whereas the demand levels of the system are all possible netflow values. Analyzing the reliability of such systems has become attractive to many researchers in recent decades. The reliability of a multi-state system can be computed in terms of minimal path vectors to demand level d, called d-MinPaths (d-MPs) (Lin, 2001; Ramirez-Marquez and Coit, 2003; Mihova and Maksimova, 2011), or minimal cut vectors to demand level d, called d-MinCuts (d-MCs) (Ramirez-Marquez *et al.*, 2003; Jane *et al.*, 1993). Both strategies extract candidates that are not minimal cut vectors by mutually comparing all pairs of vectors and removing the smaller one, if such exists. The problem of computing reliability of a multi-state system is NP-hard, but solvable (Wilson *et al.*, 2005; Provan and Balls, 1983), and commonly the inclusion-exclusion approach is used for this purpose (Provan and Balls, 1983).

Thus, the problem of searching for all *d-MCs* or *d-MPs* is one of the most important problems in multi-state network reliability, and several algorithms have been proposed as a so-

lution to this problem. Jane *et al.* (Jane *et al.*, 1993) propose a methodology that solves the problem of generating all multi-state *MCs* for the multi-state two-terminal network, obtainning a set of candidates, while Ramirez-Marquez *et al.* (Ramirez-Marquez *et al.*, 2003) optimize this procedure in such a way that their set of candidates has significantly lower cardinality.

In (Lin *et al.*, 1995), Lin *et al.* give an algorithm that finds a set of candidates for *d-MPs* and extracts all *d-MPs* by comparing all pairs of candidates and eliminating vectors that are not minimal. An approach for elimination of nonminimal candidates without comparison is given by Forghani-elahabad *et al.* (Forghani-elahabad *et al.*, 2013). Given a *d-MP* candidate, they form *m* smaller vectors (where *m* is the number of nonzero coordinates) in such a way that each of these vectors differs from the *d-MP* candidate in unit vector. If all appropriate graphs have maximum flow equal to *d*, then that vector is not a *d-MP*. The method for computing all *d-MPs* proposed in (Mihova and Maksimova, 2011) uses additional calculations that help in avoiding to obtain vectors which are not minimal.

In this paper we analyze some properties of *d-MPs* that will show the connection between *d-MPs* and flow functions to level *d* on a given two-terminal network. This helps to develop a strategy for checking whether some candidate is a *d-MP* with time complexity $O(|E|)$, which is significantly better than $O(|V|^2|E|^{3/2})$, the complexity of the strategy given in (Forghani-elahabad *et al.*, 2013). Moreover, using further analysis we give the relationship between two *d-MPs* and we propose another algorithm that directly finds all *d-MPs*. At the end, we explain the advantage of this approach, especially in the case when *d* is a maximum flow.

2. Basic Assumptions

A *two-terminal flow network* is a directed graph $G(V, E)$ with two special vertices, a source *s* and a sink $t(s \neq t)$, in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. The function *c* is called *capacity function*. Shortly, we will denote such a capacity network by $G(V, E, c)$.

A *flow* in $G(V, E, c)$ is a function $f : E \rightarrow R^+ \cup \{0\}$ that satisfies the following two constraints:

1. *Capacity constraint*: $0 \leq f(u, v) \leq c(u, v)$, for each $(u, v) \in E$, i.e., the flow of an edge cannot exceed its capacity.
2. Flow conservation:

$$f(V, v) - f(v, V) = \sum_{u \in V} f(u, v) - \sum_{w \in V} f(v, w) = \begin{cases} 0, & \{s, t\} \\ |f|, & v = s \\ -|f|, & v = t \end{cases},$$

where $|f|$ is the value of the flow.

In other words, the total flow in a node v, $f(V, v)$, must equal the total flow out the node v, $f(v, V)$, for all vertices $v \in V \setminus \{s, t\}$; the flow leaving *s* and the flow entering *t* is equal to the value of the flow.

It is assumed that if there is no edge (u, v), i.e. $(u, v) \notin E$, then $f(u, v) = 0$.

A flow is a *maximum flow* if it has the largest possible value among all flows from s to t in a given capacity network (Erickson, J., 2009).

A *pseudoflow* is a function $f : E \rightarrow R^+ \cup \{0\}$ defined on arcs that satisfy only the capacity constraints; it need not satisfy flow conservations (Ahuja and Orlin, 1993). Note that each flow function is also a pseudoflow function.

Let us assume that the set of edges in the flow network is ordered, i.e. $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Considering the edges as components, the network represents a multi-component system. It can be assumed that each component, the edge e_i, can operate in some demand level $x_i \leq c(e_i)$. The vector $\vec{x}$ is called *state vector*. In the multi-state reliability theory (Wilson *et al.*, 2005), the vector $\vec{x}$ is called *path vector to level d* if and only if the system in state $\vec{x}$ works with level equal or greater than d.

Below we introduce a few definitions that give a connection between systems and two-terminal networks.

DEFINITION 1. Let $G(V, E, c)$ be a two-terminal flow capacity network. For a pseudoflow l_c, we define *state vector $\overrightarrow{x_{l_c}}$ induced by l_c* by

$$x_i = l_c(e_i).$$

For each state vector $\vec{x}$, with $x_i \leq c(e_i)$, we define *pseudoflow function $l_{\vec{x}}$ induced by $\vec{x}$*, by

$$l_{\vec{x}}(e_i) = x_i.$$

The state vector $\vec{x}$ is called a *flow vector*, whenever $l_{\vec{x}}$ is a flow function.

Aggarwal *et al.* (Aggarwal *et al.*, 1982) defines two-terminal reliability as the probability that the network can adequately deliver a demanded flow from the source to the sink. In other words, the system is in a working state if and only if it is possible to successfully transmit the required flow from the source to the sink node. The next definition explains this more precisely.

DEFINITION 2. Let $G(V, E, c)$ be a two-terminal flow network and $\overrightarrow{x_{l_c}}$ is a state vector induced by the pseudoflow l_c. We will say that $\overrightarrow{x_{l_c}}$ is a *path vector to level d, d-P,* if and only if a flow d may be delivered in the two-terminal network $G(V, E, l_c)$. The state vector $\vec{x}$ is a *minimal path vector to level d, d-MP,* if and only if the two-terminal flow network $G(V, E, l_{\vec{x}})$ has a maximum flow d, and for each $\overrightarrow{x'} \leq \vec{x}$, the two-terminal flow network $G(V, E, l_{\overrightarrow{x'}})$ has a maximum flow less than d.

Next we give some known facts from the two-terminal network theory (Cormen *et al.*, 2009). Suppose that we have a two-terminal flow network $G(V, E, c)$. Let f be a flow in G, and consider a pair of vertices $u, v \in V$. We define the *residual capacity $c_f(u, v)$* by

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v), & (u, v) \in E \\ f(u, v), & (v, u) \in E \\ 0, & otherwise \end{cases}.$$

For the flow network $G(V, E)$ and a flow f, the residual network of G induced by f is $G_f(V, E_f)$ where

$$E_f = \{(u, v) \in E: c_f(u, v) > 0\}.$$

Note that each flow network $G(V, E)$ can be regarded as a residual network induced by a function f, where $f(u, v) = 0$ for all (u, v).

Given a flow network $G(V, E)$ and a flow f, the *augmenting path* is defined as a simple path $\mathcal{P}$ from s to t in the residual network G_f. Similarly, we will define an *augmenting cycle* as a simple cycle $\mathcal{C}$ from some node v to v in the residual network G_f. For each augmenting cycle $\mathcal{C}$ in residual network G_f, we define *augmented vector of level d' for a cycle $\mathcal{C}$, $\vec{y}^{\,\mathcal{C}, d'}$*, by

$$y_i^{\mathcal{C}, d'} = \begin{cases} d', & \text{if } e_i = (u, v) \in E \text{ and } (u, v) \text{ is on } \mathcal{C} \\ -d', & \text{if } e_i = (u, v) \in E \text{ and } (v, u) \text{ is on } \mathcal{C} \\ 0, & \text{otherwice} \end{cases}$$

for some $d' \leq \min\{c_f(e_i) | e_i \in \mathcal{C}\}$.

A *cut* (S, T) of the flow network $G(V, E)$ is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$. The *capacity* of the cut (S, T) is

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v).$$

A *minimum cut* of a network is a cut whose capacity is minimum over all cuts of the network, i.e., (S, T) is a minimum cut if for all other cuts (S', T'), $c(S, T) \leq c(S', T')$.

3. The Connection between *d-MPs* and Flow Functions to Level *d* in a Two-Terminal Network

In this section we present an approach for checking if a given flow function corresponds to a *d-MP*.

Given a flow function f, let E^f denote the set of all vertices with a positive flow, i.e. $E^f = \{e \in E \mid f(e) > 0\}$. We will refer to the unweighted graph $G(V, E^f)$ as the *graph induced by f*.

The next theorem states that a flow f corresponds to a minimal path vector if and only if E^f is acyclic. This is illustrated in Fig. 1. Namely, the flow in Fig. 1 a) is a flow function to level 3, but the state vector induced by it is not a 3-*MP* since the state vector induced by the flow of level 3 in Fig. 1 b) has induced a state vector lower then it. Note that the flow in Fig. 1 a) has additional flow through the cycle $<v_1, v_3, v_2, v_1>$, while the flow in Fig. 1 b) has no cycle.

Theorem 1. The state vector $\vec{x}$ is a *d-MP* for the two-terminal flow network $G(V, E, c)$ iff the pseudoflow function $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = d$, and the corresponding graph $G(V, E^{l_{\vec{x}}})$ induced by $l_{\vec{x}}$, has no cycles.
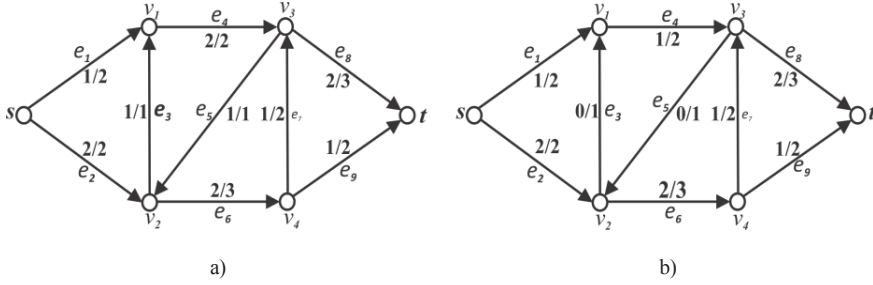
Fig. 1. a) Flow function to level 3 with $(1, 2, 1, 2, 1, 2, 1, 2, 1)$
as a vector induced by it, which is not 3-*MP*;
b) Flow function to level 3 with $(1, 2, 0, 1, 0, 2, 1, 2, 1)$
as a vector induced by it, which is 3-*MP*.

Proof. Assume that $\vec{x}$ is a *d-MP*. First, we will prove that $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = d$. Since $\vec{x}$ is a *d-P*, the maximum flow of $G(V, E, l_{\vec{x}})$ is equal to d. Then there is a flow f to level d for $G(V, E, l_{\vec{x}})$. Let $\vec{y}_f$ be the vector induced by f. It is clear that $\vec{y}_f$ is a *d-P* and $\vec{y}_f \leq \vec{x}$. Since there is no lower path vector to level d than $\vec{x}$, we have $\vec{y}_f = \vec{x}$, which implies that $l_{\vec{x}} = f$. This proves that $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = d$.

Next, let us suppose that $l_{\vec{x}}$ is a flow function and suppose the network $G(V, E^{l_{\vec{x}}})$ has a cycle. Then, $G(V, E^{l_{\vec{x}}})$ has a simple cycle, and let $e_{i_1}, e_{i_2}, \dots, e_{i_r}$, are the edges from that cycle. By $A = \{i_1, \dots, i_r\}$ we will denote the set of indices of the cycle's edges.

Let $m = \min\{x_j \mid j \in A\}$ and $\vec{y}$ is defined as $y_j = \begin{cases} m, & j \in A \\ 0, & j \notin A \end{cases}$.

We will show that the state vector $\vec{z} = \vec{x} - \vec{y}$ is also *d-P*, which, having in mind that $0 \leq \vec{x} - \vec{y} < \vec{x}$, since $c(e_i) > x_i - y_i > 0$, contradicts with the assumption that $\vec{x}$ is *d-MP*.

First we will show that the total flow for each vertex u remains the same. Clearly, if u does not belong to the cycle, the total flow in and the total flow out have no changes. If u belongs to the cycle, we have:

$$\sum_{v \in V} l_{\vec{z}}(u, v) - \sum_{v \in V} l_{\vec{z}}(v, u) = \sum_{v \in V} l_x(u, v) - m - \sum_{v \in V} l_{\vec{x}}(v, u) + m$$

$$= \sum_{v \in V} l_x(u, v) - \sum_{v \in V} l_{\vec{x}}(v, u)$$

This implies that the flow conservation constraints are satisfied and $|l_{\vec{z}}| = |l_{\vec{x}}| = d$. So $\vec{z}$ is a *d-P*.

In opposite, assume that the state vector $\vec{x}$ is such that $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = d$ and $G(V, E^{l_{\vec{x}}})$ is acyclic. We will prove that for each state vector $\overrightarrow{x'} < \vec{x}$, $|l_{\overrightarrow{x'}}| < d$.

Let us suppose that there is a path $\overrightarrow{x'}$ to level d such that $\overrightarrow{x'} < \vec{x}$. Without any loss of generality, we can suppose that $(\exists! i)\, x_i' < x_i$. Let $e_i = (w, w_1)$.

We have that

$$x_i = l_{\vec{x}}(w, w_1) > l_{\overrightarrow{x'}}(w, w_1) = x_i',$$

and for all other vertices u and v,

$$l_{\vec{x}}(u,v) = l_{\overrightarrow{x'}}(u,v).$$

Since $G(V, E^{l_{\vec{x}}})$ is acyclic, we are able to sort its vertices topologically. The same topological sort may be applied to $G(V, E^{l_{\overrightarrow{x'}}})$. Taking S to be the set of all vertices between s and w, inclusively, and $T = V \backslash S$, we will obtain a cut in $G(V, E^{l_{\overrightarrow{x'}}})$ with flow $d - (x_i - x_i') < d$. This proves that $G(V, E^{l_{\overrightarrow{x'}}})$ has maximal flow lower than d, which is in contradiction with our assumption that $\overrightarrow{x'}$ is a d-P.

Using this Theorem and Lin's algorithm (Lin *et al.*, 1995) for calculating d-MP candidates, the family of all d-MPs can be generated by the following steps:

Algorithm 1.

>   *Step* 1. Using Lin's Algorithm, find the set Q of all flow functions for which the induced vectors are candidates for d-MP.
>   *Step* 2. For each candidate $\vec{x}$ check for cycle in $G(V, E^{l_{\vec{x}}})$, and, if there is a cycle, remove it from Q.

Checking for a cycle in a graph may be simply done using DFS (Kamil, 2003), so this takes time $O(|E|)$. As a result, the time complexity of our algorithm is $O(|E|\lambda)$, where λ is an upper bound for the number of obtained candidates by Lin's algorithm. This is a significantly lower complexity than the complexity of the algorithm given in (Forghani-elahabad *et al.*, 2013), $O(|V|^2|E|^{3/2})$.

4. The Correlation between Two Minimal Path Vectors

The Ford-Fulkerson algorithm gives us a way to compute only one flow function for maximal flow in a given two-terminal network, as well as its corresponding residual network, with time complexity $O(|V||E|^2)$. The same approach may be used for computation of a flow function to level d. Using Theorem 1 we are able to find one d-MP. Here we give the connection between the two flow functions which may contribute in developing another algorithm for computing all d-MPs.

Theorem 2. Let f be a flow with $|f| = d$ in a two-terminal flow network $G(V, E, c)$ with source s and sink t, and $\mathcal{C}$ be an augmenting cycle in the residual graph $G_f(V, E_f)$. For $d' \leq \min\{c_f(e_i)|e_i \in \mathcal{C}\}$, the function f_1 defined as

$$f_1(u,v) = \begin{cases} f(u,v) + d', & (u,v) \in \mathcal{C} \cap E \\ f(u,v) - d', & (v,u) \in \mathcal{C} \text{ and } (u,v) \in E \\ f(u,v), & (u,v) \notin \mathcal{C} \text{ and } (v,u) \notin \mathcal{C} \end{cases} \tag{4.1}$$

is a flow in $G(V, E, c)$ with $|f_1| = |f|$.

Proof. To prove that f_1 is a flow function we need to show that capacity constraints and flow conservations are satisfied.

Capacity constraints.

- If $(u, v) \notin \mathcal{C}$ and $(v, u) \notin \mathcal{C}$,
  $f_1(u, v) = f(u, v)$, so
  $0 \le f_1(u, v) \le c(u, v)$.

- If $(u, v) \in \mathcal{C} \cap E$,
  $f_1(u, v) = f(u, v) + d' \ge f(u, v) \ge 0$ and
  $f_1(u, v) = f(u, v) + d' \le f(u, v) + c_f(u, v) = f(u, v) + c(u, v) - f(u, v)$
  $$= c(u, v)$$

- If $(v, u) \in \mathcal{C}$ and $(u, v) \in E$,
  $f_1(u, v) = f(u, v) - d' \le f(u, v) \le c(u, v)$ and
  $f_1(u, v) = f(u, v) - d' \ge f(u, v) - c_f(u, v) = f(u, v) - f(u, v) \ge 0$

Flow conservation. To show that the flow conservation conditions are satisfied, it is sufficient to prove that

$$\sum_{u \in V} f_1(u, v) - \sum_{w \in V} f_1(v, w) = \sum_{u \in V} f(u, v) - \sum_{w \in V} f(v, w).$$

Clearly, the last equation holds for $v \notin \mathcal{C}$, since in that case $f(u, v) = f_1(u, v)$ and $f(v, w) = f_1(v, w)$, for all u and w.

For $u \in \mathcal{C}$, since $\mathcal{C}$ is simple, u appears exactly once in $\mathcal{C}$. Let u_1 and w_1 are nodes such that $(u_1, v) \in \mathcal{C}$ and $(v, w_1) \in \mathcal{C}$. There are four possibilities:

- For $(u_1, v) \in E$ and $(v, w_1) \in E$

$$f_1(V, v) - f_1(v, V) = f_1\left(\frac{V}{\{u_1\}}, v\right) + f_1(u_1, v) - f_1\left(v, \frac{V}{\{w_1\}}\right) - f_1(v, w_1)$$

$$= f\left(\frac{V}{\{u_1\}}, v\right) + f(u_1, v) + d' - f\left(v, \frac{V}{\{w_1\}}\right) - (f(v, w_1) + d')$$

$$= f(V, v) - f(v, V).$$

- For $(u_1, v) \in E$ and $(w_1, v) \in E$

$$f_1(V, v) - f_1(v, V) = f_1\left(\frac{V}{\{u_1, w_1\}}, v\right) + f_1(u_1, v) + f_1(w_1, v) - f_1(v, V)$$

$$= f\left(\frac{V}{\{u_1, w_1\}}, v\right) + f(u_1, v) + d + f(w_1, v) - d' - f(v, V)$$

$$= f(V, v) - f(v, V)$$

- For $(v, u_1) \in E$ and $(v, w_1) \in E$

$$f_1(V,v) - f_1(v,V) = f_1(V,v) - f_1\left(v, \frac{V}{\{u_1, w_1\}}\right) - f_1(v,u_1) - f_1(v,w_1)$$

$$= f(V,v) - f\left(v, \frac{V}{\{u_1, w_1\}}\right) - (f(v,u_1) - d') - (f(v,w_1) + d')$$

$$= f(V,v) - f(v,V)$$

- For $(v, u_1) \in E$ and $(w_1, v) \in E$

$$f_1(V,v) - f_1(v,V) = f_1\left(\frac{V}{\{w_1\}}, v\right) + f_1(w_1, v) - f_1\left(v, \frac{V}{\{u_1\}}\right) - f_1(v, u_1)$$

$$= f\left(\frac{V}{\{w_1\}}, v\right) + f(w_1, v) - d' - f\left(v, \frac{V}{\{u_1\}}\right) - (f(v, u_1) - d')$$

$$= f(V,v) - f(v,V)$$

The proof is completed with

$$|f_1| = \sum_{u \in V} f_1(u,s) - \sum_{w \in V} f_1(s,w) = \sum_{u \in V} f(u,s) - \sum_{w \in V} f(s,w) = |f|.$$

Directly from the last Theorem we have the following corollary:

COROLLARY 1. Let $\vec{x}$ be a state vector for a two-terminal flow network $G(V, E, c)$ with source s and sink t such that the pseudoflow function $l_{\vec{x}}$ induced by $\vec{x}$ is a flow function with $|l_{\vec{x}}| = d$, and let $\vec{y}$ be an augmenting vector to level d' for a cycle $\mathcal{C}$ in the residual network $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$. Then $\vec{x} + \vec{y}$ is a state vector for $G(V, E, c)$ such that the pseudoflow function $l_{\vec{x}+\vec{y}}$ induced by $\vec{x} + \vec{y}$ is a flow function with $|l_{\vec{x}+\vec{v}}| = d$.

Lemma 1. Let f be a flow with $|f| = 0$ in the two-terminal network $G(V, E, c)$ such that there is an edge (u, v) for which $f(u, v) > 0$. Then the graph $G(V, E^f)$ induced by f contains a cycle.

Proof. Directly follows from two facts. The first one is that the indegree of each node in the graph $G(V, E^f)$ is strictly greater than 0 if and only if its outdegree is also strictly greater than 0. The second one is that there is a path passing through (u, v).

Lemma 2. Let $\vec{x}$ be a state vector for a two-terminal flow network $G(V, E, c)$ with source s and sink t such that $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = 0$. Then there are augmenting vectors $\vec{y}_k$, $k = 1, \ldots, r$ to levels d'_k for cycles $\mathcal{C}_k$ in $G(V, E, c)$, such that $\vec{x} = \sum_{k=1}^{r} \vec{y}_k$.

Proof. Since $l_{\vec{x}}$ is a flow function with $|l_{\vec{x}}| = 0$, from Lemma 1 it follows that $G(V, E^{l_{\vec{x}}})$ contains a cycle $\mathcal{C}_1$. Taking $d'_1 = \min\{x_i | e_i \in \mathcal{C}_1\}$ we may construct an augmenting vector $\vec{y}_1$ to level d'_1. The vector $\vec{z}_1 = \vec{x} - \vec{y}_1$ is a state vector for the two-terminal flow network $G(V, E, c)$ with $|l_{\vec{z}_1}| = 0$, too, and moreover, the graph $G(V, E^{l_{\vec{z}_1}})$ has at least one positive edge less than $G(V, E^{l_{\vec{x}}})$. If $G(V, E^{l_{\vec{z}_1}})$ has no edge (u, v) such that $l_{\vec{z}_1}(u, v) > 0$, we are fini-

shed. In opposite, we continue with this procedure of constructing an augmenting vector $\vec{y}_k$ to level d'_k for $G(V, E^{l_{\vec{z}_{k-1}}})$ and a vector $\vec{z}_k = \vec{x} - \vec{y}_k$, until $\vec{z}_k = \vec{0}$. The procedure will finish in at least $|E|$ steps. Each cycle $\mathcal{C}_k$ is a cycle in $G(V, E, c)$ since the graph $G(V, E^{l_{\vec{z}_{k-1}}})$ is a subgraph of G.

The next theorem shows that given a state vector $\vec{x}$, every other flow vector can be obtained by adding cycles from $G(V, E^{l_{\vec{x}}})$ to $\vec{x}$.

Theorem 3. Let $\vec{x}$ and $\vec{y}$ be two state vectors for a flow network $G(V, E, c)$ with source s and sink t, such that $l_{\vec{x}}$ and $l_{\vec{y}}$ are flow functions with $|l_{\vec{x}}| = |l_{\vec{v}}| = d$. Then there are augmenting vectors $\vec{y}_k$, $k = 1, \ldots, r$ of levels d'_k for cycles $\mathcal{C}_k$ in the residual network $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$ such that $\vec{y} = \vec{x} + \sum_{k=1}^{r} \vec{y}_k$.

Proof. The function $-l_{\vec{x}}$ is a flow function in $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$ from t to s with $|l_{\vec{v}}| = d$, and its residual graph is $G(V, E, c)$. Using this and the fact that $l_{\vec{v}}$ is a flow function from s to t in $G(V, E, c)$ with $|l_{\vec{v}}| = d$, we have that the function $l_{\vec{v}-\vec{x}}$ defined as $l_{\vec{v}-\vec{x}}(u, v) = l_{\vec{v}}(u, v) - l_{\vec{x}}(u, v)$ is a flow function in the residual graph $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$ with $|l_{\vec{v}-\vec{x}}(u, v)| = 0$. The state vector for $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$ induced by $l_{\vec{v}-\vec{x}}(u, v)$ is $\vec{y} - \vec{x}$. Since $|l_{\vec{v}-\vec{x}}(u, v)| = 0$, there are augmenting vectors $\vec{y}_k$, $k = 1, \ldots, r$ of levels d'_k for cycles $\mathcal{C}_k$ in $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$, such that $\vec{y} - \vec{x} = \sum_{k=1}^{r} \vec{y}_k$ (by Lemma 2).

Theorem 1, Theorem 2 and Theorem 3 are sublimated in the next theorem.

Theorem 4. Given a two-terminal flow network $G(V, E, c)$ with source s and sink t, let $\vec{x}$ be a *d-MP*. Then $\vec{y}$ is a *d-MP* if and only if $G(V, E^{l_{\vec{y}}})$ is an acyclic graph and there are augmenting vectors $\vec{y}_k$, $k = 1, \ldots, r$ of levels d'_k for cycles $\mathcal{C}_k$ in the residual network $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$, such that $\vec{y} = \vec{x} + \sum_{k=1}^{r} \vec{y}_k$.

5. Algorithm for Calculating all Minimal Path Vectors Using One Specified Path Vector and Cycles in their Corresponding Residual Network

Given a two-terminal flow network $G(V, E, c)$ with source s and sink t, let us suppose that the i-th component may operate in one of the levels from the set $\{0, 1, \ldots, M_i\}$. Assuming that the maximal flow of the network is M, the set $\{0, 1, \ldots, M\}$ is the set of all possible flows. Using the results from the previous section, we give an approach that can help us design an algorithm for computing all *d-MPs* for a given level $d \leq M$. The pseudocode for the main algorithm is the following:

Algorithm 2.

Step 1. Using Ford-Fulkerson algorithm, find one flow function f to level d.
Step 2. While $G(V, E^f)$ has a cycle, set $f = f_1$ using (4.1).
Step 3. Find all cycles in the f's residual graph, and construct their augmenting vectors $\vec{y}_k$.
Step 4. Check each $\vec{x} + \sum_{k=1}^{r} \vec{y}_k$ for cycle and print it if there is no cycle.

We can do some optimizations in order to minimize the repetition of *d-MPs* as well as to obtain vectors that are not *d-MPs*. Here, we do not discuss the strategy for enumeration of cycles.

This algorithm is useful for finding all *d-MP*s near the maximal one, since instead of adding *d* 1-*MP* vectors as in the algorithms proposed in (Mihova and Maksimova, 2011; Forghani-elahabad *et al.*, 2013; Lin *et al.*, 1995), here we obtain a candidate for *d-MP* only by one vector addition. The approach is especially useful for level *M*, since in this case the residual graph can be divided into a few connected components; each cycle must lie into exactly one of those components.

The residual graph obtained using the Ford-Fulkerson algorithm for maximum flow *M*, can be divided into strongly connected components, as explained in (Picard and Maurice, 1980; Bezakova and Friedlander, 2010). These components are used to obtain all minimum cuts. The edges lying on some minimum cut must be used with their full capacity. Moreover, each cycle must lie into exactly one of those components. This reduces the length, as well as the number of cycles. Furthermore, each *M-MP* vector can be obtained by joining the sub-vectors corresponding to each of the strongly connected components, as well as the subvector of the edges connecting a pair of strongly connected components.

Let $G(V, E, c)$ be a two-terminal network with maximum flow M and $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$ be the residual network for the *M-MP* $\vec{x}$. Assume that $G(V_k, E_k)$ are subgraphs of $\tilde{G}(V, E)$ such that V_k is the set of all nodes in the *k*-th connected component of $G_{l_{\vec{x}}}(V, E_{l_{\vec{x}}})$, and $E_k = \{(u, v) \mid u, v \in V_k\}$. Let $E' = \bigcap_k\{(u, v) \notin Vk\}$, i.e. the set of all arcs that are not in a connected component. Then each $e_i \in E'$ lies on some minimum cut and $l_{\vec{x}}(e_i) = x_i$. Moreover, for each other *M-MP* $\vec{y}$, $l_{\vec{v}}(e_i) = x_i$. On the other hand, the flow in $G(V_k, E_k)$ is equal to the flow out $G(V_k, E_k)$, i.e. $l_{\vec{v}}(V \backslash V_k, V) = l_{\vec{v}}(V, V \backslash V_k) = d_k$. Now we may propose an algorithm for *M-MP*.

Algorithm 3.

Step 1. Use Ford-Fulkerson algorithm to find a flow function *f* for maximum level *M*.

Step 2. Find strongly connected components of the residual graph and set $x_i = f(e_i)$ for all $e_i \in E'$.

Step 3. For each strongly connected component, use Algorithm 2 to compute the set of all subvectors D_k.

Step 4. Find all *M-MP*s $\vec{y}$ for which

$$y_i = \begin{cases} f(e_i), & e_i \in E' \\ l_{\vec{x}^k}(e_i), & e_i \in V_k \text{ and } \vec{x}^k \in D_k \end{cases}$$

The algorithm is illustrated in the following example.

EXAMPLE 1. Given the network in Fig. 2 a), the residual network for a maximal level 3 for the flow vector (2, 1, 0, 2, 1, 0, 2, 1) is shown in Fig. 2 b). The graph is divided into two subgraphs. Since the components in the cut must be used with their full capacity, each minimal 3-*MP* has the form $(x_1, x_2, x_3, 2, 1, x_6, x_7, x_8)$. There is only one augmenting cycle in the first strongly connected component, consisting of edges e_1, e_2 and e_3. This cycle is $(-1, 1, 1)$.

The second strongly connected component, consisting of edges e_6, e_7 and e_8 also has one augmenting cycle: $(1, -1, 1)$.

The subvectors corresponding to the first strongly connected component for which the
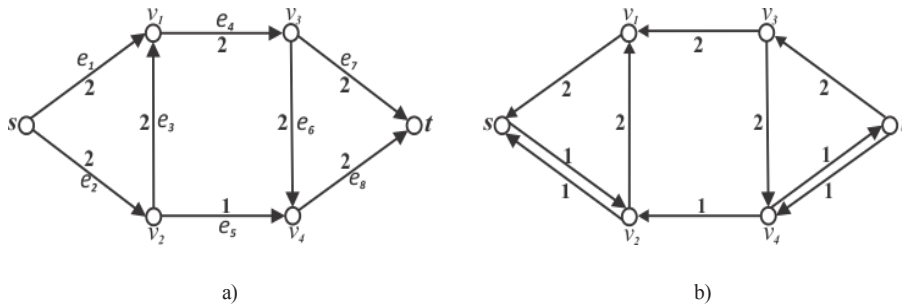
Fig. 2.: a) A flow network. b) Residual network for the flow vector (2, 1, 0, 2, 1, 0, 2, 1).

graph is acyclic are $\{(2, 1, 0), (1, 2, 1)\}$, $\{(2 - 1, 1 + 1, 0 + 1) = (1, 2, 1)\}$, while the subvectors corresponding to the second strictly connected component are $\{(0, 2, 1), (1, 1, 2)\}$. So all 3-*MP*s are all vectors obtained by joining the subvectors from this two sets together with the values of the components on the cut, i.e. $\{(2, 1, 0, 2, 1, 0, 2, 1), (1, 1, 2, 2, 1, 0, 2, 1), (2, 1, 0, 2, 1, 1, 1, 2), (1, 1, 2, 2, 1, 1, 1, 2)\}$.

6. Conclusion

Known algorithms for computing the set of all *d-MP*s commonly use network flows. In this paper we proved that a flow function corresponds to a *d-MP* if and only if the graph appropriate to that flow is acyclic. This property helped us to design an algorithm for calculating the set of all *d-MP*s, which is more efficient than the known algorithms for solving this problem. Moreover, by further analyses on the connection between two *d-MP*s, we proposed a strategy for calculating all *d-MP*s, given only one *d-MP* obtained using the Ford-Fulkerson algorithm. The proposed strategy is especially efficient for large levels. The strategy for enumeration of cycles addressed in Section 5 is one topic for our future work.

References

Aggarwal, K., Chopra, Y., Bajwa, J. (1982). Capacity consideration in reliability analysis of communication systems. *IEEE Transactions on Reliability*, 31(2), 177–180.

Ahuja, R.K., Orlin, J.B. (1993). A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 25, 89–98.

Bezakova, I., Friedlander, A.J. (2010). Counting minimum (s, t)-Cuts in weighted planar graphs in polynomial time. In: *Proceedings of 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23–27, 2010*.

Cormen, H.T., Leiserson, E.C., Rivest, R.L., Stein, C. (2009). *Introduction to Algorithms*, 3rd edition. The MIT Press

Erickson, J. (2009). *Algorithms*. University of Illinois at Urbana-Champaign.

Forghani-elahabad, M., Mahdavi-Amiri, N. (2013). A simple algorithm to find all minimal path vectors to demand level d in a stochastic-flow network. In: *5-th Iranian Conference on Applied Mathematics, September 2–4, 2013 Bu-Ali Sina University*.

Jane, C.C., Lin, J.S., Yuan, J. (1993). Reliability evaluation of a limited-flow network in terms of minimal cut sets. *IEEE Transactions on Reliability*, 42, 354–361.

Kamil, A. (2003). "*Graph Algorithms*", *CS61B*. University of California, Berkeley

Lin, J.S., Jane C.C., Yuan, J. (1995). On reliability evaluation of a capacitated – flow network in terms of minimal path sets. *Networks*, 3, 131–138.

Lin, Y.K. (2001). A simple algorithm for reliability evaluation of a stochastic – flow network with node failure. *Computers and Operations Research*, 28, 1277–1285.

Mihova, M., Maksimova, N. (2011). Estimation of minimal path vectors of multi state two terminal networks with cycles control. *Mathematica Balkanica*, 25(4), 437–447.

Picard, J.C., Maurice, Q. (1980). On the structure of all minimum cuts in a network and application. *Mathematical Programming Study*, 13, 8–16.

Provan, J.S., Balls, M.O. (1983). The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4), 777–788.

Ramirez-Marquez, J.E, Coit, D. (2003). Alternative approach for analyzing multistate network reliability. In: *Proceedings of the Industrial Engineering Research Conference (IERC), Portland, OR, May 2003*.

Ramirez-Marquez, J.E., Coit, D., Tortorella, M., (2003). Multi-statetwo-terminal reliability: a generalized cut-set approach. *Rutgers University IE Working Paper*, 03–135.

Wilson, A., Limnios, N., Keller-Mc-Nulty, S., Armijo, Y. (2005). *Modern Statistical and Mathematical Methods in Reliability*. World Scientific Publishing, Series on Quality, Reliability, and Engineering Statistics, vol. 10.

M. Mihova is an associate professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. She is a member of the board of the Computer Society of Macedonia. Her research interest is in the field of applied mathematics, more specifically applied probability and statistics, with focus on mathematical models in reliability, especially reliability of multi-state systems.

N. Stojkovikj is a teaching and research assistant of the Faculty of Computer Science, University "Goce Delchev" in Shtip. Currently, she is preparing her PhD thesis on the topic of reliability of multi-state two-terminal network.

M. Jovanov is an ass. professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is the President of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and Olympiads in informatics since 2001. He has been a team leader for the Macedonian team since 2006.

E. Stankov is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is a member of the Executive Board of the Computer Society of Macedonia and has actively participated in the organization and realization of the Macedonian national competitions and Olympiads in informatics since 2009. Currently he is PhD student at the Faculty of Computer Science and Engineering.

Tasks in Informatics with Pre-Existing Algorithms

Pavel S. PANKOV[1], Kirill A. BARYSHNIKOV[2]

[1] *International University of Kyrgyzstan*
[2] *LLC "Finance Soft", Bishkek, Kyrgyzstan*
e-mail: pps50@rambler.ru, baryshnikov.kirill@gmail.com

Abstract. Almost all tasks at informatics olympiads demand developing an algorithm. In this paper, we draw attention to tasks where some algorithm already exists. We propose classification of types of such tasks and give corresponding examples; survey both known and falling under our classification, and new (as we hope) types of tasks; consider the crucial case of a task with a known but too slow algorithm (formally presenting various known types of tasks).

We hope that some classes of such tasks would enlarge scope of tasks for use in olympiads.

Keywords: olympiads in informatics, tasks, existing algorithms, black box

1. Survey of Types of Tasks and the Aim of Paper

There is an up-to-date tendency to design various operations in informatics olympiad tasks as separate algorithms (procedures). For instance, the contestant's program must begin with the function

```
function Start (k) : Integer;
```

instead of the common

```
readln (k);
```

continue with functions of type

```
function Arr (i, k: integer): integer
```

for various integer *i* instead of the common

```
for i:=1 to k do
begin
  readln (arr[i]);
end;
```

and end with the procedure

```
procedure End (f: integer);
```

instead of producing the answer of task with

```
writeln (f);
```

This tendency can be explained as arising from numerous mistakes in input and output formats in preceding informatics olympiads of all levels.

In this paper we consider pre-existing algorithms (procedures) involved in the task due to its essence.

Such tasks can be classified firstly as follows. The algorithm itself is either unknown or is known for the contestant.

For unknown algorithms the contestant's task is either to restore the algorithm by its results or to use the algorithm for any purpose.

For known algorithms, the contestant's task might be as following:

- To implement a better equivalent algorithm.
- To implement any other algorithm related to the given one.
- To solve any other task related to the algorithm.

We survey both known and falling under our classification, and new (as we hope) types of tasks.

Section 2 considers various types of tasks with existing but unknown algorithms to the contestant. The contestant is either:

- To restore the algorithm exactly by its results or to implement an effective algorithm yielding same results („Black box" task).
- To use (or to struggle versus) the algorithm (in the form of an executable file or external library or table of results) for various purposes.

Tasks sufficiently involving an external library are called "reactive" (Opmanis, 2006).

Section 3 deals with the crucial case of acceleration of a given algorithm which is too slow. In our opinion, there arises the principal problem for all informatics similar to Church's thesis for computer science at all: what tasks in informatics can be presented in this form, or in another words: is an algorithmic language sufficient to state tasks in informatics?

Section 4 considers the problem to extract information about a hidden object only by a known algorithm (as a turbid or narrow window).

Section 5 considers tasks on the analysis of known algorithms including ones with arbitrary wandering.

2. Tasks with Unknown Algorithms

2.1. *Tasks of „Black Box" Type*

Such task may contain the following sentences:

- *Given an unknown algorithm in the form of an executable file, external library or table of results and/or some information about it.*
- *Restore the algorithm by your handle calls*. The number of calls might be bounded.

Or:

- *Write a program what will restore the algorithm by calls within given time"* or *"... and in bounded number of calls"*;
- *Write an effective program yielding same results.*

The simplest example of this type of task with unknown numbers bases requiring knowledge of the bisection method.

Task 1. Given the algorithm with unknown number *K*:
Algorithm 1.

```
writeln ('input integer number X in 0..10^10');
readln (X);
if X > K then writeln ('Greater') else writeln ('Leq') end;
```

and information *"K is an integer in 0..10^10."*

The task might be as follows:

- *Find K.*
- *Find K in less than 20 calls.*

By our experience, children of 5–7 years old solve this task in the range 0..1000 with great pleasure.

The programming version of this task:

There is the following function with unknown integer *K*:

```
function A (X: integer): boolean
begin
   if (X > K) return true;
   return false;
end;
```

Write a program that calls the function *A* to find the number *K*, calling the function *A* as few times as possible.

The following task is slightly more complex, and is also of interest for children.

Task 2. Given the algorithm with unknown numbers *U, V.* The user's aim is declared in itself:
Algorithm 2.

```
writeln ('Input zero and zero');
readln (X, Y);
D:=U^2+V^2;
writeln ('You are to receive HOT!');
repeat
   writeln ('Input integer X and Y');
   readln (X, Y);
   D1:=(X−U)^2+(Y−V)^2;
   if D1=0 then writeln ('HOT');
   if D1<D then writeln ('WARMER');
   if D1=D then writeln ('EQUAL');
   if D1>D then writeln ('COOLER');
   D:=D1;
until (D1 = 0);
end;
```

and information *"U and V are non-zero integers in – 10000..10000."*

An example with unknown variables:

Task 3. Given the algorithm *B* with unknown variables:

Algorithm 3.

```
writeln ('detect values of U, V, W in the listing');
For M:=1 to 100 do
begin
writeln ('input integer numbers X, Y, Z in 0..100');
readln (X, Y, Z);
  if X > U then Y:=U;
  if Y > V then Z:=V;
  if Z > W then X:=W;
writeln (X, Y, Z);
end;
writeln ('You have exceeded limit of requests');
end;
```

and information "*U, V, W are letters of the set {X,Y,Z}.*"

The user is to guess one of 27 arrangements with repetitions of three by three, by means of fewer than 100 calls of the algorithm.

Transformations of such tasks into programming versions are obvious.

Since algorithms within announced classes can be written in different equivalent forms ("if X>Y …" or "if Y<X …"), the tasks in situations mentioned in 2.1 can be different:

"After experimenting with the unknown algorithm write a program giving the same results" and a set of tests covering all branches of the algorithm.

2.2. *Games Versus Unknown Algorithms*

Such a type of tasks is well-known and we do not give references.

General task 4. Rules of a bilateral game are described and

(a) the jury's exe-file exists

or

(b) other contestants' exe-files will be built.

"Write a program which will play

(a) *versus the jury's program*

or

(b) *versus other participants' programs.*

In (a) case the aims may vary: to win; to withstand no fewer than a given number of moves; to obtain as many points as possible etc.

In (b) case to equalize all participants the program is to play versus each other one both as the first player ("Whites") and as the second one ("Blacks").

2.3. *Optimal Imitation of Unknown Algorithms*

Task 5 *"Function of functions"* ("Ugale", 2005)

Function f(x) is defined for all integers from 1 to 2000000000 and function values are positive integers. Given the table of function values for 100 hundred argument values: …

*Your task is to write **as short as** possible a program, which implements this function for all x values given in table above. Executing time for one particular test case must not exceed one second. Programs with lower **amount of source code in bytes** will get higher scores*.

(In such tasks pre-existence of a short algorithm yielding given results is meant).

3. Tasks to Accelerate Given Algorithm

These tasks have the following scheme: given an algorithm (mainly, too slow, of type of full sorting). Write a program yielding the same result in appropriate time.

We propose to write given algorithms in non-formal but clear "semi-algorithmic" language, as above. For brevity, simple non-algorithmic operations also may be involved.

There arises the principal problem: what tasks in informatics can be presented in such form? By our opinion, these types are following.

General task 6. (Find anything in a priori bounded set B).

Algorithm 4.

```
Given the set B, the large number M; |B|≤M; the boolean
function A;
foreach X in B do
begin
   if A(X):=true then writeln (X);
end;
```

The following tasks may be presented in such form: tasks on optimization, on solving equations, on finding intervals for solutions of equations (Pankov, 2013).

It's easy to add the following condition:

```
if (A(X):=false) foreach X∈B
then writeln ('No').
```

Remark 1. In the last IOIs the jury used to add conditions of the following type:
In not less than 50% tests $|B|≤M*10^{-3}$.

General task 7. (Count anything in a priori bounded set B).

Algorithm 5.

```
Given the set B, the large number M; |B|≤M; the boolean
function A:
 S:=0;
```

```
foreach X in B do
begin
   if A(X):=true then S:=S+1;
end;
writeln (S);
end;
```

Remark 2. Since IOI'2008 formulations of the following type are proposed:

```
    writeln (S mod 1000).
```

It avoids necessarily using long numbers and leaves treating such numbers or circumvention of them to the contestant.

General task 8. (Find anything in a priori unbounded set *B*).
Algorithm 6.

```
Given the countable set B and the Boolean function A;
foreach X in B do
begin
   if A(X):=true then writeln (X);
end;
```

For example,

Task 9 (confutation of Euler's hypothesis).
Algorithm 7.

```
U:=0;
Repeat
U:=U+1;
for X:=1 to U do for Y:=1 to U do
for Z:=1 to U do for T:=1 to U do
  begin
     if (X^5+Y^5+Z^5+T^5=U^5) then
     begin
        equal:=true; X1:=X; Y1:=Y; Z1:=Z; T1:=T
        goto M1:
     end;
end; end; end; end;
until equal;
M1:
writeln (X1, Y1, Z1, T1, U).
```

Remark 3. The condition
if there is no such numbers X1, Y1, Z1, T1, for any U then output 'No'
also can be added but it demands special proof here. For example:

Task 10. (Pankov, 2013). Find an interval of width 1 containing a solution of the equation
$F(X):=X^4+A[3]*X^3+A[2]*X^2+A[1]*X+A[0]=0$ or output "No", if there is no solution.

An a priori boundary is found by

Algorithm 8.

```
for X:=1,2,3… do
begin
   if (X^4−|A[3]|*X^3−|A[2]|*X^2−|A[1]|*X−|A[0]|)>0 then
   begin
      XM:=X;
      break;
   end;
end;
S:=false;
for X:=−XM to XM−1do
begin
   if (F(X)*F(X+1)≤0) then
   begin
      writeln (X, X+1);
      S:=true;
      break;
   end;
end;
if (S:=false) then writeln ('No'); end.
```

4. Tasks on Restricted Access to Object by Known Algorithm

Such mathematical tasks were initiated by (Tikhonov, 1943). There is an unknown object (underground ore body) and we can obtain only sums (integrals) of its components by means of geological methods (further: or by X-rays, i.e. tomography). Detect its shape with some error.

Such tasks in informatics can be described as applying a known algorithm to an unknown object.

We recall two examples.

General Task 11 (by the task *Giza*; see, for example, Burton *et al.*, 2008). There is an unknown ("secret") double array, given cross-section total sums of its elements (or: an algorithm counting such sums). Restore the array (or: some elements of it).

We propose an example of Tikhonov's type:

Task 12. There is a hidden double array *A[0..N, 0..N]* of numbers connected as topological torus (Pankov, 2008) defined as follows.

```
For all Y = 0..N Points (0, Y) and (N, Y) are glued and
For all X = 0..N Points (X, 0) and (X, N) are glued
```

The algorithm *S* yields the sum of each element and its eight (twenty-four …) neighbors. Find *max {A[X,Y]: X = 0..N, Y = 0..N}*.

Task 13 "Mean" (IOI'2000). There is a hidden array *A[0..N]* of different numbers and algorithm *M* which reworks each three indexes into such one that the element with such index is between two other elements. By means of using this algorithm detect indexes of the greatest and the least objects.

5. Tasks on Analysis of Given Algorithms

Such type of tasks was distinguished (Opmanis, 2009).

5.1. *Tasks on Restoration of Input Data*

We cite

Task 14 (Aivars Žogla), see (Opmanis, 2009). Given text of an algorithm that sorts the elements of number array *A[0..n−1]* from position low till position high in non-decreasing order. By taking *low = 0* and *high = n−1*, the entire array will be sorted. Your task is to find an array containing each of the numbers from *1* to *20* exactly once, for which sorting by calling procedure *sort(A,0,19),* uses the maximum number of array element comparison operations.

On the base of this task we propose:

General task 15 "Inverse algorithm". Given text of an algorithm and output data. Find input data yielding this output data (or: with any optimization, as in Task 6).

5.2. *Tasks on Arbitrary Inputs to Known Algorithm*

General task 16. Given an interactive algorithm containing the initial situation, possible inputs and final situation. Some inputs change the situation. Achieving the final situation ends the algorithm with corresponding results.

The sense of the task is the following: the user is to guess the aim of the algorithm and write an improved algorithm (without inputs!) or calculate the optimal answer.

As well as in section 3, we state that most of tasks in informatics can be reworked into this form.

We offered such tasks on national olympiads but did not meet mention on this genre.

Task 17 (this mathematical task was proposed Pankov, 1970).
Given:
Algorithm 9.

```
P:=1;
readln (N);
Repeat
  writeln ('input natural number');
  readln (X);
  if (X<=N) then
  begin
```

```
      P:=P*X;
      N:=N−X;
   end;
   if (N:=0) then
   begin
      writeln (P);
      end;
   end;
 Until (1 = 1); end.
```

Detect the greatest possible number *P* which can be obtained by this algorithm (or, for large *N*): calculate *P mod 1000*.

Remark 4.

- For *N* < 100 this is an easy exercise in dynamical programming.
- For *N* < 10000 there will be problems with too large *P*.
- For *N* < 10^100 the task is to be solved mathematically (the result is easy to be guessed by the results obtained by dynamical programming for *N* = 10, 11, 12,…).

Examples which also may be called "Arbitrary wandering":

Task 18. Computer models of verbs "pass" and "return" (Pankov *et al.*, 2009).
Comment. "Gate" is the couple of points (20, 39) and (20, 41).
Given:
Algorithm 10.

```
 X:=0; Y:=0; X1:=0; Y1:=1; L:=0; P:=false;
 Repeat
   writeln ('input one of (R,S,L)');
   readln (M);
   L:=L+1;
   if (M:=R) then
   begin
     X2:=X1+(Y1−Y);
     Y2:=Y1−(X1−X);
   end;
   if (M:=S) then
   begin
     X2:=X1+(X1−X);
     Y2:=Y1+(Y1−Y);
   end;
   if (M:=L) then
   begin
     X2:=X1−(Y1−Y);
     Y2:=Y1+(X1−X);
   end;
   if (X2:=20 and (Y2:=39 or Y2:=41)) then L:=L−1
   else
```

```
  begin
    if (X1:=20 and Y1:=40) then P:=true;
    if (X1=0 and Y1=0 and P:=true) then
    begin
       writeln ('Congratulations! You have PASSED the
       gates and RETURNED');
       writeln (L);
       break;
    end;
  end;
until (1=1); end.
```

Detect the least possible number *L* which can be obtained by this algorithm.

Task 19 (classical).

Algorithm 11.

```
Given integer number XM≠0; natural numbers F {jump
forward}; B {jump backward};
X:=0;
Repeat
  writeln ('input one of (−,+)');
  readln (S);
  L:=L+1;
  if (S:='−') then X:=X−B;
  if (S:='+') then X:=X+F;
    if (X:=XM) then
  begin
     writeln (L);
     end;
  end;
until (1=1); end.
```

Detect the least possible number *L* which can be obtained by this algorithm or "No" if it is not possible.

We did not meet the following type of tasks on games (pendant to 2.2).

General task 20. Rules of a bilateral game are described and the jury's algorithm is given both as listing and as an executable file.

Examine the listing and write a program which will successfully play versus the same algorithm as an executable file.

6. Conclusion

There exist special programs "optimizing" programs (for instance, withdrawing from cycles at programming level, organizing some operations within registers, not with memory

at microprogramming level etc.). The last sections of this paper are devoted to "optimizing" algorithms at "intellectual" level.

We hope that types of tasks considered in this paper would both enlarge the scope of tasks in informatics olympiads and contribute to the general problem of computer science in practice: relations between algorithm-statement-of-task and algorithm-solution-of-task.

References

Tikhonov, A.N. (1943). On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39 (5), 195–198.

Pankov, P.S. (1970). A problem. *Mathematics in School,* 5. (In Russian)

"Ugale" (2005). *Team Competitions in Mathematics and Informatics "Ugale"*. (In Latvian).
http://www.uvsk.apollo.lv/p113.htm

Opmanis, M. (2006). Some ways to improve olympiads in informatics. *Informatics in Education*, 5(1), 113–124.

Burton, B.A., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, 2, 16–36.

Pankov, P.S. (2008). Naturalness in tasks for olympiads in informatics. *Olympiads in Informatics*, 2, 115–121.

Opmanis, M. (2009). Team competition in mathematics and informatics "Ugale" – finding new task types. *Olympiads in Informatics,* 3, 80–100.

Pankov, P., Dolmatova, P. (2009). Geometrical problems in interactive computer presentation of notions of natural languages. In: *Actual Problems of Control Theory, Topology and Operator Equations. International Jubilee Conference at the Kyrgyz-Russian Slavic University.* Shaker Verlag, Aachen, 85–87.

Pankov, P.S., Baryshnikov, K.A. (2012). Tasks of a priori unbounded complexity. *Olympiads in Informatics,* 6, 110–114.

Pankov, P.S. (2013). Tasks in informatics of continuous content. *Olympiads in Informatics,* 7, 101–112.

P.S. Pankov (1950), doctor of physical-math. sciences, prof., corr. member of Kyrgyzstani National Academy of Sciences (KR NAS), is the chairman of jury of Bishkek City OIs since 1985, of Republican OIs since 1987, the leader of Kyrgyzstani teams at IOIs since 2002. Graduated from the Kyrgyz State University in 1969, is a main research worker of Institute of theoretical and applied mathematics of KR NAS, a professor of the International University of Kyrgyzstan.
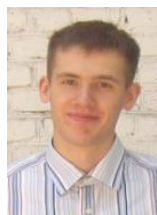
K.A. Baryshnikov (1985), LLC "Finance Soft", Bishkek, Kyrgyzstan. Participated in IOI'2002, in training the Kyrgyzstani teams for IOIs in forthcoming years. Graduated from the Kyrgyz-Russian Slavic University in 2007.

More Algorithms without Programming

Jakub RADOSZEWSKI

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
e-mail: jrad@mimuw.edu.pl

Abstract. It has been 4 years since the publication of "Algorithms without Programming" (*Olympiads in Informatics*, 4, 2010). In the past four years the set of algorithmic riddles proposed in that paper has been used on different occasions, including an olympic quiz and classes with high school and gymnasium students. It turned out that some of the problems fit better than the others, moreover the set has been extended with several more examples. We present here what we have learned in this period about teaching algorithmic and mathematical thinking without a computer.

Keywords: programming contests, non-programming tasks.

1 Introduction

In the first paper on algorithms without programming (Kubica and Radoszewski, 2010) we have presented several examples of pen-and-paper tasks, formulated using basic notions of combinatorics, that encapsulate a number of crucial concepts of algorithm design. The tasks could be solved using trial and error, however, such methods are quite painful to execute "by hand" (and such solutions are error-prone). This task set was designed for individual study for students primarily interested in mathematics and was originally published in a Polish popular monthly.

Since 2010 the task set has been used on several different occasions. First it formed the basis of an olympic quiz at an open event called the First Polish Informatics Camp held for high-school and gymnasium students during ACM International Collegiate Programming Contest Finals 2012 in Warsaw, Poland. The tasks were distributed on small pieces of paper and the solutions were checked automatically using a computer program. For this purpose the tasks had to be extended with several subtasks, so that the students would not exchange their answers too easily.

Next the task set was introduced to basic and intermediate level olympic programming camps, organized mainly for mathematically talented high-school and gymnasium students at the University of Warsaw, Poland, and to a series of popular introductory lectures to computer science. Problem solving sessions were a form of break from regular programming sessions taking place in computer rooms. They also turned out to be a networking and, at the same time, a competitive event for students. However, their main purpose was to show that computer science is not only about tackling technical is-

sues and to enhance students' problem-solving skills. The task set required a significant change to fit well for this application. Now we selected tasks in which the computations were kept at a strict minimum. The solution to each task required basically a single idea and at the same time could be obtained by students within just a few minutes.

Below we list other works, devoted mainly to specific competitions which contain related examples of non-programming tasks used in informatics training. It is worth noticing that, contrary to pen-and-paper competitions, for the classroom usage there is no need to keep inventing new tasks once every period of time. The ability to solve such tasks is just a bridge to solving regular programming tasks focused on algorithms, not an end in itself. In particular, whenever possible, we add a link to a programming task that is the base of the particular non-programming task, so that the students see a direct connection between the two types of tasks.

The "Beaver" contest consists of short tasks (each to be solved within 3 minutes) on informatics and computer literacy. The key features of the tasks: "attraction, invention, tricks, surprise", "thinking, not guessing answers", and "independence from any curriculum", see Dagienė (2006), Dagienė and Futschek (2008), are the same as in the task set that we propose. The apparent difference is that the tasks at the Beaver contest are to be solved on a computer, whereas we generally aim at pen-and-paper competition. We also aim only at a subset of the list of topics from the Beaver competition which generally fits within the following categories mentioned by Dagienė and Futschek (2008): *Structures, patterns and arrangements* (combinatorics and discrete structures) and *Puzzles* (logical puzzles).

There are several national informatics competitions and olympiads which consist of pen-and-paper tasks of similar flavour in at least one stage. This includes the South African Computer Olympiad, SACO (Merry *et al.*, 2008), Australian Informatics Competition, AIC (Burton, 2010) and Dutch Olympiad in Informatics (van der Vegt, 2012). The works of Burton (2010) and van der Vegt (2012) contain several examples of such tasks. The former provides key characteristics which apply also to our task set, namely "puzzle-based setting" and "no assumed knowledge". Again, the scope of AIC is actually wider than what we consider; our tasks fit in the *Algorithmic tasks* category mentioned by Burton (2010). The AIC contains also three-stage tasks, an idea similar to the idea of subtasks that we consider (with a slightly different motivation).

Tasks of an algorithmic flavour with purely mathematical statements can also be found in Ugāle team competition (Opmanis, 2009) and Project Euler (projecteuler.net). There is a substantial difference between the format of the two and the format of our task set: the former can be solved by the students with the aid of a computer. A more comprehensive description of tasks types which go beyond simple programming tasks can be found in Hakulinen (2011) and Forišek (2013). Yet another approach to introduction of elements of computer science in an attractive form can be found in the books of Bell *et al.* (1998), Vöcking *et al.* (2011), Forišek and Steinová (2013).

We present our task set with each section devoted to one task. Each task contains 5 subtasks that are normally distributed among students. The task descriptions are followed by a solution description and some methodological comments on the usual performance of students on the particular task. Two of the tasks are tasks from Kubica and

Radoszewski (2010) properly adapted to the new setting. Afterwards in a Conclusions section we list an updated list of conditions from Kubica and Radoszewski (2010) that a task from the set should satisfy.

2. Polygon

2.1. *Problem*

I own a parcel of a polygonal shape. It has 10 sides and its area equals 23 (that is, it contains 23 unit squares). The corners of the parcel are: B2, B5, E5, E4, F4, F8, I8, I3, C3, C2, see figure.
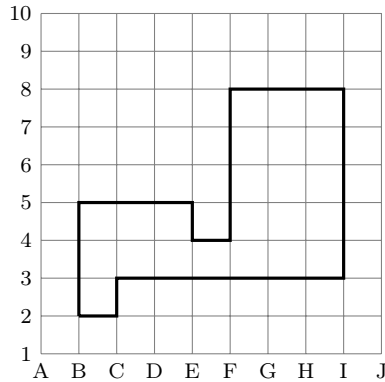
Can you draw a polygon with:

  (a) 6 sides and area 6?
  (b) 8 sides and area 8?
  (c) 10 sides and area 10?
  (d) 12 sides and area 12?
  (e) 14 sides and area 14?

Or a polygon with 13 sides and area 13? All sides of the polygon must be contained in grid lines. Each two consecutive sides must be perpendicular.

This riddle is based on a task from Algorithmic Engagements 2011
(http://main.edu.pl/en/archive/pa/2011/geo).

2.2. *Solution*

For all subtasks the answer is positive and there are numerous different solutions.
  Below we draw two different parcels for subtasks (d) 12 and (e) 14:
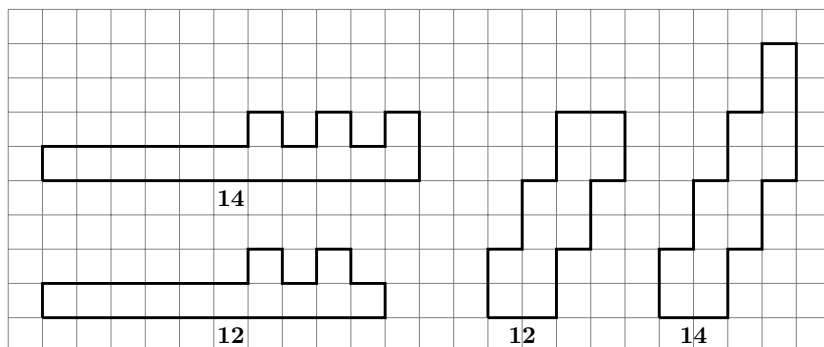
There is no polygon with 13 sides and area 13. Actually there is no polygon with an odd number of sides in which every two consecutive sides are perpendicular. This is because all even-numbered sides must be vertical and all odd-numbered sides must be horizontal (or *vice versa*).

2.3. *Methodological Comments*

At first the students usually solve the subtasks using trial and error. After all the subtasks have been solved, the students can be asked to solve a sixth subtask, with 20 sides and area 20, and then encouraged to provide a general structure of a polygon for any given even number of sides and unit squares. Construction of such scheme requires elements of algorithmic thinking.

On the other hand, the students usually put some effort in trying to draw a polygon with an odd number of sides and area. They are curious why such a polygon does not exist. Sometimes they attempt their own intuitive arguments. The main difficulty in providing such an argument is to note that the odd number of sides is the sole reason why no such polygon exists.

3. Palindromes

3.1. *Problem*

A palindrome is a word which is the same when read from left to right and from right to left. Examples of palindromes are `noon`, `radar`. Some words may be divided into **even** length palindromes (therefore, for example, the palindrome `noon` could be used in a division, while `radar` could not). For example, the word `aabbaaabbaaa` can be divided into 3 even length palindromes:

$$\text{aabbaaabbaaa} = \text{aabbaa} \mid \text{abba} \mid \text{aa}$$

What is the smallest number of even length palindromes in a division of the word:

(a)   `aabbaabaabaabbbb`?

(b)   `aaaaabbaaabbabbabbbb`?

(c)   `baabbbbaabaabaabbbbb`?

(d)   `aabbaabaabaabbbbbbbb`?

(e)   `aaaabbaaabbabbaabbbb`?

This riddle is based on a task from 2nd Polish Olympiad in Informatics
(`http://main.edu.pl/en/archive/oi/2/pal`).

3.2. *Solution*

Below are the optimal divisions of the words from all subtasks:

(a)   `aabbaabaabaabbbb = aa|bbaabaabaabb|bb`

(b)   `aaaaabbaaabbabbabbbb = aa|aaabbaaa|bbabbabb|bb`

(c)   `baabbbbaabaabaabbbbb = baab|bbbaabaabaabbb|bb`

(d)   `aabbaabaabaabbbbbbbb = aa|bbaabaabaabb|bbbbbb`

(e)   `aaaabbaaabbabbaabbbb = aa|aabbaa|abba|bbaabb|bb`

There is something special about the words selected for these subtasks. They share the following property: if we try to select to the division the longest even palindrome which is a prefix of the word, it cannot be extended to an optimal solution. The same applies if we take the longest such suffix to the division.

3.3. *Methodological Comments*

This task is to be solved using trial and error. Most commonly the students come up with suboptimal solutions at first. What they learn from this task is that greedy is not necessarily optimal. A correct algorithmic approach to this problem is via dynamic programming (which the students do not need to use here). Interestingly enough, if we were to divide the initial word into the *maximum* number of even length palindromes, a greedy approach would work!

4. Wagons

4.1. *Problem*

Let us consider a track siding: an incoming track A, an exit track B and two auxiliary tracks 1, 2. Track A contains 7 wagons numbered 1 through 7. The wagons arrive on track A in the following order:
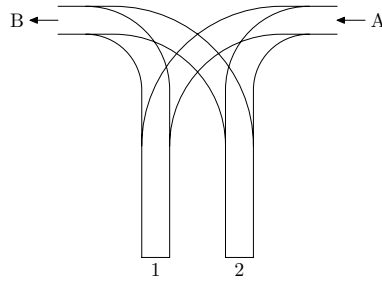
(a)   6, 3, 2, 5, 1, 7, 4

(b)   5, 2, 4, 1, 6, 3, 7

(c)   6, 2, 5, 1, 3, 7, 4

(d)   7, 5, 2, 4, 1, 6, 3

(e)   6, 1, 3, 5, 2, 7, 4

The wagons are to exit via track B in increasing order of numbers 1, . . . , 7. Each wagon is to be moved from track A to one of the tracks 1, 2 and then to track B exactly once. There can be arbitrarily many wagons on each track at the same time.

Can this task be completed successfully? If so, to which auxiliary track should the subsequent wagons be moved?

(For example, if the initial order of wagons was 5, 2, 6, 4, 1, 3, 7 then the answer would be positive. The wagons could be moved using the auxiliary tracks 1, 1, 2, 2, 1, 1, 1 respectively.)

This riddle is based on a task from 17th Polish Olympiad in Informatics
(http://main.edu.pl/en/archive/oi/17/kol).

4.2. *Solution*

The subtasks are constructed in such way that a solution exists (however, there exist permutations of wagons 1 , … , 7 which cannot be sorted using two auxiliary tracks).

Note that at all moments of time all wagons on each auxiliary track must be sorted from the smallest to the highest number (from the beginning of the track). There is a natural greedy approach to this problem: under the aforementioned order-condition, always put the next wagon on the auxiliary track where the first wagon has the smallest number. However, for all given subtasks this strategy *fails*. For example, in subtask (a) after processing the first 5 wagons we would obtain wagons 1, 2, 3, 6 on one auxiliary track and wagon 5 on the other auxiliary track. Next we move wagons 1, 2, 3 to track B in this order and there is no auxiliary track to move wagon 7 to.

Using trial and error, for example, the following solutions can be obtained:

(a)   1, 2, 2, 1, 1, 2, 1

(b)   1, 2, 1, 1, 2, 1, 1

 (c) 1, 2, 1, 1, 1, 2, 1

 (d) 1, 1, 2, 1, 1, 2, 1

 (e) 1, 1, 2, 1, 1, 2, 1

4.3. *Methodological Comments*

The purpose of this task is similar to the task *Palindromes*. The students are not supposed to come up with any particular algorithm but to see that greedy does not work. The original task from the Polish Olympiad in Informatics was solved using a smart reduction to the problem of two-colouring of a particular graph (which seems too complex to be presented at this level).

5. Coins

5.1. *Problem*

Assume you were given coins with the values being powers of two:

$$1, 2, 4, 8, 16, 32, \ldots$$

and you had exactly one coin with each value. Then you could pay any (positive integer) amount of money using these coins (for example, $45 = 1 + 4 + 8 + 32$).

 What is the smallest (positive integer) amount of money, which **cannot** be paid using the following set of coins:

 (a) 6, 3, 2, 10, 21, 46, 1, 48?

 (b) 12, 7, 3, 2, 31, 27, 28, 1?

 (c) 27, 56, 1, 13, 60, 4, 7, 2?

 (d) 44, 39, 5, 1, 9, 1, 18, 2?

 (e) 62, 3, 26, 12, 53, 2, 1, 7?

 Keep in mind that you have exactly one coin of each kind!

5.2. *Solution*

The same task in a traditional setting has already been presented in Kubica and Radoszewski (2010). We briefly recall the solution here.

 The first step is to sort the sequence of coins in non-decreasing order; for the sequence in the first subtask we have:

 (a) 1, 2, 3, 6, 10, 21, 46, 48

Using the first two coins we can pay any integer amount from [1, 3]. With the first three, we can pay any amount from [1, 6]. By including the coin 6, we can pay any amount from [1, 12], with the additional coin 10 we can pay any amount from [1, 22], and with the additional coin 21 we can pay any amount from [1, 43]. The next coin is 46, which concludes that the amount 44 cannot be paid. The answers to the remaining subtasks are as follows: (b) 26, (c) 55, (d) 37, (e) 52.

5.3. *Methodological Comments*

Students usually find this task quite hard. This is because trial and error requires quite a lot of effort to obtain the solution and often leads to mistakes in the answer. Eventually students manage to solve the majority of the subtasks.

This task encapsulates several algorithmic concepts: sorting, elements of dynamic programming approach and binary number system (in the task statement).

6. Triangle

6.1. *Problem*

We are given 11 line segments of the following lengths:

(a)   1, 49, 11, 3, 338, 128, 30, 78, 17, 208, 6

(b)   103, 1, 15, 8, 167, 271, 5, 3, 64, 25, 38

(c)   94, 154, 5, 8, 248, 35, 2, 1, 58, 23, 13

(d)   87, 3, 20, 12, 141, 4, 228, 52, 1, 33, 8

(e)   108, 25, 178, 15, 3, 42, 9, 68, 1, 4, 286

Is it possible to pick three different line segments from the set and use them to obtain a triangle with positive area? If so, which three segments to choose?

This riddle is based on a task from 2nd Polish Olympiad in Informatics (http://main.edu.pl/en/archive/oi/2/tro).

6.2. *Solution*

Recall that a triangle with positive area can be built using segments of length a, b, c if the longest one (say c) is strictly shorter than the total length of the smaller two (that is, $a + b > c$).

The answers to all subtasks are positive. In each case there exist unique three segments that can be used to form a triangle:

(a)   30, 49, 78

(b)   15, 25, 38

(c)   13, 23, 35

(d)   20, 33, 52

(e)   42, 68, 108

It is easy to see that if a solution exists then there is also a solution formed by three consecutive segments in ascending order of lengths. For example, in the first subtask we have the following order:

(a)   1, 3, 6, 11, 17, 30, 49, 78, . . .

It suffices to stop at 78, since we have already discovered the requested triad of segments: 30 + 49 > 78.

6.3. *Methodological Comments*

I usually present this task just after the task *Coins*. Thanks to that several students discover fast that having the line segments in ascending order of lengths simplifies the solution considerably.

At the conclusion of the task one can try to argue why exactly it suffices to choose three consecutive segments in the sorted order. There is a simple greedy argument: we start with any three segments in the sorted order and show that by increasing the lengths of the smaller two (without exceeding the length of the longest one) we only raise the odds of obtaining a solution.

Young students usually do not have the habit of proving the correctness of their ideas. In this task set we try to create this habit threefold. First, by showing simple formal proof ideas like the one above. Second, by stimulating the students' curiosity ("Why does it work?"). And third, by showing that the most straightforward intuition (usually the greedy one) need not always work (as in the tasks *Palindromes* and *Wagons*).

7. Anti-binary Sets

7.1. *Problem*
An anti-binary set is a set of integers that does not contain two numbers of the form m and $2m$. For example, the set:

$$A = \{2, 3, 5, 8, 11, 13\}$$

is anti-binary whereas the set:

$$B = \{2, 3, 5, 6, 8, 11, 13\}$$

is not, since both 3 and 6 are its elements.

What is the largest size of an anti-binary set that is a subset of

(a)   {1, … , 11}?
(b)   {1, … , 12}?
(c)   {1, … , 13}?
(d)   {1, … , 14}?
(e)   {1, … , 15}?

How many different anti-binary subsets of the same size exist?

This riddle is a simplified version of a task from Algorithmic Engagements 2007
(http://main.edu.pl/en/archive/pa/2007/pod).

7.2. *Solution*

The same task in a traditional setting has been presented in Kubica and Radoszewski (2010). The most reliable way to obtain the solution is to draw a graph with vertices numbered 1 through n ($n = 11, … , 15$ depending on the subtask) and edges connecting every two nodes with numbers m and $2m$. Note that such graph is always a collection of disjoint paths. The graph for $n = 15$ is presented below.
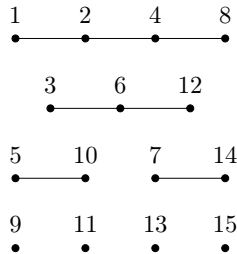
Now the task boils down to finding the largest independent set in this graph and counting all largest independent sets, which are both extremely simple. We obtain the following answers to subtasks:

(a)   12 anti-binary sets of size 7
(b)   6 anti-binary sets of size 8
(c)   6 anti-binary sets of size 9
(d)   12 anti-binary sets of size 9
(e)   12 anti-binary sets of size 10

7.3. *Methodological Comments*

This task introduces basic notions of graph theory. A good idea is to collect the solutions of both parts of the task separately. Students usually manage to solve the first part of the task (that is, to find the largest anti-binary set for a given *n*) quickly without any graphical illustration. However, they will need to perform the graph construction (either explicitly or implicitly) to count the number of largest anti-binary subsets.

The original task from Kubica and Radoszewski (2010) involved counting *all* anti-binary subsets of a set $\{1, \ldots, n\}$. While this question is nice to be solved at home with a computer (or calculator) at hand, it involves too much computation to be solved in a class. Let us note that counting *all largest* antibinary subsets solely enforces the students to use some smarter method than trial and error.

8. Conclusions

We presented several algorithmic non-programming tasks to be solved during an exercise session with a group of high-school or pre-high-school students who are just starting to program (or have never programmed before). A number of desired features of an algorithmic non-programming task were already listed in Kubica and Radoszewski (2010). These conditions also apply for the tasks presented in this paper: we aim at small instances of regular programming tasks which admit a technically simple solution that avoids using advanced techniques or classical algorithms and which do not admit a trivial suboptimal or heuristic solution that behaves better on the particular data. However, due to the specific environment considered in this paper, more restrictive guidelines apply.

Obviously we need to produce several interesting instances (subtasks) for each task. This can often be done automatically using implementations of the solutions. Task statement should be based on simple concepts, possibly include a figure to increase its attractiveness, and have a short formulation. A student should be able to come up with a solution within 5 minutes. The solution should consist of a single idea and minimum computations (errors in computations aren't fun). Note that some computations are equally easy for a computer but their complexity differs dramatically in the pen-and-paper scenario, e.g. adding 10 numbers from $\{1, \ldots, 1000\}$ vs finding their maximum. Last but not least, the solution should include some of the crucial concepts of computer science and algorithms in particular, while the task statement should avoid using any concepts of programming.

We have also introduced a new type of tasks on which the most intuitive though incorrect solution fails. In such tasks we do not expect the students to invent the model solution of the corresponding programming task. The purpose of such tasks is to encourage students to perform verification of correctness of their solutions.

References

Bell, T., Fellows, M.R., Witten, I. (1998). *Computer Science Unplugged ... Off-Line Activities and Games for all Ages.* `http://csunplugged.org/`

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

Dagienė, V. (2006). Information technology contests – introduction to computer science in an attractive way. *Informatics in Education*, 5(1), 37–46.

Dagienė, V., Futschek, G. (2008). Bebras international contest on informatics and computer literacy: criteria for good tasks. In: *3rd International Conference on Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2008*. 19–30.

Forišek, M. (2013). Pushing the boundary of programming contests. *Olympiads in Informatics*, 7, 23–35.

Forišek, M., Steinová, M. (2013). *Explaining Algorithms Using Metaphors.* Springer.

Hakulinen, L. (2011). Survey on informatics competitions: developing tasks. *Olympiads in Informatics*, 5, 12–25.

Kubica, M., Radoszewski, J. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.

Merry, B., Gallotta, M., Hultquist, C. (2008). Challenges in running a computer olympiad in South Africa. *Olympiads in Informatics*, 2, 105–114.

Opmanis, M. (2009). Team competition in mathematics and informatics "Ugāle" – finding new task types. *Olympiads in Informatics*, 3, 80–100.

van der Vegt, W. (2012). Theoretical tasks on algorithms; two small examples. *Olympiads in Informatics*, 6, 212–217.

Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., Wagner, D. (Eds.). (2011). *Algorithms Unplugged*.

J. Radoszewski (1984), assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, chair of the jury of Polish Olympiad in Informatics, co-chair of the Scientific Committee of CEOI'2011 in Gdynia, former member of Host Scientific Committees of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, and BOI'2008 in Gdynia, Poland. His research interests focus on text algorithms and combinatorics.

REPORTS

omegaUp: Cloud-Based Contest Management System and Training Platform in the Mexican Olympiad in Informatics

Luis Héctor CHÁVEZ, Alan GONZÁLEZ, Joemmanuel PONCE

omegaUp
Hacienda de Coaxamalucan 138, Col. Hda. de Echegaray
Naucalpan, Estado de México, México CP 53300
e-mail: {lhchavez,alanboy,joe}@omegaup.com

Abstract. omegaUp is an open source cloud-based online contest and training platform for the Mexican Olympiad in Informatics. It is designed to be a robust, highly scalable, low cost and secure solution. To achieve our security goals, the platform leverages *minijail*, a modern and actively maintained sandbox from the ChromiumOS project with a good security track record as well as other features for grading task solutions reliably. Students can solve past and ongoing contests as well as training problems with live feedback. omegaUp also provides easy to use administrative features enabling users to upload tasks for automatic grading without staff assistance and create their own contests using any previously uploaded task. Since omegaUp's launch 3 years ago, it has hosted more than 400 contests for both IOI training and ACM-like competitions, graded more than 100,000 submissions, and helped the Mexican Team achieve their best performance in the IOI finals to date.

Keywords: automatic grading, contest management system, omegaup, *minijail*, cloud, security.

1. Introduction

In 2010 there were several online judges and training platforms in Mexico, each one targeting a narrow community. There were training gates for one particular language, like

Karelotitlan (Karelotitlán, 2011) (only hosting Karel problems for training), and some states also had their own training platform or contest management system. National contests usually suffered from scalability problems: when bursts of load happened, live feedback was often affected and the contestants' experience was unnecessarily stressful because of the grading system. States typically did manual or semi-automated offline grading. Due to the fragmented effort, none of the platforms invested enough time in designing and implementing a solution with the scalability, security and easy-to-use manageability features in mind that national competitions demand. It was often the case that only the developer of the platform had the ability to upload tasks, set up contests and make modifications, also making the process not scalable from the human resources perspective. Last-minute corrections to the test cases, and sometimes manual verification of the student's submissions made grading a time-consuming and error-prone process, which involved several extra hours of organizers' time. The result was that training for IOI and keeping track of the student's progress was a very ad-hoc process.

omegaUp was created to become a platform that could satisfy the needs of the whole country to have a centralized, properly maintained and reliable training gate and contest management system. It was designed to have easy to use administrative features, so contest organizers across the country can create, manage, and monitor their own contests and even upload their own tasks for automatic grading. This also means having a reduced dependency on the staff that runs the platform.

omegaUp is now the official platform to host contests for both the national and several state olympiads. Instead of every region maintaining their own platform and their own similar but slightly incompatible grading methods, having a single site for the whole country provides a more homogeneous experience for contestants across the country. This is also helpful for tracking progress since the pool of previous problems is now shared and available in a single portal.

The contest environment is designed to be a robust, highly scalable, low cost and secure solution. Security is achieved by designing a role-based permission model for the whole site and isolating all untrusted components as much as possible: nodes running contestants' code are hosted in virtual machines in the cloud and leverage *minijail* (Chromium, 2010), a modern and actively maintained sandbox from the ChromiumOS project with a good security track record. It is also possible to connect to the omegaUp server in a firewall-friendly way that locks down the permissions even more to make stronger isolation guarantees.

Running contestant's code in the cloud also helps the platform achieve our scalability goal: to guarantee a good experience for all the contestants during a live event, regardless of the size of the contest. The platform leverages from Microsoft Windows Azure (Windows, 2010) to host the compiling and grading processes, making possible to scale from one grading machine to dozens of them within minutes with very low cost. The site also makes heavy use of caching systems to store the result of expensive calculations temporarily, effectively improving the number of requests per second the system can handle.

In this paper, we describe how omegaUp works and how it is used to run the Mexican Olympiad in Informatics as well as other national programming contests.

2. Informatics Competitions in México

Mexico holds its national olympiad in informatics (Olimpiada Mexicana de Informatica, OMI) on a yearly basis. Each of the 32 Mexican States train and select their best 4 contestants who participate in the national contest. Our National Olympiad has 100 participants on average. Most of the states use a similar strategy and have their own established local informatics committees and run their own State Olympiad in Informatics as well as training tracks.

Mexico started its participation in the International Olympiad in Informatics in 1992. Since then, the process of determining our IOI delegation has evolved (Cepeda and García, 2011). We currently select the best 32 competitors from our national contest and prepare them targeting the IOI to be held the year after our National Olympiad. Our 4 IOI participants are selected as the result of a training program that lasts about 10 months, consisting of 4 training camps and several rounds of online contests and practices. A group of 20 collaborators with previous international contests experience donate their time to prepare, create and translate more than 80 tasks used in the selection process.

We have identified several factors that make our informatics development challenging. Our territorial extension with respect to our economic development often bounds our training camp organization. Informatics is not considered a first-class assignment in our basic education system. Students usually arrive at a late age to competitive programming with respect to other countries. Furthermore, the understanding of English language is limited on most of our contestants in early stages, reducing the sources of self-training information they can read and practice against.

Our mission with omegaUp is to overcome the aforementioned challenges with a platform that sets our country in a position to achieve better results in the International Olympiad in Informatics and, in general, contribute to the development of computer science in our country.

3. Architecture and Design

The main design goals that omegaUp is set to solve are:

1. It must be an always-on Contest Management System where contents are provided and driven by the community itself, without site administrators' interaction.
2. It must provide a secure environment to run untrusted contestant's code, preventing cheating.
3. It must provide a scalable and low-cost contest environment where nodes could be added to the system using cloud computing services in case of an increase in load.

omegaUp uses a multi-tier service-oriented architecture, with physical separation between components, as seen in Fig. 1. The basic workflow of interaction between omegaUp services is as follows. When a contestant submits source code to be graded, the Frontend web interface relays that message to the Grader service, who orchestrates the workflow to get the code and input data (if needed) to an available Runner node that
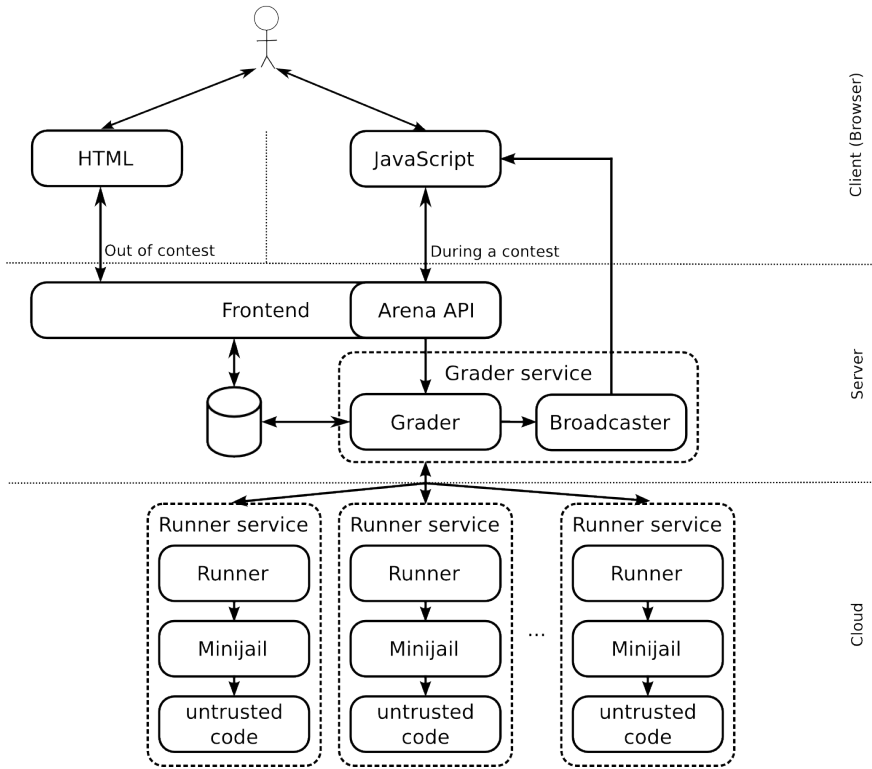
Fig. 1. omegaUp architecture.

compiles and executes the contestant's code in a sandbox. Afterwards, the Grader receives the outputs produced by the contestant's program from the Runner, compares it to the expected output and it emits a final verdict which is then sent back asynchronously to the contestant via a Broadcaster service. The next sections further explain each of the components.

All communication between components is encrypted using SSL to preserve integrity and confidentiality of the messages, and all server-side and cloud-side components use a certificate chain rooted in a self-signed Certificate Authority that provides mutual authentication to prevent contestants invoking any service directly. Isolating the expected case data is also important to provide security in depth, so even if contestants find vulnerabilities in the sandbox inside a Runner node, no expected outputs are ever stored in the machines that run untrusted code, making cheating extremely difficult.

Given the elasticity of the computing resources, it is possible to run 5-hour contests with an average verdict latency of 2–30 seconds (depending on the number of cases and time limit) with a total cost of 3 US dollars. Based on the same elasticity property, we can still achieve a good and responsive contestant experience even when there are changes to the task's test cases in the middle of the contest that require a sudden burst of hundreds of submissions for re-grading. Furthermore, the current price trend of cloud

services with major providers indicates that running a contest will cost us even less in the future.

One trade-off about running contestants' code in cloud computing services is an expected lower consistency of performance between machines: there are slight but detectable variations between instances of a cloud compute provider's service, even at the same price level (Ou *et al.*, 2012). To avoid this, other online judge services such as Sphere Online Judge have dedicated hardware clusters to evaluate contestants' code (Sphere, 2008). Nonetheless, it does not affect us too much, given that most tasks are designed to clearly distinguish between implementations of the expected complexity and a worse one. Running a 5% sample of the corpus of submissions on different clusters of the same provider at the same price level[1] resulted in a score change for at most 0.73% of the submissions. Furthermore, when comparing Amazon's m3.medium against Windows Azure's A1 instances, we found that only 2.06% of those submissions would have changed score if evaluated in the other platform.

3.1. *Frontend*

All user-facing interaction is done through the Web Interface layer, written in HTML 5 and Javascript. Users of omegaUp do not have to download any external plugin or dependency to use the service.

Since security is a priority for omegaUp and to use all HTML 5 features, support for Internet Explorer previous to version 8 was explicitly dropped. This also reduces development costs. Modern browser security features such as Content-Security-Policy and Strict-Transport-Security are used by omegaUp and further guarantee the integrity of the platform.

All business logic for contest creation and management is processed by the Frontend web service. The Frontend service is written in PHP, running on top of Facebook's HHVM and served by nginx. To maintain proper separation of concerns, all interactions between the Web Interface and the Frontend service are strictly done via JSON messages using a RESTful API (omegaUp, 2011). When a contestant submits a source code to be graded, the Web Interface calls a Frontend API to register the grading request in omegaUp. The Frontend then informs the Grader service about the new submission via Grade Request Message delivered through a REST API call.

The Frontend provides the administrative interface to manage contests and tasks, generates statistics and rankings, and hosts the main contest interface which has the standard CMS features such as clarifications, viewing past submissions, and a real-time scoreboard using WebSockets.

This layer is also the one responsible for all role-based authentication, sign-on through Google, Facebook, and native logins, as well as contest policy enforcement. There is an internationalization infrastructure built in and we support both English and Spanish, although the vast majority of task descriptions are only available in Spanish.

---

[1] m3.medium in the case of Amazon and A1 for Windows Azure

A lockdown mode is available for contests that require tighter security controls and network-level isolation. Visiting the site using an alternative URL[2] enables this mode and ensures all resources are served from the same endpoint, simplifying the firewall rules that are needed in the physical contest site to achieve the desired level of isolation. Furthermore, most of the site's features are disabled in this mode, especially those that can modify contests or problems, as well as all known scenarios in which contestants can communicate with each other through site features like viewing source code of past submissions. This mode only requires that passwords are secret and are not shared between contestants.

3.2. *Grader*

The Grader service is a fully asynchronous Scala service that communicates through JSON messages over HTTPS with client authentication. After a Grade Request Message is received, it is routed to one of several runner queues that have an associated pool of Runner nodes. This allows us to provide coarse Quality-of-Service guarantees: tasks that are designed to always finish within 30 seconds in the worst case will run in the default queue and provide a very fast response time to contestants; tasks that can take more than 30 seconds will be sent to a slower queue to avoid bogging down the whole system. Additionally, if the time to response for a contest needs to be isolated from the effects of other submissions in the system, a private runner pool can be associated with a queue that will exclusively process submissions to said contest. If a task's test cases are modified while running a contest, re-grading of submissions can be done in yet another queue to further minimize impact. This allows us to very quickly react to unexpected live contest issues and mitigate disruption to contestants. Submission states are backed by a MySQL database and Grader can rebuild the submission queues upon restart.

Each queue has an associated pool of Runner nodes that handle the request in a producers-consumers fashion. Messages to the runner nodes are also sent using JSON over HTTPS. Once the runner node finishes running the task, its output is compared in the Grader service against the expected output using one of several built-in tokenizers, or can be sent back to a Runner node if a custom grader is required. This scenario usually happens when tasks do not have a unique solution, so more untrusted code needs to be executed to come up with a verdict.

Graded submissions' results are then sent to the Broadcaster component, which updates the contest scoreboard and notifies contestants of the verdict of their submissions. Broadcaster uses WebSockets to send near-real-time updates directly to contestants' browsers, avoiding constant periodic polling and providing a low-latency solution.

3.3. *Runner*

The Runner service is written in Scala, and runs using cloud computing services. We have used both Amazon Web Services and Microsoft Windows Azure as virtual machine

---

[2] `https://arena.omegaup.com/`

providers with good results. Running a task is straightforward: the runner receives a JSON message containing the source file, the execution limits and other task parameters. The sandbox is then invoked for both the compilation and evaluation of each of the tasks' cases, and sends the results back to the grader using a combination of JSON and a custom bzip2-ed stream with the case results over HTTPS.

Runners are designed to hold the least possible amount of state and be light on configuration. Runners do not need to be pre-registered on the grader: they register themselves dynamically upon start on the default queue in case there is a need to spin off new runners. Task inputs are lazily deployed from the Grader into a Runner until a submission needs it and are cached for subsequent requests using the SHA-1 hash of the data as key. Contest administrators can request the re-grading of any submission in a way that debugging information and errors are displayed in the administrative interface, which is especially helpful to diagnose issues with interactive tasks or custom graders.

3.4. *Sandbox: minijail*

When the omegaUp project started, Moe sandbox (Mareš, 2009) was chosen as the sandbox implementation, since it was also used in the IOI. Some modifications were made to support multithreaded languages (such as Java) with very lightweight race condition exploit mitigation, multi-process support to also sandbox code compilation, and syscall interception to be able to fake dangerous calls without crashing (e.g. Java tries to open sockets during initialization).

As omegaUp evolved, it soon became obvious that the approach using a ptrace-style sandbox was not sufficient for a variety of reasons, including being vulnerable to TOC-TOU races (Isolation, 2014), introducing a large overhead per system call (Merry, 2010), and that maintaining it was costly since several updates to the kernel broke the sandbox in non-trivial ways.

For the second version of the sandbox, *minijail* from the ChromiumOS project was chosen (Chromium, 2010). It has a more modern architecture, and uses two recent Linux kernel security mechanisms: seccomp-bpf moves the syscall filtering to the kernel using compiled bytecode (Drewry, 2012), making it both very efficient and immune to race conditions while still providing syscall interception to return an error on certain syscalls instead of terminating the process; Kernel namespaces provide process-level isolation to the rest of the file system and the network.

minijail allowed the use of unmodified interpreters by providing a whitelist of allowed system calls with negligible overhead. This allowed us to expand the list of supported languages to C/C++/C++11, Pascal, Java, Python, Haskell, Ruby, and a Pascal-based Karel compiler with minimal effort. *minijail* is also actively maintained and used in a commercial application, so any exploits found are likely to be fixed quickly.

It is important to note that omegaUp considers both contestant's code execution and the compilation process as untrusted, so they are run within the sandbox. There have been Denial of Service (DoS) attacks on compilers such as the Java double parse bug (MITRE, 2010) and it is very easy to abuse the C++ error messages to generate giga-bytes of output with very small inputs (TGCEEC, 2014).

4. Contest Management

Any user of omegaUp can create and manage their own contests as well as tasks. Contest administrators are not limited to IOI or ACM-style contests. Instead, traits such as the penalty policy, a score decay factor, and scoreboard display policy are freely configurable. Tasks and contests can be configured to be either public or private, with security options to avoid information leaks.

There is support for editing task descriptions online using a slightly modified Markdown syntax, and infrastructure changes are underway to enable a peer-review system to raise the quality bar of problem statements and minimize last-minute changes. The use of Markdown instead of free-style HTML was chosen to maintain a more consistent look and feel for tasks descriptions.

Having a centralized national task repository with standardized rules and expectations has helped not only train for the IOI, but has spawned or helped improve several other regional and national level contests, since they do not have to worry about the infrastructure to make a big successful contest. We even have visitors and contests from other South American countries, such as Colombia and Bolivia.

5. Open Source and Openness

All of omegaUp's source code is freely available from GitHub with a BSD license[3]. We also run on top of a fully open source stack: nginx, HHVM, Debian/Ubuntu GNU/Linux with MySQL databases. Contributing to omegaUp is also easy since we provide a downloadable Vagrant instance that is set up to match our production configuration.

We are also committed to building an open platform: we encourage people that upload tasks to omegaUp to make them public for everybody to use and practice. This is a vast improvement from the previous status quo of contests in Mexico: once a contest was finished, task data was rarely provided, and even when it was, the data was usually lost after some time, or was provided in a format that was not useful outside of the particular contest environment used.

Community

The community around omegaUp is not limited to the contest management system. We also maintain a Q&A site (similar in spirit to StackOverflow) where students can ask and answer each other's questions. We also run a blog where task creators can post explanations to the solutions of their tasks to help students that are stuck.

Also, since the contest system is not designed exclusively for IOI-style tasks, ACM-ICPC student chapters across the country and other organizations with national programming contests like the Mexican National Open Programming Contest CONACUP (CONACUP, 2013) have chosen omegaUp to host their contests and training sessions.

---

[3] https://github.com/omegaup/omegaup/

6. Results and Future Work

We are convinced that omegaUp is well positioned to solve most of the problems it was originally designed to attack. In particular, the state of online contests in Mexico has greatly improved, a large and growing number of states use the platform for their local olympiad qualifying contests and to help with their training tracks. New types of competitions have been created thanks to the openness of the platform. Over the next months omegaUp will become easier to use for novice students and non-students, and the platform will move to a more autonomous system, where the users provide and review the site content.

Since omegaUp was launched in 2011 it has hosted more than 400 contests for both IOI training and ACM-like competitions, graded more than 100,000 submissions and currently provides more than 1,000 practice problems.

Other goals like improving the results of the Mexican Team in the IOI and building an integral training gate targeted to younger students are more long-term, so it will be an ongoing effort for a few years to come. So far we can say that, since 2011, omegaUp has helped Mexico win 4 Bronze and 2 Silver medals. This already represents roughly 50% of what we achieved from 1993 to 2011: 7 Bronze and 1 Silver medals. We believe these are just the early signs of its full potential.

7. Acknowledgments

References

Cepeda, A. and García, M. (2011). Mexican Olympiad in Informatics. *Olympiads in Informatics*, 5, 128–130.
ChromiumOS Design Docs – System Hardening (2010).
    `http://www.chromium.org/chromium-os/chromiumos-design-docs/system-hardening`
CONACUP – Concurso Nacional Abierto de Programación. (2013). `http://conacup.org/`
Drewry, W. (2012). *Dynamic Seccomp Policies* (*Using BPF Filters*).
    `http://lwn.net/Articles/475019/`
Isolation – The Confinement Principle. (2014).
    `https://crypto.stanford.edu/cs155/lectures/03-isolation.pdf`
Karelotitlán, (2011). `http://www.cmirg.com/karelotitlan/`
Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.
Merry, B. (2010). Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 8, 87–94.
MITRE CVE-2010-4476: The Double.parseDouble method in Java Runtime Environment (JRE). (2010).
    `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4476`
omegaUp REST API documentation. (2011).
    `https://github.com/omegaup/omegaup/wiki/REST-API`
Ou, Z., Zhuang, H., Nurminen, J. K., Ylä-Jääski, A., Hui, P. (2012). Exploiting hardware heterogeneity within the same instance type of Amazon EC2. 4–4.

   `https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final40.pdf`
SPOJ Clusters. (2008). `http://www.spoj.com/clusters/`
The Grand C++ Error Explosion Competition. (2014). `http://tgceec.tumblr.com/`
What is Windows Azure. (2010).
   `http://azure.microsoft.com/en-us/overview/what-is-azure/`

L.H. Chávez is an ACM-ICPC world finalist (2010) and has a bachelor's degree in computer science (2011) from Tecnológico de Monterrey, Campus Querétaro. He has been involved in several efforts to improve the state of programming contests in Mexico since 2007, and is one of the co-founders of omegaUp. He is currently employed at Google in the Chrome team and is also studying towards a MSc in computer science from Stanford.

A. González has a bachelor's degree in computer systems engineering (2012) from Instituto Tecnológico de Celaya. He created Teddy Online Judge in 2008, which hosted several national-level contests. He is currently a Software Engineer in Microsoft, working in the Operating Systems group.

J. Ponce participated in the IOI 2005 and was Mexican IOI Deputy Leader in 2009 and 2010. Has a bachelor's degree in computer science (2011) from Tecnológico de Monterrey, Campus León. He contributes to the Mexican Olympiad in Informatics coordinating the national team selection process since 2009 and currently works for Microsoft as a Software Engineer in Windows Azure SQL DB.

The Australian Informatics Competition

David CLARK[1], Mike CLAPPER[2]

[1] *The University of Canberra, Faculty of Education, Science, Technology and Maths*
 University Drive, Bruce, ACT, Australia, 2617
[2] *The Australian Mathematics Trust, Haydon Drive, Bruce, ACT, Australia, 2617*
e-mail: David.Clark@canberra.edu.au, Mike.Clapper@amt.edu.au

Abstract. The Australian Informatics Competition (AIC) is the entry-level Informatics competition in Australia. It is a pen-and-paper competition, requiring no programming experience. Most of the questions focus on testing algorithmic ability, with others testing students' ability to analyse algorithms, apply rules and use logic to solve problems. The algorithmic questions include standard algorithms such as breadth first search, dynamic programming and two person games, as well as ad-hoc algorithms specific to a particular scenario. Currently about 7,000 students enter the competition, but there are plans to expand it with on-line entries. The AIC is becoming more relevant with the introduction of algorithmic thinking as a component of the Digital Technologies strand of the Australian Curriculum.

Keywords: informatics competitions, algorithmic thinking, multiple choice questions, problem solving

1 Introduction

The Australian Informatics Competition (AIC) is the entry level Informatics competition in Australia. It is a one hour pen-and-paper competition and does not require or use any knowledge of programming or pseudocode. Students who perform well in the AIC are encouraged to enter the Australian Informatics Olympiad (AIO), an entry-level programming competition. Those who do well in the AIO may be invited to the December Training School and about 40 students will be encouraged to sit the Australian Invitational Informatics Competition (AIIO), and subsequently to enter the French Australian Regional Informatics Olympiad and the Asia-Pacific Informatics Olympiad. Results of these competitions are then used to determine attendance at the final selection school held in April to select the team of four students to represent Australia at the International Olympiad in Informatics (IOI).

The AIC is run by the Australian Mathematics Trust (AMT), which also runs the Australian Mathematics Competition (AMC), and shares some similarities with it, particularly its pen-and-paper format and automatic marking using mark-sense readers. Nevertheless there are some differences. The AMC was set up originally as a widely-accessible competition to enhance mathematical problem solving in Australian schools, and ``higher level'' mathematics such as the Mathematics Challenge and the Mathematics Olympiad were added subsequently. The original aim of the Australian Informatics

Committee was to select an Australian team for the IOI. One consequence of this is that all questions in the AIC have a title and most come with a little story, either real-world or fantasy, as in the IOI. Australia has participated regularly in the IOI since 1999, but the first AIC was not held until 2005. Nevertheless, the aim of Informatics in Australia has expanded to include raising the awareness of algorithmic thinking in Australia and the questions in the AIC are accessible to a wide range of students.

The relevance of the competition will increase significantly with the introduction of Algorithmic thinking into the Digital Technologies component of the Australian Curriculum. Similar curriculum initiatives are occurring overseas and there is increasing demand for resources to help teachers in what for most will be unfamiliar territory. Given the expectation that primary age students will begin to learn algorithmic thinking, the AIC problems committee has committed to introducing an Upper Primary Division of the AIC in 2015.

The AIC was first run in 2005. In that year it attracted a little over 2000 entries. Since then it has grown to over 7000 entries. Most participants are from Australia, with the remainder from New Zealand and Singapore.

In its scope and aims the AIC is similar to the on-line Bebras contest introduced in Lithuania in 2004, although the AIC has somewhat more emphasis in testing algorithmic thinking and does not have Bebras' interactive tasks. An overview of world Informatics Competitions is given by Burton (2010), and a report on the introduction of the AIC was given by Clark (2006). Sample AIC papers are available on the AMT website (`http://www.amt.edu.au/`) where there is also a book "Australian Informatics Competitions Book 1 2005–2010".

2. The Questions

2.1. *Format of the Questions*

The first six questions of each AIC paper are traditional multiple choice questions with five options. This type of question is familiar to students, quick to answer and easy to mark.

The multiple choice questions in the AIC have to be small enough to be understood and solved in just a few minutes whilst still having the flavour of algorithmic thinking. This brevity can, however, leave them to be susceptible to logic and/or educated guesswork.

In order to encourage a more systematic approach to problem solving, three-stage tasks are included. A three-stage task consists of a small problem to solve where there are three sets of data. The first data set is small or simple enough to be susceptible to ad-hoc techniques, but hopefully provides a basis for students to get a feeling for the problem and to develop an algorithm to be used in the remaining data sets. The answers are numbers in the range 0–999. There are three such three-stage tasks.

Each multiple choice question is worth three marks, and each stage of a *three-stage task* is worth two marks, so that the whole paper is marked out of 36.

2.2. *Types of Questions*

Unlike questions in the AMC, which classify into well-defined areas such as algebra, geometry and trigonometry, there are no readily available classifications for AIC questions. Nevertheless, the great majority of questions in the AIC fall into four broad categories: applying rules, logic, analysis, and algorithms. Some questions straddle categories, whilst some pattern matching questions do not fit into any of these categories.
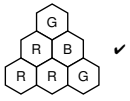
i) *Applying Rules*
Students are required to apply well-defined rules to a set of data. These questions are always multiple choice, and are typically the first or second question in a paper. They are less frequently used in the Senior paper.
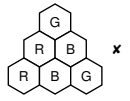
Hrossan Quilts is a typical example. It is an easy question to settle students down and, to quote one of the teachers on the problems committee, "Students like filling in grids".

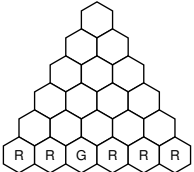---

Hrossan Quilts: 2010 Junior Q1, Intermediate Q1

The Hrossa of Malacandra make quilts of hexagonal patches in an overall triangular shape. The patches are coloured red, blue or green.
Each hexagon and the two beneath it must be the same colour or three different colours.

How many blue patches are there in the quilt below?

(A) 4   (B) 5   (C) 6   (D) 7   (E) 8

---

ii) *Logic*
These are engaging multiple choice questions early in a paper. They can require quite rigorous reasoning and case analysis.

Magic Carpet is an attractive question and, according to its proposer, "This is the one the students will be talking about afterwards."

Although in the context of the paper it is a logic question, it invites further exploration: "Can we generalise the solution to many members?" "What if there is an odd number of members?" As such, it is a good candidate for a training question for the AIO.

Magic Carpet: 2012 Senior Q4

Anna, Beth, Coral, Dianne, Emma and Francis are going camping on an offshore island. They are using their magic carpet to get to the island. The magic carpet can only take 2 people at a time, so there will need to be 9 trips, 5 across carrying 2, and 4 returning carrying 1.

Flying on a magic carpet can cause air sickness, so the speed of the carpet is determined by the rate at which the more susceptible passenger can travel.
The times that Anna, Beth, Coral, Dianne, Emma and Francis can travel one way without queasiness are 1, 2, 3, 4 and 5 minutes respectively.
What is the least time, in minutes, required to fly all members to the island?

    (A) 15   (B) 16   (C) 17   (D) 18   (E) 19

---

iii) *Analysis*

Many analysis questions require students to determine the number of operations a particular algorithm requires on a set of data. This gives students an introduction to complexity analysis of the algorithm. Another popular analysis question asks students to count the number of different routes through a network. Most analysis questions are multiple choice, although some are three-stage.

Guessing Game is a typical analysis question. Students discover the binary sort technique.

Guessing Game: 2009 Senior Q6

Ben's grandfather said to him "I have thought of a 3 digit number for you to guess. Each time you guess I will say 'Too high', 'too low' or 'correct'. You have 9 guesses. By then you should know the number."
Ben's first 8 guesses were 600 (too high), 300 (too low), 450 (too high), 360 (too low), 405 (too low), 427 (too high), 416 (too high) and 410 (too high).
By this time Ben knew that the number must be between 406 and 409. But he only had one guess left and so could not be sure that he would know the number after his last guess.
Ben's guessing strategy was flawed. After which guess was it no longer possible for him to be sure of knowing the correct number after his remaining guesses, assuming that he used the best strategy for them?

    (A) 600   (B) 300   (C) 450   (D) 360   (E) 405

---

iv) *Algorithms*

Algorithm questions are the heart of the AIC. About half of the multiple choice questions are algorithmic and they dominate the three-stage questions. All are optimization questions of one sort or another. Breadth first searches are popular, and when used in threestage questions students can be guided to discover the technique for themselves. Dynamic Programming and Two-Person Game questions are also common, and the problem setting committee takes delight in a good ad-hoc problem that requires its own special purpose algorithm.

Game is a three-stage task from the first AIC. The first diagram is easy to solve by inspection. As is the second. But even there the choice to move to the 10 then the 11 rather than the 2 then the 11 is the beginning of a breadth first search. The rather odd rule of halving the current score makes it harder to solve by inspection, and encourages an algorithmic approach.

Game: 2005 Senior Q13-15

You are playing a rather unusual game on a 4×4 grid, in which each square contains a number. You begin in the top left square of this grid, and you must travel to the bottom right square. The rules state that you must move either one square down or one square right in each turn.

To begin with you have a score of zero. Each time you move into a new square, you must halve your current score (rounding down if necessary) and then add the value of this new square. Your aim is to reach the bottom right square with the smallest score possible.

As an example, consider the following grid.

	3	9	6
1	4	4	5
8	2	5	4
1	8	5	9

The smallest possible final score for this grid is 12, which is achieved as follows.

Move	begin	down	right	down	right	right	down
Square		1	4	2	5	4	9
Score	0	1	4	4	7	7	12

What is the smallest possible score for the following grids?

	4	14	6
6	10	2	10
5	8	12	7
14	12	16	17

	2	1	1
10	11	13	7
5	7	10	8
5	5	5	10

	20	12	1
18	9	11	6
11	9	9	14
2	14	9	9

Golden iPods was used in both Intermediate and Senior papers, with different data sets. It remains one of our favourite Dynamic Programming questions.

Golden iPods: 2008 Intermediate Q13-15, Senior Q13-15

After years of research and parsecs of travel you have finally reached the third planet of Sol, ancient birthplace of humanity. You enter the great Jobs Repository and there is the object of your quest — the fabled golden iPods. You cast an expert eye over them, evaluating each. You cannot take them all. Your research indicates that taking any two adjacent iPods will cause the immediate destruction of the planet, you included. But you wish to maximise the value of the iPods you take.

For instance, if there were five iPods with values 5 6 3 1 2 you would take the 1st, 3rd and 5th, with a total value of 10. If their values were 5 8 2 2 9 you would take the 2nd and the 5th with a total value of 17.

For each set of iPod values, determine the maximum total value you can get without taking adjacent iPods.

```
1 3 1 3 4 4 4 3

1 4 4 5 8 5 3 4 1

4 3 2 5 8 6 2 2 3 2 2 2 1 1
```

The map in the two-person game Map Game is large enough to warrant a systematic approach to the solution.

Map Game: 2011 Senior Q4

You and a friend are playing a rather unusual game with a model car and a road map. You take it in turns to move the car along one section of a road on the map. (All roads on the map are one-way, left to right.) You win if you force your friend to move the car to the garage.
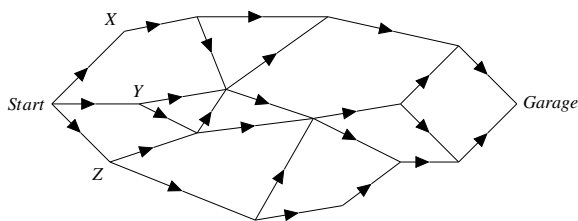
For the map above, it is your turn to go first.
You can be sure of winning if your first move is to

     (A) any of X, Y, or Z
     (B) X or Y, but not Z
     (C) X or Z, but not Y
     (D) Y or Z, but not X
     (E) Y, but not X or Z

Robot Librarian is a three-stage question with an ad-hoc algorithm. The first data set is small enough for students to fiddle, and, hopefully, to discover the algorithm.

Robot Librarian: 2014 Intermediate Q10-12

The school has acquired a robot to help the librarian. It can sort books on shelves, but only by taking a book out and placing it at either end of the shelf.
For instance, if the books A B C were on a shelf in the order B A C they could be sorted by moving the A to the front. If they were in the order C B A they could be sorted by moving the C to the end and then the A to the front, (or the A to the front and then the C to the end).
Each of the following lists represents a shelf of books. For each shelf what is the smallest number of books the robot must move to sort the books into alphabetical order?
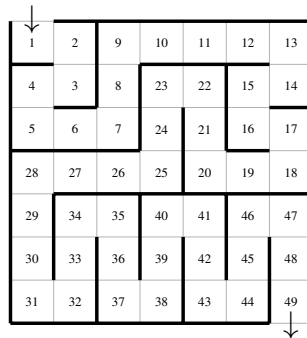
    F C A B D E

    D E C A F B G H

    D F A E C I G B J H

Putting the numbers in the cells in Maze Shortcut made it somewhat easier to find the algorithm, but not doing so could have turned it into a bit of a time sink.

Maze Shortcut: 2013 Senior Q1

The way through the maze below passes through all 49 cells.

How long is the shortest path made possible by removing one wall of one cell?

(A) 15 (B) 17 (C) 19 (D) 21 (E) 23

The algorithm in Aesthetic Skyline is not too difficult to find, but it caused some debate in the committee.

Aesthetic Skyline: 2014 Senior Q4

The council's planning committee has decided that the buildings in a new development should be arranged to provide an aesthetic skyline. This means that adjacent buildings should differ in height as much as possible. For example, consider the two arrangements of five buildings with heights of 8, 4, 3, 2 and 1 floors below:

The arrangement on the left has a total height difference of $4 + 3 + 1 + 1 = 9$ floors, whilst that on the right has a total height difference of $4 + 7 + 2 + 1 = 14$ floors. (But better arrangements can be found.)

A new development consisting of eight buildings with heights of 2, 3, 5, 2, 9, 6, 5 and 1 floors is planned.

What is the maximum total height difference for these eight buildings?

(A) 28 (B) 29 (C) 30 (D) 31 (E) 32

The concern was that the proof of optimality is not easy and some of the better students could spend time looking for a better solution. Readers are invited to admire the cute diagrams. All diagrams are written in the Tikz package in Tex

Golf can be solved by a nice ad-hoc algorithm, easily discovered by students. It was used as a three-stage task in the Intermediate paper, and as a multiple choice question (with a smaller data set) in the Junior. If it was extended to three players, it could be expressed as a Transportation Problem.

Golf: 2013 Junior Q6, Intermediate Q13-15

Yani and Na Yeon are entering as a team in a golf match. Their match score is calculated as follows. For each hole they must choose to include either Yani's score or Na Yeon's score. Overall, an equal number of scores must be chosen from each player. For example, if there are 10 holes in a game, 5 of Yani's scores and 5 of Na Yeon's scores must be included.
The aim is to make the combined score as small as possible.
For instance, suppose there were four holes and the scorecard was as follows:

Hole	1	2	3	4
Yani	4	1	4	5
Na Yeon	2	3	4	2

The smallest match score, 9, would be achieved by taking Yani's score for holes 2 and 3, and Na Yeon's score for holes 1 and 4.
For each of the scorecards below, what is the smallest possible match score?

Hole	1	2	3	4	5	6	7	8
Yani	4	1	3	2	3	2	4	5
Na Yeon	3	2	2	3	4	1	5	6

Hole	1	2	3	4	5	6	7	8	9	10	11	12
Yani	2	1	2	2	5	6	1	2	2	2	3	1
Na Yeon	4	2	3	5	4	5	4	2	6	5	2	4

Hole	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Yani	3	5	4	3	5	5	6	4	7	3	5	7	2	1	2	4
Na Yeon	1	2	1	1	5	1	2	7	3	3	2	4	1	3	3	3

---

v) *Other Types*

Not all questions fit into the above categories. Some questions straddle categories, whilst some pattern matching questions do not fit into any of the above categories.

Alphabet Sort is a three-stage pattern matching task. The first data set allows the students to discover the pattern on a smaller amount of data. The last data set invites students to work backwards.

Alphabet Sort: 2008 Junior Q10-12

A sorting program does not understand about numbers. It treats all digits as letters, so that the numbers 10, 11, 100, 101, 111 would be sorted as 10, 100, 101, 11, 111.

- If the numbers 1, 2, ..., 99 are sorted, what is the 45th number?

- If the numbers 1, 2, ..., 999 are sorted, what is the 120th number?

- If the numbers 1, 2, ..., 200 are sorted, what is the 195th number?

3. Some Practicalities

Both the AMC and the AIC include multiple choice questions, but their use is primarily to enable automatic marking. In a traditional multiple choice question, a student who does not know the answer may be able to eliminate some options, thereby increasing their chances of guessing correctly. For example in

The capital of Lithuania is

(A) London   (B) Sydney   (C) Vilnius   (D) Belgrade   (E) Riga

most Australians would be able to eliminate Sydney and many would be suspicious of London, so the chances of guessing correctly would increase to one in three. Clark and Pollard (2004) have designed scoring systems to reward partial knowledge with partial marks. In the AIC and the AMC, however, there is no partial knowledge and there are no obvious distractors to eliminate. Students solve the problem and check to find it on the list.

A further decision in the AIC was whether to penalise incorrect answers. This was done in the AMC until 2004 in an effort to deter guessing. This was only partially successful. Studies on the AMC data by Atkins, et al. (1991) indicated that in years 7 and 8, girls were more likely to guess than boys, but the situation was reversed in years 11 and 12, and in a competition with several hundred thousand entries, odd results did very occasionally occur. They were identified as a student who did poorly in previous years who suddenly came out as a prize winner. Penalties are not used in schools and were unpopular in the AMC and were dropped in 2005. Guessing was addressed by including questions whose answers were a number in the range 0–999. In the AIC there are no penalties and three-stage questions have 0–999 answers.

Unlike in the AMC, calculators are permitted in the AIC. Whilst accuracy is important, the committee decided that students who found the correct algorithm should not be penalised for an arithmetic mistake, especially as programmers would have access to them when developing and testing their algorithms. In multiple choice questions, where it is possible the options are two or more apart rather than being contiguous.

Another technique used in analysis questions is to give a range of numbers in each option.

4. Moderation

The AIC is set over one weekend by the problems committee. The chair of the committee then constructs the data, invents the story if necessary, typesets the data, writes the solutions and distributes the papers to the committee for moderation. The committee includes two teachers. Later, there is a second round of moderation by teachers. They ensure that the questions are at the right level and the language is accessible to the students. Examples of questions resolved during moderation include "Can we assume that students know what a vowel is?" (answer "No") and "Can we assume that students will know N, S, E and W?" (again answer "No"). The first AIC was set by three academics

in about two weeks, communicating by eMail. Moderation was done by university colleagues. To our relief there were very few problems, but we did stumble over one question where we defined clearly, unambiguously and concisely what we meant by a syllable. Students ignored our definition and used their own ideas. Teacher moderators would have prevented that.

5. Where to Next

There is no doubt that there is a growing concern in many countries that algorithmic and computational thinking is under-represented in the school curriculum. Whilst it used to be a part of some senior computer science courses, these have now become primarily application based, and the need for programming skills is much reduced. In Australia, this has been addressed by the introduction of a Digital Technologies strand as a part of the Technologies Learning Area. Whilst useful as a statement of intent, this will be hard for schools to implement without support and resources. Students in Years 5 and 6 are expected to be able to Design, modify and follow simple algorithms represented diagrammatically and in English involving sequences of steps, branching, and iteration (repetition), whilst by Year 8, they should Implement and modify programs with user interfaces involving branching, iteration and functions in a general-purpose programming language. [Australian Curriculum – `http://www.australiancurriculum.edu.au/technologies/digitaltechnologies/Curriculum/F-10`]

This is a very long way from what is currently happening in schools. In this context, the resource base provided by AIC questions which introduce algorithmic thinking can play an important role. This has led the AMT and the problems committee to consider the implementation of an additional competition level for Upper Primary (Years 5 and 6) students and this level will be introduced in 2015. We have also made the decision to move the competition into an on-line format from 2015. Hopefully, this will make it much easier for schools to access and implement, and it seems an appropriate format for a competition designed to encourage students to learn programming. This also opens up the possibility of marketing the competition internationally. There is currently a dearth of entry level informatics competitions and none, for instance, which have yet penetrated the US market, where the same kinds of curriculum demand are beginning to appear.

Such marketing may lead to a need to give the competition a more international name, though no decision has been taken on this at present.

Currently, Australias stocks in Informatics are high. We have just hosted a successful IOI (Brisbane 2013) and our results in IOI are very strong, particularly when compared with our population. This is due, in part, to programs (including the AIC) which allow us to identify and develop students with potential. However, in a world with an increasing demand for technological literacy, we need to encourage more schools and students to develop the skills required to cope with such demands and we believe that the AIC represents an important starting point in this process.

References

Atkins, W.J., Leder, G.C., O'Halloran, P.J., Pollard, G.H., Taylor, P.J. (1991). Measuring risk taking. *Education Studies in Mathematics*, 22, 297–308.

Burton B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

Clark, D. (2006). The 2005 Australian Informatics Competition. *The Australian Mathematics Teacher*, 62(1), 30–35.

Clark, D., Pollard, G. (2004). A measure of the effectiveness of multiple choice tests in the presence of guessing: Part 1, crisp knowledge. *Mathematics Competitions*, 17(1), 17–33.

Clark, D., Pollard, G. (2004). A measure of the effectiveness of multiple choice tests in the presence of guessing: Part 2, partial knowledge. *Mathematics Competitions*, 17(1), 34–47.

D. Clark was a committee member of the Australian Mathematics Foundation for 20 years, and has been a member of the Australian Informatics Olympiad Committee since its inception in 1998. He was treasurer for the first 10 years and was deputy team leader of the Australian teams at the IOIs from 2001–2003. He is currently the chair of the AIC problems committee, a post he has held since 2008.

M. Clapper has been a teacher of mathematics and computer science for many years, as well as a school principal. He has been a member of the AMC Problems Committee for 12 years and is currently chair of that committee. He commenced his current position of Executive Director of the AMT in January 2013.

Indonesian Olympiad in Informatics: Significant Advancements between 2010 and 2014

Yugo K. ISAL[1], M.M. Inggriani LIEM[2], Adi MULYANTO[2], Brian MARSHAL[3]

[1] *Faculty of Computer Science, University of Indonesia*
[2] *School of Electrical Engineering and Informatics, Bandung Institute of Technology*
[3] *Alumni Association, Indonesian Computing Olympiad Team*
e-mail: yugo@cs.ui.ac.id, inge@informatika.org, adi@informatika.org, bm@sirclo.com

Abstract. Indonesia has participated in the IOI since 1995 and has achieved 2 golds, 16 silvers, and 24 bronzes so far. A national report published in Olympiad in Informatics, 2010, Vol. 4 has covered the experience between 1995 and 2010. As a follow on, this article describes significant progress in the Indonesian Olympiad in Informatics (abbreviated as OKI in Indonesian) since 2010. The progress was driven by clearer structure of stakeholders and processes, updates on national curriculum, the establishment of national contest management system and more active alumni involvement which supports a more stable performance for OKI team (abbreviated as TOKI in Indonesian) in IOI. We analyze briefly each progression which might be relevant to other countries facing similar challenges.

Keywords: informatics, olympiad, training, national report, secondary education.

1. Introduction

1.1. *1995–2010*

As also explained in our previous national report (Kurnia and Marshal, 2010), the following is a short summary of Indonesian Olympiad in Informatics (abbreviated as OKI in Indonesian) between 1995 and 2010.

OKI, more frequently addressed as a team rather than as a system: OKI team (abbreviated as TOKI in Indonesian), is an informatics contest started in 1995. The main goal of TOKI is to introduce the young generation in Indonesia to informatics through a form of competition, as the formal curriculum of pre-university education in Indonesia does not include any informatics education. In addition, TOKI organizers coordinate the selection and training process of students to take part in the IOI.

The number of participants of TOKI itself has grown from 1 in 1995 to 1495 in 2009 with the primary sponsorship of the Ministry of National Education. Indonesia has participated in IOI every year since 1995 (with the exception of 2003 due to visa problems) and has achieved 2 golds, 11 silvers, and 16 bronzes up to IOI 2009.

The uneven development and accessibility of infrastructure, and the limited quantity and quality of human resource are among the main challenges faced in organizing TOKI. To overcome the challenges: multi-tiered competition structure, online training, a subset of Pascal language called Pseudopascal and OKI Bureaus came as solutions. While the measure was not there yet to judge the progress, we saw that these solutions produced some improved results.

1.2. *2010–Today*

In the past 4 years, generally, informatics competition "fever" among senior high school students has come into effect as more and more TOKI alumni managed to directly or indirectly attract juniors to participate. The effect is also amplified by the rising internet penetration in Indonesia (Lukman, 2013).

As a result, some improved results can be observed. Five more provinces were "unlocked" in the past 4 years, Aceh, Riau Islands, Lampung, Babel Islands and East Kalimantan as can be seen in Table 1 even though the ones that went to the IOI still come from the 8 provinces, with no changes in the past 4 years (please refer to Fig. 2). At the same time, even though Indonesia still cannot be classified as a top performing country in IOI, its performance was improved and became more and more stable recently as can be observed in Fig. 1.

Behind the positive fever aforementioned, some significant advancements were driven such as a clearer structure of stakeholders and processes which will be further described in Section 2 and updates on our national curriculum, establishment of our national competition information system and more active participation from alumni which will be further described in Section 3.
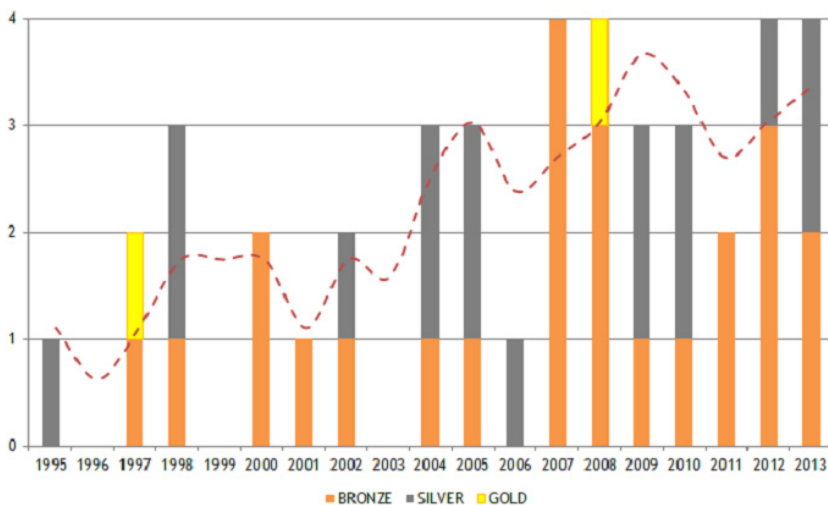

Fig. 1. Indonesian participant medal achievement at IOI 1995–2013.

Table 1

Participant province origin at the first training camp leading to IOI 2002–2014

Province	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	Total
Java														
Jakarta	10	9	4	5	8	8	6	7	5	6	6	6	4	84
West Java	4	7	4	6	4	3	1	3	4	5	4	4	5	54
Central Java	2	3	9	6	3	5	4	3	4	6	2	3	4	54
East Java	2	6	2	4	4	1	1	2	2	3		3	2	32
Yogyakarta		1	1	4	2	2	3	2	4	2	1	1		23
Banten		2	1	1	1	1	3		2	1	3	3	2	20
Sumatera														
North Sumatra				5	6	1	2	2	2	1	3	2	1	25
Jambi			2	2	4	3	2	2	4			2	2	23
Riau	1	1					1	1	2					6
South Sumatra		1				1					1	3		6
Aceh													2	2
West Sumatra								1		1				2
Riau Islands											1	1		2
Lampung										1		1		2
Babel Islands											1			1
Bengkulu														0
Bali-Nusa Tenggara														
Bali	1	3	3	2	2	1	1	1	1	1	2	1	1	20
West Nusa Tenggara	1	1	1	1							1			5
East Nusa Tenggara														0
Kalimantan														
West Kalimantan					3	2		2						7
East Kalimantan											1	1	2	4
Central Kalimantan					1									1
South Kalimantan														0
Sulawesi														
South Sulawesi							1	3	1	1			1	7
North Sulawesi					1							1		2
Gorontalo								1				1		2
Central Sulawesi														0
West Sulawesi														0
South-East Sulawesi														0
Maluku-Papua														
Maluku														0
North Maluku														0
Papua														0
West Papua														0

Fig. 2. Indonesian participant province origin at IOI 2002–2014.

2. Stakeholders and Processes

In the past 4 years, stakeholders involved in the Indonesian Olympiad in Informatics, from the beginning all the way until departure for the IOI, are getting clearer roles. As of today the structure is well illustrated in Fig. 3.

The roles of each stakeholder can be explained as follows:

1. Participants: with the help from senior, school, TOKI's network, and material available online, students are required to be self-motivated. The self-motivated aspect becomes more critical as informatics is still not included in the official secondary school curriculum in Indonesia.
2. (TOKI) Alumni: directly handle the selections and trainings, and prepares their materials. Since most of them are still active participating in many programming com-
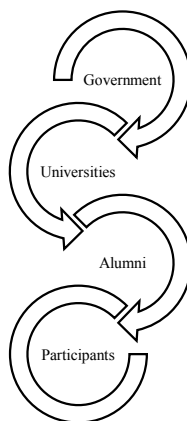
Fig. 3. Selection stakeholders.

petitions at the university level, they are more up to date and exposed to the various new problems/materials that are suitable for training in general. TOKI Alumni have become an integral part of the Scientific Committee and Technical Committee.

3. (TOKI Bureaus) Universities: maintain the curriculum, facilitating and directing the alumni and responsible for the contents and provide judges for selection purposes. There are at the moment 5 TOKI Bureaus from our cooperation with University of Indonesia, Bandung Institute of Technology, Bogor Agricultural University, Gadjah Mada University, and Sepuluh November Technology University.

4. Government (Ministry of National Education): provides and manages national budget to conduct selections and training as well as to provide scholarships for winners.

The current selection process is not much different than that was presented in (Kurnia and Marshal, 2010) as described in Fig. 4. A slight difference is that since 2010 the number of participants from each school is limited, as to allow further gap reduction between Java, Bali and Sumatra provinces and the rest, with regards to the uneven distribution of participants in the first training camp.

Because programming is not a compulsory course, the skills and knowledge of students vary (more because of students' interest and the availability of teachers as coaches), and since the selection up to the national levels involve all senior high schools students in Indonesia, the selection process up to the provincial level are meant to catch their potentials rather than their programming and problem solving skills. That is why, until now, there is no direct programming activities on the computer during the selection test up to the provincial level. The test is conducted on paper, and anything related to programming is given in Pseudopascal, as reported in (Kurnia and Marshal, 2010). Programming selection using a computer is only given in the national level.

The test in municipal and provincial levels are conducted in their own locations, and only at the national level is it conducted centrally in the same location at the same time.

| Municipal Selection | *Time: ~ April; 1 day* |
| Top participants from each municipal |
| Provincial Selection | *Time: ~ June; 1 day* |
| Top ~ 90 participants |
| National Selection | *Time: ~ September; 1 week* |
| Top ~ 30 participants |
| Training Camp 1 | *Time: ~ November; 3-4 weeks* |
| Top ~ 16 participants |
| Training Camp 2 | *Time: ~ February; 2-3 weeks* |
| Top ~ 8 participants |
| Training Camp 3 | *Time: ~ May; 2-3 weeks* |
| Top ~ 4 students |
| Training Camp 4 | *Time: ~ June; 2 weeks* |

Fig. 4. Selection processes.

3. Curriculum, Contest Management System, and Alumni Involvement

3.1. *Curriculum*

Regarding the curriculum, since 2012 the curriculum has been aligned with the IOI contest materials. By assuming that the quality of senior high school students is improving, the degree of difficulty of the test materials has been increased gradually. The significant difference is that until 2010 the test materials for national level were partly analytical and partly programming. Since 2012, the test materials for national level are entirely programming.

3.2. *Contest Management System*

One of the most important things in national competition and development is the availability of online training facility to allow interested students to learn, to keep their competition spirit, and to facilitate regular contests. It requires not only servers, but also a contest management system together with its administrator.

TOKI Alumni's technical team has successfully set up and maintain a TOKI Learning Centre, currently up and running with more than 14.000 users. Technical improvements are continually strived to increase its performance. Beginning in 2014, the team also conduct national contests and also invite contestants from other countries. The utilization of an autograder for training and teaching were also worked on by TOKI Alumni, and were implemented as undergraduate final projects in ITB (Danutama, 2013), (Chandra, 2013), and (Fernando, 2014).

The online infrastructure is also shared outside the computer olympiad contests. Nationally, Indonesia is organizing a National Olympiad in the following fields: Biology, Physics, Mathematics, Chemistry, Astronomy, Programming, Economy, and Earth Science.

Communication amongst contest' participants, technical team and TOKI Alumni is conducted through Facebook, where simple problem solving and many questions from beginners are answered and discussed. It turns out that Facebook is very effective means of communication for beginner participants.

3.3. *Alumni Involvement*

With the increasing number of alumni from year to year, a stronger and more solid alumni was built, especially since the establishment of the TOKI Alumni Association in 2011. Since most of them are still active participating in many programming competitions at the university level, they are more up to date and exposed to the various new problems/materials that are suitable for training in general. TOKI Alumni become an integral part of the Scientific Committee and Technical Committee of TOKI. Some of the alumni have worked and contributed at some prestigious IT related companies, such as Facebook, Google; and some others have successfully established their own start-ups which are recognized both nationally and internationally.

4. Conclusion and Future Work

Indonesia has participated since 1995 and will continue to actively participate in the IOI. Even though computer programming is not a compulsory course in senior high school, participating in the IOI will remain an interest for some high school students. The students learn about skills and knowledge about programming from their own interest and passion, facilitated with learning resources provided by other stakeholders. Considering the uneven distribution geographically, to select and talent scout only 4 students from about 4 million students with eligible age bracket to participate in IOI every year, is not an easy task.

The ever increasing role and contribution of TOKI alumni in the whole process of selecting and training is more recognized nowadays as the prime factor that makes it possible for the Indonesian team to raise the bar and achieve the current 'position' in IOI, as well as other programming competitions.

From the statistics presented at the beginning of this report, it can be concluded that the achievement of Indonesian team is becoming more stable gradually. The selection and training processes have become a systematic process in which the quality of the output is more measurable and predictable; no longer merely an ad hoc process. Talent scouting in provinces outside Java, Bali and Sumatra remains to discover 'hidden jewels' to improve the input. Lastly, as the outcome, the participation and achievement at the IOI also improves the spirit of competition amongst senior high school students which in turn also means improvement in quality education in Indonesia.

References

Chandra, T.N., Liem, I. (2013). Source code editing evaluator for learning programming. *ICEEI 2013*, 169–175.

Danutama, K., Liem, I. (2013). Scalable autograder and LMS integration. *ICEEI 2013*, 387–395.

Fernando, J., Liem, I. (2014). Components and architectural design of autograder system family. *Olympiad in Informatics*, 8.

Kurnia, I.W., Marshal, B. (2010). Indonesian olympiad in informatics. *Olympiad in Informatics*, 4.

Lukman, E., *Report: Indonesia Now Has 74.6 Million Internet Users; this is what they do Online*.
    http://techinasia.com/indonesia-internet-users-markplus-insight

Y.K. Isal is a faculty member of Faculty of Computer Science, University of Indonesia. He is an active organizer, coach and judge for Indonesian Olympiad in Informatics since 2006.

M.M.I. Liem is a faculty member of School of Electrical and Engineering, Bandung Institute of Technology. She is an active organizer, coach and judge for Indonesian Olympiad in Informatics since 2004. She is also coaching and advising Bandung Institute of Technology's team for ACM-ICPC International Collegiate Programming Contest.

A. Mulyanto is a faculty member of School of Electrical Engineering and Informatics, Bandung Institute of Technology. He is an active organizer, coach and judge for Indonesian Olympiad in Informatics since 2004. His special role as an organizer is in managing coordination among stakeholders such as universities and Ministry of National Education.

B. Marshal is an alumnus of Indonesian Olympiad in Informatics. He studied computer science at Nanyang Technological University. He is an active organizer, coach and judge for Indonesian Olympiad in Informatics since 2007. His special role as an organizer is in managing alumni association.

About Journal and Instructions to Authors

OLYMPIADS IN INFORMATICS is a peer-reviewed scholarly journal that provides an international forum for presenting research and developments in the specific scope of teaching and learning informatics through olympiads and other competitions. The journal is focused on the research and practice of professionals who are working in the field of teaching informatics to talented student. OLYMPIADS IN INFORMATICS is published annually (in the summer).

The journal consists of two sections: the main part is devoted to research papers and only original high-quality scientific papers are accepted; the second section is for countries reports on national olympiads or contests, book reviews, comments on tasks solutions and other initiatives in connection with teaching informatics in schools.

The journal is closely connected to the scientific conference annually organized during the International Olympiad in Informatics (IOI).

Abstracting/Indexing

OLYMPIADS IN INFORMATICS is abstracted/indexed by:

- Cabell Publishing
- Central and Eastern European Online Library (CEEOL)
- EBSCO
- Educational Research Abstracts (ERA)
- INSPEC

Submission of Manuscripts

All research papers submitted for publication in this journal must contain original unpublished work and must not have been submitted for publication elsewhere. Any manuscript which does not conform to the requirements will be returned.

The journal language is English. No formal limit is placed on the length of a paper, but the editors may recommend the shortening of a long paper.

Each paper submitted for the journal should be prepared according to the following structure:

- concise and informative title
- full names and affiliations of all authors, including e-mail addresses
- informative abstract of 70–150 words
- list of relevant keywords
- full text of the paper
- list of references
- biographic information about the author(s) including photography

All illustrations should be numbered consecutively and supplied with captions. They must fit on a 124 × 194 mm sheet of paper, including the title.

References cited in the text should be indicated in brackets, e.g., for one author – (Johnson, 1999), for two authors – (Johnson and Peterson, 2002), for three or more authors – (Johnson et al., 2002). If necessary, the page number may be indicated as (Johnson, 2001, p. 25).

The list of references should be presented at the end of the paper in alphabetic order. Papers by the same author(s) in the same year should be distinguished by the letters a, b, etc. Only Latin characters should be used in references.

Please adhere closely to the following format in the list of references:

For books:

Hubwieser, P. (2001). *Didaktik der Informatik*. Springer-Verlag, Berlin.

Schwartz, J.E., Beichner, R.J. (1999). *Essentials of Educational Technology*. Allyn and Bacon, Boston.

For contribution to collective works:

Batissta, M.T., Clements, D.H. (2000). Mathematics curriculum development as a scientific endeavor. In: Kelly, A.E., Lesh, R.A. (Eds.), *Handbook of Research Design in Mathematics and Science Education*. Lawrence Erlbaum Associates Pub., London, 737–760.

Plomp, T., Reinen, I.J. (1996). Computer literacy. In: Plomp, T., Ely, A.D. (Eds.), *International Encyclopedia for Educational Technology*. Pergamon Press, London, 626–630.

For journal papers:

McCormick, R. (1992). Curriculum development and new information technology. *Journal of Information Technology for Teacher Education*, 1(1), 23–49.
`http://rice.edn.deakin.edu.au/archives/JITTE/j113.htm`

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

For documents on Internet:

International Olympiads in Informatics (2008).
`http://www.IOInformatics.org/`

Hassinen, P., Elomaa, J., Ronkko, J., Halme, J., Hodju, P. (1999). *Neural Networks Tool – Nenet (Version 1.1)*.
`http://koti.mbnet.fi/~phodju/nenet/Nenet/General.html`

Authors must submit electronic versions of manuscripts in PDF to the editors. The manuscripts should conform all the requirements above.

If a paper is accepted for publication, the authors will be asked for a computerprocessed text of the final version of the paper, supplemented with illustrations and tables, prepared as a Microsoft Word or LaTeX document. The illustrations are to be presented in TIF, WMF, BMP, PCX or PNG formats (the resolution of point graphics pictures is 300 dots per inch).

Contacts for communication

Valentina Dagienė
Vilnius University Institute of Mathematics and Informatics
Akademijos str. 4, LT-08663 Vilnius, Lithuania
Phone: +370 5 2109 732
Fax: +370 52 729 209
E-mail: valentina.dagiene@mii.vu.lt

Internet Address

Olympiads in Informatics provides an early access to the articles by placing PDF files of the papers on the Internet. All the information about the journal can be found at:

```
http://ioinformatics.org/oi_index.shtml
http://www.mii.lt/olympiads_in_informatics
```

Olympiads in Informatics