ISSN 1822-7732

INTERNATIONAL OLYMPIAD IN INFORMATICS

VILNIUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

OLYMPIADS IN INFORMATICS

Volume 9 2015

Selected papers of
the International Conference joint with
the XXVII International Olympiad in Informatics
Almaty, Kazakhstan, 26 July – 2 August, 2015

OLYMPIADS IN INFORMATICS

The journal Olympiads in Informatics is an international open acccess journal devoted to publishing original research of the highest quality in all aspects of learning and teaching informatics through olympiads and other competitions.

`http://ioinformatics.org/oi_index.shtml`

ISSN 1822-7732 (Print)
2335-8955 (Online)

Foreword

The national olympiads do not exist in isolation, and the papers in the ninth time organized conference of the *International Olympiad in Informatics*, or the *IOI* as it is frequently called, showed how similar problems arise in different countries, and different environments. This conference concentrates on a variety of topics, and many of the ideas and experiences are drawn from the national olympiads.

In this volume, we have published a few papers directly focused on the development of the *IOI*. Stefano Maggiolo paper "An update on the female presence at the IOI" discusses the unbalanced gender ratio in computer science. With the help of the delegations that answered the prepared questionnaire, the author gathered statistics about the gender of contestants and leaders to show that the female participation at the *IOI* is even lower. Increasing the participation of female students is the most natural way of continuing to fulfil the goal of the *IOI* of promoting the discipline of informatics among young people, and eventually to improve the performances of the teams by increasing the pool of candidates. The performances of female contestants can be improved for example through training programs, and by removing stereotype threat. Also reintroducing the requirement of mixed gender for complete teams, as a way of encouraging countries to put in place programs promoting gender balance, can be considered.

The paper "Organising National Olympiads in Informatics: a Review of Selection Processes, Trainings and Promotion Activities" by Sébastien Combéfis and Alexis Paques reviews common structural elements and activities organised by several countries, for the advertisement of the national contest, the steps of the selection process and the proposed trainings. Specific activities added by countries according to cultural aspects or other country-specific characteristics are highlighted with the reasons motivating the integration of those activities. The paper summarises the key activities that could be organised by any country, with explanations about what they bring to the national contestants and motivation for their organisation. The authors draw the conclusions, that more cooperation and collaboration should be established between countries, so that informatics education communities may get closer at reaching the common goal of spreading programming and improving its presence in education.

It has been an aim of the *IOI* conference, since it was established, to become part of, and bring in, the wider pedagogical community. We have not had many papers authored from those outside the *IOI*, but one or two have appeared in each volume. In this volume we have detailed paper on the "Effectiveness of Robotics Competitions on Students' Learning of Computer Science" written by researchers Fatima Kaloti-Hallak, Michal Armoni, and Mordechai (Moti) Ben-Ari from Weizmann Institute of Science. This work

investigates students' learning of computer science as part of a research project on students' learning of and attitudes toward STEM (Science, Technology, Engineering, and Mathematics) subjects during their participation in robotics activities. The population consisted of groups of middle-school students (ages 13–15 years). The methodology used is both qualitative and quantitative using questionnaires, observations and interviews during the school year 2013–2014.

A few other papers in this volume deal with special software for generating tests or creating interactive tasks, and with selection of talented students in programming competitions. There are reasonings on several research methods: Jūratė Skūpienė analyses the "Multiple Criteria Decision Methods in Informatics Olympiads"; Michal Forišek describes a better way to teach dynamic programming, Mirzakhmet Syzdykov and Madi Uzbekov presents an "Ant Colony Optimisation applied to non-slicing floor planning".

In the second part of the volume Syria and Turkey present interesting and thoughtful country reports. Technical report from the Baltic Olympiad in Informatics organised in Lithuania this year are presented as well. Detailed report of the IOI Workshop'2015 "Creating an International Informatics Curriculum for Primary and High School Education" deals with the role of informatics in the primary and secondary education. The Workshop participants tried to encapsulate several activities that might give insight on how to treat this issue with success.

At the end the VisuAlgo – visualising data structures and algorithms through animation are presented by Steven Halim.

As always, thanks are due to all those who have assisted with the current volume – authors, reviewers and editors. A lot of work is required, not only to the write the papers, but to an extended period of reviewing and correction. Peer reviewing all of the papers takes a significant amount of time and work. Special thanks should be given to those people.

Last, but by no means least, particular thanks are due to the organisational committee for IOI'2015 in Almaty, Kazakhstan without whose assistance we would be unable to hold the conference. Their assistance, during what is an already busy period, is gratefully received.

Editors

Olympiads in Informatics, 2015, Vol. 9, 3–14
DOI: http://dx.doi.org/10.15388/ioi.2015.01

libinteractive: A Better Way to Write Interactive Tasks

Luis Héctor CHÁVEZ
omegaUp
Hacienda de Coaxamalucan 138, Col. Hda. de Echegaray Naucalpan
Estado de México, México CP 53300
e-mail: lhchavez@omegaup.com

Abstract. Interactive tasks are currently written as a set of language-dependent shims and libraries that are linked against the contestant's code to produce a single executable. This implies that task writers often need to generate three or four different libraries that need to be tested separately, for C/C++, Pascal and Java. Furthermore, the libraries must be written with care to avoid cheating, since it is possible for contestants to access the memory and opened files of the whole process. *libinteractive* solves these problems by defining an interface description language that is used to automatically generate shims in all IOI-approved languages in a way that is easily sandboxed; and a fast, portable interprocess communication mechanism that allows complete separation of the task writer and contestant code in different processes. This simplifies task creation and testing, making all tasks compatible with any future approved languages.

Keywords: interactive tasks, performance, omegaup, sandbox, security.

1. Introduction

Since its introduction in IOI 2010 until IOI 2014, all interactive tasks are distributed as a small package that contain a few files that the contestant can download to their machine, modify, compile, and validate their solution against a small set of provided inputs. The package contents are typically as follows:

- A source file created and tested by the task writer that reads task information from a file, interacts with the contestant's code through a series of well-established functions or procedures, and then either writes a text version of the contestant's response or a verdict of the solution.
- A header file that contains the function or procedure prototypes that can be included from the contestant's code.
- A template version of what the contestant is expected to implement.
- Optionally, some scripts that can be invoked to automatically compile, link, and execute the whole program and run it against some of the inputs. These scripts are usually written for the Unix shell `sh`.

All task code is language-dependent, so all previously created tasks have one version for each supported language, and they all need to be created manually by the task writer. Moreover, since the task's and contestant's code are both executed in the same process, there is no security guarantee whatsoever, and the programs must be coded defensively and obfuscated to prevent contestants from obtaining direct access to the input file or in-memory structures. The task files usually come with instructions to run and test the program, since the way they are created changes from contest to contest and is not defined anywhere or standardized. Some previous IOI tasks used some RPC mechanisms, like Regions from IOI 2009, but were done in an ad-hoc fashion. Given that the Microsoft Windows family of operating systems has between 88% (Statista, 2014) and 91% (NetMarketShare, 2014) of market share in desktop computers, the vast majority of students cannot easily test their solutions on the operating system they most likely have access to once the competition is over and they wish to train for the next one.

One solution to all the above problems, and the one implemented by *libinteractive*, is to run the code provided by the task writer and the contestant in separate processes that communicate through Remote Procedure Calls or RPC, which is a technique that makes executing code on a different process semantically similar or equivalent to calling a local function (Birrel, 1984). The separate processes can now be written in potentially different programming languages, allowing task writers to only provide a program in one language and allowing the remote procedure call machinery to perform translation on the fly. The rest of the ancillary code and scripts described above can be generated by a compiler by providing a formal description of the interface in an Interface Definition Language, or IDL (OMG, 1991).

libinteractive creates a standard, multi-platform, language agnostic, secure, and mainly transparent solution to describe, compile, run, and validate interactive tasks. This paper is structured in the following way: The second section briefly explores the state of the art in RPC libraries and automatic code generation through the *libinteractive* IDL. The third section describes the architecture and design of *libinteractive*, including a platform-specific optimization for Linux to significantly improve the performance of task execution. The fourth section explores the performance characteristics of the *libinteractive* RPC. The fifth section concludes with the results obtained so far, and points out further directions for future expansion.

2. RPC Mechanisms

The concept of RPC and automatic code generation has been around since the early 1980s, and since then several platforms have been created to serve different needs. They usually fall into one of the following two categories:

- Language/platform-specific RPC: most modern programming languages include an easy way to perform remote calls in a way that is syntactically equivalent to performing a regular function call. Java supports Remote Method Invocation, or RMI. Microsoft's C# and the rest of the languages supported by the Common Language Runtime allows for 9 different interprocess communication technolo-

gies, including COM. These solutions are well integrated into the language and platform they run on, but do require some extra code to be written, exceptions to be handled carefully by the consumer, and are not easy to consume outside of their respective languages or platforms.

- Service-oriented RPC: the main goal of these solutions is to very quickly define the interface of a service that then can be consumed through a network. The way this is done is by defining schemas in which objects are encoded into messages, which are passed around through the network. OMG's CORBA, Google's Protocol Buffers and GRPC, Facebook's (now Apache's) Thrift, and Apache's Avro are some popular service-oriented RPC platforms. All of these solutions are platform-neutral and have good performance on client/service architectures where the cost of constructing and transporting messages is negligible compared to the actual service they facilitate.

None of the solutions found had the right balance of transparency to the programmers, performance, and security. Language-specific RPCs had the best support for transparency and ease of use, but are not necessarily as performant as we wanted and were not easily portable to other environments. Some of the existing solutions being able to achieve a very low overhead for RPC calls and enabling throughputs up to several thousand messages per second, but required a multithreaded, fully asynchronous programming model, which does not work well for interactive tasks which are inherently synchronous. More highly performant platforms, like the LIMAX Disruptor even require a different programming paradigm to achieve their goals. *libinteractive* was created as a completely transparent, secure, and relatively performant RPC code generator and library.

3. Architecture and Design

libinteractive, much like any other RPC system, consists of three core components: an Interface Definition Language, a compiler that can convert IDL files into code and metadata, and the actual RPC mechanism used to communicate between processes and signal them. An optional component is provided to improve throughput when running on a Linux system: a kernel module that reduces the overhead of the RPC mechanism without compromising its security and the isolation between processes.

3.1. *The Libinteractive IDL*

The interface definition language chosen by *libinteractive* is based heavily on Web IDL (W3C, 2012), developed by the W3C as a way to express interfaces in JavaScript/ECMAScript and then later used by all web browser vendors in their own documentation. Its syntax resembles Java and allows for attributes to describe properties of various elements of the interface.

The building blocks of a *libinteractive* IDL file are the interface blocks, which describes what procedures or functions are implemented by which of the processes, which

are written just like interfaces in Java. The type system is similar to Java's but in order to better support multiple languages, there are a few restrictions on the types, which closely match the C language's semantics and limitations. There are six primitive types that can be used as parameter or return types: `bool`, `char`, `short`, `int`, `long`, and `float`. Procedures use the special return type `void`. Single- and multi-dimensional arrays of any of the primitive types can also be used for parameter types given that all their dimensions except possibly the first are compile-time constants. In the cases where an array dimension is variable, it must be passed in as a parameter that comes before in the parameter list, and it must have a Range attribute describing the lower and upper bounds of the value of that number in order to calculate the maximum size in bytes of the array. For instance, the task Parrots from IOI 2011 (Fakcharoenphol, 2011), can be described with the following IDL snippet:

```
interface Main {
    void send([Range(0, 65535)] int n);
    void output([Range(0, 255)] int n);
};

interface encoder {
    void encode([Range(0, 64)] int N, int[N] M);
};

interface decoder {
    void decode([Range(0, 64)] int N, [Range(0, 320)]
int L, int[L] X);
};
```

In the above example, for the `encoder.encode` procedure, it is known that the array `M` can have up to `N` elements, which in turn must be an integer between 0 and 64. This information is used to perform runtime parameter validation, as well as simplifying the protocol and memory management. By convention, the first interface in the IDL file is called `Main`, and it represents the program that the Task writer has created. The rest of the interfaces will be run in separate processes using the functions and procedures implemented by the contestant. This means that *libinteractive* supports isolating an arbitrary number of processes. `Main` is allowed to call the functions and procedures of the other interfaces, and the interfaces can call the functions and procedures of `Main`, but not any member of other interfaces. This is done to both simplify the design of *libinteractive* as well as to avoid cheating. The full syntax and semantics of the *libinteractive* IDL can be found on the project's documentation website[1].

3.2. *The libinteractive Compiler*

Once the task writer describes the contract between the main process and the one implemented by the contestant in the libinteractive IDL, the compiler can be used to generate

[1] `https://omegaup.com/libinteractive/`

all the files needed for contestants to compile, run, and test their solution against a set of predefined sample inputs. For a given platform and programming language, the compiler generates four sets of files:

1. A platform-dependent script that contains all the commands needed to compile and run the task: a Makefile, used in Linux/Mac OS X/Unix environments with the `make` command; or a .BAT file for Windows, that can be invoked directly as `run.bat`.
2. Language-dependent header files that can be included in both the task writer and contestant's programs, that expose the interface(s) that can be called.
3. Language-dependent utility functions that handle the serialization/deserialization of the parameters and handle all RPC invocation and signalling.
4. A platform-dependent run driver, written in C, which prepares the environment for the processes, executes them and prints out anything written to standard output/standard error. In Windows, the driver also orchestrates all the RPC communication between all processes.

Once the files are generated, they are packaged into a .zip file that contestants can download together with general instructions and examples.

One of the design goals of *libinteractive* is that it should be possible, if both the task writer and the contestant's source files are written in the same language, to compile them both into the same executable. This means that there is no additional syntax or unfamiliar semantics to be learned in order to write a *libinteractive* task or a solution for it.

The compiler is written in Scala, and is typically invoked as a standalone Java application, but it can also be used as a library. omegaUp (Chávez, 2014) leverages this and invokes the library to validate uploaded tasks, as well as generating all files that are to be consumed by contestants for all platforms and languages ahead of time.

3.3. *Execution Flow*

Programs compiled with libinteractive perform some initialization before handing control over to either the task writer's or the contestant's code. In this initialization phase, the RPC transports are created and prepared for communication, and then control is transferred to the normal program entrypoint (int main() in C/C++, public static void main(String[] args) in Java) for the Main process. Non-Main processes proceed to wait until Main interacts with them. Once execution reaches a point where a call to a function on another interface is made, all function parameters are serialized into a message in a transport-specific way, and is sent to the other process. The caller then waits until the callee acknowledges having finished execution of the call. The original caller then resumes execution and the callee goes back to waiting for a message. The acknowledgement itself is another message that might contain a return value, so all processes will be sending messages and waiting for a reply until one process terminates. If the process that terminates is Main, it is treated as a normal termination, otherwise it is an abnormal termination and the execution is treated as a failure.

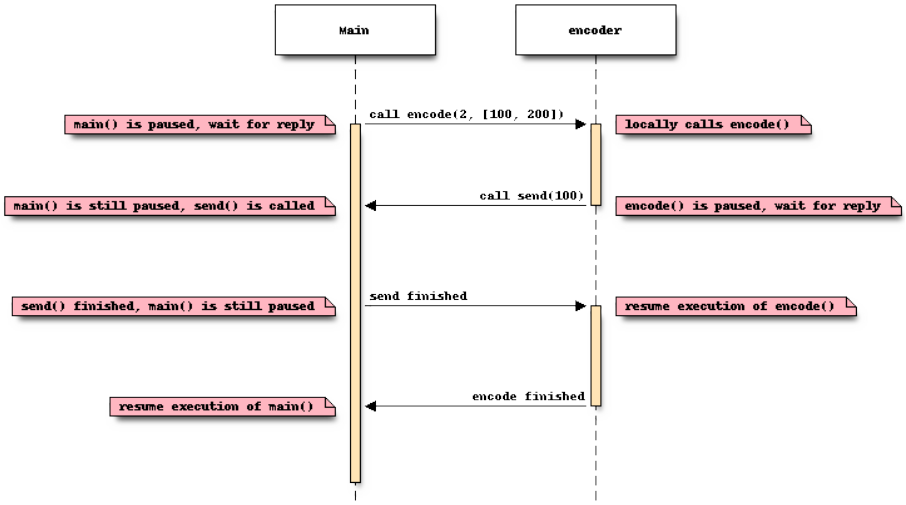


Fig. 1. Sequence diagram for message interaction in Parrots from IOI 2011.

One important thing to mention is that when a process is waiting for a message reply, a method call message can be received. This enables nested calls to be performed between the processes, as shown in Fig. 1.

3.4. *RPC Transports and Mechanisms*

The compiler can abstract away all of the lower-level details so that both task writers and contestants do not need to worry about exactly how the RPC calls are made, and in fact it is possible to choose from two available mechanisms: a cross-platform one that uses named pipes (Vaught, 1997) that contestants can use on their machines to test their solutions, and one designed for Linux based on shared memory (Stevens, 2003) that offers much lower roundtrip latency, but is not as portable and requires a Linux kernel module as described in the next subsection.

3.4.1. *Named Pipes*

Named pipes are available for all modern platforms, and they are simply streams of bytes that have endpoints in different processes. One process has the endpoint that can be written to and the other one reads from its endpoint. Pipes usually have blocking reads, which means that when one process reads from the pipe before it has any data in it, it will wait until the other process writes to it. This makes signalling easy, since each process is either doing computation or waiting for the other one make or return a call. The message encoding is also very simple, since the IDL mandates that all para-meters have a fixed size on compile time, or its size can be derived with only parameters that are already available: the binary representation of each parameter is written to the stream in a format compatible with C, which means that the memory of each parameter is copied directly

into the pipe one after the other. Arrays are serialized in row-major order. In order to distinguish between the different functions available to each interface, each function is assigned a unique, random 32-bit integer id during compilation, which is then prepended to each message before the parameter list. A 4-byte random cookie is appended to each message in the stream and then validated on the reply to avoid replay attacks. The actual data that goes through the pipe would then be similar to the one in Fig. 2.

Each interface pair has a pair of named pipes, one for outgoing messages and the other for incoming messages. Every time a message is received by one of the interfaces, it locally invokes the procedure or function and then sends the result of the function or a simple acknowledgement in case of procedures to the other pipe in the pair so that the caller can resume execution.

Named pipes do have one downside, that is shared amongst several of the RPC mechanisms outlined in the second section: they need to copy all the data into the pipe in one process, into the kernel, out of the kernel, and then out of the pipe for the other process. Typically this is done very fast for small messages shorter than 4 kb, but it becomes slower the larger the messages are. Also, since writes to a pipe are designed to not block unless the kernel buffer that receives the data from the pipe is full, the writing process will regain execution until it issues a read to the other pipe in the pair in order to wait for the response from the other process, which wastes a small amount of time.

3.4.2. *Shared Memory*

There is a second RPC mechanism available in all modern operating systems: shared memory. Two or more processes can request the operating system to allocate a flat memory location that can be accessed by all processes simultaneously[2]. Typically, shared memory is used together with a signalling mechanism that lets the other process know when it is safe to read from the shared memory without reading garbage, enforcing processes to take turns reading and writing the shared memory area. Given that the memory is never touched by the kernel, the cost of copying data around is greatly reduced. The overhead of the RPC call is now dominated by the cost of making a context switch between the processes, and can change depending on the signalling mechanism used. Semaphores and mutexes are the fastest ones available in typical Linux installs. One downside to using these synchronization primitives is that they are not easy to sandbox, since they require some extra system calls and access to a broader part of the filesystem

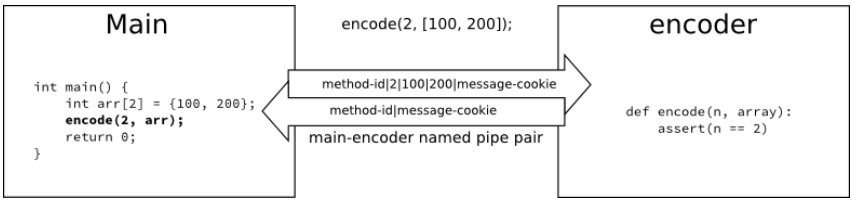


Fig. 2. Binary protocol for the named pipe backend.

[2] There are some caveats regarding cache coherence and consistency of the shared memory.

to be allowed in order to work properly. Programmers should also code very defensively when using them, since it is possible to deadlock if the process that currently holds the semaphore or mutex dies unexpectedly.

In *libinteractive*, since all messages have either fixed sizes, or contain range attributes to know the maximum size of any arrays passed as parameters, it is possible calculate an upper bound on message sizes on compile time. This information is used to implement a simple slab allocator (Bonwick, 1994) that allows for individual memory areas that are deallocated to be placed in a per-message-size linked list and quickly reused to avoid fragmentation. Since each interface handles its own shared memory, the number of message sizes has an upper bound on n, the number of functions and procedures in each interface. Given that most tasks are designed in a way that there is a constant amount of function call nesting across interfaces, at any given point in time there are at most $O(n)$ live messages and so the per-interface shared memory size is bounded. The generated code automatically manages all memory allocations needed, as well as validation of the input parameters and handling of all error conditions, so the whole process is transparent to that task writers and contestants.

Since the contestant's process also needs to modify the internal data structures of the allocator, all allocator calls are validated for consistency and the process aborts if it detects any modifications.

3.5. *transact Linux Kernel Module*

We must recognize that there is an unavoidable amount of overhead that is introduced by any RPC system. Most platforms deal with that through parallel processing, but interactive tasks are inherently serial and one process must wait for the other to respond before continuing. Most of this overhead comes from the context switching that the operating system must perform in order to stop running one thread/process and run another in a way that all processes are isolated from each other, but even that can be optimized: libinteractive also includes an optional Linux kernel module called transact that can provide significantly lower context switching cost in the scenario where there are exactly two processes switching back and forth from each other.

transact provides a fast, simple, file-based synchronization mechanism between exactly two processes that only uses the four most basic Linux system calls: `open` to acquire the lock, `write` to signal which of the two processes is the one owned by the task writer, `read` to make the context switch, and `close` to signal the other process that the current process is done with the lock and will shut down. Since the kernel manages the data structures that back the locks, it is possible to atomically force a context switch and yield control to the other process, reducing the overhead up to 20%. It is also resistant to deadlocks, since once a process is shut down, the kernel automatically closes all open files and will signal the other process that the other endpoint has died. By using transact and blocking thread creation at the sandbox level, it is possible to guarantee that there is exactly one process in each pair running at any point in time, so concurrent modification to the shared memory area is not possible.

Using transact is not a requirement for using *libinteractive*, but it improves performance significantly and allows to better measure the amount of time that the contestant's code is actually using to solve the task instead of being wasted in waiting for the kernel or serializing messages.

3.6. *Sandbox Compatibility*

libinteractive was designed with omegaUp's minijail sandbox in mind, which uses seccomp-bpf to do system call filtering, so it had to avoid using dangerous syscalls. It only relies on open, close, read, and write with named pipes transport, and additionally uses mmap when using transact. To also avoid having to relax filesystem sandboxing, a mode was added where all files that are to be shared among processes are all contained in a separate directory that can be mounted with read-write privileges (and additionally dev permissions with transact) in all sandboxed containers. This also makes it compatible with isolate (Mareš, 2012), the sandbox currently being used in IOI.

4. Performance Analysis

In a default, 64-bit install of Ubuntu Linux on a single-core virtual machine running on a AMD Opteron 4171 HE (used in some Windows Azure datacenters), we have measured that a well-written blocking RPC call has a wall-time context switch overhead of roughly 4.5–5.9 microseconds, depending on the RPC mechanism used to make the call. User time overhead (the CPU time spent executing the contestant's program exclusively) is much lower, on the order of around 0.5 microseconds. With the transact module, it is possible to lower both the user-time and wall-time overhead by 25%. Fig. 3 shows both user- and wall-time overheads for different message sizes.

Running interactive programs in multi-core machines is not recommended with *libinteractive*, since it makes the wall-time overhead significantly larger: over 10x in AMD processors. The operating system needs to ensure memory coherence between all processor cores and caches, so in the case where execution is transferred from one core to another and there is a data dependency between them, an additional synchronization step must be performed that further increases latency. Intel processors are also affected, but not as dramatically. If multi-core machines are to be used, it is possible to force all processes to run in the same CPU core by forcing their affinity, giving the same results as single-core environments.

Regarding variability of measurements, for small messages (<100 bytes), the user time overhead can vary up to ±0.4 microseconds per call in the worst case when both contestant and problemsetter processes perform negligible amounts of processing and the RPC costs dominate. The use of transact makes both the overhead and variability of measurements lower. Fig.4 shows a boxplot with the distribution of user-times for a 16-byte message.

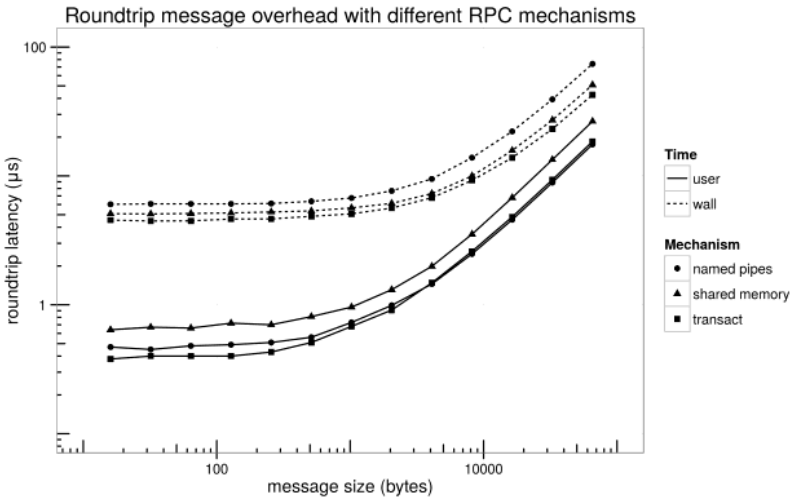


Fig. 3. User- and wall-time overheads for different message sizes.

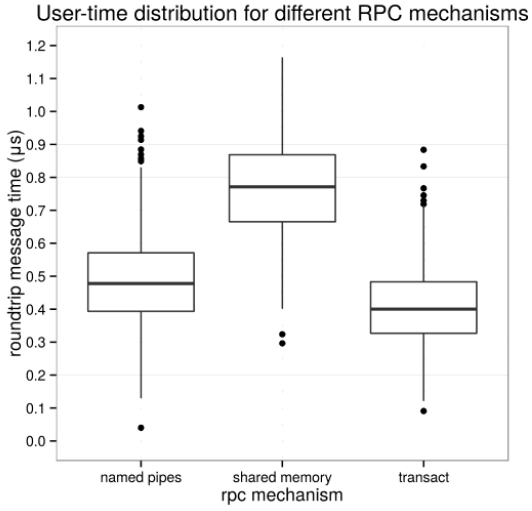


Fig. 4. User-time distribution.

One example of a good task for *libinteractive* is IOI 2013's Cave (Pouly, 2013), which has a limit of 70,000 messages per case and both contestant and problemsetter processes do non-trivial computation between each message. Both source code files were unmodified and were compiled in both a single binary and as *libinteractive* programs with the transact signalling mechanism. The single binary using the official solution finished in 23.91 s, with a user-time of 23.29 s. The *libinteractive* binaries finished in 38.19 s (+59.72%), with a user-time for the contestant process only of 20.14 s (-13.53%).

Despite the wall-time overhead, the user-time measurement was lower since all the processing related to reading the input file and performing validation was not accounted for. In general, tasks where the problemsetter code needs to do significant processing tended to fare better when run under *libinteractive*, whereas tasks with higher number of roundtrip calls tended to fare worse.

5. Conclusion

libinteractive is an excellent option to write interactive tasks that do not require a huge amount of roundtrips since it only requires a single source file in one language to be able to interact with the contestant's code. It is also platform-independent which allows contestants to practice writing solutions in the operating system they have access to. It was also designed to be sandbox friendly, and is compatible with both minijail and isolate. All code has been released through GitHub under the BSD license (except the transact kernel module which has a GPL license to match the Linux kernel's license)[3].

There are still a few things that we would like to do to improve the user friendliness of *libinteractive*, like IDE integration. Finding ways to further reduce the RPC overhead, especially on Intel processors, and supporting more data types like strings and structs/records are also high on the list.

Acknowledgments

The rest of the omegaUp development team and volunteers, especially Ethan Jiménez for his feedback during beta testing.

References

Birrel, A.D., Nelson, B.J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1).

Bonwick, J. (1994). The slab allocator: an object-caching kernel memory allocator. *In USENIX Summer*, 1994, 87–98.

Chávez, L.H., González, A., Ponce, J. (2014). omegaUp: cloud-based contest management system and training platform in the Mexican olympiad in informatics. *Olympiads in Informatics*, 8, 169–178.

Fakcharoenphol, J. (2011). Parrots. *The 23rd International Olympiad in Informatics*.

Mareš, M., Blackham, B. (2012). A new contest sandbox. *Olympiads in Informatics*, 6, 100–109.

NetMarketShare – Desktop Operating System Market Share (2014). `http://www.netmarketshare.com/operating-system-market-share.aspx?qprid`

[3] `https://github.com/omegaup/`

=10&qpcustomd=0&qpsp=2014&qpnp=2&qptimeframe=Y
Statista – Global market share held by operating systems Desktop PCs from January 2012 to December 2014 (2014). http://www.statista.com/statistics/218089/global-market-share-of-windows-7/
Stevens, R. (2003). *UNIX Network Programming*, Vol. 2, Second Edition: Interprocess Communications. Prentice Hall, 303–323.
Object Management Group (1991). The Common Object Request Broker: Architecture and Specification.
Pouly, A., Charguéraud, A. (2013). Caves. *The 25th International Olympiad in Informatics.*
Vaught, A. (1997). Introduction to named pipes. *Linux Journal*, #41.
W3C – Web IDL (2012). http://www.w3.org/TR/WebIDL/

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.

Olympiads in Informatics, 2015, Vol. 9, 15–26
DOI: http://dx.doi.org/10.15388/ioi.2015.02

Organising National Olympiads in Informatics: a Review of Selection Processes, Trainings and Promotion Activities

Sébastien COMBÉFIS[1,2], Alexis PAQUES[2]
[1] *Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM)*
Promenade de l'Alma 50, 1200 Woluwé-Saint-Lambert, Belgium
[2] *Computer Science and IT in Education ASBL, Belgium*
e-mail: s.combefis@ecam.be, alexis.paques@csited.be

Abstract. For a country to be allowed to send a national delegation to the International Olympiad in Informatics (IOI), it must organise a national competition to select the national delegation. In addition to the competition, trainings can also be proposed to the selected contestants to train them specifically for the IOI. How to organise the national contests is at the discretion of countries, so far as fairness is ensured among all the potential candidates. This paper reviews common structural elements and activities organised by several countries, for the advertisement of the national contest, the steps of the selection process and the proposed trainings. Specific activities added by countries according to cultural aspects or other country-specific characteristics are highlighted with the reasons motivating the integration of those activities. Based on the review, this paper summarises the key activities that could be organised by any country, with explanations about what they bring to the national contestants and motivation for their organisation.

Keywords: national Olympiad in informatics, national IOI delegation selection process, trainings activities and camps.

1. Introduction

When considering competitions as a tool to support and strengthen education, opinions differ, even though most people agree that education and competitions are closely related. It is natural for children to compete and competitions are also important in adult life; competitions should therefore be part of education (Verhoeff, 1997). In particular, all the activities revolving around competitions, and all the material produced, if used properly, enhance teaching and learning of concepts, such as in informatics, for example (Combéfis and Wautelet, 2014).

The first international Olympiad in the field of informatics was organised in 1988 by the Association for Technical Culture of Slovenia (Zrimec, 1989). The *International*

Olympiad in Informatics (IOI) is a competitive programming competition that supports the education of informatics, recognised and supported by UNESCO, launched in Bulgaria, in 1989 (Manev *et al.*, 2007). In particular, this competition is a good force for promoting programming and algorithm design fields of computer science. It is also an opportunity for countries all over the world to promote and push informatics in the education of young pupils, through the organisation of *National Olympiad in Informatics* (NOI). Many other regular international Olympiad in informatics are organised such as the *Balkan Olympiad in Informatics* (BOI) launched in 1993 at the initiative of Romania, the *Central-European Olympiad in Informatics* (CEOI) first organised in 1994 and again founded by Romania gathers nine countries, and the *Baltic Olympiad in Informatics* (BOI) created in 1995, which started with only three participating countries and now encompasses about 60 participants from nine countries.

This paper reviews how NOIs are organised in various countries, and how they are used to foster the spread of informatics in schools. It also puts forward good practices and highlights difficulties encountered by some countries.

Section 2 summarises the goals of NOIs, the main organisational and promotional difficulties and the structure of institutions being in charge of organising NOIs.

Section 3 reviews the selection process and, more precisely, it examines how the candidates are graded and selected to be part of the national IOI delegation.

Section 4 covers the trainings proposed to the contestants and the national delegation, before the IOI. It also sums up additional activities organised in countries to promote informatics and attract pupils to take part in the NOI.

Finally, the last section concludes the paper with some open questions and suggestions to improve the overall participation to the various existing national/regional/international Olympiads in Informatics and to better disseminate informatics in schools.

2. National Olympiad in Informatics

Each country that wants to send a national delegation to the IOI must organise a national contest to make the selection of the national IOI delegation. This section summarises the goals and the organisation of the National Olympiads in Informatics (NOIs) as well as how they are promoted in the country.

2.1. *Goals of the National Olympiads in Informatics*

According to the institutions organising the NOI in various countries, informatics Olympiads play an important role in the introduction of informatics, in particular of programming, in secondary schools. In some countries, the NOI is the biggest ICT related competition, such as in Mongolia, for example (Choijoovanchig *et al.*, 2007).

The main goals of the NOI are similar for most countries, that is, to encourage the teaching of programming in schools. More specifically, the main goals can be summarised as follows:

- The NOI can stimulate the interest for informatics and programming among secondary school pupils and teachers. In China, for example, it has been demonstrated that the NOI plays a role in the promotion and popularisation of information technology in secondary schools (Wang *et al.*, 2007).
- Talented teachers and pupils are brought together thanks to the NOI, which fosters the promotion of the contest in schools and collaboration between several institutions (schools, universities, associations and ministries). It is therefore a way to identify those talented pupils, and encourage them to pursue further opportunities in the profession and enrol in computer science related programs at universities.
- All the educational material produced for NOI, mainly tasks with solutions, syllabuses, training material or handbooks, can be used to motivate teachers to start activities related to informatics in their schools.
- Finally, the most obvious goal is simply to find talented pupils that will be part of the national IOI delegation and will succeed in bringing back medals from the IOI.

2.2. *Promotion of National Olympiads in Informatics*

Organising a nationwide contest and promoting it among teachers is not an easy task, as highlighted in (Pohl, 2007). A direct consequence is a low participation rate to the NOI. Several reasons have been highlighted:

- The responsibility for schools is not always at the level of the nation, which makes NOI difficult to promote, such as in Germany or in Belgium, for example (Combéfis and Leroy, 2011; Mukund, 2013; Pohl, 2007).
- There can be a lack of a centralised contest organisation, which makes it difficult for teachers in schools to decide which contest to recommend to their pupils. Indeed, the market of contests is large since NOI is not the only prestigious scientific contest with a corresponding international Olympiad, and NOI is not the only informatics related contest in most countries.
- Informatics is not a mandatory subject in schools – it is taught at most as an elective subject – which means that pupils do not know how to program nor have any knowledge in algorithm design and consequently do not participate in programming contests. A direct side effect is that there are only few professional informatics teachers, making the level of informatics education in schools very low.
- Financial issues can also arise in some countries. For example, Finland lost its main sponsor, which vanished into its possibility to organise a three-step NOI (Koivisto, 2013). The country had to consider using online competitions instead of in school contests – this dramatically reduced the number of contestants.

A direct consequence of those issues is that some potential talented candidates may miss the opportunity to participate in the national Olympiad and therefore be part of the national IOI delegation, just because they was not aware of the informatics Olympiad.

They may have discovered a passion for programming and may have shown good programming skills, but are instead just pursuing a normal life, without informatics…

In addition to these difficulties related to the organisation of the NOI, it is not always easy for some countries to participate and send a delegation to the IOI. Several reasons have been put forward:

- Pupils from some countries have insufficient English skills, which make it difficult for them to understand the tasks and to use online resources such as online Olympiads.
- The financial situation can also prevent a country from participating at the IOI every year: the low annual budget can make it impossible to cover all annual expenses. For example, Japan was not able to participate to all the IOIs (Tani and Moriya, 2008).
- It is not always possible for people from developing countries to get a visa to enter some developed countries, preventing them to participate to some IOIs. For example, Mongolia and Bangladesh were not able to attend several IOIs (Choijoovanchig *et al.*, 2007; Kaykobad, 2013).

It is not easy to address these issues, as they depend mainly on political decisions and on the economic situation of countries. Nevertheless, countries have taken some actions. Some countries are saving their money and decided to only participate to some IOIs. For the visa issue, some countries are participating online and then compare themselves to others thanks to the public rankings.

Promoting the NOI is also difficult. Contacts have to be found within schools, and human resources have to be allocated to go to schools and explain to the teachers what is the informatics Olympiad about. Some countries have developed interesting promotional materials to spread the word about informatics:

- Bangladesh convinced a newspaper to allocate space to publish math puzzles and problems for their young readers (Kaykobad, 2013). This action was a large success as thousands of pupils stormed into the office of the newspaper with their solutions. Doing the same with small algorithmic problems could be a good promotion vector.
- Georgia started developing an online Olympiad thanks to the Olympiad alumni, now working as professional programmers at various companies who gathered money from several sponsors (Mandaria, 2013). This platform allows contestants to compete more frequently and is a useful tool for identifying talented pupils.
- Development of books with materials about informatics, solved tasks with detailed solutions and explanations, and theoretical concepts related to programming, time complexity and algorithm design, for example (Mandaria, 2013).

Motivating pupils to participate at NOIs can be done in several ways. For example, Thailand is offering scholarships for contestants selected for the national IOI delegation (Malaivongs, 2013). Most countries offer medals and/or prizes to contestants that perform well on several stages of the NOI.

2.3. *Organisation of National Olympiads in Informatics*

Depending on the country, the NOI is either organised by a single institution or cooperatively by several institutions. In most countries, a national association has been created to promote informatics, such as the Croatian Computer Science Association, the Computer Society of Macedonia, the Italian Association for Informatics, the China Computer Federation or the Institute for the Promotion of Teaching Science and Technology in Thailand, for example. A committee is then put in place, with collaborators from the Ministry of Education, universities, high schools and ICT industries, to organise the national Olympiad.

Several aspects have to be taken into account when organising and managing a NOI. The separation of roles is more or less clear, depending on the country. For example, the organisation of the Italian Olympiad in Informatics is split among three groups (Casadei *et al.*, 2007), which occurs in many countries:

- The **scientific group** is responsible for the definition of the selection process and trainings, the composition of the national team for the IOI delegation.
- The **administrative group** takes care of the contacts with the schools and handles the logistics of the selection process.
- The **technical group** is in charge of the creation of tasks, the evaluation of the programs written by the contestants and has to organise teaching and trainings for the winners of the national Olympiad.

Another potential difficulty for some countries is related to the schedule. The IOI usually takes place during July or August, which is not holiday in some countries, such as Thailand, for example (Malaivongs, 2011). Many countries must adapt their schedules to the national IOI delegation process.

3. Selection Process

The selection process for the national IOI delegation varies by countries, though they share several key points. In most countries, the final of the national Olympiad is simply a small-scale copy of the IOI, which is preceded by multiple selection steps. Some countries have added pen-and-paper rounds in addition to more traditional computer rounds, such as in Belgium (Combéfis and Leroy, 2011).

The format of the NOI is chosen according to the skills that the country organisers want their contestants to develop. For most countries, the NOI is focused on algorithm design and practical programming skills; the contest is therefore centred on solving tasks on a computer. But some countries, such as Slovakia, are focusing on the thinking process (problem solving process), arguing that it is what they will need in their future lives if they chose a career in computer science; the contest having therefore more paper rounds supervised by human judges (Forišek, 2007; Forišek, 2013).

One important point of interest for the selection process for many countries, such as Latvia and China, for example, is the establishment of strict rules that define how the national IOI delegation is selected (Opmanis, 2013; Wang *et al.*, 2010).

3.1. *Organised Contests and Levels*

The selection process for the national IOI delegation expands through one or two years, depending on the country. Limiting the duration of the selection process to one year makes it easier to have candidates participating more than once to the IOI. Extending the selection process for two years allows the candidates to be more trained, which increases the chance to reach first positions in the ranking during the IOI. In most countries, the national Olympiad is split into several stages starting with local contests, followed by the provinces/regional contests, and finally leading to one nationwide final.

Here are the most common stages of national Olympiads:

- **School competitions** are organised in schools by teachers, and are generally a compulsory step, but without a qualifying meaning, such as in Latvia (Opmanis, 2013). Such a stage is very useful for promotional purposes, and is rather easy to organise, as the teachers handle it locally. It allows pupils to test their willingness to participate in the national Olympiad. Practically, this can either be a pen-and-paper contest, which is easier for the teacher, or a computer-based contest, in which case the best solution is to propose an online platform to support the contest.
- **Regional competitions** are organised by regions/provinces/districts, and have a qualifying status. The main goal for contestants is therefore to qualify for the next level of competitions. The qualified contestants are generally selected based on one unique nationwide ranking. However, in some countries, at least one contestant by region is selected, for promotion and equity purposes. Such a stage is generally organised into a set of schools where enthusiastic teachers supervise the contestants that participate in the contest through an online platform.
- **Country competition**, often referred to as final, is organised as an on-site competition whose location can change every year, generally hosted by a local university or university college.
- **Selection competition** is an additional level of competition that some countries organise to select contestants that will be part of the national delegation for regional contests such as the BOI or CEOI, and for the national IOI delegation. That stage is usually very similar to the IOI, namely a two-day on-site competition based on programming tasks to solve with a computer.

Some countries do not work in such a structure. For example, Thailand only has several nationwide contests interleaved with training camps, which results in the selection of about 100 contestants for the NOI (Malaivongs, 2013). Another quite spread habit is to organise some of the selection steps as online contests as it is the case in Japan and Germany, for example (Pohl, 2007; Tani and Moriya, 2008). Finally, some countries,

such as India, are explicitly organising two separate contests in the first stage, one pen-and-paper style to test algorithmic insight and one on computers to test programming skills (Mukund, 2013).

Table 1 shows a summary of an average participation rate, in term of the number of contestants, for the different stages of the NOI in several countries. It reveals that some countries are trying to have widespread local competitions, to reach a lot of pupils (mainly for promotion) and to allow the highest number of them to enter the competition, whereas other countries are more focused on the selection for the IOI, directly starting with regional or final competition.

3.2. *Grading Systems*

Being able to automatically grade the programs produced by the contestants is very important for a good NOI. In some countries, produced code is inspected and graded manually by a jury, either with a precise grading scheme, or with a less systematic grading consisting of the attribution of a numerical score to the proposed solutions (Pohl, 2007).

Most countries have developed their own automatic grading systems able to safely execute code and to run them against test sets in order to establish a scored ranking (Chávez *et al.*, 2014; Kostadinov *et al.*, 2010; Maggiolo *et al.*, 2014; Mareš and Blackham, 2012; Zhao *et al.*, 2013).

Many countries adopted *Contest Management System* to organise their contests, a distributed system for running and organising a programming contest (CMS, 2015; Maggiolo *et al.*, 2014). The main concerns of such graders are flexibility, efficiency, safety and security, independence to the programming language, and accuracy in execution time measurements. More general grading systems whose main goal is to be

Table 1
Participation to the different stages of the NOI for several countries

Country	Rounds	School/Internet	Regional	Final	Selection
BE	3	–	~150	~40	~15
BR	2	~8,000	–		
CN	4	~80,000	~12,000	?	?
FI	3	~4,000	~200	~20	–
IN	3	–	~8,000	~300	~25
JP	2	~250	–	~50	~15
MX	3	~15,000	~2,000	~100	–
RO	3	?	?	~400	–
RS	5	~350	~150	~75	?
SK	2–3	~150	?	~30	~10
TH				~100	–

embedded in learning platforms that can be used to train pupils have also been produced more recently (Combéfis and le Clément de Saint-Marcq, 2012; Urbančič and Trampuš, 2012).

3.3. *Scoring and Selection of the IOI Delegation*

Most countries do have a simple way to select the four pupils to form the national delegation for the IOI, just selecting the four candidates having obtained the largest scores during the final round of the national Olympiad. In some countries, as in Macedonia, the sum of all the scores obtained at each stage of the national Olympiad is used to select the IOI delegation (Janceski and Pacovski, 2007).

In China or in Germany, candidates have the opportunity to defend themselves during an oral defence in front of a jury, as part of the selection process (Wang *et al.*, 2007; Pohl, 2007). This additional interview is a great opportunity to check the English proficiency level of the candidates, for example. Other criteria such as the previous year medallist at the national Olympiad, at regional Olympiads and at the IOI are taken into account in the selection process, for example in Latvia (Opmanis, 2013).

Finally, in some countries, an additional contest, whose style is the same as the one of the IOI, is organised for the contestants that got the best scores during the NOI, in order to select the four contestants that will represent the country at the IOI. Usually, that final contest takes place after a training camp lasting several days that allows pupils to learn advanced algorithms and sharpen their programming skills.

4. Trainings and Additional Activities

Training contestants at various steps of the NOI is important since informatics is not present in school curricula of most countries. The most widespread activity is training camps, but countries are also organising other kinds of activities more specifically, to promote informatics or to foster cooperation between countries. For example, centres of excellence have been deployed in Romania, developing trainings for six disciplines, one of which is informatics (Cerchez and Andreica, 2008).

4.1. *Training Camps*

Trainings camps are organised by many countries. These camps last from a few days to weeks and are mainly organised at different moments in the selection process. They are mainly organised once the national IOI delegation has been selected, to train them for the IOI. But there are also training camps organised before the NOI final, or camps dedicated to the best contestants from the NOI, to help in the selection of the national IOI delegation, such as in Japan (Tani and Moriya, 2008).

Some countries prepare their contestants for the IOI during several years, starting with the basics of programming, and then going to more advanced concepts in algorithmic methods. In addition to those more theoretical concepts, contestants are also training their code-writing skills. Trainings camps are mainly organised by teachers, but most countries also integrate former IOI contestants as tutors (Tani and Moriya, 2008). Usually, training camps are especially dedicated to train the contestants participating in NOI, or those selected for the IOI. Countries are starting to organise camps whose purpose is simply to spread informatics, providing introductory courses in computer science (Anido and Menderico, 2007).

In addition to those on-site supervised training camps, some countries encourage their contestants to take part in online contests organised by other countries or organisations, such as USACO, Chinese ACM-ICPC Online Judge, and Google Code Jam, for example. It is clear that using tasks from past IOIs is possible. Using those resources available online is a cheap way to propose trainings when lacking of human resources to supervise them (Combéfis and Wautelet, 2014). Correspondence camps, organised in Japan and in Slovakia, are a solution to do this supervision (Forišek, 2007; Tani and Moriya, 2008). It is clearly a good way to tackle the lack of qualified informatics teachers, that are only located in a few cities in most countries.

4.2. *Regional Olympiads and Cooperation*

Finally, in addition to the IOI, there exist other regional Olympiads, such as the Balkan Olympiad in Informatics (BOI), the Baltic Olympiad in Informatics (BOI), the Central-European Olympiad in Informatics (CEOI) and the French-Australian Regional Informatics Olympiad (FARIO), for example.

Allowing the best pupils of the national Olympiads to participate at those regional Olympiads is a good training as highlighted by several countries. Of course, money has to be found to cover the participation expenses, except for online contests such as FARIO, for example.

Generally speaking, more cooperation between countries should be put in place, since the goals of every country regarding informatics and education are mainly the same. As it is the case with Slovakia, countries could be collaborating to prepare problems and help for their NIO and to prepare training camps (Forišek, 2007).

4.3. *Promotion Purpose*

As previously stated, national Olympiads are often also used as a tool for promoting informatics in schools. Some countries have introduced innovations in the national contest to help this promotion purpose.

Latvia introduced a special “first subtask” for every task of the country competition that can be solved by hand without the need of finding and implementing an algorithm to

solve them (Opmanis, 2013). The main reason is to avoid the so-called "0-frustration" of contestants who understood the idea of the task, but were not able to write the algorithm to solve it. In Japan, four of the tasks of the first round are relatively simple, for the same reason (Tani and Moriya, 2008).

As highlighted in (Isal *et al.*, 2014), establishing clear roles for the stakeholders involved in the NOIs for the promotion purpose is very important. The authors propose four levels of stakeholders starting with the participants and alumni contestants followed by universities and government. Each of these layers has to advertise the NOI since they can have different impacts and reach different publics.

5. Conclusion

To conclude this paper, IOI and related activities lead by participating countries are useful for promoting informatics in schools. Nevertheless, Olympiads need supplemental activities that must be organised to reach the informatics promotion goal that is promoting computer science and fostering its presence in education.

An observation that is made in several countries is that members of the national IOI delegation can often attend only one IOI since they are too aged. Two main reasons explain that observation: informatics is not in the curriculum of primary (8–12 years old) and secondary schools (12–18 years old), and selection processes sometimes last two years. One possible solution is to start the competition earlier and organising mini and junior Olympiads, such as the Junior Balkan Olympiad in Informatics, or to propose promotion activities such as the junior summer camps organised in New Zealand (Phillipps, 2010). Australia also went into this direction with their *Australian Informatics Competition* (AIC) which is an entry-level pen-and-paper competition targeted to youngers (Clark and Clapper, 2014). Collaborating with other contests targeted to younger pupils, such as the Bebras contest (Futschek and Dagiene, 2009), can also help to attract more people to join NOIs and IOIs.

Another big issue raised by most countries is the lack of materials that can be used by schools' teachers and for the trainings of contestants (Ilić and Ilić, 2012). Some countries are doing the tasks that are used in NOIs publicly available. Solutions as well as explanations of the solutions must also be available for such material to be useful and exploitable.

This review highlights that most countries share common ideas in the organisation of their NOI. Having a local competition level, that is not necessarily mandatory, is useful for promoting informatics among pupils and teachers. The second important activities are training camps. Two kinds of camps are organised: interleaved camps during the NOI that teach pupils algorithms and programming and selection camps with the best contestants from NOI to choose the national IOI delegation. Both camps are very important and should be organised if possible.

Finally, maintaining alumni contestants involved in the organisation of NOIs is also a good solution to help promote. Alumni associations can be found as it has been done in Indonesia (Isal *et al*, 2014).

To conclude, five recommendations can be highlighted:

1. Promotion of informatics should *start earlier*, with initiatives and contests dedicated to younger pupils.
2. *Collaboration and links* with existing informatics contests, not necessarily related to programming, should be made.
3. Learning *materials* should be produced for teachers and trainers, to be used with their pupils and trainees.
4. *Training camps* and *entry-level contests* should be organised.
5. Relations with *alumni contestants* should be maintained to keep them involved in NOIs.

To work on those recommendations, more cooperation and collaboration should be established between countries, so they may get closer at reaching the common goal of spreading computer science and improving its presence in education.

References

Anido, R., Menderico, R. (2007). Brazilian olympiad in informatics. *Olympiads in Informatics*, 1, 5–14.

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

Cepeda, A., Garcia, M. (2011). Mexican olympiad in informatics. *Olympiads in Informatics*, 5, 128–130.

Cerchez, E., Andreica, M. (2008). Romanian national olympiads in informatics and training. *Olympiad in Informatics*, 2, 37–47.

Chávez, L., González, A., Ponce, J. (2014). omegaUp: cloud-based contest management system and training platform in the Mexican olympiad in informatics, *Olympiads in Informatics*, 8, 169–178.

Choijoovanchig, L., Uyanga, S., Dashnyam, M. (2007). The informatics olympiad in Mongolia. *Olympiads in Informatics*, 1, 31–36.

Clark, D., Clapper, M. (2014). The Australian informatics competition. *Olympiads in Informatics*, 8, 179–189.

CMS (2015). *Contest Management System*. `https://cms.readthedocs.org/en/v1.2/`

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.

Combéfis, S., Wautelet, J. (2014). Programming trainings and informatics teaching through online contest. *Olympiads in Informatics*, 8, 21–34.

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

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*.

Ilić, A., Ilić, A. (2012). IOI Trainings and Serbian competitions in informatics. *Olympiad in Informatics*, 6, 158–169.

Isal, Y.K., Liem, M.M.I., Mulyanto, A., Marshal, B. (2014). Indonesian olympiad in informatics: significant advancements between 2010 and 2014. *Olympiads in Informatics*, 8, 191–198.

Janceski, M., Pacovski, V. (2007). Olympiads in informatics: Macedonian experience, needs, challenges. *Olympiads in Informatics*, 1, 66–78.

Kaykobad, M. (2013). Bangladesh olympiads in informatics. *Olympiads in Informatics*, 7, 163–167.

Koivisto, J. (2011). The national computer olympiads and the IOI participation in Finland. *Olympiads in Informatics*, 5, 147–149.

Kostadinov, B., Jovanov, M., Stankov, E. (2010). A new design of a system for contest management and grading in informatics competitions. In: *Web Proceedings of ICT Innovations 2010*. 87–96.

Maggiolo, S., Mascellani, G., Wehrstedt, L. (2014). CMS: a growing grading system. *Olympiads in Informatics*, 8, 123–131.
Mandaria, G. (2013). Olympiads in informatics: the Georgian experience. *Olympiads in Informatics*, 7, 168–174.
Malaivongs, K. (2011). Preparing students for IOI: Thailand country report. *Olympiads in Informatics*, 5, 150–154.
Manev, K., Kelevedjiev, E., Kapralov, S. (2007). Programming contests for school students in Bulgaria. *Olympiads in Informatics*, 1, 112–123.
Mareš, M., Blackham, B. (2012). A new contest sandbox. *Olympiads in Informatics*, 6, 100–109.
Mukund, M. (2013). The Indian computing olympiad. *Olympiad in Informatics*, 7, 175–179.
Opmanis, M. (2013). Latvian olympiad in informatics – lessons learned. *Olympiads in Informatics*. 7, 78–89.
Phillipps, M. (2010). The New Zealand experience of finding informatics talent. *Olympiads in Informatics*, 4, 104–112.
Pohl, W. (2007). Computer science contests in Germany. *Olympiads in Informatics*, 1, 141–148.
Tani, S., Moriya, E. (2008). Japanese olympiad in informatics. *Olympiads in Informatics*, 2, 163–170.
Urbančič, J, Trampuš, M. (2012). Putka – a web application in support of computer programming education. *Olympiads in Informatics*, 6, 205–211.
Verhoeff, T. (1997). The role of competitions in education. In: *Proceedings of the Future World Educating for the 21st Century Conference and Exhibition*.
Wang, H., Yin, B., Li, W. (2007). Development and exploration of Chinese national olympiad in informatics (CNOI). *Olympiads in Informatics*, 1, 165–174.
Wang, H., Yin, B., Liu, R., Tang, W., Hu, W. (2010). Selection mechanism and task creation of Chinese national olympiad in informatics. *Olympiads in Informatics*, 4, 142–150.
Zhao, Q., Wang, F., Yin, B., Sun, H. (2013). Arbitrer: the evaluation tool in the contests of the China NOI. *Olympiads in Informatics*, 7, 180–185.
Zrimec, M. (1989). The report of Slovenian association of technical culture organisation. In: *Proceedings of the International Congress on Education and Informatics: Strengthening International Cooperation*. ED.89/WS/62.

Dr. S. Combéfis obtained his PhD in engineering in November 2013 from the Université catholique de Louvain in Belgium. He is currently working as a lecturer at the École Centrale des Arts et Métiers (ECAM), where he is mainly teaching informatics. He also got an advanced master in pedagogy in higher education in June 2014. 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.

A. Paques is studying electronics engineering at École Centrale des Arts et Métiers (ECAM), in Brussels. He is now a first year master student. In addition to his passion for engineering, he likes programming. He has also been involved in the Bebras final in 2015, where he taught informatics to secondary school students. Since then, he has started to contribute to several projects related to informatics teaching for pupils.

Olympiads in Informatics, 2015, Vol. 9, 27–37
DOI: http://dx.doi.org/10.15388/ioi.2015.03

Methodology for Characterization of Cognitive Activities when Solving Programming Problems of an Algorithmic Nature

Gilberto CUBA-RICARDO, María T. SERRANO-RODRÍGUEZ,
P. Alberto LEYVA-FIGUEREDO, Laura L. MENDOZA-TAULER
José de la Luz y Caballero University of Pedagogical Science of Holguín. Cuba
e-mail: {gilberto.cuba, mariat, albertoleyva, laura}@ucp.ho.rimed.cu

Abstract. This paper shows a methodology for characterization the students' cognitive activity when solving programming problems of algorithmic nature. It also reveals the methodology stages and the dimensions assumed to assess the process using several methods, techniques and tools.

The paper describes some characteristics and regularities from the behavior of three students when they solve a programming problem.

Keywords: programming contests, problem solving, metacognition.

1. Introduction

Programming contests are constituted by problems that contestants must solve in a short period of time in relation to its level of complexity (Verhoeff, 1997). Hence, some of the results obtained are less than what it is expected from a student.

The informatics coach must diagnose the students' development stage when mastering these algorithms. They must also diagnose their skills when solving these kinds of problems. These elements constitute a challenge from a pedagogical point of view. As it is important for the coach to influence and improve students' work, identifying their deficiencies is critical.

The source code constitutes the solution given by the students to a specific problem. The worksheets for such contestant are the space where they write some models or techniques related to the problem. This source code and their worksheets don't allow for the complete analysis of their skills and shortcomings. This is because the elements are results of the final state of a process lasting between 60 and 90 minutes. Consequently, a current issue is the study of the cognitive activity developed by the student when solving programming problems (Deek *et al.*, 1999; White and Sivitanides, 2002).

Specifically, Hosseini *et al.* (2014) show a new approach for assessing the exercise solutions. Here, they check the intermediate steps during implementation of an algorithm solution. This alternative allow Hosseini *et al.* (2014) to conclude that the study of these intermediate steps helps the coach to know the most common paths used by students when solving programming problems, and thus, provide better feedback to students. However, these authors only consider at each intermediate step, the source code developed by the student.

Surakka and Malmi (2004) list the cognitive skills that successful computer programmers must have when solving programming problems. These cognitive skills are the results of many experienced programmer opinions through the application of two rounds of the Delphi method. However, programmer opinions were not contrasted with information gathered from other practical methods.

In such case, the study of cognitive activity when students solve programming problems should not only consider the analysis of the aforementioned process results, but also the assessment of the thinking states of the students while they solve the problem.

In this sense, considering the possibilities offered by the computer to register the interactions given by the user in short periods of time, it is decided then to search for an alternative to register the students' behavior while solving programming problems. In addition, a methodology was developed to guide the process considering some process and outcomes indicators.

The methodology was applied to three senior high school Informatics contestants.

2. Screenshot Software

When seeking informatics applications that met the register requirements, some were found which were oriented to monitoring and registering the traces left by the user during their interaction with the computer. Their usage is supported by a conception involving and compromising the computer's security for malicious purpose. It is very common to recognize them as malware, or relate them to monitoring processes in enterprises that monitor informatics security. However, the application explained below, has different purposes in relation to the scientific research, mainly, to reveal behavior when solving programming problems of algorithmic nature.

Without going further on the security matter, some of the names that identify them are *Spyware, Spybot, Keylogger, Computer Monitor, PC Audit*, etc. Almost all of those found are registered under owners' license and with a great amount of setting options which slow the output of the computer. Plus, it makes usage for more specific purposes like only recording screenshots in short periods of time difficult.

Thus, a very simple informatics application was developed, which allowed the accomplishment of the stated objectives without reinventing those that already have their established goals related to the malware or security. Nevertheless, the core of the research wasn't the development of a new sophisticated application that permitted the registration of the entire user's interaction with the computer. Up to this moment, we

have only implemented the registration of snapshots of the computer desktop state over short periods of time.

The name given to this application was *Screen Shooter System Tray*. It has an executable that can be instantiated from any location in the computer. Installations requirements do not depend on frameworks or third party libraries to allow execution. It only runs on Microsoft Windows XP or later versions.

Its visual recognition is presented as an icon in the computer's operating system's tray, next to the clock. It has a context menu with the basic options to enable and disable the recording. After the execution of the application, the image recording is saved by default in the same folder of the executable file, with a one-second difference between screenshots.

Since the number of images saved in one hour is greater than 3600, it is necessary to reduce image sizes. For that, images are saved as JPEG format in grayscale with 8 bits depth of colors.

3. Methodology for Characterizing the Cognitive Process of Solving Programming Problems

The methodology used was not only aimed at a final assessment of such a process. There are several manual methods to do so, allowing checking the solution of an exercise with the data sets produced with the objective of getting a grade. There are also Internet Websites, that devote their content to publishing exercises and evaluating the solutions submitted by the users. (Kolstad and Piele, 2007; Revilla *et al.*, 2008; Verhoeff, 2008; Mares and Blackham, 2012; Maggiolo *et al.*, 2014).

The objective of the present methodology was to determine the characteristics shown by students during the programming problem solving process, at the time they evaluate the quality of the final outcome, expressed in the source code of the problem's solution algorithm. For a better understanding and materialization of it, the methodology was organized into five stages:

1. Preparation for the process.
2. Recording the process of exercise application.
3. Analysis, processing and assessing of the partial reports (observation, desktop pictures, source code of the algorithm solution).
4. Interview session.
5. Final assessment.

In this regard, deep studies on the characterizations of the student's cognitive activity while solving problems have no recent antecedents in the area of mathematics in the works of Shoenfeld (1985) and Cruz (2002). On this basis, common features have been analyzed and certain procedures, which characterize the programming problem solving process, have been imported from Mathematics.

In Informatics, specifically in the Programming area, the work of Deek *et al.* (1999) has been found. These authors developed their research in the introductory courses of

problem solving and programming taught to university students from the New Jersey Institute of Technology. They used a method to assess processes developed by the students, which allowed them to readjust the content and related teaching methods, supported by a six-stage model.

Their arguments allowed us to clarify the variables to consider in the characterization process, and we used some of the indicators presented in their work. However, their instruments are not used, since they do not take advantage of the interview potential with each student.

Specifically, for the implementation of the methodology, we recognized that training and developing metacognitive processes in the students, has an important role in the solving programming problems of an algorithmic nature. Hence, one of the dimensions taken into consideration was *metacognition*. The other one was the use of *complementary tools* that help the contestant solve the problem. The third one was *solving the stated problem*, which is a source code in a Programming Language (PL) reflecting an individualized representation of the solution algorithm.

The first two dimensions have a process nature, while the third one is a result. This demonstrates the objective of the methodology, essentially aimed at the characterization of solving programming problems of an algorithmic nature.

3.1. *Stage (I) Preparation for the Process*

A careful selection of the exercise was essential to carry out the characterization process. For that, the previous knowledge of the students was taken into consideration, along with what it is needed from the problem itself to find a solution algorithm. It was also guaranteed that there were several solutions of the problems and the data sets which allowed us to assess the students' answers.

The exercise was prepared to be read by the student from a document reader in each computer. This permitted us to know through screenshots when the contestant consulted the document for the exercise. Furthermore, the computers' clocks were synchronized when applying the same programming problem at the same time.

When collecting the data which enables characterizing the process, three main methods were used: registration of participant observations, the screenshot recording and the interviews. The first two were developed while solving the problems and the last one, once the process was completed.

The purpose of the observation in the development of the activity, was aimed at describing in as much detail as possible, the students' behavior when solving problems. Hence, the registration of the observation focused on writing what the students did in every time interval. For that reason, it was designed as a three-column table containing the description of the action, including the start and end time.

While solving the problem the student took some notes and reflected. These reflections are show when he is not reading the exercise or not writing on the worksheet. To be sure of the second action, the students were asked some questions.

3.2. *Stage (II) Recording the Process of Exercise Application*

This activity began using the proposed application that captures the computer screenshots used by the students (*Screen Shooter System Tray*), at the time it gave them the start command. From then, every observation moment was registered on worksheets. The students' doubts were also assisted regarding the interpretation of the problem, which were also registered like the other actions. Doubts were essential elements to determine the students' comprehension levels in relation to the exercise. This obviously helped orient the questions of the interview.

With a little difficulty some of the students' behavior was also registered such as anxiety, despair, uneasiness, satisfaction, success, joy, fear, defeatism, etc, which allowed a comprehensive assessment of the process. Some of these behaviors were identified through questions, because it was difficult to diagnose from the observation.

Likewise, the students were told when they had 5–10 minutes left to finish the exercise like in any other contest of this kind. At this point, the students' decisions changed, which were also registered as much as possible in order to fulfill the assessment strategy developed by them when solving problems under pressure.

When the time dedicated to do the task finished, the application recording the screenshots was stopped. The images were collected along with the programs source code to be processed. Worksheets were also gathered and added to each student's observation sheet.

The students were not allowed to tell each other their experiences until the interview process was over. This decision was mainly based on the fact that they could readjust not only their solving algorithm, but their behavior and opinions. It was considered that if this exchange of ideas could happen, the results of the students' answers in the interview would have altered.

3.3. *Stage (III) Analysis, Processing and Assessing of the Partial Reports*

It was decided that the time for the problem solving process, and the interview sessions wouldn't exceed 24 hours. This was based on the following: the larger the time between the two activities, the less the possibilities to really know the characteristics and cognitive behaviors developed by the students when solving the problems. If the time exceeded 24 hours, the students could forget why they determined certain decisions that were revealed during the observation process. Therefore, once the recording process was finished, every screenshot was further analyzed, and the interview began right after that, with the objective of clarifying those elements that were not very clear in the interpretation process, and to clarify some hypothesis that surfaced from the analysis process.

When assessing the previously declared dimensions, some indicators were taken into consideration that enriched the process in its abstraction. In the metacognitive

dimension, the indicators were focused on: the metacomprehension of the problem and the given solution, its planning, the conscious use of strategies and techniques to solve the problems, the self assessment of the steps followed and the whole process as such.

In relation to this foundation, the studies of Weinstein and Underwood (1985) were consulted. They proposed questionnaires representing inventories dedicated to assess the learning strategies, which, among others, metacognition can be found. The review of such questionnaires permitted formulating a guide to questions to be asked in the interview stage.

Most of these indicators were best suited to the interview stage. However, from the analysis of the desktop screenshots, along with the students' notes in the worksheet, plus the notes from the observation, the guide of questions and topics to ask during the interview were planned. As an example, some of the topics and questions asked of the students in the interviews were:

- Could you read and fully understand the exam exercise? How and/or when do you know you had correctly understood the exercise?
- Was the exercise difficult or easy to answer? Why? What elements did you take into consideration to determine the difficulty of the exercise? What was more difficult: understanding the exercise or searching for an algorithm is solution that fulfilled the requirements of the exercise, and/or to code in a PL the algorithm solution found?
- Did you feel ready to solve this kind of exercise? Was there any kind of content you had not mastered and felt you needed to solve the exercise? Which one? What do you do in such situation, when there is an exercise you cannot completely solve during a contest?
- Did you use any particular strategy to solve the exercise? Can you tell us about it? Have you yourself defined any steps to follow, or that you had already planned beforehand to solve the exercise? What techniques, tools, steps or ways did you use to solve the exercise successfully?
- When you found an algorithm is solution for solving the problem, how did you know it was correct to start to code it and that it will be successful? Or did you just start coding it and check at the end with the data sets if it is right or wrong?
- Was the exercise similar to others previously answered by you during the training period or in any other contest?
- What were your notes in the worksheets used for? Can you explain some of them?

Most of these questions have their own purpose. The first one, deciphers the characteristics of the students' thought, along the problem solving process; the second one, educational, evidenced when the student is introspective and consciously gains the knowledge of how certain processes are developed in his thought. Therefore, these interviews were very important because they make possible the development and training of metacognitive skills in students.

When analyzing the dimension related to the additional tools that enabled solving the

problem, it was considered that: the computer's operating system, the Integrated Development Environment (IDE), debugger and PL, are part of this group. When recognizing the skill level of the students' development while using these tools, the understanding of their basic concepts is also taken into consideration.

Some problems detected after the observation and the analysis of the screenshots, are presented below:

- The keyboard configurations do not allow a suitable output of the symbols needed to code the algorithmic solution in the selected language.
- Inadequate mastery and familiarization of IDE, which makes the necessary operations impossible to code with the required speed.
- IDE configurations do not facilitate a larger visibility of the source code and access to the most used objects.
- The debugger possibilities were wasted.
- Insufficient mastery of the PL used to code the algorithm solution, mainly evidenced when the student doesn't know about the range of the data types, when he doesn't recognize the syntax errors indicated by the compiler, and when he doesn't interpret the functionality of part of the program code.

Finally, when analyzing the final solution of the problem as a source code of such an algorithm, the following were taken into account: the total grading obtained during the program assessment, the source code style, the relations between the source code and the algorithm solution, and the data structure in their relation to code portions supporting the input, output and processing of the data.

The clarity of the source code, as a manifested characteristic of the solution program submitted, is related to the planning done by the student during the initial process of the solution conception. Consequently, the analysis of its links and the search of explanations by the students is needed. Its effectiveness was checked through the assessment of the program in execution with every data set.

As the final solution of the exercise was not very clear, it was necessary to investigate some matters related to the source code and the solution algorithm found. Upon this foundation, the following questions were planned:

- Explain briefly the solution algorithm found for the exercise. First of all, express it in a mathematical way according to the data and the unknown data. Then, explain yourself, demonstrating in praxis the use of the data structures, as well as the control structures used in the programming language.
- How did you determine the data structures you were going to use? Which do you determine first, the structures or the algorithm?
- When you started to code the program, did you do it thinking about the solution algorithm of the exercise? Did it change during the time used to solve it? Can you describe the main differences or changes produced between the algorithm as first conceived and the one delivered? How do you know if the coded program solves the exercise?

Once the interview guide is organized, it is added to the source code, the students' worksheet and the observations made in order to move on to the next stage.

3.4. *Stage (IV) Interview Session*

The interviews were individual and clarified the hypothesis and doubts observed from the students' behavior. The need to have devices that allow recording videos become evident. It was also verified, that during the activity other questions surfaced that should have been planned and arranged as part of the interview guide. Other matters that prevailed were the students' exhaustive explanations about the algorithm they followed, and the way they took it to the "new" worksheets.

At the end of the interview, the student was shown the algorithm planned as the solution to the exercise in case it did not match there, and they were asked for opinions about it. Likewise, the relation between the presented algorithm and the source code that solves it was explained. During this explanation, he was also shown some modeling techniques that enable a better representation of the information of the problem and the relations established between them.

The exemplification of this element not only shows to the student the way to solve the stated problem, as the interview not only gives the teacher those elements he must teach in order to improve the solution of the programming problem; but also, it allows the student to think of his metacognitive activity, of his techniques and strategies applied to solve the problem, and of his skills. In general, this kind of procedure influences the student self regulation from the very moment he knows how his knowledge is developed.

3.5. *Stage (V) Final Assessment*

Reaching this point, an exchange cycle with the contestant is closed, where most of the elements explained by him are corroborated with his worksheet, along with the registration of the observation.

In order to continue clarifying the problem solving process developed by the students, and after a first screenshot review, a further and exhaustive review was undertaken, where some elements were revealed constituting problems, wasting of time or difficulties that obstruct the speed of the problem solving process.

In general, after the analysis of several of the represented data, we concluded that the students demonstrated a tendency to think and solve the problem as they coded the solution algorithm. This aspect confirms the strong union between the construction of the problem solution algorithm and its code. (Deek *et al.*, 1999).

This process is evidenced when the student starts to code the solution of the problem; reinterprets the written code and compares it to his algorithm solution represented in his mind. Then, he corrects the possible mistakes little by little, and optimizes the source code in correspondence to the planned objectives.

This phenomenon is shown as an unfinished idea, and in the way the student implements the source code, he evaluates and controls mentally its parts. This interpretation is carried out through internal simulations, and such portions are closely related forming little algorithms that in its general structure, determine the algorithm solution of the exercise presented.

Taking into account this behavior, it was observed that students have difficulties when interpreting the problem, as well as when planning the search for the solution. Similarly, it was evidenced that leading the heuristic search in the construction of the solution algorithm, was made unconsciously and not in a self controlled way, which would facilitate the correct selection of the strategies to solve problem of this nature.

Throughout the process, it was also shown, that the students check the partial status of the functioning of the source code implemented, for example: data reading and writing, small algorithm of filling data, preprocessing and simple algorithms expressed as subroutines.

It was also concluded that the students demonstrate, in some cases, the classic use of trial and error strategies. This strategy is exhibited when making small changes in the source code, compile and run, data entry and retrieving the result, assessing the answer and evaluating and comparing the new answer with the previous ones and their corresponding source code changes (Cuba-Ricardo *et al.*, 2014).

In relation to the evaluations made, the cycled is repeated as many times as the student has patience for, new combinations have to be tested, or changes are made during the process.

4. Conclusions

Monitoring the students' behavior, allowed saving the construction process of the solution algorithm in its transcription or coding to a PL of the oriented objects paradigm. This aspect was later contrasted with the interview with the students, which helped reveal the students' notes in the worksheets, as well as the reasoning followed when determining the solution algorithm.

In general, the observation process of the source code construction, in its different evolutionary status, is of greater value for its analysis rather than observing the final program, due to:

- The analysis of the process along with its explanation by the student, promotes the effective valuation of the programming techniques and particularly that of problem solving.
- Along with the interview with the student, his way of thinking can be known.
- It allows determining which data structures or algorithms are more difficult for the student.
- It reveals the most common skills when using the tools that make possible the solving process of programming problems.

5. Future Work

The most important task remaining in our work is porting the software to an open source operating system. This allows the application of such methodology to another context,

and encourages us to search for other results that should improve the dimensions and indicators in each methodology stage.

On the other hand, the intention is to capture more images in less time intervals, and the best solution is to save screenshots as video format instead of images.

The final idea is to determine automatically some behavioral patterns manifested by the students when solving programming problems. This should avoid the need for extensive human review; and should be possible through new software, that receives the video as input and after processing the information it returns the time intervals in which these patterns appear. We must also define these patterns.

References

Cruz Ramírez, M. (2002). *Estrategia Metacognitiva en la Formulación de Problemas para la Enseñanza de la Matemática*. PhD, ISPH "José de la Luz y Caballero", Holguín.

Cuba Ricardo, G., Leyva Figueredo, P.A., Mendoza Tauler, L.L. (2014) Learning strategies of informatics contestants. *Olympiads in Informatics: International Journal*, 8, 35–48. http://www.ioinformatics.org/oi/pdf/v8_2014_35_48.pdf

Deek, F.P, Hiltz, S.R, Kimmel, H., Rotter, N. (1999). Cognitive assessment of students' problem solving and program development skills. *Journal of Engineering Education*, 88(3), 317–326.

Hosseini, R., Vihavainen, A., Brusilovsky, P. (2014) Exploring problem solving paths in a Java programming course. *Psychology of Programming Interest Group Annual Conference.* 65–76.

Kolstad, R., Piele, D. (2007). USA computing olympiad (USACO). *Olympiads in Informatics: International Journal.* 1, 105–111. http://www.mii.lt/olympiads_in_informatics/htm/INFOL016.htm

Maggiolo, S., Mascellani, G., Wehrstedt, L. (2014). CMS: a growing grading system. *Olympiads in Informatics: International Journal*, 8, 123–132. http://www.ioinformatics.org/oi/pdf/v8_2014_123_132.pdf

Mares, M., Blackham, B. (2012). A new contest sandbox. *Olympiads in Informatics: International Journal*, 6, 100–109. http://www.mii.lt/olympiads_in_informatics/htm/INFOL094.htm

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

Schoenfeld, A.H. (1985) *Mathematical Problem – Solving*. New York, Academic Press.

Surakka, S., Malmi, L. (2004). Cognitive skills of experienced software developer: Delphi study. In: Korhonen A., Malmi, L. (Eds), *Kolin Kolistelut–Koli Calling 2004. Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education.* Finland, Koli, 37–46

Verhoeff, T. (1997). *The Role of Competitions in Education*. Paper presented at the Future World: Educating for the 21st Century. A Conference and Exhibition at IOI' 97. http://olympiads.win.tue.nl/ioi/ioi97/ffutwrld/competit.pdf

Verhoeff, T. (2008). Programming task packages: peach exchange format. *Olympiads in Informatics: International Journal*, 2, 192–207. http://www.mii.lt/olympiads_in_informatics/htm/INFOL019.htm

White, G.L., Sivitanides, M.P. (2002) A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. *Journal of Information Systems Education*, 13(1), 59–68.

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.

M.T Serrano-Rodríguez is a Specialist in Pedagogy and Psychology, and Bachelor of Education specializing in Chemistry. She presents publications in scientific events as FIMAT, ENFIQUI and Pedagogy, with themes related to ICT and Chemistry. She is a researcher of educational orientation for the use of ICT in the learning process.

A. Leyva-Figueredo holds a Ph.D. in Pedagogical Sciences. He is the Director of the Center of Studies for Labor Education and Coordinator of the Doctorate Collaborative Curricular Program at the University of Pedagogical Sciences of Holguín. He has 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 Center 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.

Olympiads in Informatics, 2015, Vol. 9, 39–44
DOI: http://dx.doi.org/10.15388/ioi.2015.04

Efficient Range Minimum Queries using Binary Indexed Trees

Mircea DIMA[1], Rodica CETERCHI[2]

[1] *Hickery, Martir Closca st., 600206 Bacau, Romania*
[2] *University of Bucharest, Faculty of Mathematics and Computer Science*
14 Academiei st., 010014 Bucharest, Romania
e-mail: mircea@hickery.net, rceterchi@gmail.com

Abstract. We present new results on Binary Indexed Trees in order to efficiently solve Range Minimum Queries. We introduce a way of using the Binary Indexed Trees so that we can answer different types of queries, e.g. the range minimum query, in $O(\log N)$ time complexity per operation, outperforming in speed similar data structures like Segment/Range Trees or the Sparse Table Algorithm.

Keywords: binary indexed tree (BIT), least significant non-zero bit (LSB), range minimum query (RMQ).

1. Introduction

The Binary Indexed Tree, introduced by Peter M. Fenwick in (Fenwick, 1994), is a data structure that maintains a sequence of elements (e.g. numbers) and is capable of computing the cumulative sum of consecutive elements, between any two given indexes, in time complexity $O(\log N)$ and also update the value at a given index.

We show how to use the structure of the Binary Indexed Tree so that it will support other types of operations besides summation, e.g. range minimum query, maintaining the same time complexity of $O(\log N)$.

2. Binary Indexed Trees

2.1. *Problem Presentation*

Consider an array A indexed from 1 with N integers and the following types of operations:

1. Update – change the value at an index i, (e.g. $A[i] = v$).
2. Query – find the value of $\min(A[i], A[i+1], \ldots, A[j])$, for $1 \le i \le j \le N$.

The Binary Indexed Tree, as presented by Peter Fenwick, cannot efficiently answer these kinds of queries, because, for determining the sum of $A[i \ldots j]$, it needs to compute the difference between the sum of the first j elements and the sum of the first $i-1$ elements.

2.2. *Defining the BIT*

A BIT is not a Binary Tree, the name "Binary Indexed" comes from the fact that the nodes are indexed from 1 to N with labels written in binary, and it uses this binary representation to define the parent node for each node.

BITs are in fact the binomial trees of (Cormen *et al.*, 1990). We construct them inductively, starting with B_0, a tree with a single node. We will construct two varieties, the *left* and the *right* binomial tree. The *left binomial tree* B_{k+1} is obtained from two copies of left binomial trees B_k, by attaching the first of them as the leftmost child of the root of the second one. The *right binomial tree* is obtained in a mirror-like fashion, by attaching the second B_k as a right child of the root of the first B_k.

Starting from the array A, from its set of indexes, we build a left binomial tree BIT1 (Fig. 2.1) and a right binomial tree BIT2 (Fig. 2.2).

The binomial tree B_k (either left or right) has precisely 2^k nodes and height k. If we write the array indexes in binary, in the left binomial tree BIT1 we have $parent(i) = i + 2^{LSB(i)}$, and in the right binomial tree BIT2 we have $parent(i) = i - 2^{LSB(i)}$. This enables us to climb up either tree in $O(\log N)$.

Each node will keep aggregated data for all the nodes in its subtree. For instance in the first tree (Fig. 2.1) node 12 keeps the minimum value of nodes 9, 10, 11 and 12 which is the subarray $A[9 \ldots 12]$. Similarly, in the second tree (Fig. 2.2) node 12 keeps the minimum value the subarray $A[12 \ldots 15]$.

Since the parent of a node can be computed with a formula, we can store the trees in two arrays:

1. BIT1$[i]$ = minimum value of subarray $A[i - 2^{LSB(i)} + 1, i]$.
2. BIT2$[i]$ = minimum value of subarray $A[i, i + 2^{LSB(i)} - 1]$.

Computing these two arrays can be done in $O(N)$ with a bottom-up algorithm.

Let us consider the following array A with 15 positive integers:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A_i	1	0	2	1	1	3	0	4	2	5	2	2	3	1	0

You can see below how we computed the data stored in node 12:

BIT1$[12]$ = min(A$[9]$, A$[10]$, A$[11]$, A$[12]$) = min$(2, 5, 2, 2) = 2$

BIT2$[12]$ = min(A$[12]$, A$[13]$, A$[14]$, A$[15]$) = min$(2, 3, 1, 0) = 0$

In the following figures Fig. 1 and Fig. 2, the number inside a node is the index associated with that node. The number below the node is the aggregated minimum value of its subtree.

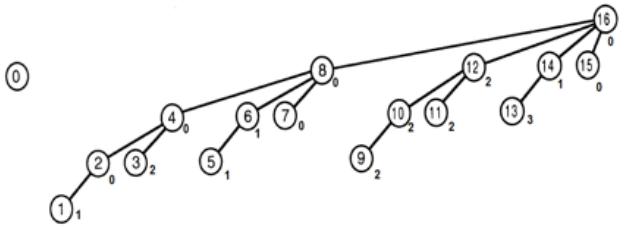


Fig. 2.1 Binomial Tree corresponding to BIT1 (node 16 is fictive) (Fenwick, 1994).

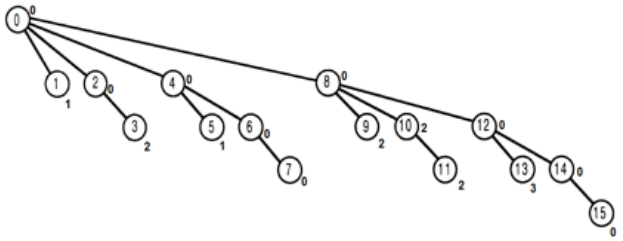


Fig. 2.2 Binomial Tree corresponding to BIT2 (node 0 is fictive) (Fenwick, 1994).

We shall exemplify the Least Significant Bit for better understanding:
LSB(216) = LSB(11011000) = 00001000 = 8 because there are 3 zeros at the end.

2.3. *Query operation*

For two given indexes i and j of the array $1 \le i \le j \le \mathrm{N}$, we want to answer the question: What is the minimum value among $\mathrm{A}[i], \mathrm{A}[i+1], \ldots, \mathrm{A}[j]$?

We start from node i in the first tree (Fig. 2.1) and climb the tree through its parent as long as the node index is less than or equal to j. We do the same thing in the second tree starting from node j and climbing the tree through the parent. In both cases we reach the same node and it splits $A[i \ldots j]$ in subarrays that are found either in BIT1, BIT2 or the value of the common stop node.

Let us exemplify by doing the query operation for the subarray $A[5 \ldots 13]$.

We start from node 5 and we climb the first tree (Fig. 2.1) while the current node's index is less than or equal to 13. We stop at node 8 because the next node, the parent of 8, is 16 which is greater than 13 and contains in its subtree the nodes 14, 15 and 16 which are not included in our subarray $A[5 \ldots 13]$. So far we passed by the nodes 5, 6 and 8. We take the minimum values corresponding to nodes 5 and 6 from the second tree found in BIT2. Looking in Fig. 2.2, node 5 keeps the minimum value for $A[5]$ and node 6 keeps the minimum value for $A[6 \ldots 7]$.

Similarly, we start from node 13 and climb the second tree (Fig. 2.2), passing by nodes 13, 12 and 8. We take the minimum values corresponding to nodes 13 and 12 from the first tree. Looking in Fig. 2.1, node 12 keeps the minimum value for $A[9 \ldots 12]$ and node 13 keeps the minimum value for $A[13]$.

We can observe that $A[5 \ldots 13]$ is now partitioned in the following subarrays:

$A[5 \ldots 5]$, $A[6 \ldots 7]$, $A[8]$, $A[9 \ldots 12]$, $A[13]$.

An important thing is that we get to the same node 8 for both traversals. We prove this happens every time:

Consider the subarray $A[i \ldots j]$ we want to make the query on. We know that $i < j$ and, because the order on integers is the same as the lexicographic order on their binary representation, we can write the indexes in binary like this (we consider that the indexes can be represented with n bits and $p+1$ is the first bit on which i and j differ):

$i = c_1 c_2 \ldots c_p 0\, i_{p+2} \ldots i_n$

$j = c_1 c_2 \ldots c_p 1\, j_{p+2} \ldots j_n$

When we iteratively add 2^{LSB} to i we will get at some point to $k = c_1 c_2 \ldots c_p 10 \ldots 0$ and if we iteratively subtract 2^{LSB} from j we will get to the same k. This is the common node where we stop.

Because the query climbs the two trees by following the parent link and because the height of a binomial tree with 2^K nodes is K, the time complexity of the query operation for a subarray is $O(\log N)$ where N is the size of the subarray.

2.4. *Update Operation*

Suppose we need to update the array at index p with the value v ($A[p] = v$).

We have to update all the tree nodes that have p in their subtree. We start from node p in the first tree (Fig. 2.1) and climb the tree until we reach the root (an index greater than N). For each node i we pass by, we consider its associated interval that defines its subtree: $[i - 2^{LSB(i)} + 1, i]$ (e.g. [9,12] is the associated interval of node 12). We can observe that the generated intervals include the index p because the parent's subtree expands and includes the node's subtree.

We want to update the minimum value of the associated interval of a node, be it $[x, y]$, where $y = x + 2^{LSB(x)} - 1$. If the minimum value of that interval is at an index q, $x \le q \le y$, different from p, then we update the interval by taking the minimum value between v and $A[q]$. If the minimum value is at index p, then we have to take the minimum values of intervals $[x, p-1]$ and $[p+1, y]$.

If we compute the minimum values using two queries, the time complexity of the update will be $O(\log^2 N)$.

We make the following observation: when we generate the associated intervals of the nodes we pass by, we can cover the whole interval $[p+1, y]$ by starting from node $p+1$ and climbing the first tree (Fig. 2.1). So instead of doing a query for every node we update, we compute the results of the queries on the fly by climbing the tree once.

Analogously, we can update all the intervals of the form $[x, p-1]$ by starting from node $p-1$ and climbing the second tree (Fig. 2.2). The same algorithm is applied for updating both trees.

Since we are climbing each tree three times and the height of a binomial tree with 2^K nodes is K, the amortized time complexity of the Update operation is $O(\log N)$.

2.5. *Experiments and Results*

We wanted to find out how the Binary Indexed Tree compares to a similar data structure called Segment Tree (also known as Range Tree), since it supports both update and query operations in the same time complexity of $O(\log N)$.

We implemented these two data structures in C++ and ran them on a 3.5 GHz Intel Xeon-Haswell server with 8 GB of RAM on Ubuntu 14 operating system with gcc 4.8 compiler.

The initial array had 100K random integers and we ran 10M random updates and 10M random queries. In Table 1 is what we found (times are in seconds):

While there is not a big difference on the Build step, we see a 47% reduced time for updates and 77% reduced time on queries.

3. Conclusions

In the current paper we intended to adapt the Binary Indexed Tree so that we can solve different types of operations, using as an example the Range Minimum Query problem, and maintaining the original time complexity of $O(\log N)$. The RMQ can be solved using a Segment Tree or other data structures like quadtree, but the Binary Indexed Tree proved to be 2–4 times faster in practice due to its simple iterative implementation.

Due to the structure of the Binary Indexed Tree, it can be extended in multithreading and distributed environments obtaining $O(\log(\log N))$ time complexities per operations (Elhabashy *et al.*, 2009). Also the data can be distributed among multiple nodes. This data structure can be used as indexes for databases in a distributed manner.

In conclusion, the Binary Indexed Tree has the following advantages:

- Is faster than other data structures that allow the same types of operations.
- Can be adapted for a large number of distinct operations: sum, minimum, maximum, greatest common divisor (gcd), greatest common factor (gcf), etc.
- Can be extended on multi-core and distributed platforms.

Acknowledgements

We thank dr. Florin Manea from the University of Bucharest and the University of Kiel for fruitful discussions and insightful comments on the topic of this paper.

Table 1

Operation Type	Segment(Range) Tree time	Binary Indexed Tree time
Build 100K array	0.0009 s	0.0006 s
10M Updates	1.274 s	0.672 s
10M Queries	2.397 s	0.551 s

References

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (1990). *Introduction to Algorithms*. MIT Press, McGraw-Hill, 1st edition.

Demaine, E., Sen, S., Lindy, J. *Advanced Data Structures*. Massachusetts Institute of Technology 6.897.

Elhabashy, A., Mohamed, A., Mohamad, A. (2009). An enhanced distributed system to improve thetime complexity of binary indexed trees. *World Academy of Science, Engineering and Technology*, 3(6), 121–126. http://waset.org/publications/5410/an-enhanced-distributed-system-to-improve-thetime-complexity-of-binary-indexed-trees

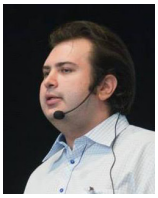
van Emde Boas, P., Kaas, R., Zijlstra, E. (1977). *D*esign and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10, 99–127.

Fenwick, P.M. (1994). A new data structure for cumulative frequency table, Software-Practice and Experience, 24(3), 327–336.

Fischer, J., Heunn, V. (2006). Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In: *CPM'06 Proceedings of the 17th Annual conference on Combinatorial Pattern Matching. Heidelberg*, Springer-Verlag Berlin, 36-48.

Topcoder Inc. (2014a). *Binary Indexed Trees*. https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/

Topcoder Inc. (2014b). *Range Minimum Query and Lowest Common Ancestor*. https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/



M. Dima (1989) – Co-Founder of Hickery.net, Senior Software Engineer, Former Software Engineer Intern at Facebook, Invited Host Scientific Committee Member at IOI 2013 Australia, problem setter at Romanian Olympiads, Programming Contest Veteran

R. Ceterchi (1953) – Associate Professor at the Faculty of Mathematics and Computer Science, University of Bucharest specialized on Algorithms and Data Structures, author of over 40 papers in national and international journals.

Olympiads in Informatics, 2015, Vol. 9, 45–55
DOI: http://dx.doi.org/10.15388/ioi.2015.05

Towards a Better Way to Teach Dynamic Programming

Michal FORIŠEK
Comenius University, Bratislava, Slovakia
e-mail: forisek@dcs.fmph.uniba.sk

Abstract. We give an overview of how various popular algorithm textbooks deal with the topic of dynamic programming, and we identify various issues with their expositions. In the second part of the paper we then give what we believe to be a better way of presenting the topic. While textbooks only contain the actual exposition, our paper also provides the rationale behind our choices. In particular, we managed to divide the topic into a sequence of simpler conceptual steps that are easier to learn.

Keywords: algorithms, dynamic programming, memoization, task analysis.

1. Overview

Dynamic programming is a standard paradigm used in the design of efficient algorithms. This approach is usable for problems that exhibit an optimal substructure: the optimal solution to a given instance can be recursively expressed in terms of optimal solutions for some sub-instances of the given instance.

Dynamic programming comes in two basic flavors. The top-down approach, usually called *memoization*, is based on implementing the computation of the discovered recursive relation as a recursive function, and then adding a cache so that each sub-instance only gets evaluated once. The bottom-up approach, usually called *dynamic programming*, essentially evaluates the same recurrence but in an iterative way: the algorithm designer specifies an order in which the sub-instances are processed, and this order is chosen in such a way that whenever we process a particular instance, all its needed sub-instances have already been processed and their optimal solutions are already known by the algorithm.

Below, we use the term *dynamic programming (DP)* to cover both flavors. When talking specifically about the iterative approach we will use the term *iterative DP* or *bottom-up DP*.

Despite being conceptually easy, dynamic programming is notorious for being hard to learn. Quoting Skiena (2008): "[Until] you understand dynamic programming, it seems like magic."

Different textbooks use very different approaches to present dynamic programming. The canonical way of presenting dynamic programming in algorithm textbooks is by showing a sequence of tasks and solving them using dynamic programming techniques. What is usually missing is:

- Rationale for choosing these specific tasks and their order.
- Notes on potential pitfalls when presenting the tasks and their solutions.

In this paper we aim to fill in those missing gaps. More precisely, the paper consists of the following parts:

- We present the way dynamic programming is exposed in multiple standard algorithm textbooks.
- We analyse those expositions and identify a set of possible pitfalls that often confuse and mislead students.
- We present our suggested version of a better order in which to teach the individual concepts related to dynamic programming, and we argue about the benefits of our approach.

2. Algorithm Textbooks

Throughout this paper we are going to refer to the way dynamic programming is treated in some of the canonical algorithm textbooks. In particular, we examined the following ones: Cormen *et al.* (2001), Dasgupta *et al.* (2006), Kleinberg and Tardos (2006), Sedgewick (1998), Skiena (2008). When refering to these textbooks below, for better readability we will use the following shorthand instead citations: CORMEN, DASGUPTA, KLEINBERG, SEDGEWICK, and SKIENA.

Below we give a brief summary how each of these textbooks introduces dynamic programming.

CORMEN prefers and almost exclusively uses a bottom-up approach. Dynamic programming is introduced using the following sequence of tasks and texts:

1. Assembly-line scheduling.
2. Matrix chain multiplication.
3. A general overview of iterative dynamic programming and memoization.
4. Longest common subsequence.
5. Optimal binary search tree.

DASGUPTA also prefers and almost exclusively uses a bottom-up approach, on a rather large set of solved tasks:

1. Shortest paths in a DAG.
2. Longest increasing subsequence.
3. A warning against exponential-time recursion.
4. Edit distance.
5. Knapsack.
6. A note about memoization.
7. Matrix chain multiplication.

8. Shortest paths.
9. Independent sets in trees.

KLEINBERG first introduces dynamic programming using a top-down approach, but then uses a bottom-up iterative approach in all following problems. The entire exposition looks as follows:

1. Weighted interval scheduling.
2. A general overview of iterative dynamic programming and memoization.
3. Segmented least squares.
4. Subset sums and Knapsack.
5. RNA secondary structure.
6. Sequence alignment (and optimizations to reduce memory use).
7. Shortest paths.

SEDGEWICK prefers a top-down approach. Only presents two problems:

1. Fibonacci numbers.
2. Knapsack.

Note that the new 4th edition (Sedgewick and Wayne, 2011) no longer contains a section on dynamic programming.

SKIENA is also in favor of starting with the top-down approach. His exposition proceeds in the following order:

1. Fibonacci numbers: recursively, with memoization, iteratively.
2. Binomial coefficients.
3. Edit distance.
4. Longest increasing subsequence.
5. Linear partition.
6. Context-free grammar parsing.

3. Critical Analysis

In this section we show the results of our analysis of the expositions used in the textbooks mentioned above. We mostly focus on tasks used in multiple textbooks.

3.1. *Matrix Chain Multiplication*

Statement: Given is a sequence $M_1, \ldots, M_n$ of rectangular matrices such that the product $M_1 \times \cdots \times M_n$ can be computed. Clearly, it can be computed as a sequence of $n-1$ standard matrix multiplications, and the result does not depend on their order. Given the assumption that multiplying an $a \times b$ and a $b \times c$ matrix takes $\Theta(abc)$ time, what is the most efficient way of computing the entire product?

The problem is solved by dynamic programming over all intervals. I.e., the states can be described by pairs of indices i, j such that $i \leq j$. For each state we compute the best

solution for the matrices in the given range.

Issues with this problem:

- Incomprehensible to students who lack background in linear algebra. The problem feels unnatural and the cost function seems arbitrary.
- Unnecessary clutter: the input is a sequence of ordered pairs of integers. There are other similar problems on integer sequences and/or strings.
- Lack of practical motivation. Finding a clear practical application for this algorithm is probably impossible.
- The existence of a much better solution. The $\Theta(n^3)$ DP algorithm shown in textbooks is an overkill, Hu and Shing (1982, 1984) gave a different $O(n \log n)$ solution for this problem.

3.2. *Shortest Paths in DAGs*

Statement: Given is a weighted directed acyclic graph (DAG). Find the shortest path from vertex 1 to vertex n.

This is a very good problem to be used at some point during the instruction on dynamic programming – mostly because it is the most general one. Essentially all dynamic programming solutions can be viewed as computations on directed acyclic graphs: the states of the computation (i.e., sub-instances we are solving) are the vertices of the DAG, the recursive relation determines the edges, and the order in which an iterative DP solution evaluates the states must correspond to a topological order of this graph.

Issues: DASGUPTA uses this problem as the first problem on which a dynamic programming approach is presented. We strongly advise against that. While we agree that the concepts mentioned in the previous paragraph are important, we believe that the proper time and way to learn them is by abstraction after already being familiar with many specific problems solved using dynamic programming.

Additionally, this problem requires students to be able to store and access a graph, and the data structures needed to do so efficiently are more involved than simple static arrays. A detailed analysis of the time and space complexity is also non-trivial as there are two different parameters (the number of vertices and the number of edges). This is especially true if one tries to solve this problem using recursion without memoization.

3.3. *Longest Common Subsequence*

Statement: Given two sequences (or strings), find one longest sequence that occurs as a (not necessarily contiguous) subsequence in each of them.

Related problems: Edit distance (Levenshtein distance) between two strings, DNA sequence alignment.

This is, and certainly should be, the gold standard among introductory problems solvable using dynamic programming. The solution only requires basic arrays, the sub-

problems and the recurrence are natural, and each of the subproblems can be evaluated in constant time.

Later, this problem can be used when discussing the differences between the top-down and the bottom-up approach.

This problem also leads to an advanced topic: Hirschberg's implementation (Hirschberg, 1975) of a bottom-up DP solution that can reconstruct an optimal solution in linear memory.

Issues: The time complexity of the brute force approach (recursive search without memoization) is hard to analyse exactly and it is often neglected in textbooks. CORMEN only mentions it to be "exponential-time" without any details, while DASGUPTA and KLEINBERG completely avoids mentioning it. SKIENA is the only one to address it, showing a (non-tight) 3^n lower bound for his version of the Edit distance problem.

3.4. *0–1 Knapsack*

Statement: Given is an integer weight limit W and a collection of n items, each with an integer weight w_i and an arbitrary cost c_i. Find a subset of items that has a total weight not exceeding the given limit and the largest possible total cost.

Related problems: Knapsack where arbitrarily many copies of each item are available. Coin change problems.

This is a reasonably natural class of problems. Again, their advantage is that the implementation only requires basic tools and that the recurrence relation is simple.

Issues: The whole notion of pseudopolynomial time. At some point, the students need to be explained why an algorithm that runs in $O(nW)$ is not considered a polynomial-time algorithm. While this issue is orthogonal to the concept of dynamic programming, it is an inherent part of this task and it should come up during its analysis. From experience, this may be the most challenging part of the problem for the students.

SEDGEWICK avoids the topic of pseudopolynomial time completely. It is just mentioned that the algorithm is only useful if the capacities are not huge. KLEINBERG also avoids this topic completely.

DASGUPTA addresses the topic with a single brief note: "[...] they can both be solved in $O(nW)$ time, which is reasonable when W is small, but is not polynomial since the input size is proportional to $\log W$ rather than W."

3.5. *Fibonacci Numbers*

Statement: Given n, compute the n-th Fibonacci number.

Fibonacci numbers are an excellent source of what is possibly the simplest non-trivial recurrence relation. They can easily be used to demonstrate the effect of memoization, as a straightforward recursive function that computes their values runs in exponential time.

Issues: The only issue with this very simple problem is that the numbers themselves grow exponentially and their values quickly exceed the range of standard integer variables in most languages. And even if you use a programming language with arbitrary precision integers (e.g., Python), the size of these numbers plays a role in estimating the time complexity of efficient programs.

A common way to address this issue is to modify the problem: instead of computing the exact value of the n-th Fibonacci number we aim to compute its value modulo some small integer. (E.g., if the modulus is 10^9, we are in fact computing the last 9 decimal digits of F_n.) Here we would just like to remark that the Fibonacci sequence modulo any m is necessarily periodic and this observation leads to asymptotically more efficient algorithms.

4. Our Approach to Teaching Dynamic Programming

In this final section we give a detailed presentation of how we suggest to teach dynamic programming. For each task used we clearly state and highlight the new concepts it introduces, and we argue why our way of introducing them works.

Note that we intentionally start with the top-down version of dynamic programming, i.e., by adding memoization to recursive functions. This is intentional and very significant. The main purpose of this choice is to show the students how to break up the design of an efficient solution into multiple steps:

1. Implement a recursive algorithm that examines all possible solutions.
2. Use the algorithm to discover a recursive relation between various subproblems of the given problem.
3. Add memoization to improve the time complexity, often substantially.
4. Optionally, convert the solution into an iterative bottom-up solution.

The more traditional approach that starts with iterative DP requires students to do steps 2 and 4 at the same time, without giving them good tools to do the analysis and to discover the optimal substructure. In our approach, step 1 gives them such a tool: once we have the recursive solution, the arguments of the recursive function define the subproblems, and we can examine whether the function gets called multiple times with the same arguments. If it does, we know that the problem does exhibit the optimal substructure, and in step 3 we mechanically convert our inefficient solution into an efficient one.

Lesson 1: Fibonacci numbers

Goals: Observe a recursive function with an exponential time complexity. Discover the source of inefficiency: the function executes the same recursive call many times.

Fibonacci numbers have a well-known recurrence: $F_0 = 0$, $F_1 = 1$, and $\forall n > 1 : F_n = F_{n-1} + F_{n-2}$. In our presentation we use the following Python implementation:

```
def F(n):
  if n==1 or n==2:
    return 1
  else:
    return F(n-1) + F(n-2)
```

Note that this implementation neglects the case $n = 0$ and uses $n = 1$ and $n = 2$ as the base case. This is intentional, the purpose is a more elegant analysis later.

We can now run this program and have it compute consecutive values of the Fibonacci sequence:

```
for n in range(1,100): print( n, F(n) )
```

The first few rows of input will appear instantly but already around $n = 35$ the program will slow down to a crawl. We can empirically measure that each next value takes about 1.6 times longer to compute than the previous one.

What is going on here? The easiest way to see it is to log each recursive call:

```
def F(n):
  print('calling F(' + str(n) + ')')
  ...
```

Already for small values like $n = 6$ we quickly discover that the same function call is made multiple times. Here it is instructional to show the entire recursion tree for $n = 6$, we omit the picture here to conserve space.

Here, a suitable homework is to leave the students analyse how many times $F(n - k)$ gets called during the computation of $F(n)$. For our version of the implementation the answer to this question are again precisely the Fibonacci numbers.

Alternately, we can just directly estimate the whole time complexity: when computing $F(n)$, each leaf of the recursion tree contributes 1 to the final result.

(This is the rationale for our choice to use $n = 1$ and $n = 2$ as base cases.) Hence, there are precisely $F(n)$ leaves and thus precisely $F(n) - 1$ inner nodes in the recursion tree. In other words, the *running time* of the computation of $F(n)$ is clearly proportional to the *value* of $F(n)$, which is known to grow exponentially.

Lesson 2: Memoization

Goals: Learn about memoization and conditions when it can be applied.

One of the points that is woefully neglected in traditional textbooks is the difference between *functions* in the mathematical sense and in the programming sense. The output of a mathematical function only depends on its inputs: $\cos(\pi/3)$ today is the same value as $\cos(\pi/3)$ tomorrow. For a function in a computer program, two consecutive calls with the same arguments may often return different values. There are lots of different reasons why this may happen. For instance, the output of the function may depend on global variables, on environment variables (such as the current locale settings), on pseudorandom numbers, on the input from a user, etc. (Listing these is actually a lovely exercise for students!)

Given the above observation, memoization is a very straightforward concept for students: we simply want to avoid computing the same thing twice. And it should now be clear that any function in our program that is also a function in the mathematical sense can be memoized. (Such functions are sometimes called *pure functions*.)

An interesting historical note: memoization is not only useful when it comes to improving the asymptotic time complexity. For instance, many early programs that produced computer graphics used precomputed tables of sines and cosines because table lookup was faster than the actual evaluation of a floating-point-valued function.

Lesson 3: Fibonacci numbers revisited

Goals: See the stunning effect memoization can have.

By applying memoization (using a simple array) to the Fibonacci function, each of the values $F(1)$ through $F(n)$ is only computed once, using a single addition. Therefore, we suddenly have a program that only performs $\Theta(n)$ additions to compute the value $F(n)$: quite an improvement over the original exponential time. The above Python program can now easily compute $F(1000)$.

(Note that Python operates with arbitrarily large integers. The n-th Fibonacci number is exponential in n and therefore has $\Theta(n)$ digits. In the RAM model, the actual time complexity of the above algorithm is $O(n^2)$, as each addition of two $O(n)$-digit numbers takes $O(n)$ steps.)

Here it is important to highlight the contrast: exponential time *without* vs. polynomial time *with* memoization. It is also instructional to draw a new, collapsed version of the entire recursion tree for $n = 6$.

Lesson 4: Maximum weighted independent set on a line

Goals: Encounter the first problem solvable using dynamic programming. Learn how to write a brute force solution in a good way, and how to use memoization to "magically" turn it into an efficient algorithm.

Statement: Given is a sequence of n bottles, their volumes are v_0 through v_{n-1}. Drink as much as you can, given that you cannot drink from any two adjacent bottles.

This problem has a very short and simple statement and only requires a simple one-dimensional array to store the input. But the main reason why we elected to use this as the first example will become apparent once we implement and examine a brute force solution for this problem.

Our goal is to implement a recursive solution that generates all *valid* sets of bottles and chooses the best among them. Such a recursive solution can be based on a simple observation: either we choose the last bottle or we don't. If we don't, we want to find the best solution from among the first $n - 1$ bottles.

If we do, we are not allowed to take the penultimate bottle and therefore we are looking for the best solution among the first $n - 2$ bottles.

```
v = [ 3, 1, 4, 1, 5, 9, 2, 6 ]
def solve(k):
```

```
''' returns the best solution for the first k bottles '''
if k == 1: return v[0]
if k == 2: return max( v[0], v[1] )
return max( solve(k-1), v[k-1] + solve(k-2) )
```

It should now be obvious that the time complexity of this program is exponential – in fact, the number of recursive calls needed to evaluate $solve(n)$ is precisely the same as the number of calls needed to evaluate $F(n)$ in our first lesson.

A key observation to make here is that `solve` can be considered a pure function. Even though it does access the global variable `v`, its contents remain the same throughout the execution of the program. Hence, we may apply memoization to `solve` in order to reduce the time complexity from $\Theta(\phi^n)$ to $\Theta(n)$.

(Note that `solve` can easily be turned into a true pure function if we pass a constant reference to `v` to `solve` as a second argument.)

Lesson 5: An iterative solution to the previous problem

Goals: Learn about the duality between the top-down and the bottom-up approach.

The subproblems solved by the memoized recursive solution can be naturally ordered by size. The same recurrence can now be used to write an iterative solution. A side-by-side comparison of both programs helps highlight parts that remained the same / only changed syntactically.

Optional lesson 6: Paths in a grid

Goals: Developing a bottom-up solution directly.

Statement: Given is a grid. Count all shortest paths along the grid from $(0,0)$ to (a,b).

The answer is obviously the binomial coefficient $\binom{a+b}{a}$, but the path-based point of view allows a natural formulation of a recurrence relation: Let $P(x,y)$ be the number of ways to reach (x,y). Each path that reaches (a,b) goes either through $(a-1,b)$ or through $(a,b-1)$. Hence, $P(a,b) = P(a-1,b) + P(a,b-1)$.

Statement: Now some grid points are blocked by some obstacles. Count all paths that go from $(0,0)$ to (a,b) in $a+b$ steps and avoid all obstacles.

The recurrence remains the same, only now the blocked grid points have $P(x,y) = 0$. It is instructional to solve small instances on this problem on paper – in fact, many of our students are familiar with this problem from earlier Math classes.

Lesson 7: Longest common subsequence

Goals: Investigating the differences between the top-down and the bottom-up approach.

For this problem, we first show the entire process. First, we show a recursive solution that generates all common subsequences, starting by comparing the last elements of both sequences. Then, we show that adding memoization improves this solution from

a worst-case exponential one into a solution that runs in $O(n^2)$.

Finally, we convert the solution into an equivalent iterative one.

Afterwards, we focus on the following points:

- Memory complexity. The iterative solution can easily be optimized to use $O(n)$ memory only, the recursive one cannot.
- Execution time. So far, iterative solutions were better as they didn't perform the additional work related to function calls. However, in this problem we can easily find inputs (e.g., two identical sequences) where the recursive solution outperforms the iterative one. The lesson here is that the top-down approach only evaluates the subproblems it actually needs, while the iterative approach doesn't know which subproblems will be needed later and thus it must always evaluate all of them.

Lesson 8: Longest increasing subsequence in quadratic time

Goals: Examining the first example where a subproblem isn't evaluated in constant time. Understanding how this is reected in the time complexity estimates. Most importantly, seeing that the subproblems don't have to be instances of the original problem.

Statement: Given a sequence of numbers, compute the length of its longest increasing subsequence.

In our opinion, this problem is one of the most important ones in teaching dynamic programming properly. When compared to previous problems, this one is much harder for beginners. Here's the main reason: *The subproblems we need to solve aren't actually instances of the original problem.*

This problem is used by DASGUPTA and SKIENA. However, DASGUPTA just reduces it to paths in a DAG without explicitly mentioning the conceptual step where we change the problem. SKIENA treats the problem properly: notably, asking the question "what information about the first $n-1$ elements of [the sequence] would help you find the solution for the entire sequence?" (Still, note that this question is factually incorrect: you are, in fact, supposed to look for information about the first $n-1$ elements that would help you find *the same information* for all n elements.)

We suggest actually emphasizing the redefinition of the problem. First, we illustrate on an example that knowing the length of longest increasing subsequence in the first $n-1$ elements is useless for solving the same problem for the first n elements. Only then we ask the question how to modify the problem in a way that would be useful. And the question is easily answered by using our approach: we can easily write a recursive solution that generates all increasing subsequences by choosing where to end and then going backwards. Converting this program into an $O(n^2)$ one requires just the mechanical step of adding memoization.

Optional follow-up lessons

After the above sequence of problems and expositions the students should have a decent grasp of the basic techniques and they should be ready to start applying them to new

problems. Still, there are more faces to the area of dynamic programming. Here, we suggest some possibilities for additional content of lectures:

- Problems where the state is a substring of the input: Edit distance to a palindrome. Optimal BST.
- Problems solvable in pseudopolynomial time: Subset sum, knapsack, coin change.
- Optimalizations of the way how the recurrence is evaluated: Hirschberg's trick for LCS in linear memory; Knuth optimization for Optimal BST; improving Longest increasing subsequence to $O(n \log n)$ by using a balanced tree to store solutions to subproblems.

5. Conclusion

Above, we have presented one possible way in which dynamic programming can be introduced to students. Our opinion is that the main improvement we bring is the systematic decomposition of the learning process into smaller, clearly defined conceptual steps.

References

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V. (2006). *Algorithms*. McGraw-Hill.

Hirschberg, D. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6), 341–343.

Hu, T.C., Shing, M.T.(1982). Computation of Matrix Chain Products (part 1). *SIAM Journal on Computing*, 11(2), 362–373.

Hu, T.C., Shing, M.T. (1984). Computation of Matrix Chain Products (part 2). *SIAM Journal on Computing*, 13(2), 228–251.

Kleinberg, J., Tardos, É. (2006). *Algorithm Design*. Addison-Wesley.

Sedgewick, R. (1998). *Algorithms in C++*. Addison-Wesley, 3rd edition.

Sedgewick, R., Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional, 4rd edition.

Skiena, S.S. (2008). *The Algorithm Design Manual*. Springer-Verlag, 2nd edition.

M. Forišek is an assistant professor at the Comenius University in Slovakia. Since 1999 he has been involved in organizing international programming competitions, including the IOI, CEOI, and ACM ICPC. He is also the head organizer of the Internet Problem Solving Contest (IPSC). His research interests include theoretical computer science (hard problems, computability, complexity) and computer science education.

Olympiads in Informatics, 2015, Vol. 9, 57–73
DOI: http://dx.doi.org/10.15388/ioi.2015.06

Introducing tcframe: A Simple and Robust Test Cases Generation Framework

Ashar FUADI
Indonesia Computing Olympiad Alumni Association
e-mail: fushar@gmail.com

Abstract. Preparing test cases is a vital step in a programming contest. Creating all test cases manually by hand is hard and error-prone, so they should be generated by programs. There have been several attempts at creating a framework for test cases generation, that involve writing a generator program that generates the test cases, and a validator program that validates whether the produced test cases conform to the constraints. This paper proposes a simpler yet robust framework, called tcframe, for generating test cases especially for programming contest problems. The proposed approach involves writing a single self-validating C++ generator program as opposed to writing two separate programs. The framework API is designed in such a way that the resulting generator program is easy to read and modify. Using this framework, programming contest organizers can produce generator programs with a consistent and similar structure across all problems.

Keywords: test case, test cases generator, test cases framework.

1. Introduction

In the past few years, competitive programming contests have been on the rise. Worldwide, the number of online programming contests has been increasing. Specifically in Indonesia, there have been more universities and high schools that have started organizing programming contests. Students have many opportunities for participating in programming contests in each school year.

In a competitive programming contest, contestants are given several problems to solve under a predetermined time. A contestant is considered to solve a given problem if he/she can write a program which produces correct output for each of the problem setter's secret input data (Halim and Halim, 2010). Therefore, the correctness of the secret input data (test cases) itself is really important and the programming contest organizers should put a lot of efforts in creating the test cases.

It is common that the people who prepare programming contest problems, including the test cases, are the ones who have participated in some programming contests in the past. The problem is that having experience in participating in a lot of contests usually

does not automatically make us a good in preparing test cases. It does help a bit, as for example, a seasoned contestant will be aware that many kinds of tricky cases should be included in the test cases. However, we feel that it is not enough to be a good test cases preparer.

There are at least two aspects in learning how to produce good test cases. The first is learning how to systematically write a test cases generator program. The second is learning how to come up with a strong set of test cases that catch as many bugs as possible in contestants' submissions. In this paper, we will be focusing on the former aspect.

We realize that there are currently very few learning resources on how to systematically create test cases for programming contests. Many people simply don't know how or where to even start. In our experience, this lack of knowledge usually results in each problem setter inventing their own test cases generator program.

The programs are usually not standard: some have parameters that are easy to modify; some do not; some write the test cases directly to files, some to the standard output, and so on.

The above situation is actually dangerous: it becomes hard for a person to modify or even understand other people's generator programs. Imagine a possible situation where we have to do some last-minute changes to a problem's constraints (for example: adding very easy subtask for beginners), but the person that wrote the generator program is not available at the moment. Other people will have to spend a considerable amount of time to understand and modify the program to produce the desired changes.

To eliminate the mentioned problems, we developed a test cases generator framework called **tcframe**. We did not create tcframe at once. Instead, we begin by using an existing library called testlib. It is a library designed for generating test cases for ICPC-style programming contests. It has been used for many Russian programming contests (Mirzayanov, 2008). Using testlib, we have to write a generator program that outputs exactly one test case, and a validator program that checks whether a particular test case conforms to the constraints.

We found that it was not quite suitable for IOI-style contests, so we created a wrapper around it, called tokilib (Fuadi, 2014). The wrapper essentially makes it possible to generate multiple test cases at once, and to check the constraints based on the assigned subtasks of each test case. We have been using it for preparing Indonesian IOI training camps throughout 2014 and the scientific committee members have been very satisfied with it.

Finally, we invented a quite significantly better approach that involves only writing a single self-validating generator program. We decided to rewrite the framework from scratch and give it a new name, tcframe. We are currently in the process of completing the development of tcframe.

The rest of the paper is organized as follows. In Section 2 we propose a formalization of test cases organization. Section 3 talks about the existing framework attempts and how we incrementally refined them to finally create tcframe. Section 4 explains our implementation of the proposed approach. In the last section, Section 5, we conclude the paper and offer many possible future works for tcframe.

2. Test Cases Organization

In order to create a framework, we have to formalize several aspects related to the organization of test cases. In this section, we propose such formalization. This section contains the definitions and relations between the aspects.

2.1. *Test Case*

We define a *test case* as a pair of input values and the corresponding output values. The input must satisfy the *constraints* (defined later in this section). We will talk mostly about test case inputs. Therefore, for simplicity, if not explicitly clarified, a test case will mean a test case input.

Test cases should be hidden from contestants. However, there are some test cases that are given in the problem statements. They are given so that the contestants will not misunderstand the input/output formats. Such test cases are called *sample test cases*. In contrast, test cases that are not given in problem statements are called *official test cases*.

2.2. *Input Variables*

We define *input variables* as variables which compose test cases inputs. They are usually declared in the input format section of a problem statement. For each value that appears in a test case input, it must be either a constant or (part of) an input variable.

For example, consider the following problem:

Statement. Given an integer N, and an array consisting of N integers, compute the product of all integers, modulo 10007!"

Input format. The first line consists of a sentence “Dengklek has N integers", where N is an integer. The next line contains N space-separated integers A_1, A_2, ..., A_N."

Output format. Output a single line containing a single real value: the mean, printed with two digits after the decimal point.

In this problem, for this input:

```
Dengklek has 3 integers
10 20 30
```

- The strings “Dengklek”, “has”, and “integers” are constants. If we consider whitespaces, they are constants as well.
- The integer 3 is N.
- The integer 10 is A_1.
- The integer 20 is A_2.
- The integer 30 is A_3.

2.3. *Constraints*

We define a *constraint* as a boolean predicate which limits the possible values of the input variables in test case inputs. The value of the predicate must be completely determined by the values of input variables only. For example, $1 \leq N \leq 100$ is a valid constraint for the previous problem.

2.4. *Subtasks and Test Groups*

We define a *subtask* as a set of one or more constraints. Subtasks are numbered with consecutive integers starting from 1. A test case is said to satisfy a subtask if it satisfies all constraints in the subtask.

One purpose of introducing subtasks in a problem is to create a nice score distribution (van der Vegt, 2009). Since 2010, all IOI problems have used subtasks in their constraints. Besides having a good distribution, the scores are also predictable since the points allotted to each subtask are usually fixed.

Let's consider a typical problem, which has only N as an input variable, and has three subtasks as follow:

1. $1 \leq N \leq 100$
2. $1 \leq N \leq 1000$
3. $1 \leq N \leq 10000$

We must assign a set of test cases to each subtask. The most common way is to consider each subtask independently. This means that the test cases generation for each subtask is independent to the other subtasks. In this way, it is quite unlikely that two subtasks have a test case with exactly the same content.

However, this has a serious problem. If each subtask is considered independently, then theoretically it is possible that a submission solves subtask 2, but not subtask 1. This can happen if subtask 1 has a tricky case that is not present in subtask 2. However, this situation does not make sense: if a solution fails a test case having $1 \leq N \leq 100$, then logically we should not let it pass subtask 2, since that test case satisfy $1 \leq N \leq 1000$ as well.

Our proposed solution to the above problem is as follows. Instead of each subtask having a set of independent test cases, we *assign* each test case to a set of subtasks. A submission is then considered to solve a subtask if it solves all test cases assigned to it. For convenience, we define a *test group* as a set of test cases, numbered with consecutive integers starting from 1. If two test cases are assigned to the same set of subtasks, then they are in the same test group.

Formally, we propose the following assignment rules:

- Each test case is assigned to a set of subtasks.
- For each test group, all official test cases in it have the same set of subtasks.
- If a test case is assigned to a set of subtasks S, then:
  - The test case must satisfy all subtasks $s \in S$.
  - The test case must not satisfy any of the subtasks $s \notin S$.

In the proposed rules, instead of generating independent set of test cases for each subtask, we organize the test cases generation as follows.

- Test group 1: generate test cases that satisfy $1 \le N \le 100$. Assign them to subtasks $\{1, 2, 3\}$.
- Test group 2: generate test cases that satisfy $101 \le N \le 1000$. Assign them to subtasks $\{2, 3\}$.
- Test group 3: generate test cases that satisfy $1001 \le N \le 10000$. Assign them to subtasks $\{3\}$.

Let's consider another problem, this time with two input variables N and K.

1. $N = 1$; $1 \le K \le 100$
2. $1 \le N \le 100$; $K = 1$
3. $1 \le K, N \le 100$

Generate the test cases as follow:

- Test group 1: consists of only one test case $N = K = 1$. Assign it to subtasks $\{1, 2, 3\}$.
- Test group 2: generate test cases that satisfy $N = 1$; $2 \le K \le 100$. Assign them to subtasks $\{1, 3\}$.
- Test group 3: generate test cases that satisfy $2 \le N \le 100$; $K = 1$. Assign them to subtasks $\{2, 3\}$.
- Test group 4: generate test cases that satisfy $2 \le N,K \le 100$. Assign them to subtasks $\{3\}$.

Perhaps it will be understood better by drawing the Venn diagrams of the subtasks, as depicted in Fig. 1. Our advice is to have a test group for each closed region in the resulting Venn diagram.

Note that the above solution still depends on the grader system used for the contest. If the grader system only support independent set of test cases for each subtask, then a possible workaround is to have multiple copies of a test case for each of the assigned subtasks. This will, however, result in a test case being evaluated multiple times. In the

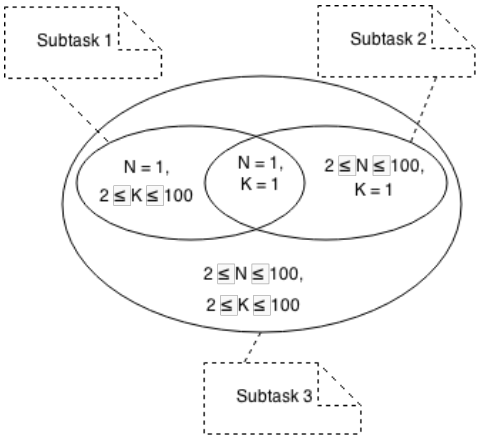


Fig. 1. Example test groups Venn diagram.

current implementation of grader mentioned in (Fernando and Liem, 2014), the above idea has been enforced. Each test case will never be evaluated more than once. Each subtask result is then deduced from the test case evaluation results.

Having formalized the test cases generation, we are now ready to discuss how to design a framework that supports it.

3. Test Cases Generation Framework Designs

There have been several previous efforts related to test cases generation that we are aware of. For each of them, we will explain the key designs, the problems that arise, and proposed solutions. Eventually, the proposed solutions are then used as foundations for writing the tcframe framework.

For each problem, there should be a set of test cases, a generator program that generates the test cases (if any), and a validator program that validates the correctness of the test cases, as proposed in (Diks *et al.*, 2008). All previous works require us to write a generator and a validator program. tcframe takes a little step further and only requires us to write a self-validating generator program.

When explaining a framework, we will assume that an official solution to the problem we are currently considering is already available. This means that our job is only generating test cases (not writing the solution).

3.1. *testlib*

testlib is a library created by Mike Mirzayanov *et al.* To generate test cases using the testlib library, we need to write two programs: a *generator program*, which outputs a test case input, and a *validator program*, which validates the produced input.

Generator program. This program, when run, will produce a single test case input file. To have some variations in the values of the input, the generator program usually contains some randomizations. The randomization parameters and seed can be passed as the program's command line arguments.

testlib provides a bunch of functions related to randomization that we can use.

For example, generating weighted random values, generating random strings based on a regex pattern, etc.

For example, here is a sample pseudocode of a generator program using testlib.

```
int maxN = the program's 1st argument

int N = randomize between 1 and maxN
println N
for 0 <= i < N:
    int a = randomize between 0 and 1000
    println a
```

Validator program. After a test case input file has been produced by the generator, we need to validate whether the values in the input file conform to the constraints. This is done by the validator program. It takes the input file as standard input and validates the values using the provided functions.

For example, here is a sample pseudocode of a validator program using testlib.

```
read an integer N and verify that 1 <= N <= 100000
for 0 <= i < N:
    read an integer a, and verify that 0 <= a <= 1000
    if i + 1 < N:
        read a space
    else:
        read a newline
verify that it is EOF
```

We found that testlib is missing several features. First, the generator program only produces a single input file in a single execution. So, to produce test data that has many test cases, we have to write a script and call the generator program several times using different arguments. Second, the validator program does not support subtasks: it is not possible to validate a test case on a specific subtask.

These features are missing because testlib was designed for ICPC-style problems.

Fig. 2 shows the diagram of steps for generating test cases using testlib.

3.2. *tokilib*

Recall that in summary, testlib does not support generating multiple test cases at once for problems with subtasks. To fill this missing feature, we wrote a wrapper around testlib, called *tokilib*. We call it wrapper because it is using testlib's generation and validation functions.

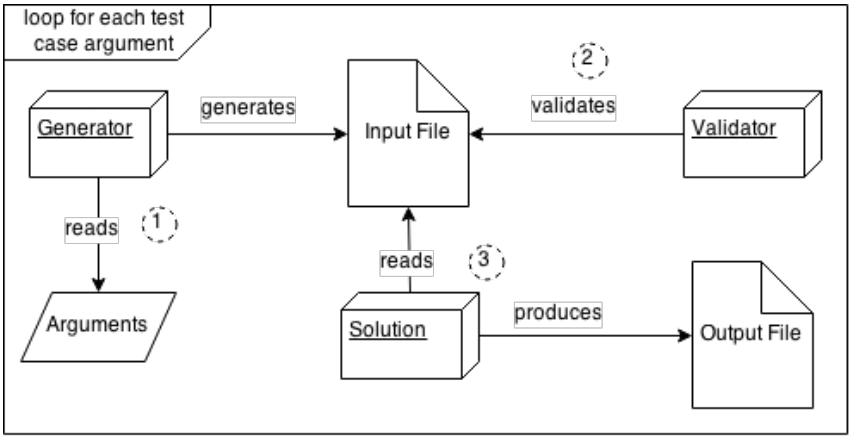


Fig. 2. testlib generation diagram.

Improved generator program. A generator program now consists of several functions, each of which defines a test group (called *batch* in tokilib). Each test group can be assigned a set of subtasks. Then, *test case definitions* follow, in the format as described in the next section below.

Component-based test cases. We introduce the concept of *components* in a test case as follows. Suppose we have a labeled tree data structure of N nodes as the test case. We can break down the structure into two *independent* components: a tree consisting of N nodes, and a list of N labels for the nodes. This way, we can generate each component independently, and then mix them to produce strong test cases.

For example, for the first component, we can have the following two variations: a binary tree and a random tree. For the second component, we can also have two variations: a list consisting of equal labels, and a list consisting of random labels.

For each component, we choose a suitable representation, which might be different from the input format section. For example, the tree can be represented as follows. Consider the tree as a rooted tree, with node 1 as the root. The representation is a vector of N elements. The i-th (one-based) element is the number of the parent of node i, or 0 (no parent) if i = 1 (Manev *et al.*, 2010).

Each component is declared as a global variable in the desired representation.

Then, a test case definition can be defined as a sequence of statements that assign the correct values to the components. Finally, we must implement a print() function, which prints the components according to the input format. Here is a pseudocode of a sample generator program in tokilib:

```
variables:
    int N
    int[] parents

print():
    println(N)
    for 0 <= i < N:
        if parents[i] != 0:
            println(i + " " + parents[i])

batch1():
    assignToSubtasks({1, 2})
    beginTC(); N = 7; binaryTree(); equalLabels(); endTC()
    beginTC(); N = 10; randomTree(); equalLabels(); endTC()
    beginTC(); N = 100; binaryTree(); randomLabels(); endTC()
    beginTC(); N = 100; randomTree(); randomLabels(); endTC()

batch2():
    assignToSubtasks({2})
    beginTC(); N = 500; randomTree(); randomLabels(); endTC()
    beginTC(); N = 1000; randomTree(); randomLabels(); endTC()
```

Improved validator program. A validator program in tokilib combines the schemes used in testlib's validator and tokilib's generator. It consists of a list of *subtask definitions*. A subtask definition consists of a list of *constraint definition*s, each of which is an assignment to a *constraint boundary variable*. A constraint boundary variable is just a

variable that holds the varying values of a constraint across subtasks, for example, min/max possible value of *N*. It is declared as a global variable. Finally, we must implement a validate() function, which validates a test case according to the assigned subtasks. Here is a pseudocode of a sample validator program in tokilib:

```
variables:
    int maxN

validate():
    read N and verify that 1 <= N <= maxN
    read tree edges
    verify that the edges form a tree

main():
    beginSubtask(); maxN = 100; endSubtask()
    beginSubtask(); maxN = 1000; endSubtask()
```

Improved usage. testlib requires us to manually call the validator program on the generated input file, and then run the solution to produce an output file.
This is not the case in tokilib: after the generator program produces an input file, the validator program will be automatically run against it. If it passes, the solution will be automatically run against it to produce an output file.

Fig. 3 shows the diagram of steps for generating test cases using tokilib.

3.3. *tcframe*

After using tokilib for generating test cases for many problems, we noticed that the part of validator that parses and checks whether the values are printed according to the input format looks similar and repetitive across all problems. We then wondered whether checking the input format in the validator can be eliminated.

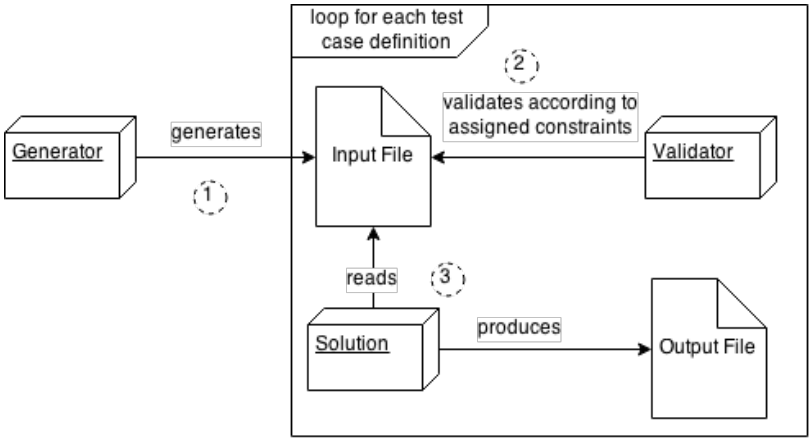


Fig. 3. tokilib generation diagram.

Let's begin with the reason why input format validation has been necessary in the first place. It has been because it is us humans who type the actual code for printing the values, and humans are error-prone. In addition, there are many ways to print the input variable values, because we may be manipulating the test case components using a different representation that defined in the input format section. For example, suppose the test case consists of an undirected graph.

The input format section might state that the graph is given as a list of edges, but we might be manipulating the graph using an adjacency list instead. One implementation for printing the edges is to print them while traversing the graph using, for example, breadth-first search. Then, if we make mistakes in the traversal code, we might print duplicate edges, or we might miss some edges.

So, we had an idea. If we hand over the input variables printing part to the framework, then the input variables parsing and input format validation should not be necessary anymore. To make this possible, we need to formalize how to declare the input variables and specify the input format. Remember that this has not been formalized before: we are free to decompose the structure into any components we want.

We propose the following formalization:

- The framework only works on "primitive" input variables. For example: scalars, vectors, and matrices of basic types (integers, floating-points, strings). Note that these are the most commonly used structures in input format sections.
- The framework provides an API to arrange the input variables in certain formats. For example, space-separated scalars in a single line, space-separated elements of a vector in a single line, multiple lines each containing an element of a vector, etc.

In this way, we only need to specify the input variables arrangement, and to assign values to the input variables for each test case. Then, the framework will take care of printing the values according to the specified input format. Assuming the framework is always correct, we now don't need to parse the variables and validate the input format in the validator program anymore.

Here is the pseudocode of the desired scheme:

```
inputVariables:
    int N
    int[] u, v

inputFormat():
    singleLine(N)
    multipleLines(N-1, {u[i], v[i]})
```

The interesting thing about this scheme is that the pseudocode is really similar to the input format section in the problem statement, which is nice because it will be easy to verify.

Now, the only thing left in validator program is constraints checking. Previously, this is done after we parse the input variables from the input file produced by the generator. Since the parsing part is now eliminated, we can do the check directly on the input variables declared in the generator, as a list of *subtask definitions*, each of which consists of a list of *constraint definitions*. As the input format validation is already similar to the

corresponding section in the problem statement, it would be nice if this part can also be made similar to constraints/subtasks section in the problem statement.

Here is the pseudocode of the proposed scheme of this part:

```
subtask1():
    declareConstraint({1 <= N and N <= 100})
    declareConstraint({graph is valid tree})

subtask2():
    declareConstraint({1 <= N and N <= 1000})
    declareConstraint({graph is valid tree})
```

Therefore, the validator program is now eliminated completely. As the generator program now contains both test cases generation and validations, we call it with another name: *runner program*.

Finally, the test groups/test cases definitions are similar to those in tokilib, except that we also make the syntax more declarative.

```
testGroup1():
    assignToSubtasks({1, 2})
    declareTestCase({N = 7; binaryTree(); equalLabels()})
    declareTestCase({N = 100; randomTree(); randomLabels()})

testGroup2():
    assignToSubtasks({2})
    declareTestCase({N = 200; binaryTree(); equalLabels()})
    declareTestCase({N = 1000; randomTree(); randomLabels()})
```

In summary, Fig. 4 depicts the flow of test cases generation using the mentioned idea so far.

We felt that this change is very significant compared to tokilib, so we decided to consider this as a brand new framework and give it another name, *tcframe*.

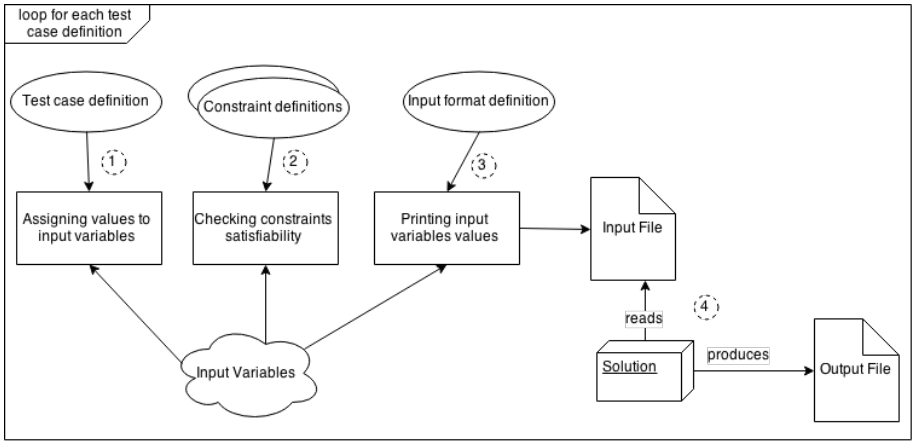


Fig. 4. tcframe generation diagram.

4. Implementation Details

In the previous section, we described motivations and ideas behind tcframe. This section will explain how tcframe is implemented in more detail.

The code is hosted on GitHub (`https://github.com/ia-toki/tcframe/`). At the time of writing, it is on version 0.3.0 and it is not ready for public use.

We chose to write tcframe in C++. One of the main reasons is because it is a popular language used by competitive programmers. In addition, C++ has preprocessor macros, which will be used extensively for producing concise but powerful code.

Let's begin with some philosophies that we want to achieve in the implementation of tcframe.

- It should be possible to write only a single runner program as opposed to writing two programs: generator and validator programs.
- The resulting runner program code should be concise, declarative, and should resemble the problem statement.
- The overall syntax should be well structured so that it is easy to extend in the future.

Fig. 5 depicts a proposed high-level class diagram to meet the above requirements.

Let's discuss each part of the class diagram in more details. Note that tcframe is not final yet so some part definitions may change in the future.

4.1. *Specification Classes: BaseProblem and BaseGenerator*

Recall that testlib and tokilib both require writing two separate programs: generator and validator. This scheme has an advantage that the validator cannot access any user-defined functions in the generator. This is important because to test a system (the generator), we

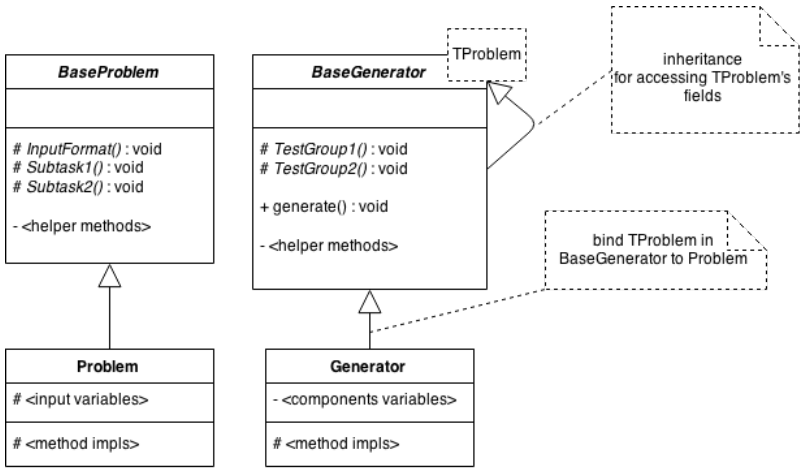


Fig. 5. tcframe class diagram.

should only use things outside the system. However, to implement tcframe, this cannot be the case: both the generation and validation steps need to access the input variables. However, we also want to prevent the validation step from using any user-defined functions used in the generation step.

To satisfy both seemingly contradicting requirements, we restructure parts of the generator and validator programs into two classes: input variables, input formats, and subtask definitions go into *problem specification* class, while test cases definitions go into *generator specification* class. Both classes then go to just a single program. The definitions are implemented as virtual (abstract) methods in these classes, which must be overridden in concrete classes. By convention, let's name them Problem and Generator. Any user-defined functions then should be declared private in Generator.

As mentioned above, the input variables go into the Problem class, as member variables. The Generator class then must be given access to the member variables.

We notice that there are two ways to implement this requirement: by composition and by inheritance.

By composition. The Generator will hold an object of type Problem, whose member variables are declared public. This is nice, but now whenever we want to assign values to input variables in the Generator class, we must type the object name (e.g., problem.N = 100). We feel that this way is not concise and pleasant.

By inheritance. To avoid writing an object name before each input variable, we chose to use inheritance. The input variables are to be declared as protected member variables in Problem. Then, we make BaseGenerator inherit Problem. BaseGenerator now can access the input variables. Generator is also able to access them as well, since it inherits BaseGenerator.

Finally, to make the above scheme work for any Problem class, we use templates in the following way. Here is the declaration of BaseGenerator:

```
template<typename TProblem>
class BaseGenerator : public TProblem;
```

And here is the declaration of Problem and Generator:

```
class Problem : public BaseProblem;
class Generator : public BaseGenerator<Problem>;
```

4.2. *Definitions: Input Format, Constraints/Subtasks, Test Cases/Test Groups*

The definitions are realized as virtual member functions in the base specification classes. We must implement those functions in the concrete specification classes, and put the definition items inside them by making API calls. They should have something in common: if any step in the whole generation process fails, then the definition item that causes the failure should be presented to the users so that the users can fix it. For example, if

an input variable value does not satisfy a constraint, something like “Error: constraint $1 \le N \le 100$ not satisfied" should be output.

To support the above requirement, the definition items will be implemented using C++ macros, which supports the stringization trick of the input parameters.
The next section will show the planned syntax for the macros.

Input format. It consists of one or more *input segments*, which will be printed one after another. Currently, the following input segment types are supported: space-separated scalars/vectors in a single line, lines each containing an element of vectors, and grid. For example, the definition of a line segment (the first type) is made by calling this macro:

```
LINE(A, B);
```

which then expands to something like:

```
inputFormat.addLineSegment("A, B", A, B);
```

The above call defines a line that consists of input variable A, followed by a space, followed by input variable B.

Constraints/subtasks. BaseProblem declares virtual methods Subtask1() ... SubtaskX() for a finite number X (currently it is set to 10). To define a subtask, implement any of the mentioned methods in the Problem class. Inside the method, we can define one or more constraints, by calling this macro:

```
CONS(1 <= N && N <= 100);
```

which then expands to something like:

```
constraints.add("1 <= N && N <= 100",
        [this] { return 1 <= N && N <= 100; });
```

In tokilib, we implement a constraint definition as an ordinary C++ statement.

In tcframe, we will use a new feature in C++11: lambda closure, for each constraint. This has several advantages. For example, each constraint is now independent from any other and they can be called in any order. It also makes the framework more extensible; for example, it becomes possible to write a plugin based on tcframe that only prints all constraint descriptions.

Test cases/test groups. BaseGenerator declares virtual methods TestGroup1() ... TestGroupX() for a finite number X (similar to subtasks, currently it is just set to 10). To define a test group, implement any of the mentioned methods in the Generator class. Inside the method, we can define one or more test cases, by calling this macro:

```
CASE(N = 100, randomArrayElements());
```

which then expands to something like:

```
testCases.add("N = 100, randomArrayElements()",
        [this] { N = 100, randomArrayElements(); });
```

Similar to constraint definitions, test case definitions also make use of lambda closures. Note that we choose comma operators rather than semicolons for separating input variable assignments. This way, the test case definition looks more "declarative": it consists a *list* of assignments to the input variables, rather than *statements*.

Finally, we also want to be able to use component-based in tcframe. This can be achieved by declaring the component variables in the Generator class, use them in test case definitions, and then convert them to the actual input variables in the Problem class before the end of each test case definition.

5. Conclusion

We have presented the ideas behind creating tcframe, and how we are currently implementing it. We also suggested a formalization on test cases organization.

We hope that tcframe will allow more people to be able to create good test cases systematically. For programming contests with multiple authors, we hope that tcframe allows the authors to be able to work together creating test cases more collaboratively.

6. Future Works

We are aware of several possible other improvements and new features that can be implemented in tcframe.

6.1. *Output Validation*

testlib, tokilib, and tcframe all share a problem: the produced test case outputs, obtained by running the solution against the generated test case inputs, are not validated. This is actually very dangerous because the solution might have some mistakes and print the output not according the output format, or have some invalid values.

We can validate the outputs in a similar way as we do to the inputs:

- Declare *output variables*. For example, in a single-integer answer, we can call it $answer$.

- Declare *output format*. The framework can then use this format for parsing the produced outputs, validating the format, and storing the values to output variables.
- Declare *output constraints*. For example, if a problem requires the answer modulo 10007, we have an output constraint $0 \leq answer \leq 10006$. The framework can then use it for validating the output variables.

6.2. *Answer Checker*

For problems with several possible solutions, we need a checker that compares the judge's and contestant's answers. Our class structures make it easy to add this functionality. We can build a BaseChecker class on top of Problem class, similar to BaseGenerator. The check can be then somehow implemented as follows.

```
class Checker : public BaseChecker<Problem> {
public:
    bool check(const Problem& contestant, const Problem& judge) {
        return fabs(contestant.answer - judge.answer) < 1e-9;
    }
};
```

6.3. *Offline Solution Checker*

Instead of just producing test cases, we can let tcframe take another solution as input, then report whether the outputs produced by that other solution match the outputs produced by the official solution. This way we can effectively "submit" a solution without using the online judge. We can limit memory and time limit using, for example, **ulimit** UNIX command.

6.4. *Public Library for Commonly Used Structures*

We want to provide a public place where people can contribute by writing generators that generate common test cases structures. For example, polygons for convex hull problems, coin values for coin change problems, etc. The submitted generators should generate strong test cases. For example, there should be convex polygons, the coin values should be in such a way that greedy solutions will fail, etc.

This will make it even easier for beginners to generate test cases. They can browse the library for the structures they need and modify the generator for their problems.

References

Diks, K., Kubica, M., Radoszewski, J., and Stencel, K. (2008). A proposal for a task preparation process. *Olympiads in Informatics*, 2, 64–73.

Fernando, J. and Liem, I. (2014). Components and architectural design of an autograder system family. *Olympiads in Informatics*, 8, 69–79.

Fuadi, A. (2014). *tokilib*. `https://github.com/fushar/tokilib/`

Halim, S., Halim, F. (2010). *Competitive Programming*. `Lulu.com`

Manev, K., Yovcheva, B., Yankov, M., Petrov, P. (2010). Testing of programs with random generated test cases. *Olympiads in Informatics*, 4, 76–86.

Mirzayanov, M. (2008). *testlib*. `https://github.com/MikeMirzayanov/testlib/`

van der Vegt, W. (2009). Using subtasks. *Olympiads in Informatics*, 3, 144–148.

A. Fuadi is a fellow in Indonesian Computing Olympiad Alumni Association. He participated and obtained a silver medal in IOI 2010. He has been a scientific committee member for IOI training camps for Indonesian teams 2011–2014. Graduated from Faculty of Computer Science, Universitas Indonesia in 2014.

Olympiads in Informatics, 2015, Vol. 9, 75–88
2015 *Vilnius University*, *IOI*
DOI: http://dx.doi.org/10.15388/ioi.2015.07

Metamorphic Testing and DSL for Test Cases & Checker Generators

Ryan Ignatius HADIWIJAYA, M. M. Inggriani LIEM
Data and Software Engineering Research, School of Electrical Engineering and Informatics
Institut Teknologi Bandung
e-mail: ryan.ign54@gmail.com, inge@informatika.org

Abstract. In programming competition, a problem setter must prepare a task description, program solution, test cases and sometimes a checker. Test cases should be able to capture all possible cases; therefore, its preparation is time-consuming. Metamorphic Testing (MT) is a property-based testing method where relationships are defined between input and output to alleviate a test oracle problem. The success of MT relies on the existence of a Metamorphic Relation which is comprised of two interrelated relations: the test-case relation and the test-result relation. MR can be used for automated test-case generation and verification of results. In this research, we defined a Domain Specific Language (DSL) to describe metamorphic relations that will be used for test case and checker generation of programming tasks. Our method has been tested for tasks with Knapsack, Greedy, and Dynamic Programming solutions, and it has been proven, reliable, reusable and more systematic.

Keywords: test case generation, programming task checker, programming competition, Knapsack problem, Greedy problem, metamorphic testing.

1. Background

In Indonesia, autograder systems are used for national training programs and for the selection of IOI participants. To prepare a contest or training session, we have to define a problem set, which includes a description of the task, program solutions, and test cases (input as well as output). Some tasks also need checker. Up until now, the preparation has been carried out manually by a problem setter, including preparation of input and output test cases. Manual test case preparation is time-consuming and nearly impossible for a complex problem with a large amount of data. Therefore, test cases are generated by programs written one by one in a manner, specific to each problem set.

In IOI, there are two types of tasks, namely batch tasks and interactive tasks. In this paper, we focus on the batch task, which is judged by a grader and based on black-box testing. However, grading is more than testing because the grader must judge and give a score for each subtask that refers to a contestant's solution. A batch task has one or more subtasks, which in turn have constraints and scores. A good black-box testing method

uses all of the values in the input domain as input test cases. This would be impossible if the input data domain had very large (or even infinite) values. The problem setter must select reasonable values to be used in grading. This problem is known as the test-case selection strategy. If test cases are selected manually, either intuitively, or randomly, then their coverage is not guaranteed. The programming of a task also has time constraints that require appropriate test cases. An incorrect solution can be judged as an acceptable answer when the test cases do not precisely reflect the conditions. On the other hand, a redundant test cases will consume unnecessary time of execution and consume CPU resources. Good test cases must have sufficient test coverage and reasonable quantity and properties. Therefore, a test-case specification is needed. We aim to write a test-case generator based on specifications so that it is self-documented, and the scientific committee can verify its coverage and quality.

Some tasks may have many possible solutions. In order to optimize the autograding process, the team writes a checker instead of generating all possible solutions. The checker is used to compare contestant output to output test cases. Usually, the checker is also made ad-hoc and by writing a specific program for a specific task. It is difficult to verify its correctness. A faulty checker can make an incorrect answer become acceptable. In our research, we aim to provide the problem setter with a checker specification.

Before using the system explained in this paper, test cases for Indonesian training programs are actually being generated by the program as much as possible and not completely manual. However, test-case generation source code depends on problem setter and not driven by specific method. Checkers are programmed one by one specifically for each task and are not generated. Metamorphic Testing has the potential to be implemented as a method for improving test case and checker generation which implies an improvement in the automatic grading process.

2. Related Works

Our work is inspired by Metamorphic Testing (MT) (Chen *et al.*, 1998; Chen *et al.* 2004; Zhou *et al.*, 2004; Mahmuda *et al.*, 2011; Barus *et al.*, 2011) and Domain Specific Language (DSL) (Im *et al.*, 2008; Ghosh, 2011). Chen *et al.* (1998) which suggest using Metamorphic Testing for test case generation. Test case generation based on Metamorphic Relation (MR) could be automated (Gotlieb and Botella, 2003), with the MR coded directly in a general programming language. In our approach, we generate test cases (input, output) and a checker for a programming task by defining a Domain Specific Language for representing MR and input/output.

2.1. *Metamorphic Testing and Metamorphic Relation*

Metamorphic Testing (MT) is a technique to generate follow-up test cases based on existing test cases that have not revealed any failure. MT generates follow-up test cases by making reference to the metamorphic relation (MR).

An MR refers to two types of relations. First, by referring to the MR of the target function, follow-up test cases can be automatically constructed, executed, and checked to further verify the program. Metamorphic testing is to be used in conjunction with a test-case selection strategy S. Test set T generated from S must also exist in the first place.

Second, MR refers to the verification of testing the output (test result). Suppose we have a metamorphic relation R of function *f*, of which p is an implementation. The second relation refers to necessary properties of the target function *f* where if any of these properties does not hold, then program p is faulty. Metamorphic testing makes use of the relationship between the inputs and outputs, and involves multiple executions of p.

For example, if $f(a) = e^a$, then the property $e^a \times e^{-a} = 1$ is a typical MR. For a test case $a = 0.3$, metamorphic testing generates its follow up test case $a' = -0.3$ and then runs the program again on a'. The relation of the two outputs is checked against the expected relation $p(0.3) \times p(-0.3) = 1$. If this identity does not hold, then a failure is immediately detected (Zhou *et al.*, 2004). Another example of trivial MR is $\sin x = \sin(\pi - x)$ for a program that computes $\sin(x)$.

In testing, successful test cases are test cases which do not reveal any failure of the program. In a contest, successful test cases are test cases that reveal a correct answer and give a full score. Therefore, successful test cases have been considered useless in conventional testing because they do not reveal any failures. In other words, in a conventional testing, the successful test cases are discarded or retained. In contrast, metamorphic testing can be employed to make use of the successful test cases. In the context of the programming task, this idea will be used for validating the result of the generator (test input), and to accept or reject a generated test case.

Another example of follow-up test cases is illustrated in Fig. 1 (Murphy, 2010). Fig. 1 illustrates an example of a metamorphic relation to sum all elements of an unsorted numerical array. Permute, add, multiply, include and exclude are five examples of metamorphic relation. Five sets of new test cases can be generated based on an initial

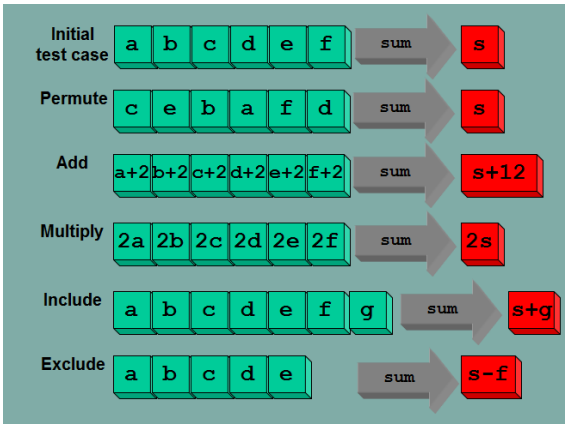


Fig. 1. Example of Metamorphic Relation for Sum Function (Murphy, 2010).

successful test case in order to reveal faults in the program. The output of these new test cases can be determined easily by its metamorphic relation, and this can save time and reduce the cost of making test cases.

2.2. *Domain Specific Language*

A DSL is a programming language that is targeted for a specific domain. It contains syntax and semantics that models concepts at the same level as abstraction of the problem domain.

Compared to GPL (General Purpose Language), DSL is shorter and simpler (Ghosh, 2011). DSL is easier to understand by domain experts. By using DSL, users can focus on the problem and deliberate from detail implementation. DSL is designed to be used intuitively.

A domain-specific language is created specifically to solve problems in a particular domain and is not intended to be able to solve problems outside that domain (although it may technically be possible). DSL for the business domain is defined to externalize business rules and computations, such as tax calculations, salary calculations, or financial engineering.

Examples of domain-specific languages include HTML and SQL for relational database queries, YACC grammars for creating parsers, regular expressions for specifying lexers, Csound for sound and music synthesis, and the input languages of GraphViz and GrGen, software packages used for graph layout and graph rewriting.

DSL is also used in automated test case generation (Im *et al.*, 2008). We intend to define a specific DSL to solve the generation of test cases of a programming task, based on MR.

3. Problem Statement and Objectives

When test cases are generated randomly, the coverage is not guaranteed and the generator is not reusable. More than that, its documentation is not preserved. Test case generation that contains initial specification can solve this problem. DSL offers precise and simple expressions well known by experts of the domain. Specifications written in a DSL will preserve the documentation of test cases and the checker. MT is property-based testing and provides a method for automated generation of test cases by defining MR. MT will improve the quality of test cases so that the autograding process will be more robust. During the grading process, the checker must not only check the properties of the output, but also checks the relations of many executions.

In this research, we combine the idea of Metamorphic Testing by defining the Metamorphic Relation with a Domain Specific Language. We aim to deliver a solution to a problem setter, so that the problem setter as the domain expert can express test cases and a checker by a specification written in a DSL.

The advantages of our approach are :

1. The problem setter focuses on specifications rather than on a program.
2. The system provides a reusable library of common test cases and checkers, since algorithmic solutions can be grouped by a variety of techniques such as Knapsack, Greedy, Dynamic Programming, Geometry, etc. It uses standard data structures (arrays, trees, graphs, etc.) since each techniques and data structure has a common MR.

4. Proposed Solution

First of all, we define a DSL grammar to represent MR, input/output variable names, constraints and their values, input/output format, and checker expression. A part of the DSL grammar represents the name of the class, and its features and six main declarations are presented in Code 1. The complete grammar is accessible in `https://github.com/ryanignatius/CheckerDSL/tree/master/Grammar`. Our system will read the specification, and generate test cases and checkers. The problem setter is not required to write a program, compared to the usage of a framework or an existing library (Mirzayanov, 2008).

```
Class:
    'class' name=ValidID '{' features+=Feature* '}';
Feature:
    ChkVariableDeclaration | Method | Format | Check | MR | Score;
ChkVariableDeclaration:
    type=ChkTypeReference ('[' sz+=CHK_NUMBER ']')* name=ValidID
      ('(' limit1=Limit (';' limit+=Limit)* ')')?
      ('value' '{' spValue=SpValue '}')?;
Method:
    'op' type=JvmTypeReference name=ValidID
      '('(params+=FullJvmFormalParameter
        (',' params+=FullJvmFormalParameter)*)?')'
       body=XBlockExpression;
Format:
    InputFormat | OutputFormat;
Check:
    check='check' '{' ( chk+=(ChkExpression | ChkLoopExpression) )* '}';
MR:
    mr='MR' num=INT '{'
        (mrExp+=(ChkExpression | ChkLoopExpression))*
        followup=FollowUp
        property=Property
        '}';
Score:
    'score' '{' (scores+=ChkScoreExpression)+ '}';
```

Code 1. A Part of DSL Grammar.

In the auto-grading process, a grader executes a contestant's program with a corresponding input test case, and then compares the execution result with the output test cases. If the output of the contestant's program is equal to output test cases, then the

grader judges it as a correct answer. For some tasks, contestant outputs are checked by provided checker(s). The contestant obtains a score for each correct input-output set and the final score for a task is visualized on a scoreboard. In our case, the checker does not only check a single-run output. The checker checks MR and other properties defined in the specification.

By using MR, the relation between the output of one run to another run (related by MR) can be checked. This will increase test robustness. When a relation between two outputs does not conform to the defined MR, then the grader will judge it as a wrong answer. We define each MR as a checker. A checker is a predicate that can check whether a set of output corresponds to a predefined MR, simply checks the property of the output, or checks the coverage of the input.

The problem setter must write a problem solution, a base-test case (a set of minimum test cases), and a specification file. The specifications are written in the DSL and consist of six declarations:

1. **Variable declaration**. The problem setter declares variable names, variable constraints, and test-case domain partitions that will be used in other sections. A variable can be declared as a JAVA primitive type, an array or a specific data structure such as a graph, tree or list. Each variable is generated as a private attribute in the generated JAVA file. For each attribute, functions are also generated to read, write, and validate.
2. **Input/output format declaration**. The problem setter declares the input and output formats, where values of variables will be read or written. The system will generate functions to read and write all variables that have been declared in the previous section. The read/write function will validate the input or check the output based on the constraints that have been defined in the variable declaration section.
3. **Predicate declaration**. A predicate is a function returning a boolean. This predicate will be used to generate a checker. If the input or output to be checked passes all tests by invoking the predicates, then the output will be judged as a correct answer.
4. **MR declaration** consists of follow-up and properties. Follow-up will be generated based on the MR, and properties are used to ensure that MR is satisfied.
5. **Other function declarations**. The problem setter can define specific functions. The system provides predefined functions such as *sort, swap, min, max, check if a number is a prime number*, etc. If a function is not defined in the library, the problem setter must implement it. The problem setter can enrich his environment by registering his function in the library.
6. **Score declaration**. The problem setter defines the score distribution for each subtask. In IOI, the score is given when a contestant's program passes all test cases in the subtask.

Test case and checker generation are described by the work flow depicted in Fig. 2. The first phase of the process consists of two parts that can be carried out in any order. The first part is processing the DSL specification file within the XText framework (Xtext, 2014) and produce a file named `GeneratedClass.java`. This file is then compiled with the given `MainGenerator.java` and `LibraryFunction.java`. `Main-`

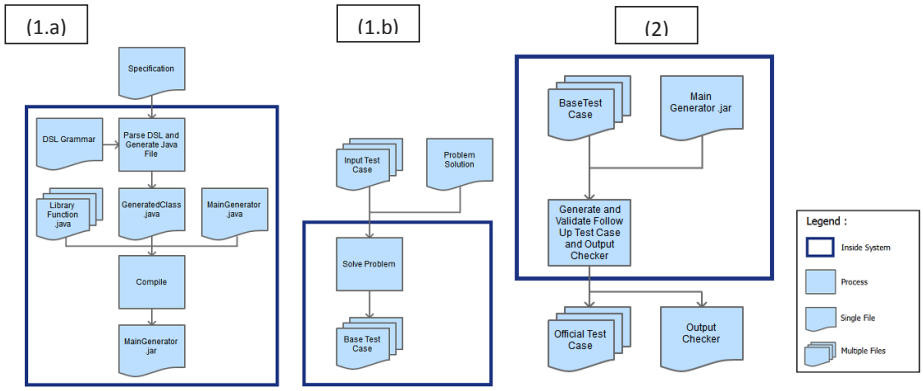


Fig. 2. Workflow of Test Case and Checker Generation.

`Generator.java` is a main program that receives parameters from the problem setter (output checker, input-output test cases, minimum number of test cases, mapping of test cases to subtasks). `LibraryFunction.java` contains predefined functions, that can grow as we may find other generic functions in the future. The result of this compilation is a jar file that will be used in the second phase. The second part is the generation of base test cases by running the problem solution with the given input.

The second phase is the generation of input files, output files, score files and checkers. The generation process is repeated and for each generation the program will validate input and output (defined in the specification). The system will reject test cases that do not comply with the specification, and repeat the process until the problem setter obtains sufficient test cases described in the specification. For each MR and given test case, the system will generate a follow-up test case. A corresponding checker will also be generated for checking the MR of the given test case output with generated follow-up test-case outputs.

5. System Architecture

The user of our system is the problem setter. The architecture of the system consists of four layers as can be seen in Fig. 3. The first layer is Java, containing JVM as the runtime environment. The second layer is IDE, whose framework contains Eclipse and XText, running on JVM. The DSL grammar, JVM Model Inferrer, Library Function, and Main Generator are put in the Developer layer. This layer is provided by us. On top of the third layer is the user layer, where a problem setter defines the specification and obtains generated classes. The problem setter interacts with the system through components in this layer. For each task, the problem setter writes a module. Checker specification is defined by a user using the XText component. Checker specifications are parsed by the XText parser using DSL grammar. If parsing is successful, then a checker will be generated by XText based on the existing JVM Model Inferrer. This generated file will be compiled

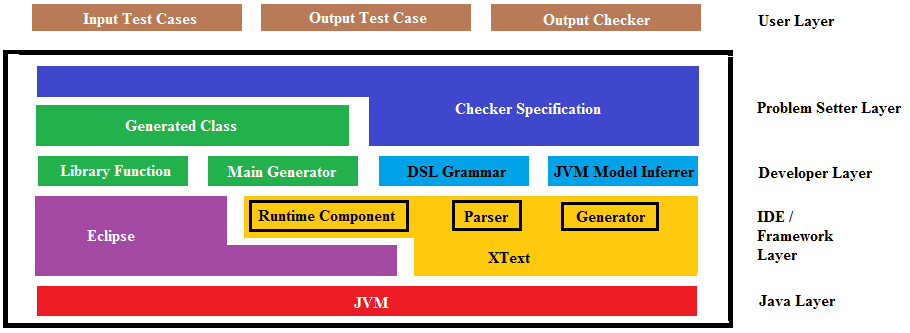


Fig. 3. System Architecture.

by the Library Function and Main Generator. The execution of these files will produce files (test-case input, test-case output, score, and checker).

DSL grammar is implemented using the XText framework, a plugin for the Eclipse IDE. Eclipse's features such as autocomplete and automatic error checking are also available while the problem setter defines the specifications. These specifications will be translated into a `.java` file. XText is used because of its availability as part of the Eclipse. Eclipse is a cross platform IDE, independent of a specific Operating System. Xtext is also integrated with JAVA so that our DSL can take advantage of the existing JAVA data type.

Some metamorphic relations are common in mathematical functions (Murphy, 2010), such as Additive (increases or decreases numerical values by a constant), Multiplicative (multiplies numerical values by a constant), Permutative (permutes the order of elements in a set), Invertive (takes the inverse of each element in a set), Inclusive (adds a new element to a set), Exclusive (removes an element from a set), and Compositional (creates a set from a number of smaller sets). These relations are generally applicable to tasks that deal with numerical inputs and outputs. Since many tasks in IOI involve numerical input and output, these relations are frequently applied. Therefore, we also have implemented these relations in our Library Function as reusable MR.

6. Case Studies

We applied the methods and tools to generate test cases for tasks with Knapsack, Greedy and DP solutions. More than that, we also demonstrate that the generic knapsack MRs can be used as a reusable specification for a more specific knapsack problem.

6.1. *Knapsack*

The Knapsack program accepts three sets of integers. Two n-tuple sets, $P = \{p_1, p_2, \ldots, p_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ represent the profits and the weights of n items,

respectively; while another m-tuple set $C = \{c_1, c_2, \ldots, c_m\}$ contains the capacities of m knapsacks. The outputs of Knapsack are one n-tuple set $Y = \{y_1, y_2, \ldots, y_n\}$ and one positive integer TP. $y_i = j$ (where $i = 1, 2, \ldots, n$ and $j = 0, 1, \ldots, m$) states that the i^{th} item should be put into the j^{th} knapsack. If $y_i = 0$, it means that the i^{th} item will not be selected into any knapsack. TP represents the total profit of the picked items. The Knapsack program attempts to calculate the optimal solution and thus to maximize the total profit. (Mahmuda *et al.*, 2011).

For generic knapsack problem, we adopted 10 Metamorphic Relations defined by Mahmuda (Mahmuda *et al.*, 2011) and translated into 10 MR declarations, MR1 to MR10. These MRs will be used to generate input test cases and checkers to check the relation between outputs. Examples of MR1 and MR5 are translated into DSL expressions :

1. **MR1**: Swap the k^{th} and the l^{th} items, where $1 \leq k < l \leq n$, and $p_k \neq p_l$ or $w_k \neq w_l$. We can get the follow-up test case $T' = \{P', W', C\}$, where $P' = \{p_1, p_2, \ldots, p_l, \ldots, p_k, \ldots, p_n\}$ and $W' = \{w_1, w_2, \ldots, w_l, \ldots, w_k, \ldots, w_n\}$. The output corresponding to T' is $O' = \{Y', TP'\}$. We should have $Y' = \{y_1, y_2, \ldots, y_l, \ldots, y_k, \ldots, y_n\}$ and $TP' = TP$.

   MR1 expressed in DSL :

```
MR 1 {
  (select(k,l) where 1<=k and k<l and l<=n and p[k]!=p[l]
      or w[k]!=w[l])
  followup {
    (p' = swap(p,k,l))
    (w' = swap(w,k,l))
  }
  check {
    (y' = swap(y,k,l))
  }
}
```

2. **MR5**: Change the capacity of the 1^{st} knapsack to a new value c_1', where c_1' is equal to the sum of the weights of all items put into the 1^{st} knapsack. We can get the follow-up test case $T' = \{P, W, C'\}$ where $C' = \{c_1', c_2, \ldots, c_m\}$. The output corresponding to T' is $O' = \{Y', TP'\}$. We should have $Y' = Y$ and $TP' = TP$.

   MR5 expressed in DSL :

```
MR 5 {
  (def c1 = sum(w) where y[i]==1)
  followup {
    (c'[1] = c1)
  }
  check {
    }
}
```

This problem has multiple values that can produce an optimal total profit. Therefore, we have to define a checker to verify the correctness of a solution. A checker (source code) will be generated from the specification to check the following properties :

1. The total profit of the output produced must be equal to the total profit generated in the answer.
2. The sum of all profits of the item must be equal to the total profit.
3. The sum of weight of all items in each knapsack must be less than or equal to the capacity of the corresponding knapsack.

MR1 to MR10 will be used in follow-up test-case generation. We give an illustration of test-case generation for MR1 and MR5 in Table 1.

Complete implementation of the DSL specification of the Knapsack problem, all generated input, output, and checkers are accessible in:

`https://github.com/ryanignatius/CheckerDSL/tree/master/Examples/Knapsack/Knapsack1`.

6.2. *Specific Knapsack*

From a knapsack case study, we can see that MR1 to MR10 can be used for other tasks with a knapsack solution, for example "Polo the Penguin and The Test" (`http://www.codechef.com/problems/PPTEST`). Here is an example of how test cases and checkers from the knapsack case study can be reused for another task that has the nature of a knapsack problem. In this task, there is one knapsack. The amount of time represents the capacity of the knapsack. Tests represent the items that must be put in the

Table 1

Test-case generation example for MR1 and MR5 of a Knapsack problem

Test Case	File Input	File Output	Explanation
Original Test Case	3 2 5 4 8 2 3 5 6 1	1 1 0 9	Input and Output are defined by a problem setter. Input: 1st line: 3 items to be put in 2 knapsacks 2nd line: profit of each item 3rd line: weight of each item 4th line: capacity of each knapsack Output : 1st line: knapsack number for each item Total Profit = 9
MR1	3 2 5 8 4 2 5 3 6 1	1 0 1 9	File input and output are generated based on the original test case and MR1 (by swapping the 2nd and 3rd item) Output: Total profit = 9
MR5	3 2 5 4 8 2 3 5 5 1	1 1 0 9	File input and output are generated based on the original test case and MR5 (by changing the capacity of the 1st knapsack to the sum of the weights of all items put into the 1st knapsack)

knapsack. Profit is analogous to the number of tests contain this question (C[i]) multiplied by the number of points of this question (P[i]). However, MR10 is not applicable since the number of knapsacks is one. We replace MR2 (to add profit to an item) by MR11 and MR12. MR 11 is to add the number of tests (C[i]) to an item. MR12 is to add the number of points to an item (P[i]). We also remove the variable y, since the task asks for the total profit only.

We generate test cases and checker for another example ("farmer"), is taken from IOI 2004 task (`http://www.ioinformatics.org/locations/ioi04/contest/day2.shtml#p2`). This task can be modeled as a knapsack problem, so we can use the same MRs of the knapsack problem to generate test cases and checkers for this task. In this task, there is one knapsack. The number of cypress trees to be selected represents the capacity of the knapsack. Fields and strips represent the items. The number of trees in each field represents a profit and weight of the item. The number of trees in each strip represents weight for the item and the profit for this item equals to the weight of this item minus one. MR10 is not applicable since the number of the knapsack is one. We replaced MR1 (to swap two items) by MR11 and MR12. MR11 is to swap two fields and MR12 is to swap two strips. We replaced MR6 (to add a new item) by MR13 and MR14. MR13 is to add a new field and MR14 is to add a new strip. We replaced MR7 (to delete an item) by MR15 and MR16. MR15 is to delete a field and MR16 is to delete a strip. MR3 and MR4 are not applicable to this task since the weight and profit of an item can't be manipulated individually.

In this case study, we have demonstrated the reusability of our system and how to modify an existing metamorphic relation for a variant of a Knapsack problem.

Detailed implementation of DSL specification for this Knapsack problem, the generated input, output and checker are accessible here:

`https://github.com/ryanignatius/CheckerDSL/tree/master/Examples/Knapsack/Knapsack2`

and

`https://github.com/ryanignatius/CheckerDSL/tree/master/Examples/IOI%20Task/farmer`.

6.3. *Greedy*

The Greedy program (key-lock problem) receives input that is a set of keys $K = \{k_1, k_2, \ldots, k_x\}$ and a set of locks $L = \{l_1, l_2, \ldots, l_y\}$, where $x, y > 0$. For every pair (k_m, l_n), we define $r(m, n)$ as a relationship between key k_m and lock l_n such that $r(m, n) = 1$ if k_m opens lock l_n and $r(m, n) = 0$, otherwise. (Barus *et al.*, 2011)

We adopted nine Metamorphic Relations defined for this problem from Barus (Barus *et al.*, 2011). This problem does not need a checker, therefore the checker session is "NONE". Examples of MR and DSL expressions for Greedy problems are given as follows.

1. **MR3**: Adds an insecure lock column

MR3 expression in DSL :

```
MR 3 {
  followup {
    (y' = y+1)
    (m' = addColumn(m))
    for (i,x){
      (m'[i][y] = 0)
    }
  }
  check {
  }
}
```

2. **MR8**: Adds an exclusive lock to an unselected key

MR8 expression in DSL :

```
MR 8 {
  (select(k) where not contain(o,k))
  followup {
    (y' = y+1)
    (m' = addColumn(m))
    for (i,x){
      (m'[i][y] = 1 where i==k)
      (m'[i][y] = 0 where i!=k)
    }
  }
  check {
    (numKey' = numKey+1)
    (o' = add(o,k))
  }
}
```

An illustration of test-case generation for MR3 and MR8 are given in Table 2.

Detailed implementation of DSL specification for the Greedy problem, the generated input, output and checker are accessible here:

`https://github.com/ryanignatius/CheckerDSL/tree/master/Examples/Greedy/Greedy1`

6.4. *Other Case Studies from the Indonesian National Informatics Olympiad*

The same method has been used for test cases and checkers generator for some tasks in the Indonesian National Informatics Olympiad with MR corresponding to a DP (Dynamic Programming) solution. Complete definition of the tasks, MRs, test-case specification and the generated input, output, and checkers are accessible from:

`https://github.com/ryanignatius/CheckerDSL`

Table 2

Test-case generation example for MR3 and MR8 of a Greedy problem

Test Case	File Input	File Output	Explanation
Original	3 4 1 0 1 0 1 0 1 1 0 1 0 0	2 2 3	Original test case defined by problem setter. 1st line: the number of keys and the number of locks. 2nd line until the last line contains the definition of each key. Each number in each line defines the relation between a key and a lock.
MR3	3 5 1 0 1 0 1 1 0 1 1 1 0 1 0 0 1	2 2 3	File input and output are generated based on the original test case and MR3 (adding an insecure lock column)
MR8	3 5 1 0 1 0 1 1 0 1 1 0 0 1 0 0 0	3 1 2 3	File input and output are generated based on the original test case and MR8 (adding an exclusive lock to an unselected key)

7. Conclusion

A specifications-based test-case generator has been built for improving test-case generation and checkers. The generator has been used for Indonesian training-program task definition. The relation between input and output is checked by running the contestant's program. Whereas in the classical way checker is written to check one output run, in our system the checker is capable of checking the relation between two or more output executions, based on the Metamorphic Relation. Instead of writing a program, a problem setter writes a specification based on a DSL grammar. The specifications contain variables and their values, input-output format, input-output values, input-output constraints, score, MR and predicates representing a checker. The specifications will then be used to generate test cases and checkers. However, the usage of this system is not intuitive unless the problem setter has a minimum knowledge and understanding of MR and our DSL.

For a problem class, MR represents a property that can be reused in other similar problems in the same domain. We have proven the reusability of metamorphic relation for Knapsack, Greedy, DP, and numerical problems for generating test cases and checkers for an Indonesian national training task and IOI task. By using our method and tools, a problem setter can take advantage of previous experience and enrich the system.

The system also provides a library of predefined functions that will grow along with the experience of the problem setter. This generator will be useful for simple problems in which the input-output relationship can be expressed easily, even without writing solutions. This is the case in preliminary selection, such as national preparation where we have to conduct training programs with many simple tasks.

In this version, the code is generated in JAVA. In the future version, the generated code can also be applied to other languages, such as C, C++, or Pascal, by changing the DSL grammar.

References

Barus, A.C., Chen, T.Y., Grant, D., Kuo, F.C., Lau, M.F. (2011). Testing of heuristic methods: a case study of greedy algorithm. In: Zbigniew H. *et al.* (Eds.), *3rd IFIP TC 2 Central and East European Conference on Software Engineering Techniques (CEE-SET 2008), Brno, Czech Republic, 13–15 October 2008. (Lecture Notes in Computer Science, 4980)*. 246–260

Chen T.Y., Cheung, S.C., Yiu, S.M. (1998), *Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01*. Hong Kong, Department of Computer Science, Hong Kong University of Science and Technology.

Chen, T.Y., Huang, D.H., Tse, T.H., Zhou, Z.Q. (2004). Case studies on the selection of useful relations in metamorphic testing. In: *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*. Madrid, Spain, 569–583

Ghosh, D. (2011). *DSLs in Action*. Manning Publication, 2011

Gotlieb, A., Botella, B. (2003). Automated metamorphic testing. In: *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*

Im, K., Im, T., McGregor, J. D. (2008). Automating test case definition using a domain specific language. In: *ACM-SE 46 Proceedings of the 46th Annual Southeast Regional Conference on XX*. 180–185.

Mahmuda, A., Liu, H., Kuo, F.-C. (2011). On testing effectiveness of metamorphic relation: a case study. In: *Fifth International Conference on Secure Software Integration and Reliability Improvement, Jeju Island Korea, 2011.*

Mirzayanov, M. (2008). *Testlib*. `https://code.google.com/p/testlib`

Murphy, C. (2010). *Metamorphic Testing Techniques to Detect Defects in Applications without Test Oracles*. Columbia University Dept of Computer Science tech report cucs-010-10.

XText. (2014). `https://eclipse.org/Xtext`

Zhou, Z.Q., Huang, D.H., Tse, T.H., Yang, Z., Huang, H., Chen, T.Y. (2004). Metamorphic testing and its applications. In: *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*. Japan, Software Engineers Association.

R.I. Hadiwijaya is a student in Informatics Engineering, Institut Teknologi Bandung, and an assistant in Programming Laboratory, Data & Software Engineering Research Group. He is doing his research in development of a systematic checker and test case generation for the automated grading system as a part of his final project, under supervision of Inggriani Liem.

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

Olympiads in Informatics, 2015, Vol. 9, 89–112
DOI: http://dx.doi.org/10.15388/ioi.2015.08

The Effectiveness of Robotics Competitions on Students' Learning of Computer Science

Fatima KALOTI-HALLAK, Michal ARMONI,
Mordechai (Moti) BEN-ARI
Weizmann Institute of Science, Israel
e-mail: fatima.kaloti.hallak@gmail.com, michal.armoni@weizmann.ac.il,
moti.ben-ari@weizmann.ac.il

Abstract. This work investigates students' learning of computer science (CS) as part of a research project on students' learning of and attitudes toward STEM (Science, Technology, Engineering, and Mathematics) subjects during their participation in robotics activities. The population consisted of groups of middle-school students (ages 13–15 years) who participated in the FIRST® LEGO® League competition. The methodology used is both qualitative and quantitative using questionnaires, observations and interviews during the school year 2012–2013, and mainly group interviews during the school year 2013–2014. A *representational model* was used during the interviews to facilitate externalizing the students' understanding of STEM concepts. The analysis used the revised Bloom Taxonomy (BT) to study the students' meaningful learning. Two CS concepts were investigated: input-output and interfacing with sensors. The results showed that during their preparation for the competition, almost all the students demonstrated meaningful learning, although some students reached higher levels of the BT than others.

Keywords: computer science, middle-school, extra curriculum, robotics, competitions, FLL.

1. Introduction

Recently, the number of schools that participate in robotics activities has increased and a few of them have tried to integrate such activities into their school curriculum. In order to improve learning of STEM subjects and to increase enrollment, educators have suggested that robotics be integrated into schools at many levels from middle school through college (Anderson *et al.*, 2011). In particular, competitions like the FIRST® LEGO® League (FLL) competition are the primarily type of robotics activities in schools. Most of the existing literature shows that students can be motivated and enthusiastic about participation in robotics activities. However, there are very few empirical studies that demonstrate improvement in students' learning of STEM. This research project focuses on investigating students' learning of and attitudes toward each of the STEM subjects during their participation in robotics activities (Kaloti-Hallak, 2014). This paper presents results on the learning of CS; subsequent publications will present

the results on learning of other STEM subjects, and the results of the investigation of students' attitudes.

Our research concerned the achievement of learning by middle-school students participating in the FLL competition. The data was collected by using a variety of instruments: pre- and post-questionnaires, observations and interviews. The analysis was primarily qualitative based on the cognitive process dimension of the (Revised) Bloom Taxonomy (BT) (L. Anderson *et al.*, 2001). Quantitative analyzing of the questionnaires was also conducted.

The research question (limited to the scope of this article) is:

What scientific content knowledge do students learn during their participation in the FLL competition?

a. To what extent is CS content knowledge is learned?
b. To what extent is the students' learning meaningful?

The background is given in Section 2, followed by the presentation of the methodology in Section 3. The data analysis is in Section 4, the findings are presented Section 5, and they are discussed in Section 6 and concluded in Section 7.

2. Background

2.1. *Robotics in education*

Robots have been used in both community outreach programs and academic institutions at all educational levels (M. Anderson *et al.*, 2011), even in special needs education (Virnes *et al.*, 2008). Robots, as physically manifested computing devices, inherently show students how the programs that they write can impact the real world. Robots are generally used to motivate students' interest in further study of science and technology (Lauwers *et al.*, 2009). It facilitates hands-on programming, increasing the quality of interaction between the child and the robot (a command to the robot is followed by the feedback of the robot's behavior), and improves the quality of instruction and intervention (Virnes *et al.*, 2008).

Fagin and Merkle (2003) examined the effectiveness of robotics in encouraging first-year university students to select computer science or computer engineering. In general, results were negative: test scores were lower in the robotics sections than in the non-robotics ones, and the use of robots did not have any measurable effect on students' choice of discipline. Summet *et al.* (2009) assigned a robot to each student, defined a curriculum and developed an interactive environment. They claimed that the approach was successful, and that it encouraged more students' to enroll in a higher level CS classes. However, the authors were somewhat equivocal about the results, noting that the "robots approach does not appear to be doing harm."

Lauwers, Nourbakhsh and Hamner (2009) worked with CS educators to investigate the features of robots that are well-suited to the learning goals of CS in an introductory university-level CS course. The results were compared with courses taught previously

without robots. They found a significant increase in students' positive attitudes and motivation to learning with robotics activities; the students also completed all the assignments and tests scores were significantly higher than in prior years. However, retention did not improve.

Martin (2006) claimed that the physical imperfections in robots and the environment can help students deal with unexpected problems better than computer-only activities. Studies have shown that students' response to a problem can be inadequate, because they engage in trial-and-error until they succeed or give up. For example, Sullivan (2008) showed that even academically advanced middle-school students who were motivated to join a robotics research group and eventually succeeded in solving problems did so through trial-and-error.

2.2. *Robots and Robotics Competitions*

Several kinds of robots have been used in education. For example, the Scribbler robot (Summet *et al.*, 2009), the iRobot Create (Anderson *et al.*, 2011), the Topobo robot (Virnes *et al.*, 2008), or the Thymio (Riedo *et al.*, 2013). One of the most widely used educational robots is the LEGO® MINDSTORMS® kit. Robots consist of the mechanical robot platform, motor(s), an onboard computer and a system for communicating with a desktop computer used for programming, sensors and software for programming the robot. The software can be a visual programming environment, or it can be an adaptation of an ordinary programming language.

Most schools engage in the robotics activities through competitions, including: the Trinity College Fire-Fighting Home Robot Contests (TCFFHRC) (`http://www.trincoll.edu/events/robot/`) (Verner and Hershko, 2003), the Botball contest (`http://www.botball.org/`) (Miller and Stein, 2000), Robo Fest (`http://robofest.net`), and the FLL (`http://firstlegoleague.org`) or the FIRST Robotics Competition (`http://www.usfirst.org/roboticsprograms/frc`) (Melchior *et al.*, 2005).

The FLL is a yearly competition for children in grades 4 to 8 using the LEGO® MINDSTORMS® kit. The kit contains LEGO® bricks as well as motors, gears and sensors. Programs are written on a personal computer using a visual programming environment called LabView and downloaded to the NXT controller, so that the robot can run the program without being tethered to the computer. Students can download instructions for constructing several robots; normally, one or more of these robots are built to obtain experience before trying to design a new robot.

The FLL competition consists of three parts:

1) The students are required to design and build a robot that fulfils missions in a scenario that changes every year; robots that fulfil the largest number of missions in the shortest time win the competition.
2) A scientific project that challenges students to create an innovative solution for a specific problem.
3) Developing core values that emphasize teamwork.

My research focuses on the first part of the competition that requires students to design and build robots.

The FLL competition's missions are designed according to an authentic theme; in year 2012–2013, the first year of the research was conducted, the FLL theme was "Senior Solution" and contained missions that simulate assisting senior citizens in areas that they may find difficult. A score is associated with each mission and can be accumulated. In year 2013–2014, the FLL theme "Nature's Fury" and contained missions that simulate helping people prepare, stay safe or rebuild in case of natural disaster. Yearly competitions have different themes, but they have common goals: promoting robotics in education and encouraging systems-thinking, problem solving, and teamwork skills.

We choose to work within the context of the FLL competition for several reasons: (i) it offers a different theme each year that is related to STEM and real world problems; (ii) requires that the students make a presentation on the subject; (iii) it targets students as young as 9 years old; (iv) it is the one that is available to us.

2.3. *The Bloom Taxonomy and Meaningful Learning*

The Bloom Taxonomy was first described in 1956 as a hierarchical model for the cognitive domain that organizes the cognitive aspects of learning into six hierarchical levels: knowledge, comprehension, application, analysis, synthesis, and evaluation (Bloom *et al.*, 1956). Given its popularity through the years, the taxonomy has been condensed, expanded and reinterpreted in a variety of ways (Forehand, 2012; Johnson and Fuller, 2006). The model was revised by Anderson *et al.* (2001) with a number of significant changes to the terminology, structure and emphasis. The revised structure has two dimensions: a cognitive process dimension with the original categories of Remembering, Understanding, Applying, Analyzing, Evaluating, and Creating, and a knowledge dimension with the categories Factual, Conceptual, Procedural and Meta-Cognitive. The new version is referred to as the *revised* Bloom Taxonomy (Thompson et al. 2008). We chose to work with the revised BT because it appears to offer appropriate categories for evaluating students' meaningful learning.

Ausubel (1963, 2000) defined meaningful learning as the subsumption or incorporation of new learned material into the student's cognitive structures. The goal of meaningful learning is to teach students concepts that will be recalled and used; therefore, meaningful learning strategies must build complex knowledge structures in the learner's mind (Ausubel, 2000). It is commonly accepted today that generalizations cannot be presented or given to the learner, but can only be acquired as a product of problem-solving activities. Meaningful learning occurs when students build the knowledge and cognitive processes needed for successful problem solving. The five categories of the taxonomy (understanding, applying, analyzing, evaluating and creating) are increasingly related to transfer, while the remembering category is related to retention (Mayer, 2002).

Technology can make learning more meaningful. Howland, Jonassen and Marra (2011) present five characteristics that are necessary to achieve meaningful learning using technology: active, constructive, intentional, authentic, and cooperative. In constructionist learning (Turkle and Papert, 1991), students engage in active cognitive processing. The revised BT cognitive processes describe the range of students' cognitive activities in meaningful learning that it is the way students can actively engage in the process of constructing meaning (Mayer, 2002).

LEGO® MINDSTORMS® are conjectured to facilitate meaningful learning (Miller, Nourbakhsh and Siegwart, 2008). Robotics competitions with LEGO® MINDSTORMS® require that students collaborate in order to accomplish the tasks required, such as the missions of an FLL competition. The students start with an assigned authentic project, and as they become more familiar with the technology, they achieve more control over constructing and programming the robot and can implement their own creative ideas.

2.4. *Computer Science Concepts*

Performing robotics activities requires mastery of CS concepts. Computer science encompasses far more than programming (Denning and McGettrik, 2005). Denning *et al.* (1989) coined the phrase *discipline of computing* to combine the analysis and abstraction of computer science with the abstraction and design of engineering. The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation and application. It includes nine subareas: algorithms and data structures, programming languages, architecture, numerical and symbolic computation, operating systems, software methodology and engineering, database and information retrieval systems, artificial intelligence and robotics, and human-computer communication. See Denning *et al.*, (1989) for details of each subarea.

Among all the CS concepts that the students demonstrated, we chose to focus on two:

(a) input-output (not just from the screen and keyboard) which appears in the programming languages subarea;
(b) interfacing with sensors from artificial intelligence and robotics subarea.

These concepts were chosen because: (a) students displayed a range of engagement with the concepts during the robotics activities; (b) the concepts were not previously known by most of the students; and (c) the concepts touch on more than just programming, since students must identify the different sensors and their capabilities, understand how they can be used for input and output, as well as the algorithmic transformation of the input data to the output data.

The definitions of the concepts that we use here are based on robotics books such as Trobaugh (2010), Kumar (2009) and Martin (2001). In more detail:

a. Input-output: Computer programs receive input data, perform computations and then produce output data. In robotics, the input comes from sensors that measure the dynamic environment of the robot (light, color, proximity, touch) and output (motors). This concept will be introduced when the students connect an input such as light sensor and output such as the motors to the NXT controller. The controller has four input connectors for sensors and three output connectors for motors. The expected learning outcomes can range from just identifying the input-output ports (limited learning) to explaining the difference between the input and output devices and how they work (more extensive learning), up to the highest level of devising new solutions based on these concepts.
b. Interfacing with sensors: Sensors in robotics can be used for detecting an object or for following a line on the mission table using a light or color sensor. The ultrasonic proximity sensor can detect this distance to an object and the touch sensor detects when the sensor touches something in the environment. The interface with these sensors is by programming the controller. The expected learning outcomes can range from identifying the techniques and algorithms used in interfacing with sensors (limited learning) to modifying the techniques or creating new ones (more extensive learning).

3. Methodology

3.1. *Population*

The research population consisted of middle-school students in Israel and the Palestinian Territories, aged 13–15 years, who participated in FLL robotics competition. The robotics activities were extracurricular, after school, on weekends and during vacations. Most of the teachers have no background in robotics; they are trained for a few months before they teach robotics and supervise the activities.

During the year 2012–2013, eight groups participated in the FLL competition. Six groups (Group 1, Group 2, Group 5, Group 6, Group 7 and Group 8), a total of 47 students (34 females and 13 males) participated in the competition for the first time, while Group 3 and Group 4, a total of 15 students (all female), had previous experience with the robotics. Ten of them had participated in the previous competition in 2011–2012.

The students who were interviewed by the researcher are referenced by S associated with a number. A summary of groups is presented in Table 1.

Five groups (Group 1, Group 2, Group 5, Group 6 and Group 7), a total of around 37 students (29 females and 8 males), who had participated in the 2012–2013 competition continued on to participate in the 2013–2014 competition. Some of the teachers were the same in both years, as were many of the students. Each group included students who participated in the previous year's competition, as well as new students who had never participated before in robotics activities. The school of Group 7 initiated a regular robotics class in 2013–2014 and added it to their curriculum. Six of the students who had

never participated before in robotics activities participated in both the robotics class and the robotics competition. The interviews were conducted with two students (S16 and S17) who did not have experience in competitions, but who participated in the regular school robotics class. For two of the groups (Group 4 and Group 8) we were not able to collect data of their activities, and Group 3 did not participate in the competition for the year 2013–2014. A summary of the groups is presented in Table 2.

The robotics activities started around the first months of the school year and ended on the competition day, usually during the last two months of the school year.

The resources available to the students were the *LEGO*® kit, together with its manual, online tutorials and books, and handouts provided by the organizers. In the absence of a fixed curriculum and textbook, the students searched for information from all these resources.

Table 1

Research participants in the school year 2012–2013

Group Number	Students and Gender	Students with experience in robotics	Students Interviewed
1	9 F	-	S1, S2, S3
2	6 F	-	S8, S9
3	8 F	4	S12, S13
4	7 F	6	-
5	7 M	-	S4, S5
6	10 F	-	S6, S7
7	7 F	-	S10, S11
8	6 M, 2 F	-	S14, S15
Total	**62 (49 F, 13 M)**	**10 F**	**15 (11 F, 4 M)**

Table 2

Research participants in the school year 2013–2014

Group number	Students and gender	Students with experience in robotics	Students interviewed	Teachers
1	7 F	3	-	Different
2	5 F, 2 M	2	-	Same
3	Didn't participate in competition year 2013–2014			
4	Not able to collect data from this group			
5	6 M		-	Same
6	7 F	-	-	Same
7	10 F (6 of whom participated in robotics class)	4	S16, S17	Different
8	Not able to collect data from this group			
Total	**37 (29 F, 8 M)**	**9 F**	**2 F**	

3.2. *Research Instruments*

The data collection instruments included (a) pre- and post-questionnaires consisting of Likert-scale items to investigate the students' attitudes toward learning STEM and robotics activities; (b) observations during the school year; (c) semi-structured interviews conducted by the researcher with 2–3 students from each group; and (d) group interviews conducted by the judges during the competition day[1]. The observations and the interview were recorded on video.

The interview used a *Representational Model* (RM), defined as an inscription, image, analogy, physical construction or computer simulation that facilitates the externalization of students' knowledge and understanding. Students were asked to express the design of their robots (including aspect of mechanics, electronics and software) graphically or in writing, and to instruct the robot to accomplish one of the FLL missions using flowcharts, pseudo-code, or any other notation of their choice. Before the interview, a table with materials that students might use to express their thoughts was prepared: paper, pencils, crayons, rulers, a protractor, and LEGO® pieces.

The students were asked to relate to three types of activities (tasks) in their representational models: a) the engineering of the robot; b) the mathematics required to instruct the robot to perform the missions; and c) the programming and CS concepts required. In this paper we focus on the programming and CS concepts. In each case, an alternative task was prepared to be used if the student did not cooperate in the initial task.

For example, the students had drawn a path that the robot should take to reach the mission's location, determined the distance and angles the robot should take, and decided on the functions the robot needs to perform, such as lowering a handle or picking something up. The students were now asked to write or draw the programming instructions and to explain these instructions. If the student couldn't write or draw the program instructions, a written program prepared in the NXT environment is presented as an alternative task and the students were asked to explain it.

3.3. *The Operationalization of Meaningful Learning of the CS Concepts*[2]

The categories or levels[3] of the cognitive process dimension of the revised BT were used to analyze the meaningful learning of CS concepts during the robotics activities. The first level of the BT, **remembering**, promotes retention, while the five levels above **remembering**, promote transfer (Mayer, 2002). Mayer claims that in some subjects, you need to start with **remembering** in order to get to a meaningful learning goal, and we believe that this is true in robotics. However, in robotics, meaningful learning requires more of the students than simply recalling or recognizing factual knowledge.

[1] The first author was asked by the judges to participate in some of these interviews.
[2] Words in **bold** in this section and below refer to the categories of the revised BT and *italic* is used for the operational definition of each category.
[3] We use levels in preference to categories.

Therefore, we required that students achieve higher levels than **remembering** in order to demonstrate meaningful learning. The higher the level the more meaningful learning the students gain.

For the concept of input-output, we merged the **understanding** and the **applying** levels, because if the students *realize* the concept they also need to *implement* the related techniques. We merged the **analyzing** and the **evaluating** levels, because if the students *integrate* knowledge related to the whole design, they are also *differentiating*, *comparing* and *criticizing* the different performances of their design.

As for the concept of interfacing with sensors, we merged the **applying** with the **analyzing** level, because when the students *implement* a solution to a task, they go directly to *integrating* knowledge related to the whole design and *testing* the solution.

The BT levels for the CS concepts were operationalized as follows:

1. **Remembering:** The students demonstrate the remembering level if they are:
   a. (Input-output) *naming, listing* or *memorizing* facts and terms as they had been taught or mentioned before without demonstrating any meaningful grasp of the input-output functionality. For example, the students are mentioning the port sides of the NXT to attach the sensors and the sides for motors as they have been told without realizing the difference between them.
   b. (Interfacing with sensors) *naming*, *listing*, *memorizing* or *identifying* the available sensors and the command(s) responsible for interfacing with those sensors. For example, the students are *listing* the available sensors as have been told without knowing or trying to use them.
2. **Understanding:** The students demonstrate the understanding level if they are:
   a. (Input-output) *realizing*, *recognizing* and *implementing* the main ideas or knowledge of the input-output concept in a new problem.[4] For example, the students are *recognizing* that the sensors are input devices and connect them through the input ports, same with motors connecting them through the output ports of the NXT.
   b. (Interfacing with sensors) *recognizing* the facts (properties), techniques or algorithm related to the use of sensors' command(s). For example, the students are *recognizing* the NXT blocks responsible for operating the sensors. However, they are not trying to use any sensor.
3. **Applying** the students demonstrate the applying level if they are:
   a. (For input-output this level was merged with **understanding**).
   b. (Interfacing with sensors) *using* the relevant knowledge, techniques or algorithm to interface with the sensors in solving a problem.[5] For example, the students *use* the light sensor to follow a black line on the mission table ground. They are *using* a specific technique.
4. **Analyzing** the students demonstrate the analyzing level if they are:

[4] For this concept, the definition refers to the merged **understanding / applying** level.
[5] For this concept, the definition refers to the merged **applying / analyzing** level.

a. (Input-output) *explaining* the related knowledge of the input-output and *differentiating* between their functionality in relation to the overall structure of the robot.[6] For example, the students are *explaining* the robot's sudden stopping position (away from the mission table edges) as the reason is related to the approximate sensor measurements.
b. (For interfacing with sensors this level was merged with **applying**).

5. **Evaluating** the students demonstrate the evaluating level if they are:
   a. (For input-output this level was merged with **analyzing**).
   b. (Interfacing with sensors) *criticizing*, explaining and *modifying* the algorithm used for interfacing with sensors according to the robot's test results. For example, if the students used PID controller technique, they *modify* the algorithm to improve the robot's movements.
6. **Creating** the students demonstrate the creating level if they are:
   a. (Input-output) *coming up* with a new, alternative or unexpected solution, or *devising* a new strategy for using the input-output concept.
   b. (Interfacing with sensors) *coming up* with a new, alternative or unexpected algorithm.

4. Data Analysis

The students of each of the eight groups worked as a team on the activities for the FLL competition. Some students worked more on the scientific project part, but during the observations, the interviews and the competition day group interview, students used the plural pronoun 'we' (rather than 'I'), thus expressing the group's involvement and mutual responsibility. Therefore, we take the interviewed students as representative of the group as a whole, and not just to assess the learning of individual students.

The analysis of the data from the school year 2012–2013 focused on the observations of the students during the activities, and on the interviews of 15 students after the activities, while the analysis for year 2013–2014 focused on the group interviews during the competition day, except for Group 7, whose data was collected during the year and focused on the observations and the interviews of two students. The students who were interviewed by the researcher are referred to by S associated with a number as presented in Table 1 and Table 2 in section 3.1 above.

The research data collected were analyzed quantitatively and qualitatively. The quantitative analysis was relevant only to the investigation of students' attitudes and will be reported separately. The qualitative analysis examined the transcriptions of the observations and interviews that were videotaped. The transcriptions were analyzed according to the BT, as operationalized above. The analysis of the students' verbalization during the observations and their interaction with the representational models were inspired by Chi's (1997) verbal analysis for quantifying qualitative data.

[6] For this concept, the definition refers to the merged **analyzing / evaluating** level.

The qualitative analysis regarding the CS discipline was performed as follows:

1. Segmenting each group's data according to the activity: the beginning, the middle and the end of the year (the competition).
2. Presenting the segments in tables; the rows of each table demonstrated a specific concept accompanied with detailed information and explanation.
3. An analysis for all the groups was conducted according to the segments presented in step (1) and further analyzed according to the concepts.
4. Another analysis was carried out for each of the concepts related to all groups.
5. The findings were summarized. The learning level of each group for each concept was presented in a table.

To ensure validity of the qualitative analysis, the researcher was not involved in the teaching and learning process, but only in preparing the research instruments and performing the data analysis. Triangulation among instruments was used to ensure the accuracy of the results. In addition, an independent analysis of the results was performed by a colleague to ensure theoretical validity. The few disagreements that occurred were negotiated until a consensus was reached.

5. Findings

This section presents the findings of learning CS during the preparation for the FLL local competitions. For each concept, the findings are summarized in a table for each group and then explained according to the BT. For groups that were investigated in both 2012–2013 and 2013–2014, there are two rows for the group in the table, one (white) for the first year and one (gray) for the second year. The results regarding learning are depicted by an arrow that starts at the initial BT level and ends at the BT level the students achieved at the end of the activities. The results of all groups in year 2013–2014 refer to the final period of the robotics activities (the competition day), except for Group 7 where the results refer to the whole period from the beginning of the activities until the competition day in 2013–2014.

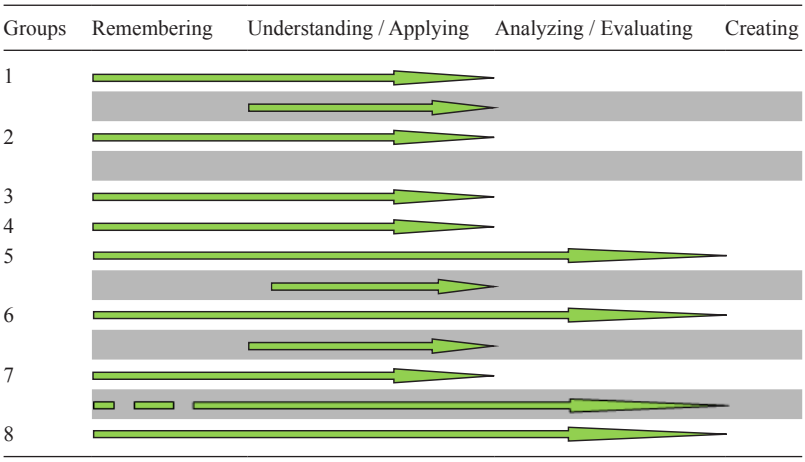
5.1. *Input-Output*

5.1.1. *Summary of the Learning Levels that Were Achieved*

Table 3 summarizes the results of the input-output concepts.

Regarding the groups who had participated in the robotics activities for the first time in year 2012–2013, three out of the six groups (Group 5, Group 6, and Group 8) demonstrated learning at the **analyzing / evaluating** level of the BT. The groups started the activities at the **remembering** level. Three groups (Group 1, Group 2 and Group 7) started at the **remembering** level and at the end demonstrated a learning level of no more than the **understanding / applying**. The same results were demonstrated for the two groups (Group 3 and Group 4) who had previous experience in the robotics activities.

Table 3
Results for the computer science concepts – input-output



In year 2013–2014, one group (Group 7) demonstrated learning at the **analyzing / evaluating** level, and three groups (Group 1, Group 5 and Group 6) demonstrated learning at the **understanding / applying** level, while the data that was collected for Group 2 were not sufficient to demonstrate the learning level for this concept. The groups' starting level could not be determined because the data were gathered only during the group interview on the competition day. The results showed that for the year 2013–2014 Group 5 and Group 6 demonstrated a lower level of what they demonstrated the year before.

The data of Group 7 were collected throughout the school year and showed that the new students demonstrated the **remembering** level at the beginning of the activities (presented in the table above as dashed arrows) and joined the **understanding / applying** level with the experienced students when they participated in the school robotics class. Later in the activities, all students demonstrated the **analyzing / evaluating** level.

5.1.2. *Examples for the Learning Levels and Expanded Observations*
The following points were observed and can be illuminate and broaden these results:

- **Limited learning**: All the groups / students *memorized* the connection between the input-output devices with the NXT controller as they had been told, such as assigning the port 'A' always for the motor that is responsible for moving the manipulators. For example, the student S2 of Group 1 *named* the ports during the interview:

  S1: [writing on one side of the NXT (see Fig. 1)] 'A' [port] is for the [motor connected to the] robot handle [a manipulator, a LEGO® construction for dragging and lifting] always. And 'C' [port] is for the left wheel and 'B' for the right one [wheel]. Here is the USB for the computer. Now here [writing on the other side of the NXT] we have '1', '2', '3', and '4'.

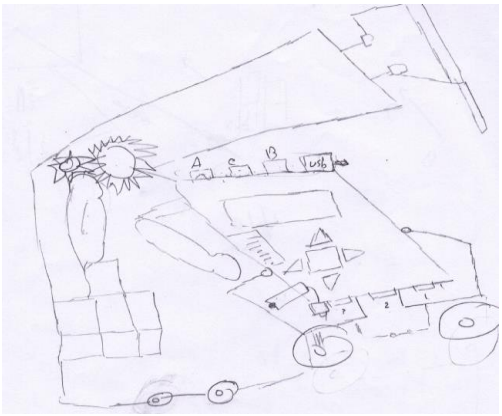


Fig. 1. Students' drawing of their robot.

When the students were asked if it is necessary to use that order, S12 of Group 3 responded during the interview:

> S12: Well, 'B' and 'C' [ports] … we can switch them, but 'A' [port], it has to be for [connected with] the handle [manipulator].

Students invariably used the same three outputs in this order. They *memorized* the location of input ports and the output port, but they did not know the difference since they only followed what they had been told.

- **The difference between input and output:** The location of the input ports on the NXT controller is on the side opposite of the location of the output ports. All students *realized* the locations and their connected devices, but treated them as all input or output. For example, Student S2 of Group 1 described the connections during the interview:

> S2: The NXT is the brain of the robot. I mean without it the robot would not move. When we connected and downloaded [the program] from the computer to the NXT, the robot moved … of course, by using the wires that are connected to the motors. The motors caused the handle [manipulator] and the wheels to operate. Here are the outputs 'A', 'B', and 'C' … '1', '2', '3', and '4', each of these are outputs for the sensors [use].

S2 *realized* the places and the connections of the ports with the NXT controller, *described* the download of the program that caused the manipulators to operate by the motors, and *mentioned* that the wires connect the sensors, motors and the USB with the NXT controller. However, she referred to them as output, then and during the interview:

> S2: No, these are inputs [thinking] these are inputs [pointing to the '1', '2', '3'and '4'], all of them. [Explaining by showing confidence] these four

> [pointing to '1', '2', '3' and '4'] are inputs for the sensors, and these [pointing to the 'A', 'B' and 'C'] are inputs for the motors' movements. … Actually in the NXT there are not only 7 outputs, we have 8 outputs. The 8th is the speaker …. No … these are outputs not inputs ... ah ... yes outputs ... the speaker is … it gives sound. For example, when the robot reaches the destination, it gave a sound of 'Good bye' … the sound is an output.

Some of the students, similar to S2, were confused when they were asked to explain the difference between input and output. Eventually, after reflection, S2 was able to *understand* the distinction between inputs and outputs, when she *recalled* the knowledge concerning the speaker as an output component and *applied* it to another situation (the sensors and motors). These students demonstrated the **understanding / applying** level but not higher.

- **Developing a non-viable mental model**: A few students confused downloading a program into NXT controller with inputting sensor's data. Although they knew that speakers are an output, they found this inconsistent with the mental model that the insertion of all wires is input. For example, Student S3 of Group 1 said during the interview:

> S3: One output for the USB. The USB is an input ... we write a program and input it [download] from the computer to the NXT. The outputs are the sensors … the same as the motors. Outputs are from the motors to the NXT. The input, which is the only USB, which we used to download the program … these [pointing to the ports] are the input, from the motor to the NXT.

S3 referred to the connection of the USB as input and showed using their hands the downloading the program from the computer to the NXT. Student S1 mistakenly thought that the activity of connecting wires to the ports means input and that there is no difference between input and output:

> S1: I know there are inputs and outputs but I do not know what the difference is or where are they. According to what I know we connect the wires, these [the components that are connected by the wires with the NXT] are inputs. … Both [input and output] have the same meaning.

The mental model they created when they inserted the wires to the NXT caused them to call it input. S1 did not specify what are the inputs or the outputs but realized where each should be connected with the NXT controller.

- **Explaining the input-output devices' functionality but not the concepts**: Some students *realized* the functionality of the input-output devices. For example, Student S12 of Group 3 during the interview said:

> S12: The motors need different programming orders than the sensors. The sensors sense something [surroundings], I mean if the robot needs to do a mission, the robot will do it even if the place is different. And if something got different [in the environment] the sensor reflection rate would change. It

> is like a local plan, we do not know the values before the sensor operates … we have wires, these wires transfer the data by electronic charges, and the motor which is a device that transfers these data or order to movement.

S12 *explained* how the sensors read the surroundings and accordingly the robot moves or operates. S12 *described* how the data transferred and caused the robot to move. However, when S12 was asked about the concepts' terms:

> S12: I heard that the inputs are the programs that we download from the computer to the NXT controller. While that outputs are the missions that the robot accomplish, or may be the orders that we download from the NXT to the computer.

S12 thought that the process of downloading from and to the computer represent the terms input and output. She did not connect what she just explained about the sensors and the motors with the concept's terms. Therefore, S12 and other students reacting similarly demonstrated the **understanding / applying** level.

- **Implementing and explaining the data flow**: After connecting the motors to the NXT, the students were asked to explain the data flow from the sensors through the NXT and then to the motors. Most of the students *exemplified* the data flow by talking about the medicine mission. For example, Student S5 of Group 5 *exemplified* during the interview:

> S5: The robot gets the information from the sensor, analyzes it … For example, [if the sensor reflects a color of] black or white, then the "reversed feeding" [as the student called it] which is when the program evaluates and orders the motor to move accordingly.

Some of the groups *listed* all the inputs and outputs peripherals in the robot, *described* the data flow from the input to generate output, *applied* the information about the concepts by using different sensors and controlling the robot's movements. They were able to *differentiate* between the concepts and relate the knowledge to the overall structure and behavior of the robot.

Moreover, the students were able to *use* the information about input and output for the *purpose* of moving from one programmed mission to another in a single touch sensor press, or *using* the NXT buttons (as if they were sensors) to select the specific program for each mission and thus gain more time for accomplishing the missions within the limited competition time. For example, Student S4 of Group 5 said during the interview:

> S4: We used the buttons on the NXT as sensors; as it [the robot] waits until we press on one of the NXT button, then it will do the next mission [instead of clicking several buttons on the NXT fetching for the right program for a specific mission].

S4 *explained, manipulated* and *evaluated* the inputs and outputs to serve their needs. They expressed the *manipulation* to the overall structure either by using the

NXT buttons or the touch sensor. S4 and other students reacting similarly demonstrated the **analyzing / evaluating** level, although it could also be considered to be **creating** because it was so unexpected.

- **Connecting the information gained in a regular technology class with the robotics activities**: Some of the students *recalled* their knowledge of the previous years' regular technology classes within the context of the robotics activities. For example, Student S10 of Group 7 said during the interview:

> S10: I did not recognize the connection [of what she learned in the technology class and what she learned during the robotics activities] before [the interview]; this is awesome! … Now I understood the concept [input-output] more [than her understanding from the technology class].

S10, during the interview, *recalled* that the concepts input-output were mentioned in her regular technology class last year and *explained* the relevant of these concepts to the activities. However, this was only during the interview, when she was asked if they learned the concept input-output before the activities.

5.2. *Interfacing with Sensors*

5.2.1. *Summary of the Learning Levels that Were Achieved*

The students needed to program the sensors for two missions: 1) using a color sensor to detect the green medicine; and 2) using a light or color sensor to enable the robot follow the black or colored line on the competition table. Table 4 summarizes the results regarding the concept of interfacing with the sensors.

Regarding the groups who had participated in the robotics activities for the first time in year 2012–2013, one group (Group 5) out of the six groups, demonstrated learning at

Table 4

Results for the computer science concepts – interfacing with sensors

Groups	Remembering	Understanding	Applying / Analyzing	Evaluating	Creating
1					
2					
3					
4					
5					
6					
7					
8					

the **evaluating** level of the BT, and one (Group 7) demonstrated learning at the **applying / analyzing**. Both groups started the activities at the **remembering** level.

Three groups (Group 1, Group 6 and Group 8) demonstrated learning at the **understanding** level. The groups started the activities at the **remembering** level. Only one group (Group 2) demonstrated no more than the **remembering** level. As for the two groups (Group 3 and Group 4) who had previous experience in the robotics activities, they started learning at the **remembering** level and achieved learning at the **applying / analyzing** level.

In year 2013–2014, one group (Group 7) out of five demonstrated the level following the one they demonstrated the year before. They started at the **applying / analyzing** and achieved learning at the **evaluating** level. Two groups (Group 2 and Group 5) demonstrated learning at the **applying / analyzing** level. The students of Group 2 started at the **understanding,** which was the level following the one they had demonstrated the year before, while Group 5 demonstrated a level less than the one they had demonstrated the year before. Two groups (Group 1 and Group 6) demonstrated no more than **understanding**. These two groups' starting level was not clear because the data were gathered only during the group interview on the competition day.

The data of Group 7 were collected throughout the school year and showed that the new students demonstrated the **remembering** level at the beginning of the activities (presented in the table above as dashed arrows) and joined the **applying / analyzing** level with the experienced students when they participated in the school robotics class. Later in the activities, all students demonstrated the **evaluating** level.

5.2.2. *Examples for the Learning Levels and Expanded Observations*

The following points were observed and can be illuminate and broaden these results:

- **Limited learning**: All students followed the instructions presented in the LEGO® kit booklet to connect the light sensor. They *recognized* that the sensor block in the NXT software is used for the programming to interface with the sensors. They *named* the different kinds of sensors and briefly *listed* the purpose of using each one. For example, Student S15 of Group 8 *listed* the available sensors during the interview:

  > S15: The light sensor and color sensor for detecting the colors, the Gyro sensor, for detecting and measuring the angles, and the ultrasonic to let the robot avoid hitting the wall [competition table edges]. At the end we decided not to use any of the sensors...

  S15 *listed* the kinds of sensors and their uses but the students of Group 8, at the end of the activities, caused the robot to move according to the measurements instead of using any sensors. Therefore, the students of Group 8 and other students reacting similarly demonstrated the **remembering** level.
- **Implementing using sensors for specific reason**: Most of the students described the overall process of the sensors' functionality. For example, Student S12 of Group 3 used the touch sensor for the purpose of moving after getting hit into the competition table edges. she said:

> S12: We used the touch sensor to avoid hitting [the robot] into the wall [competition table edges]. We used to have hard time when the robot hits into the wall and stop [not doing the next order]. The touch sensor made the robot stop and then move forward.

The robot switches to the next order / command (or in this case block) in the program when it finishes the current one. When the robot are forced to stop because of hitting an edge, it is actually operating to finish the order of moving until the time or distance assigned ended. The students of Group 3 *used* the touch sensor to cause the next command to operate and thus keep the robot moving. These students and other students reacting similarly demonstrated the **applying / analyzing** level.

- **Using the dual- or multi-state technique / algorithm**: One of the NXT blocks is a block for interfacing with sensors. By filling out the needed parameters, the sensor operates accordingly. For example, Student S3 of Group 1 explained the method used for interfacing with the color sensor:

> S9: we programmed the robot so the sensor detects a color. The sensor's light turned on, and when it [the light attached with the color sensor] is on, the sensor detects the colors around it. There are colors that either reflect or absorb the light. The colors that reflect the light have rates of more than 50. While the colors that absorbs the light, usually have rates of less than 50. For example, we want the robot to move on [following] the black line [drawn on the competition table ground], the black color absorbs the light, so it is less than 50. So when the sensor detects a color that has a rate of less than 50, we assign an order [command or added a block] to do some action such as stop, turn or move.

S3 *explained* the relevant knowledge of light reflection, *described* the method of reading the rate of reflection and *manipulate* accordingly. This method called *dual-state* method. Most of the groups' students referred to this method when describing the interface of sensors.

A few of the groups' students descried the *multi-state* method. For example, Student S6 of Group 6 mentioned during the interview:

> S6: We determined the range of the colors; black, white, green and orange. Then we assigned the value in a loop. I remember using the switch [conditional statement block], if yes [the option within the range of a specific color], the robot should keep checking, otherwise stop … and things like that [for the rest of the colors]. But we did not have time to do it; we left the medicine mission to the end.

S6 *described* the algorithm by using the visual blocks related to sensors for detecting several colors. S6 also *interpreted* the process related to the repetition and conditional statements. However, the students of Group 6 did not actually *implement* the program; instead, they decided to discard and skip the accomplishment

of the medicine mission. Therefore, the students of Group 6 and the other students reacted similarly demonstrated the **understanding** level.

- **Using the PID technique / algorithm and comparing the results**: A PID controller modifies the output proportional (P) to the input value, its integral (I) and its derivative (D). This concept was mentioned by one of the mentors, but the students used only the simplest P (proportional) controller. The algorithm was used for the purpose of following the black line on the competition table ground. Student S4 explained the algorithm during the interview:

> S4: First we dragged the light sensor [the software block] … we had several laws that we wrote on a separate paper … We started applying these laws one after another: First thing we subtracted the 'perfect point' -50, in order to cause the robot move on the edge that had 50% white and 50% black, then we looked at the 'proportional constant', which was the multiplication by 0.9, because we did not want to cause the robot to move in a [noticeable] zigzag way. Then the program had to decide one of the two directions – subtract or add. … [With an appropriate speed] … we added the motor 'B' [added a condition block and then a move block for 'B'] and on the other [side of the condition block] motor 'C' [move block]. It depends on the robot's location; if it is on the right or the left side of the black line. All these are inside a big loop … So the robot moved right, smoothly and straight.

In addition, Student S5 of the same group gave an example:

> S5: If the sensor saw 60% [the reflection light rate] that means the robot was going to the white color a little, so we had an order to decrease that motor power and increase the other motor [each responsible for moving the front wheels], and the robot turned.

Both S4 and S5 of Group 5 *described, exemplified, checked* and *critiqued* the proportional method and *modified* the solution that caused the robot to move on the colored line with hardly any zigzag. Student S4 *compared* the experience of using sensors with the experience when the group started the activities without using any sensors. S4 reached to the conclusion that the group was wasting their time when they did not use any sensors. The Group 5 started the activities without using sensors, because they thought that using sensors might ruin the behavior of the robot, especially on the competition day that may have different environment (an issue mentioned by all groups). Subsequently, the students of Group 5 decided to *use* three sensors, which they *programmed* without assistance; they were highly motivated to extract information from sources such as online resources.

Although the students of Group 5 were confident of their work, the robot did not behave as expected on the competition day. Therefore, the students of Group 7, in year 2013–2014, *realized* the problem and *developed* an alternative plan. Student S16 *explained* the use of the gyro sensor during the interview:

S16: I agree that the use of sensor was beneficial. But for the competition we were afraid of using sensors. We had two programs one interface with sensors and one without sensors. We used the one [program] without sensors.

The groups who *used* the sensors and *explained* the algorithms demonstrated the **applying / analyzing,** while the one who *criticized* the algorithms and tried to *modify* it to get better results demonstrated the **evaluating** level.

5.3. *Two Items in the Attitude Questionnaires that Dealt with CS*

Although the investigation of attitudes is beyond the scope of this paper, we bring two items of the questionnaires that are relevant to CS learning. (1) "I think computer science knowledge is necessary for robotics." (2) "My future career will not be in computer science." The results showed that around 85% of the students agreed on the importance of having computer science knowledge for the robotics activities. The percentage was high when they started the activities and slightly increased by the end. For the second item, most of the students responded 'not sure' in both the pre- and the post-questionnaires.

6. Discussion

6.1. *Meaningful Learning According to the Bloom Taxonomy*

Almost all the students demonstrated meaningful learning as a result of participation in the robotics competitions. Most of the groups demonstrated learning up to the level of **understanding / applying**, except for one group that demonstrated learning at the low level of **remembering** for the concept of interfacing with sensors. Some of the students reached higher levels of the BT like **analyzing** or **evaluating**. A closer examination of the findings yields the following results.

Most of the students achieved the level of **understanding** or **applying** for both concepts, that is, they connected the input-output devices with the NXT controller correctly and tried to interface with sensors, as required for completing one of the missions. However, the scope of their learning was narrow: they did not distinguish between inputs and outputs, and eventually decided to discard the missions that required the use of the sensors. Indeed, both concepts necessitate exploring and searching for information, in order to reach higher levels of learning.

Fewer students reached the level of analyzing or evaluating level. These students explained the concepts, differentiated between inputs and outputs, and used the relevant knowledge in modifying the structure or the algorithms. The common characteristics of these groups were the exploration of the resources and discovering new solutions, with or without help from the teacher or mentor. For example, Group 7 which reached the evaluating level for both concepts had a teacher who employed a guided discovery learning pedagogy, while there was limited involvement by the teacher in Group 5 which

also reached the evaluating level for both concepts. This was compatible with Sullivan and Moriarty (2009); they examined the teacher's reflection on teaching and learning robotics through the discovery learning method, and found that the experience of finding information and solving unexpected problems was effective. We conclude that students and teachers need to be encouraged to work together in exploring and acquiring knowledge and in discovering new solutions.

Only one group (Group2) did not exceed the **remembering** level for the concept of interfacing with sensors. The students only listed the kinds of sensors available and their purposes. They could not describe their algorithm for interfacing with the light sensor even though it was part of the design. We believe that this poor performance was related to the teacher-centered instructional pedagogy that they experienced. The context—accomplishing the competition missions—certainly facilitates the use of these two CS concepts; however, the teacher's detailed instruction on how to use the sensors did not foster high levels of exploration. This is in contrast with the achievements of the other groups who reached higher levels of learning with teachers who were not using a fully teacher-centered pedagogy.

6.2. *Factors that Affect Learning in Robotics Competitions*

We found that certain factors characteristic of the robotics competitions seemed to play a role in determining the learning levels that the students achieved: (a) the competitive nature of the activities; (b) the teaching pedagogy; (c) the unstable nature of the design of the robots; (d) the curricular position of the activities.

The competitive nature of the activities. Two aspects of the competition influenced the students' learning: the mission requirements and the limited time available. The competition had a positive impact on students' learning, because they were challenged to solve problems in order to accomplish the missions. This is consistent with Melchior *et al.* (2005) who found that the FRC competition promoted a positive academic trajectory for its students. However, the competitive environment also had some negative impact. For example, some students did not attempt certain missions, because of the limited time they had thought that they would not be able to succeed in interfacing with the sensors. A few groups were able to manage their time and accomplish most of the missions—including the harder ones—so we cannot conclude that time limitations were the only negative factor for those students.

Teaching pedagogy. The results showed a high variability in the learning that the students achieved. As discussed above, discovery learning was explained variability between groups. Students experienced a teacher-centered pedagogy at the beginning of the activities, enabling to achieve the remembering level. Since most of the teachers had limited knowledge of robotics and the robotics competition, a shift to a learner-centered pedagogy occurred in most groups when the students and teachers realized that the shift was necessary in order to accomplish the missions. Both the students and the teachers (or in some cases the students alone) searched the available resources to construct more knowledge and to solve the unexpected problems. This supports the claim by Virnes, Su-

tinen, and Kärnä-Lin (2008) that the advantage of robotics activities is that they offer opportunities for exploration due to the frequent occurrence of unexpected problems. This raises the possibility that teacher-centered pedagogy may not be effective in the context of robotics competitions.

The unstable nature of the design of the robot. Martin (2006) noted that the physical variability in real-world robots and the environment can help students deal with unexpected problems. Our findings showed that the teachers did not hide the fact that "robots do not drive straight" and pointed out that sensors are unreliable. Some students became discouraged and tried to eliminate sensors in their design, while other students took it as a challenge. Indeed, the solutions that the students presented as a challenge, did not work very well in the competition, but the students acknowledged the benefit of using sensors and felt proud of their accomplishments. This may have been due to the students' determination to succeed, which was observed during their design sessions and reported in the interviews.

The curricular position of the activities. Most of the students from Group 7 who had participated in 2012–2013 competition also participated in a robotics course that the school introduced the following year. The new students who joined the group demonstrated the remembering level at the beginning of the activities and were able to catch up with the students who had participated in the competition the previous year. All students eventually demonstrated learning up to the evaluating level, exploring information outside the scope of the available resources and producing impressive robot designs. This supports the findings of Melchior, et al. (2005) that the FRC helped participating schools in introducing robotics courses in fostering a positive school spirit. An alternative explanation might be that the new students were helped by their more experienced teammates, and they could have reached a similar level of learning if the activities had been extracurricular. However, the observations and interviews showed that the experienced teammates did not mentor the new students; furthermore, additional material was taught in class that had not been part of the competition. In other groups who participated for a second year in extra-curricular activities, we could not observe similar results. Although no direct help by the experienced students was observed, nevertheless, the new students seemed to learn faster in this context.

7. Conclusions

The research showed that robotics competitions are effective in achieving meaningful learning of computer science concepts. Most students reached the middle levels of the Bloom Taxonomy and some reached higher levels. The most successful learners were those who engaged in exploration of resources in order to learn new concepts and to solve problems they encountered.

The competitions had both positive and negative effects. On the positive side, many students displayed a determination to accomplish the missions that led to effective learning behaviors. On the negative side, learning opportunities were sometimes pushed aside

in favor of constructing robots that tried to accomplish the missions. Further research is needed to determine the relative advantages and disadvantages of robotics competitions when compared with curricular robotics activities.

References

Anderson, L., Krathwohl, D., Airasian, P., Cruikshank, K., Mayer, R., Pintrich, P., Rath, J. and Wittrock, M. (eds.) (2001). *A Taxonomy for Learning and Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman.

Anderson, M., McKenzie, A., Wellman, B., Brown, M., Vrbsky, S. (2011). Affecting attitudes in first-year computer science using syntax-free robotics programming. *ACM Inroads*, 2(3), 51–57.

Ausubel, D. (1963). *The Psychology of Meaningful Verbal Learning.* New York, NY, Grune & Stratton.

Ausubel, D. (2000). *The Acquisition and Retention of Knowledge: A cognitive View*. Dordrecht, the Netherlands, Kluwer Academic Publishers.

Bloom, B., Krathwohl, D. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals, by a Committee of College and University Examiners. Handbook 1: Cognitive Domain*. New York, Longmans.

Botball website retrieved on April 2015 from `http://www.botball.org`

Chi, M., (1997). Quantifying qualitative analyses of verbal data: a practical guide. *The Journal of the Learning Sciences*, 6(3), 271–315.

Denning, P. (2010). The great principle of computing. *The Scientific Research Society*, 98, 369–372.

Denning, P., McGettrick, A. (2005). Recentering computer science. *Communications of the ACM*, 48(11), 15–19.

Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, A., Young, P. (1989). Computing as a discipline. *ACM*, 32(1), 9–23.

Fagin, B., Merkle, L. (2003). Measuring the effectiveness of robots in teaching computer science. *ACM SIGCSE Bulletin,* 35(1), 307–311.

FIRST Robotics Competition website retrieved on April 2015 from
http://www.usfirst.org/roboticsprograms/frc

FLL site Retrieved on April 2014 from `http://firstlegoleague.org`

Forehand, M. (2012). Bloom's Taxonomy, Georgia.
http://projects.coe.uga.edu/epltt/index.php?title=Bloom%27s_Taxonomy

Howland, J.L., Jonassen, D., Marra, R.M. (2011). *Meaningful Learning with Technology*. (4th Ed.) Pearson Education, Inc.

Johnson, C., Fuller, U. (2006). Is bloom's taxonomy appropriate for computer science? In: Berglund, A. (Ed.). 6th *Baltic Sea Conference on Computing Education Research* (*Koli Calling 2006*). Finland, Koli National Park, 115–118.

Kaloti-Hallak, F. (2014). The effect of Robotics Activities on Studetns' Learning and Attitudes. *ICER'14.*

Kumar, D. (2011). *Learning computing with robots.* Institute for Personal Robots in Education.

Lauwers, T., Nourbakhsh, I., Hamner, I. (2009). CSbots: design and deployment of a robot designed for the CS1 Classroom. *ACM SIGCSE Bulletin*, 41(1), 428–432.

Martin, F. (2001). *Robotic Explorations: A hands-on Introduction to Engineering*. Upper Saddle River, N.J.: Prentice Hall.

Martin, F. (2006). Real Robots Don't Drive Straight. American Association for Artificial Intelligence.

Mayer, R. (2002). Rote versus meaningful learning. *Theory into Practice*, 41(4), 226–232.

Melchior, A., Cohen, F., Cutter, T., Leavitt, T. (2005). More than Robots: An Evaluation of the FIRST Robotics Competition Participant and Institutional Impacts: Center for Youth and Communities, Brandeis University

Miller, D., Nourbakhsh, I., Siegwart, R. (2008). *Robots for Education.* Springer. 1283–1301.

Miller, D.P., Stein, C. (2000). "So that's what pi is for!" and other educational epiphanies from hands-on robotics. In: Druin, A., Hendler, J. *Robots for Kids, Exploring New Technologies for Learning*. Morgan Kaufmann Publishers, 219–244.

Riedo, F., Chevalier, M., Magnenat, S., Mondada, F. (2013). Thymio II, a robot that grows wiser with children. In: *IEEE Workshop on Advanced Robotics and its Social Impacts* (*ARSO*), 187–193.

Robofest website retrieved on April 2015, from http://robofest.net
Sullivan, F. R. (2008). Robotics and science literacy: thinking skills, science process skills and systems understanding. *The Journal of Research in Science Teaching,* 45(3), 373–394.
Sullivan, F.R., Moriarty, M.A. (2009). Robotics and discovery learning: pedagogical beliefs, teacher practice, and technology integration. *The Journal of Technology and Teacher Education*, 17(1), 109–142.
Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., Balch, T. (2009). Personalizing CS1 with robots. *ACM SIGCSE Bulletin*, 41(1), 433–437.
TCFFHRC website retrieved on April 2015, from http://www.trincoll.edu/events/robot
Thompson, E., Luxton-Reilly, A., Whalley, J., Hu, M., Robbins, P. (2008). Bloom's taxonomy for CS assessment. *Tenth Australian Computing Education Conference (ACE 2008) 79*.
Trobaugh, J. (2010). *Winning Design! LEGO Mindstorms NXT*. Apress
Turkle, S., Papert, S. (1991). Epistemological pluralism. In: Harel, I., Papert, S. (Eds), *Constructionism*. Norwood, NJ, Ablex, 116–126.
Verner, I.M., Hershko, E. (2003). School graduation project in robot design: a case study of team learning experiences and outcomes. *Journal of Teaching Education*, 14(2). 40–55.
Virnes, M., Sutinen, E., Kärnä-Lin, E. (2008). *How Children's Individual Needs Challenge the Design of Educational Robotics*. Paper presented at the 7th international conference on Interaction design and children.

F. Kaloti-Hallak is a PhD student in the Department of Science Teaching of the Weizmann Institute of Science. She holds master degrees in Computer Information Systems from Eastern Michigan University in 2000 and Science Teaching from the Weizmann Institute of Science in 2011. Her research interests include middle-school computer science education and human-computer interaction. She has been a lecturer at the Department of Computer Science and Information Technology of Al-Quds University.

M. Armoni is a senior scientist at the Department of Science Teaching, Weizmann Institute of Science. She received her PhD from the School of Education in Tel-Aviv University, and her B.A. and M.Sc. in computer science from the Technion, Israel Institute of Technology. She has been engaged in computer science education for more than 20 years as a lecturer and a teacher, as a curricular developer, and as a researcher. She has co-authored several textbooks for high schools and for junior high schools. Her research interests are in the teaching and learning processes of computer science, specifically of various computer science fundamental ideas.

M. Ben-Ari is a professor in the Department of Science Teaching of the Weizmann Institute of Science. He holds a PhD degree in mathematics and computer science from the Tel Aviv University. He is the author of several textbooks on elementary computer science, mathematical logic and concurrent programming. In 2004, he received the ACM/SIGCSE Award for Outstanding Contributions to Computer Science Education, and in 2009 he was designated an ACM Distinguished Educator.

Olympiads in Informatics, 2015, Vol. 9, 113–125
DOI: http://dx.doi.org/10.15388/ioi.2015.09

Different Approaches for Making the Initial Selection of Talented Students in Programming Competitions

Bojan KOSTADINOV, Mile JOVANOV, Emil STANKOV,
Marija MIHOVA, Biljana RISTESKA STOJKOSKA

Faculty of Computer Science and Engineering, University Ss. Cyril and Methodius
st. Rugjer Boshkovikj 16 Skopje, Macedonia
e-mail: bojan.kostadinov@gmail.com, {mile.jovanov, emil.stankov,
marija.mihova, biljana.stojkoska}@finki.ukim.mk

Abstract. Competitions in informatics are one of the most useful ways of engaging pupils to the field of computer science. Many national and international competitions are carried out each year, with the organizers of each of them attempting to reach out to as many potential students as possible. In this paper, we analyse several different contest types, how each of them aids to promote computer science by engaging students in competitions, and how they can help provide the initial selection of talented students for the later stages of the competitions. Based on that, we propose seven criteria for the classification of the contests. Further we present our rich experience in the organization of Macedonian contests. Finally, we propose different approaches for making the initial selection of talented students in programming competitions and discuss ways for alleviating some of the issues with standard contests, for example enabling feedback and introducing difficulty divisions.

Keywords. programming contests, types of contests, promoting informatics, identifying talented students.

1. Introduction

In this paper, we will show several approaches for attracting talented students interested in participating in informatics workshops or contests, and conducting an initial selection between them. Although the focus of this paper is mostly targeted at the organization of competitions in informatics, the presented ideas can also be used by educational organizations and governmental entities to identify talented students in the STEM (Science, Technology, Engineering and Math) fields. As the demand for skilled workers in science, technology and mathematics increases, linking the rise of these fields with global competitiveness, it's increasingly vital that governments and institutions find

ways to detect talented students early on in their primary and secondary education. Because identifying potentially gifted and talented students has never been an exact science (Goodhew, 2009), making it difficult to measure and analyse, we will present several different kinds of approaches to this problem, as well as the advantages and disadvantages of every one of them.

The paper is organized as follows. In Section 2 we will present several popular competitions that are currently organized throughout the world, and attempt to provide a general overview of possible contest types and the advantages and disadvantages of each of them. We present this through seven criteria that we identified, and which we propose for the purpose of classification of contests. In Section 3 we focus on the types of informatics competitions organized by the Computer Society of Macedonia, compare them with the international contests, and describe the biggest challenges that we are facing at the moment. We will also briefly describe several other contests that take place in our country. Section 4 builds up on the details presented in the previous sections, and compares how suited each contest type is to the problem of attracting pupils, and making an initial selection between them. In Section 5 we provide directions for future work and summarize our findings.

2. Overview of Contests

Competitions are a major factor in education (Verhoeff, 1997). A lot of countries use different forms of competitions in order to encourage students to perform better in school (for example, in order to earn scholarships or grants), to bring the best out of them (by enrolling students into special programs based on their interests), or to promote cooperation by grouping students into teams and teaching them (through competitions) the importance of collaboration between people with different talents and preferences.

Whatever the goal is, competitions must be fair in order to make them engaging to students (or people, in general). That is, students should have (pretty much) the same chances of winning – compared to other students/teams. In this paper we describe several competition types that satisfy these criteria – including competitions that are graded manually (by people), or automatically (by machines).

Besides fairness, a very important attribute of competitions is how well they accommodate to the different skills possessed by students. With regards to informatics competitions, they should be designed in such a way that the difficulty of (at least some of the) problems is approachable to most of the students. This can be achieved by dividing students into age divisions, knowledge groups, or by organizing a series of contests with increasing difficulty.

Before we attempt to define the criteria by which competitions can be reviewed, we will first provide an overview of several active informatics competitions. We will use this overview, as well as our knowledge about other contests, to describe strategies for dividing, analysing and improving competitions.

In the following list, we present some of the most popular, and some of the most interesting competitions organized in the world today. Although they have different grad-

ing criteria, deliverables or durations, each one of them motivates and attracts a large number of participants every year.

- **The International Olympiad in Informatics (IOI)** is an annual programming competition, reserved for secondary school students. Students compete by solving problems (of algorithmic nature) in one of several available programming languages (C, C++, Pascal and, starting from 2015 – Java). In the last few years, participants receive full feedback for their submissions, which wasn't the case during earlier Olympiads. The contest is usually organized on two competition days, and countries are limited to 4 competitors each.
- **ACM International Collegiate Programming Contest (ACM-ICPC)** is an annual programming competition involving university teams. The competition is initially organized in several regions, with regional winners attending the World Finals. During a contest, the teams are given several hours (usually 5) to solve multiple programming tasks using one of the available programming languages (C, C++, or Java). Teams receive feedback when they submit a wrong solution for a task, allowing them to make changes to their solution and submit a new one.
- **Google Code Jam** is an international programming competition, initially created as a means to identify top engineering talent for potential employment at Google. Participants are given several tasks that they need to solve, using any programming language and development environment. Currently, competitors receive an input file, for which they must generate a correct output in a limited amount of time. For most of the tasks, there are actually two input files – an easy input (for which feedback is provided), and a hard input (which is judged after the contest has finished).
- **Bebras** is a very popular international competition, whose goal is to promote informatics and computer science to students of various ages. The contests are made up of a set of short questions, which can be answered – according to the organizers – without prior knowledge about informatics, even though the tasks are clearly related to informatics concepts, and are designed to motivate students in computational thinking.
- **Infomatrix** is an International Informatics Project Competition organized to encourage young people to apply their creativity, imagination and passion to make a difference in the world through technology. The competition reached more than 60 countries and 460 schools in 2014. Students can compete in 5 categories: Programming, Computer Art, Hardware Control, Short movie and Robotics (Mini Sumo, Line Follower, Lego Sumo and Lego Line Follower). Some of the criteria by which projects are judged in the Programming, Computer Art, Hardware Control and Short movie area are: Originality, Quality of the project documentation and the content, Usability, as well as whether or not the project reflects the current interests in the specific area.
- **The Australian Informatics Competition (AIC)** is a pen-and-paper style competition, organized as an entry-level competition in informatics in Australia. Questions are designed to test algorithmic ability, logic and the students' ability to analyse algorithms. Since its introduction in 2005, the number of students that

take part in the competition has increased from 2000 students, to more than 7000 students in the last year (Clark and Clapper, 2014).

When defining the scope and rules for a competition, organizers can decide to give students either a predefined set of tasks (or questions) to complete, or they can give broad definitions of topics, and require students to showcase their own projects or work. For competitions where students work on a predefined set of programming tasks or questions, the grading is usually much easier and faster – in some cases, even automated using specialized grading systems. Participants either upload their solutions to an online system, or they write them down (or mark answers) using pen and paper. For competitions based on projects, a judging committee formed by the organizer usually does the grading manually.

Contests can be classified according to multiple criteria. There is no substantial previous work on this matter. After a careful review of the work in (Pohl, 2006), and a look at the rules of multiple competitions around the world, we propose a classification that we believe is useful with regards to currently popular competitions in informatics (both in our country and internationally).

Competitions in informatics can be roughly classified by these seven criteria:

- What participants need to work on (types of ***problem***s/tasks given to the contestants)?
- What do participants need to ***deliver***?
- How will participants provide their output/product (***submission method***)?
- What is the ***duration*** of the competition?
- What is the ***scoring*** system?
- How is the ***grading*** done?
- What sort of ***feedback*** do participants get during the competition?

Some of the more popular options for each criterion are presented in Table 1.

Compared to previous research work on the overview of competitions in informatics, we chose to give feedback a lot more emphasis. Feedback is one of the most important parts of a competition. For most programming competitions, the increasing trend is to provide participants with some sort of feedback regarding the correctness of their solutions. One of the bigger issues of the International Olympiad in Informatics

Table 1
Criteria for classifying competitions in informatics

Criterion	Possibilities
Problems	Project, Set of tasks, Set of questions, Mixed
Deliverables	Answers, Source Code, Executable, Working Robot
Submission method	Pen and paper, files on an USB drive, online grading system
Duration	Short competition, Long/marathon competition
Scoring	Equal points for each task (question), different points, scoring based on how quickly the task is solved
Grading	Manual, Semi-automatic, Automatic
Feedback	No feedback, Partial feedback, Full feedback, Pointing out mistakes, etc.

(and the organization of national competitions for selecting the top students for the IOI) has been that the automatic grading process may be unfair – since small mistakes may lead to solutions which score very few points (especially for tasks which are hard to test/debug during the limited competition time), and good competitors can end up with bad results. Providing feedback during the competition is one good solution to this problem, although one has to consider that the feedback may influence the goals and form of the contest.

Programming competitions can be easily classified using the criteria provided above. Thinking about the possible criteria and how they can be amended for a given competition can lead to better communication between organizers and participants, to fairer grading, and ultimately to the organization of better competitions.

There are hundreds of different informatics competitions organized each year (by a wide range of organizers), and each of them needs to properly inform participants about the terms of the competition, and what exactly is expected from them. As an example, the International Olympiad in Informatics is a competition where: (1.) students work on a predefined set of tasks created by the organizers, and are asked to (2.) deliver their source code, by means of (3.) uploading it to an online contest system, (4.) during a limited competition round of 5 hours, where (5.) each task is worth the same amount of points, and the provided solutions are (6.) graded automatically using a contest management system. Although results are published after the competition is done, (7.) students receive feedback for their own solutions while the competition is running. It is worth pointing out that before deciding on a competition type on the basis of the criteria presented above, organizers must first take into account the available resources, the competition's goal, and the expected number of participants.

3. Macedonian National Competitions in Informatics

Competitions in informatics are held in Macedonia since 1990, and, by the end of 2014, there were 25 national contest cycles – which include multiple competitions each year, selection contests for international competitions, as well as training camps. Every year the contestants go through many levels of competition so that the best could be selected. The selected pupils represent themselves and Macedonia at the (Junior) Balkan Olympiad in Informatics (BOI/JBOI), the International Olympiad in Informatics (IOI), and at other smaller regional competitions. The main organizer of the competitions in informatics for primary and secondary school pupils is the Computer Society of Macedonia.

The format of the competitions evolves each year, depending on many factors, such as the number of interested pupils, available resources, inclusion of programming in the school's curricula, etc. The interest for the competitions in informatics, presented by the number of participants at each competition level, is presented in Table 2. Presently, the competitions are organized for primary and secondary school pupils, and include: School Qualification Competition, Regional/Municipality Competition, National Competition, National Olympiad, and (potentially) Selection Contests for International Competitions.

Table 2
Number of students participating in the official competitions in informatics in Macedonia

	Year					
	2009	2010	2011	2012	2013	2014
Regional competition	51	55	118	209	290	341
National competition	44	45	68	95	118	101
National Olympiad	22	23	19	21	21	21

In order to support several competition types, enable a large number of students to participate in the competitions, and introduce as many pupils as possible to the art of programming, all of the competitions in informatics that are part of the national contest cycle (accredited by the Ministry of Education), are organized using the MENDO competition management system (Kostadinov *et al.*, 2010).

The usage of a grading system has a lot of advantages with regards to the organization of programming competitions, but it also has several disadvantages. Specifically, MENDO is an interactive e-learning system that was developed following the goal of integration of all previously used modules for organization of programming contests, in a single compact environment: supporting the uploading of the competition tasks (organizers) and the solutions (contestants), evaluation and automatic grading of the uploaded solutions, publishing results, communication, training, lectures intended for learning programming languages and student improvement, collaboration, and feedback. In order to provide automatic grading, students solve the tasks by writing a solution (usually a console program) in one of several available programming languages, which is later compiled, added to a grading queue, and eventually executed on several test cases by sending various input data, whilst time, memory, output and security limits are being enforced on top of it. The system's support for multiple contest types allows us to organize different competitions each year (Jovanov *et al*., 2013).

Tasks are a powerful way to test the user's knowledge, but they can also influence users in a negative way – for example, a student can be stuck on a certain task due to inexperience (printing data in a wrong format, etc.). In order to help users, MENDO offers several means of providing feedback (during training or competitions): solution-specific analysis to match problems with a predefined set of mistakes, the ability to download or examine test cases, and the option to view the author's solution to a task. This helps us to attract new participants without much involvement from their principals and tea-chers, and to keep them interested in practicing without worrying that they can get stuck on specific tasks. The number of sent and graded submissions by students, on the MENDO system, increases each year, as can be seen on Fig. 1.

The usage of a grading system and competitions based on solving programming tasks helps the Computer Society of Macedonia organize competitions in a similar environment to the one available at the International Olympiad in Informatics. In reality, as long as the international competitions in informatics are organized as they are, in order to select the best students for the international competitions, the national contests need to accept similar grading and organizational practices. However, organizing competi-

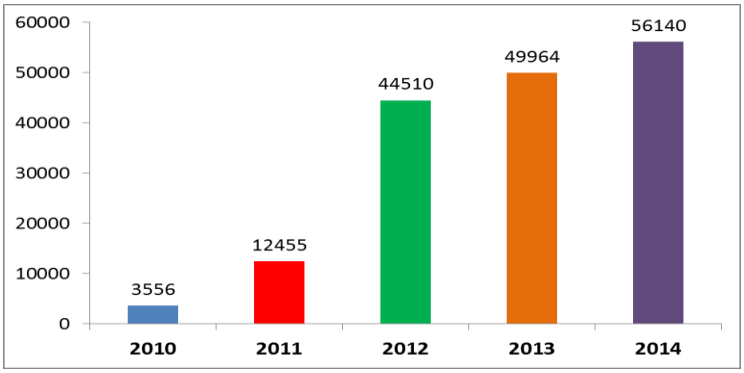


Fig. 1. Submitted solutions on the MENDO grading system.

tions which utilize an automated grading system is not always the perfect option, as it can have several downsides: it requires resources (ie. a computer for each student, servers, etc.), low insurance that students solve the tasks by themselves (for online competitions), difficulties with making sure students have the same environment on their computers as is used on the competition grading system and persuading students to participate in a competition that requires having a specific prior knowledge and abilities, to list the most common. The work presented in (Stankov *et al.,* 2013) addresses some of the above mentioned issues, and there we explore one alternative way of assessment of the contestants' solutions.

In the case of the national contest cycle organized by the Computer Society of Macedonia (and we believe similar problems exist at other countries as well), the biggest issues we face are attracting young students, and providing an initial selection between them. Currently, the problem of attracting students is confronted by visits to schools and organization of promotional events there. Additionally, the promotion is done through teachers which have been involved in earlier contest cycles. Most importantly, we try to keep the pupils online community active during the year using the MENDO platform, and we inform them about the upcoming events.

The first competition that students participate in is the School Contest, which is used by schools to select the best pupils, which they can later send to the Regional Contest. However, organizing a competition like this has the downside of requiring schools to make internal preparations for organizing such a competition (so some schools simply decide not to do it), as well as that each school has an assigned quota (number) of participants they can send to the Regional Contest (which may not be an accurate estimate of skills compared to other schools). Therefore, it is important for us to analyse other possible contest types (presented in the next section), the advantages and disadvantages of each of them, and to try and improve the current process that we use for selecting students early on in the competition cycle.

Besides the official contests for primary and secondary school organized by the Computer Society of Macedonia, there are other contests organized in our country. Some of the more popular competitions, which are based on solving algorithmic tasks, are

the CodeFu competitions (multiple contests each year), and the ACM-ICPC national competition. The CodeFu competitions are organized using specific rules (5 tasks, each worth different number of points), and the ACM-ICPC national competition is organized according to the ACM-ICPC rules. A large number of students who have programming abilities (both on high school and university level) also participate at other competitions – like Startup Weekend Skopje, where they work in teams with other students.

4. Approaches for Conducting an Initial Selection of Students

National competitions in informatics are important – both in terms of selecting the best students that will later represent the country at international competitions in informatics, as well as in terms of promoting informatics locally. Agreeing on a system for the national competitions in informatics is one of the first things to consider when starting a new competition cycle. In our experience, the most important thing that should be considered is how the initial selection of students should be done (which is the first real contact that participants have with the competitions). Different approaches have their own advantages and disadvantages. For example, allowing more students in the initial stages of the competitions poses a significant logistical problem – tasks, access to computers, greater number of people involved in the organization of the competitions, travel expenses, etc. These problems are more pronounced in diverse countries such as Macedonia – having multiple ethnic cultures, with students who want to participate using their own mother tongue. On the other hand, limiting the number of students (per school, or per municipality) means that lower number of students will benefit from the organization of the competitions.

In the following list, we present several options for conducting the initial selection of students. Later, we discuss possible strategies for alleviating some of the potential issues, and several approaches for making the competitions more fair and interesting.

- **Solving programming tasks (online)** – this approach is currently used by a significant number of countries in the IOI community. The biggest advantage of having an online competition organized in such a way is that there is (practically) no limit to the number of participants that can take part in the competition. Also, since one of the main goals of the organizations that are part of the IOI community is to select the best students for international competitions, having a competition in which students solve IOI-style programming problems is an optimal way to select the students which are good in solving such tasks. One example of this approach is USACO (USA Computing Olympiad), which invites students to an on-site camp (where the IOI team is selected) based on the results of web-based contests organized throughout the year, and the performance of students on their on-line training pages (Kolstad and Piele, 2007). The two biggest disadvantages in organizing these types of competitions are that students must have previous experience with solving these types of tasks, as well as the difficulty involved in making sure that students are graded solely on their performance (that they aren't cheating).

- **Solving programming tasks (onsite)** – this approach is currently used by many countries. Similarly to the previous option, students solve programming tasks on a computer, but this time – all of them are gathered at one (or multiple) onsite locations – in contrast with solving the tasks online (from home or school). Due to the serious logistical problems in organizing such competitions (organizing staff at every location, a computer for every participant, etc.), only a limited number of students can be accepted to participate at such competitions. On the other hand, the results at such competitions can be taken as official (and a very good estimate of the actual students' abilities). If regulations for issuing certificates imply that competitions must be organized onsite (and are imposed by governing institutions in the country), this approach may be the best alternative to the online competitions mentioned above. The problem of pre-selection of the students can be delegated to the teachers in Informatics in every school. The Macedonian model features a fixed number of places for competitors per school (between 2 and 5, determined every year at the beginning of the cycle), and additional number of places (determined by the number of high achieving students from that school in the previous year's cycle).
- **Solving tasks using pen and paper** – in many countries that organize different types of contests, these are by far the competitions that attract the highest number (and the most diverse group) of students – a good example is given in (Clark and Clapper, 2014). Although these competitions are very different from the IOI (thus, they cannot be used to select students for international competitions), the biggest advantages they have is that there is no need for hardware resources, but the competitions can include tasks and/or questions of diverse difficulty and types (from solving multiple-choice questions to writing and analysing programs). One of the biggest issues with the previous two options (solving programming tasks on a computer) is that less-experienced students can hardly solve any of the problems – because their solutions are checked using a computer, they usually receive a very low number of points. On the other hand, pen and paper rounds are usually checked by people (teachers), which end up posing logistical problems of their own. We believe that organizing a pen and paper competition is a very good choice if organizations would like to attract more students to computer science.

If organizations have enough resources to support multiple competition types, we believe that it is a very good idea to organize both onsite programming competitions where students work on IOI-style tasks, and contests (or events) that are specifically designed to promote informatics. Besides organizing a custom-made pen and paper round, there are other very good choices that are currently expanding to multiple countries. One such example is the Bebras competition – an initiative that promotes Computational Thinking among pupils. The contests can be organized during normal school hours - thus allowing a very large percentage of students to take part.

On the other hand, there are multiple approaches that can be taken to mitigate the issues with online and onsite programming contests. Clearly, with regards to making sure that students are ranked solely based on their knowledge and abilities during

online competitions, there are multiple software systems that can detect plagiarism, analyze students using a web camera, or identify suspicious activities (multiple participants using the same IP address, or one participant using multiple addresses), which can later be investigated manually.

Without sufficient hardware and human resources, onsite competitions are very hard to organize correctly. Even if the number of participants is limited, there are several approaches that can be used to enable organizers to make fair decisions. For example, hard limitations to the number of students that a school can enter into a competition, can be mitigated using results from other online contests (like TopCoder or Codeforces) – inviting students which have done good in other contests, or allowing schools to register more participants based on the results of last-years competitions, for example inviting students who have earned certificates at the national competitions, out of the normal quota.

In Table 3, we summarize approaches organizations may take on to improve the process of organizing a competition by which the initial selection of students is made.

Finally, it's worth mentioning that feedback during competitions can be used as mitigation to the problem of having users who score a low number of points due to lack of experience. Feedback is also very useful for making sure that students who make small mistakes do not end up with a too low number of points. However, having feedback requires additional resources. It needs better hardware, so the submissions can be

Table 3

Summary of the approaches organizations may take to improve the process of organizing a competition by which an initial selection of students can be made

Approach	Advantages	Disadvantages
Organizing competitions onsite	Fair results, can issue certificates.	Requires additional hardware and human resources.
Organizing pen and paper rounds	Attracts a large number of participants, promotes informatics.	Less programming involved, can't be solely used to select best competitors for international competitions.
Organizing parallel competitions (Bebras, etc)	Attracts a large number of participants, promotes informatics.	Can't be used to select competitors for international competitions, usually requires hardware and more involvement from teachers.
Tasks of varying difficulty	Tasks are approachable by more students.	More tasks need to be created and they must be ordered correctly in the final set of tasks because less experienced students have problems in determining their difficulty. This can be hard to do, as students can be good at different things.
Age categories, Difficulty divisions	More students can get involved; they can solve tasks that are designed for their level of knowledge and ability (i.e. beginner groups)	Requires additional human resources, due to the need for additional tasks and questions to be prepared and (potentially) judged.
Feedback during competitions	Results are more fair, as small mistakes don't lead to drastic changes in results (unless the student is not able to debug – identify or fix mistakes).	It must be ensured that test cases and solutions (i.e. the feedback) are correct and they adhere to the task description.

graded in real time. Also, it requires significant preparations before the contests, which demands a greater team of people who will work on tasks. Having an incorrect official solution, test cases, or grading system during such competitions that allow feedback, can lead into doubt of the fairness of the entire competition, due to the incorrect notifications given to students.

5. Conclusion

Competitions are a very important part of education. Given the rise of informatics and computer science in the world, multiple organizations are using competitions to promote informatics to a diverse group of young students. Today, there are many national and international competitions in informatics being organized each year – some of them having different rules and regulations. Some of the most popular international competitions in informatics are the International Olympiad in Informatics, ACM-ICPC, Google Code Jam and Bebras.

In this paper we identified several criteria for reviewing different competitions in informatics, depending on what participants need to work on, the duration of the competition and the type of grading done, the scoring system, what participants need to deliver, how they will provide their output or product, and the type of feedback given during the actual competition.

The most critical part of creating a national contest cycle is defining the parameters to which the initial selection of students will comply. Popular mechanisms used as an initial contact with students are online (or onsite) programming competitions and pen and paper rounds. In the paper, we have explored different approaches with their advantages and disadvantages, and we tried to give a summary that can be useful to organizations that will organize contests in the future, or to organizations that are planning to introduce changes in their current competition cycles.

Also, we have tackled the issue of increasing the number of participants in the competitions. Using strategies like: making stronger connections with teachers, organizing Bebras-style competitions, tracking (sustaining of an online community using apps or social media) and appropriately awarding users who introduce other people to the competitions in informatics, can be very helpful. Age divisions, feedback, valuing results at other competitions and organizing parallel events to promote informatics are just some of the things that organizations can use to further engage students to the art of computer science.

Acknowledgement

The research presented in this paper is partly supported by the Faculty of Computer Science and Engineering in Skopje.

References

ACM International Collegiate Programming Contest (ACM-ICPC) (2015). http://icpc.baylor.edu/

Bebras – International Contest on Informatics and Computer Fluency (2007–2015). http://bebras.org

Clark, D., Clapper, M. (2014). The Australian Informatics Competition (AIC). *Olympiads in Informatics*, 8, 179–189.

CodeFu Coding Competition (2015). http://codefu.mk

Computer Society of Macedonia. http://zim.mk

Goodhew, G. (2009). *Meeting the Needs of Gifted and Talented Students*. London, Continuum International Publishing Group.

Google Code Jam (2008–2015). https://code.google.com/codejam

Infomatrix (2015). http://www.infomatrix.ro/

International Olympiads in Informatics (IOI) (1989–2015). http://www.ioinformatics.org

Jovanov, M., Kostadinov, B., Stankov, E., Mihova, M., Gusev, M. (2013). State competitions in informatics and the supporting online learning and contest management system with collaboration and personalization features MENDO. *Olympiads in Informatics*, 7, 42–54.

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

Kostadinov, B., Jovanov, M., Stankov, E. (2010). A new design of a system for contest management and grading in informatics competitions. In: *ICT Innovations Conference 2010, Web Proceedings.* 87–96.

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

Stankov E., Jovanov M., Madevska Bogdanova A., Gusev M. (2013), A new model for semiautomatic student source code assessment. *CIT. Journal of Computing and Information Technology*, 21(3), 185–194.

Verhoeff, T. (1997). The role of competitions in education. In: *Future World: Educating for the 21st Century: a conference and exhibition at IOI 1997.*

B. Kostadinov is currently working as a software engineer. In 2014, he defended his MSc thesis in Intelligent information systems at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is one of the organizers of the Macedonian national competitions in informatics. He has participated at IOI as a contestant and also as a team leader for the Macedonian team.

M. Jovanov is an assistant professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. As the President of the Computer Society of Macedonia, he 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 at International Olympiads in Informatics since 2006. His research interests include development of new algorithms, future web, and e-education, and he has authored more than 40 research peer reviewed papers.

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 a Ph.D. student at the Faculty of Computer Science and Engineering. His research includes analysis of program code correctness using different techniques, and its application to e-learning.

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.

B. Risteska Stojkoska obtained her Ph.D. degree in 2013 with the thesis "Data fusion in wireless sensor networks." Currently she works as an Assistant Professor at the Faculty of Computer Science and Engineering (FCSE), University "Ss. Cyril and Methodius", Skopje, Macedonia. She is an author of more than 30 peer reviewed papers. Her research interests include positioning in wireless sensor networks; smart home; smart grid; wireless power transmission; and intelligent and embedded systems. She serves as TCP member of more than 10 international conferences.

Olympiads in Informatics, 2015, Vol. 9, 127–137
DOI: http://dx.doi.org/10.15388/ioi.2015.10

An Update on the Female Presence at the IOI

Stefano MAGGIOLO
London, U.K.
e-mail: s.maggiolo@gmail.com

Abstract. The gender ratio in computer science is already very unbalanced; we gathered statistics about the gender of contestants and leaders to show that the female participation at the IOI is even lower. We look at existing programs trying to increase gender balance at a national level and offer some suggestions, including (re)-introducing mixed gender requirements for complete teams.

Keywords: gender imbalance.

1. Introduction

There is no need for statistics to realize how much the IOI are unbalanced in terms of gender distribution: it is far too easy to notice the gender skew just by walking around during the IOI week, or by looking at the audience of the opening ceremony.

But we need statistics when we want to compare the IOI with the general trend in computer science and in math-intensive fields, and if we want to track progress. For example, the organization of the IMO is now recording the gender distribution of the contestants for recent editions, and many past editions have also been back-filled (International Mathematical Olympiad).

We asked the national delegations to provide this piece of historical data, and to share the programs they have in place to improve gender balance. This article is mainly about presenting the results of this survey (section 2), and comparing the IOI's gender ratio with those of other relevant institutions and companies (section 3).

At the international level there have not been many initiatives in this direction. The main one happened in 1995, when the Netherlands, as the organizing country of the IOI, allowed delegations of five contestants (instead of the usual four) when the team was gender mixed. Despite having a reasonable success, with four times as many female as the average edition, the program was not reproposed by other hosting countries, nor picked up by the International Committee.

In section 4 we present arguments for the IOI to adopt more actions to reach female contestants, in accordance with its goal of promoting computer science among young

people. Finally, in section 5, we offer suggestions for possible initiatives, both at the national and international level.

2. Data

We prepared a questionnaire to collect historical data regarding female participation at IOI, and we sent it to the contact person of each member country, as listed in the IOI's website. Given the low number of replies, the questionnaire was also extended to the two mailing lists ioi-announce and ioi-discuss.

2.1. *Format*

The questionnaire was composed of three parts, asking:

1. To describe the national programs having effects on the female participation at the IOI.
2. To estimate the total and female participation at three stages of the team selection: the "base" (all the participants), the "national olympiad", and the "training".
3. To mark the gender of each member of the official team (contestants and leaders) participating in the past IOI editions.

Names and pictures (where present) of the team members were obtained from the IOI statistics website (Kalinicenko). The same source was used to get all the participation numbers we present in the rest of the section.

2.2. *Participation*

Overall, 36 delegations responded (44% of the 81 countries participating in IOI 2014). We completed the gender assignment for the remaining 61 countries, for what we could infer from several, not necessarily correct, sources: gender-defining names, pictures in IOI-related websites, notes in the IOI newsletters, and generic web search. These approaches still left about 5% to 10% of the contestants marked as "not sure" for years up to 2001, and 0% to 5% since 2002, especially for countries that lack gender-defining names.

Given the generally low female participation, false negatives could change significantly the numerical results, even if a big change is not very likely. In any case, we believe that our conclusions are not impacted, especially given that the number of contestants of the last few editions for which we lack the assignment is very low. We encourage the IOI to collect the gender of the participants and make it available on the statistics website (Kalinicenko), to ease future investigations.

2.3. *Results*

Per year results. Table 1 contains the gender data aggregated by year, and Fig. 1 shows the percentage[1] of female participants, as contestants and leaders separately.

It is clear from the figure that there is not an obvious trend in the female participation. For leaders, if any, there is a slightly decreasing trend, but the data for years before 2000 is far from complete. For contestants, the period between years 1998 and 2006 is quite striking, having 2% or less female contestants for 9 consecutive years.

Table 1

Per-year participation at IOI, broke down by gender and role (leader or contestant). The columns ``F'' and ``M'' indicate the number of female and male participants, respectively; the column ``?'' the number of participants for which we were unable to assign a gender; the column ``F %'' the percentage of female participants among those we were able to assign a gender to.

Year	Leaders					Contestants				
	Tot	F	M	?	F %	Tot	F	M	?	F %
1989	14	0	6	8	0.0	37	1	21	15	4.5
1990	2	1	1	0	50.0	38	2	32	4	5.9
1991	17	2	15	0	11.8	68	1	63	4	1.6
1992	79	10	65	4	13.3	171	7	151	13	4.4
1993	29	2	27	0	6.9	107	5	94	8	5.1
1994	88	7	78	3	8.2	189	8	165	16	4.6
1995	30	2	28	0	6.7	210	19	172	19	9.9
1996	106	8	95	3	7.8	218	3	200	15	1.5
1997	95	10	81	4	11.0	221	7	199	15	3.4
1998	51	5	46	0	9.8	241	2	224	15	0.9
1999	54	6	48	0	11.1	172	2	158	12	1.3
2000	137	18	116	3	13.4	268	4	251	13	1.6
2001	75	8	67	0	10.7	261	5	241	15	2.0
2002	55	5	50	0	9.1	272	4	256	12	1.5
2003	50	5	45	0	10.0	261	3	247	11	1.2
2004	148	12	134	2	8.2	298	4	286	8	1.4
2005	138	10	127	1	7.3	281	5	274	2	1.8
2006	142	10	130	2	7.1	289	5	273	11	1.8
2007	140	10	128	2	7.2	280	11	269	0	3.9
2008	143	14	126	3	10.0	284	9	267	8	3.3
2009	156	18	138	0	11.5	302	14	288	0	4.6
2010	156	11	144	1	7.1	306	10	291	5	3.3
2011	150	12	138	0	8.0	303	4	299	0	1.3
2012	162	11	150	1	6.8	316	3	313	0	0.9
2013	150	12	138	0	8.0	299	6	293	0	2.0
2014	161	10	150	1	6.3	315	10	305	0	3.2
Overall	**2528**	**219**	**2271**	**38**	**8.8**	**6007**	**154**	**5632**	**221**	**2.6**

[1] All percentages are over the number of participants for which we were able to determine gender; in other words, we ignore participants with no gender assignment.

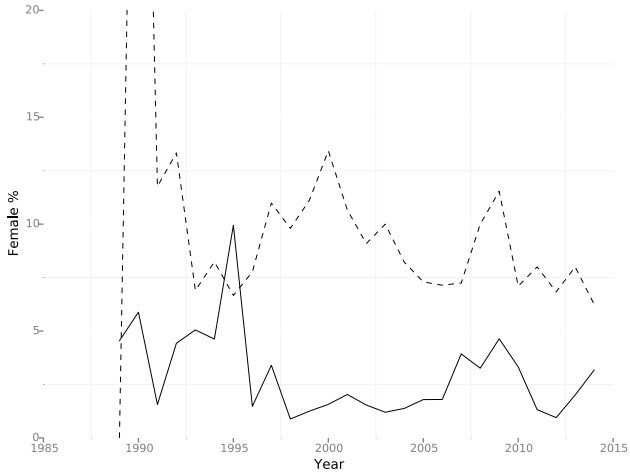


Fig. 1. Percentages of female participation by year, for contestants (solid line) and leaders (dashed line).

On the other hand, it is very visible the peak in 1995 for female contestants, caused by the rule allowing delegations with 5 contestants if the team was gender mixed. It is interesting to notice that this rule was in reaction to the low female participation in the previous years, but the situation before 1995 was much better than after (the average female participation was 4.4% in 1989–1994 and 2.2% in 1996–2014).

Per country results. Fig. 2 is a histogram of the number of countries by percentage of female contestants[2], after filtering for countries with at least 40 contestants. Among the 69 remaining countries, 22 have never had a female contestant, and only 6 countries have had more than 5% of female contestants. The country clearly differentiating, bringing to the IOI three times as many female contestants as the second ranked, are the Netherlands, those delegations are composed of females for the 22%.

Performances. One of the reason that the experiment in 1995 was not repeated was that countries had difficulties in recruiting female contestants able to compete with their

Table 2

Percentages of contestants obtaining a medal, by gender

Gender	No medal (%)	Bronze (%)	Silver (%)	Gold (%)
Female	77.9	9.1	11.0	1.9
Male	49.2	25.1	17.0	8.6

[2] Here we ignore the fact that contestants and leaders can participate in multiple years, and we treat these cases as if they were different participants. Therefore, to simplify the language, we may write "a contestant" instead of "a participation of a contestant in a certain year".

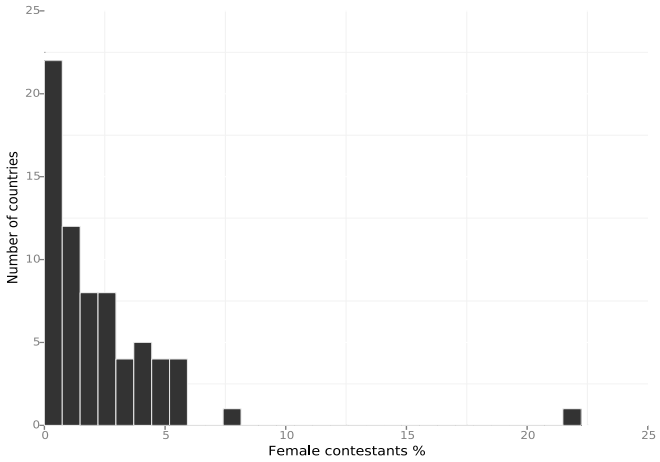


Fig. 2. Histogram of number of countries by percentages of female contestants. Only countries with at least 40 contestants are shown, which reduces the total number of countries to 69.

male counterparts. We cannot offer statistics on the distribution of scores, because for most editions the ranking is available only for contestants with a medal. Table 2 instead shows the percentage of female and male contestants that obtained a medal. Indeed, we can see that the percentages of female contestants not receiving an award is much higher than for males. An interesting phenomenon, for which we do not have an explanation apart from the small sample, is that female contestants received more silver medals than bronze medals.

At the national level. We asked in the questionnaire about the female presence at three stages of the national selection in the last year: the "base" (at the first selection), the "national olympiad" (at the main national competition), and the "training" (when pupils receive lectures to improve their chances at the IOI).

Due to the wildly varying ways in which national selections works, it is difficult to unify the 30 responses received to this question in a single outlook. Nonetheless, some observations can still be made.

- *Female participation declines as the selection progresses.* This is quite expected, as the competition's base is in high school, where the gender ratio is more homogeneous than at universities and in IT companies. The last selection stage, training, shows an average female participation only slightly higher than at the IOI. At the first stage instead, many countries have double-digit female participation, and some reach parity.
- *Female participation at the training level is highly correlated with programs trying to increase it.* We also asked delegations to describe their actions to equalize the gender ratio, and the presence of such programs was correlated with a double-digit presence of females at the training level. Again, this is expected, but it is nice to see that these programs have results.

Programs to improve the gender ratio. Ten countries described programs geared towards, or having the effect of improving the ratio of female contestants. The following are the main ideas in these programs.

1. *Extra training and competitions.* Three countries have or had training camps and competitions aimed specifically at attracting females. This can be very helpful for promising students that might not have the occasion to try programming and computer science at all.
2. *Increase of the selection pool.* Two countries collaborated with math competitions for high school students to attract more talent. The gender ratio in these competitions is usually more balanced, and many of these students (regardless of gender) have potential and discover being interested in computer science despite not having any background in programming.
3. *Use of non-conventional task types.* The selection process can emphasize experience (that male participants are more likely to have). This can be avoided by using different type of tasks, for example mathematical, geometric, or graphical, especially at the initial levels.

3. Comparison

We showed that female participation at the IOI is low in absolute number: on average, 2.6% of the contestants and 8.8% of the leaders are female. Still, this could be common in the field and not specific to the IOI. The situation of women in IT and computer science, both in education and in the workforce, is indeed far from perfect, but the reality is that it is difficult to find environments as unbalanced as the IOI.

Education. In 1980, Bachelor Degrees[3] in Computer Science awarded to women were about 35% of the total (Camp, 2001). This number steadily declined in the following years; in 2011, they were about 12% of the total (The Computing Research Association). Masters Degrees and Ph.D. in Computer Science have higher percentages: again in 2011, respectively 25% and 18% were awarded to women (same source). If we include Information Sciences, percentages increase to 18% for Bachelor Degrees, 27% for Master Degrees, and 19% for Ph.D. (National Center for Education Statistics, 2014). For a comparison, Bachelor Degrees in Mathematics awarded to women have been quite stable in the past 30 years oscillating between 40% and 47%.

One might object that the IOI are an elite competition, and that might explain the difference with the percentages of Degrees awarded, but similar numbers hold for elite universities, like Stanford (Smythe, 2012).

The female presence at the International Mathematical Olympiad is steadily increasing since the 1970s, and is now just below 10% (International Mathematical Olympiad).

[3] For availability reasons, the numbers mentioned in this section will refer to the United States. Without implying that the United States are representative for the whole world, we can agree they are a leading and influential country for both computer science education and IT companies.

Workforce. In academia, in 2012, women comprised 25% of all computer science assistant professors, 18% of associate professors, and 13% of full professors, and these numbers are increasing (National Center for Women and Information Technology, 2014).

Even looking at the recipients of the most prestigious awards, the female presence is higher than at the IOI: as of 2014, 3 out of 62 winners of the Turing Award were women, for a percentage of 4.8%.

In the private sector, percentages are slightly higher, at around 30% for "computer and mathematics" professions. Some top IT companies (Double Union) recently divulged diversity data for their tech employees: the figures are lower, ranging between 15% and 20%, but again much higher than the IOI's.

4. Why Should We Care?

Female participation at the IOI is much lower than in post-secondary education and in the workforce, even in top institutions and companies. This comparison should already give a warning sign, but there are also other, more significant, reasons to improve gender balance.

Promoting computer science. One of the goal of the IOI is to promote computer science among young people. Since the first editions, the number of students involved in the IOI increased exceptionally, thanks to many countries joining the competition, and to countries already participating that managed to reach more and more students.

We would claim that many countries are now hitting a wall, already reaching most students that *would like* to participate in the IOI. The obvious next step is to encourage students that *do not know* that they would like to participate; and given its current low participation rate, the group that has the most headroom for growth is that of female students.

Increasing the performance of the team. A common argument is that regardless of the effort, eventually one will need to face that girls show less interest in computer science and programming than boys.

This might be true, but not a reason not to do anything about it. For a start, the situation has not always been like this, it is enough to look at the numbers of Bachelor Degrees awarded to women in the 1980s, or at the fact that many women have been pioneers in the field of computer science (The Ada project).

Also, the correlation between national level programs and female presence at the training camps shows that it is possible to attract female contestants with the necessary skills to be part of the national team at the IOI. Indeed, one good selfish reason for establishing these programs is that increasing the pool of candidates can also increase the average level of the team, and therefore of the IOI.

Numerous researches show that gender mixed teams work better (Simard, 2007). This is admittedly a stretch for the IOI, as the competition is individual; nonetheless, training camps are social events and, in our experience, the learning approach is very

collaborative, with a lot of horizontal knowledge distribution. Incidentally, increasing horizontal collaboration is among the steps taken by universities that have been successful in recruiting more women, like for example by implementing pair programming (National Center for Women and Information Technology, 2014).

Fairness. It is established that most people have implicit stereotypes that influence their decisions even if these prejudgments do not arise at a conscious level. For women in computing, a research showed that 70% to 80% of the subjects, regardless of gender, have implicit stereotypes associating science and tech with males more than females (Nelson, 2014).

Having unconscious biases is difficult to avoid, therefore it is important to be aware of having them and to limit their influence on our conscious decisions.

Another factor limiting fairness is stereotype threat, that is, situations that make people feel to be at risk of confirming the negative stereotypes about their group. A very relevant example is when a group is a strong minority and does not reach a critical mass.

Experiments show that performance is lower when stereotype threat is present, and higher when it is counterbalanced by a credible narrative of the path to success, by reminding of positive stereotypes, and the possibility of self-improvement, by highlighting positive examples, and role models (The Ada project).

5. What Can We Do?

5.1. *Principles*

Based on the previous section, we can define some guiding principles for actions that can have a positive impact on gender balance. From these we will derive the suggestions proposed for the IOI organization at national and international levels.

- *Fight bias (unconscious and not).* The IOI are driven by automatic scoring, that is not biased by definition. At the early stages though, there are many occasions in which bias might play a role: contests with subjective judging, decisions on which students to focus on, or which students have the most potential, etc.
- *Nurture potential not already expressed.* It is true that female students, like other socioeconomic groups, are in general less involved in programming courses, and are less likely to try programming on their own. Nonetheless, with the adequate motivation and training, hidden potential can develop, creating contestants able to compete for a place in the team.
- *Provide a welcoming environment for everybody.* Especially at the training level, teachers should take an active role in creating a welcoming environment. This does not only mean being inclusive, but also opposing stereotypes, providing positive examples, and creating a critical mass to fight isolation.

5.2. *National Level*

Here we suggest some concrete steps that can be applied to IOI selection and training at a national level.

- *Take an unconscious bias test.* It is important to be aware of unconscious biases one might have, and take the necessary countermeasures. An implicit bias test is a good starting point; for example Project Implicit (Greenwald et al.) helps highlighting conscious or unconscious biases on, for example, gender and science inclination.
- *Attract talented students without programming knowledge.* Unconventional programming tasks, or mathheavy problems, might help identifying talented students that did not have the opportunity or motivation to practice programming.
- *Collaborate with math competitions and other similar activities.* Many students passionate and talented in mathematics simply have never had the opportunity or the motivation, to start practicing programming and studying computer science. This is true regardless of the gender, but math competitions tend to have a more balanced gender ratio.
- *Offer entry level training for younger students.* This goes hand in hand with the previous points: when the goal is to attract students without an explicit knowledge in programming, it is a necessity to give them the opportunity to learn.
- *Showcase gender diversity.* If not already present, invite female teachers to training camps, and create opportunities to talk about important women in the history of computer science.

5.3. *International Level*

The most significant initiative the IOI has taken was the temporary rule allowing delegations of five contestants if gender mixed[4]. A rule in this spirit (either by forcing gender mixed teams, or by "gifting" them with one additional contestant) was promoted by the delegation of the Netherlands since 1992; despite having support from most members of the International Committee, a consensus was not reached, and the proposal did not pass.

In 1995 the Netherlands hosted IOI and obtained funding for a fifth contestant, therefore the rule was implemented "for free" in that year. The participation to this program was positive: among the 44 delegations with at least 4 contestants, 20 (45%) had a team of 5.

On the other hand, some delegations were concerned by the difficulty in recruiting competitive female contestants. Moreover, there were loud concerns about the consequences of such a rule. In particular, that it could lead to the perception that females are

[4] All the information regarding this program are extracted from reports and regulations available at the IOI website, and from personal communications with Ries Kock.

less qualified than males, and to more segregation at a national level (like having separate selections for males and females). Because of this, and most probably also due to a lack of funding, the rule was not replicated in any of the following editions, remaining a single episode in the history of the IOI.

We believe that it is time to reintroduce the requirement for complete teams to be gender mixed.

On a practical level, the IOI in 1995 was very young, and it is safe to assume that the pool of students reached by the national selections was much smaller, and that high schools were much less likely to provide programming (or even computer literacy) classes. This, together with the fact that the rule was announced just one year earlier, probably made difficult for national delegations to find competitive female students.

We believe that now, with larger bases for the national selections, established training programs, and more widespread opportunities to learn to program, it would not be as hard as it was in 1995 to find suitable female students, especially if given two or three years to prepare.

The fundamental idea behind this proposal is not that we want the IOI to have a stronger female presence, and therefore we impose it with artificial rules. On the contrary, we want a higher, self-sustained, female presence because it is fair, and it is the easiest way to expand the reach of the IOI. But we feel that the process towards this goal requires a bootstrapping phase, in which countries needs to be encouraged to devise programs like those outlined before. Our hope is that this rule will lose its reason to exist in just a few years.

6. Conclusions

With the help of the delegations that answered our questionnaire, we showed that the female participation at the IOI is very low, even comparing it with relevant academic institutions and workplaces; but we also found evidence that programs to improve female participation actually make a difference in the number of female contestants reaching the training phase of the national selections.

We argued that increasing the participation of female students is the most natural way of continuing to fulfill the goal of the IOI of promote the discipline of informatics among young people, and eventually to improve the performances of the teams by increasing the pool of candidates. We also presented evidences that the performances of female contestants can be improved through training programs, and by removing stereotype threat (for example, creating a critical mass).

Finally, we suggested to reintroduce the requirement of mixed gender for complete teams, as a way of encouraging countries to put in place programs promoting gender balance, arguing that most countries have now a more mature selection process, able to cope with this requirement, especially if given enough time to prepare.

Acknowledgments.

We would like to thank Ries Kock for kindly providing some background on the 1995 edition hosted by the Netherlands; Nandana Dutt and Selen Basol for useful discussions and for suggesting bibliography sources; Flavia Poma for reading the draft and suggesting several improvements; and all the people answering to the questionnaire, many of whom also communicated their support to this initiative.

References

Camp, T. (2001). Women in computer science: reversing the trend. *Colorado School of Mines*.
http://www-2.cs.cmu.edu/~women/resources/aroundTheWeb/hostedPapers/Syllabus-Camp.pdf
Double Union . Open diversity data. http://opendiversitydata.org/
Greenwald, T., Banaji, M. and Nosek, B. *Project implicit*.
https://implicit.harvard.edu/implicit/takeatest.html
International Mathematical Olympiad. Timeline.
https://www.imo-official.org/organizers.asp
Kalinicenko, E. *International Olympiad in Informatics - Statistics*.
http://stats.ioinformatics.org
National Center for Education Statistics. (2014). Digest of Education Statistics.
http://nces.ed.gov/programs/digest/2014menu_tables.asp
National Center for Women and Information Technology. (2014). NCWIT scorecard.
http://www.ncwit.org/sites/default/files/resources/ncwitscorecard_08122014_lowres.pdf
Nelson, B. (2014). The data on diversity. *Communications of the ACM*.
http://cacm.acm.org/magazines/2014/11/179827-the-data-on-diversity/fulltext#R5
Simard, C. (2007).Barriers to the advancement of technical women. *Anita Borg Institute for Women and Technology*.
Smythe, S. (2012). Stanford CS Department strives for gender parity. *The Stanford Daily*.
http://www.stanforddaily.com/2012/10/19/stanford-cs-department-strives-for-gender-parity/
The Ada project. Pioneering women in computing technology.
http://www.women.cs.cmu.edu/ada/Resources/Women/
The Computing Research Association. Computing degree and enrollment trends.
http://www.cra.org/uploads/documents/resources/taulbee/CS_Degree_and_Enrollment_Trends_2010-11.pdf

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. From 2006 to 2013 he collaborated 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 and IOI 2014.

Olympiads in Informatics, 2015, Vol. 9, 139–145
DOI: http://dx.doi.org/10.15388/ioi.2015.11

The Estimation of Winners' Number of the Olympiads' Final Stage

Aleksandr MAIATIN, Pavel MAVRIN, Vladimir PARFENOV, Oksana PAVLOVA, Dmitrii ZUBOK

ITMO University, Saint-Petersburg, Russia
e-mails: mayatin@mail.ifmo.ru; pavel.mavrin@gmail.com; parfenov@mail.ifmo.ru; pavlova.ifmo@gmail.com; zubok@mail.ifmo.ru.

Abstract. It is a complex and actual task to determine promising candidates for Olympiad's final stage from the participants' number of remote qualifying stage on the basis of their points scored during the qualifying stage. In this paper estimations of winners and awardees' number of the Olympiad final stage are made depending on the value of the passing score of the Olympiad's final stage. As part of the mathematical approach of mass Olympiad participants' results evaluation, data on indices of problems' solvability and participants' distribution in the qualifying stage according to the type of solved problems and scored points are used. The proposed approach can be applied by methodical committee and the jury of massive Olympiad in the process of the contest problems' development and determination of the passing score into the Olympiads' final stage.

Keywords: mathematical statistics, Olympiads, the criteria for results' evaluation.

1. Introduction.

The popular form of organizing Olympiads in Informatics and programming around the world consists of two stages – remote qualifying stage and the final intramural. The features of the remote stage are a great number of participants and a similar set of problems. The originality and novelty of problems are mostly inherent to the final stage. After the qualifying stage the Organizing Committee of the Olympiad decides whom to admit to the final. This decision is often based on a simple ranking of participants' scores and methodically founded principle of identifying promising or prospective participants. One can consider the time for solution of the problem or the number of attempts to solve the problem in the case of Olympiads in programming.

Methodists have a few problems. Firstly they have to decide what types of problems to devise, the number of problems and how to determine the estimation of their difficulty using points. The crucial rule here is the tradition of Olympiads. The second problem – how to determine the number of points p_{th} for qualifying participants into the final

stage. And the third one – how to determine the winners of the finals. The last two issues are resolved with a help of expert estimation.

Let's proceed from the fact that the aim of the Olympiad or one of the stages of the competition is to identify as many talented teenagers as possible. But the more the finalists are, the more resources are involved into the Olympiad conducting process. It is necessary to find a compromise. So we must be able to build more or less reliable forecast of the winners' number depending on the participants' number.

One can apply optimization techniques or methods of game theory, but we use the method of extrapolation and linear regression.

One of the important steps that must be carried out is to determine the estimation of the complexity of the problem using points. One can consider a few approaches to the determination of the numerical weights of typical problems of qualifying stage: expert estimation (Option "A"), which is a priori and estimation by solvability index of respective problems of the qualifying stage (Option "B"), which is a posteriori estimation.

Nowadays it is an actual issue to determine the passing score "p" in the final stage of the Olympiad, as among those who are not admitted to the final stage could be the ones who would be able to cope with the proposed problems and could become winners and awardees. To solve this problem it is required to estimate the probability of problems solving by the participants of the final stage, and, accordingly, it is required to estimate the number of winners and awardees of the final stage depending on their results in the qualifying stage and the expected set and level of problems complexity of the final stage.

2. Method of Estimating the Passing Score

First of all each problem of the final stage should match the group of the problems of qualifying stage according to theme and difficulty level. For these groups of problems of qualifying stage one can calculate average index of solvability according to the complete data. Then let us make a table of correspondence between the indices solvability of problems of the final stage and the average solvability indices corresponding groups of qualifying stage problems.

To estimate the number of winners and awardees of the final stage of the Olympiads depending on the passing score at $p < p_{th}$, method of extrapolation is used (Krug *et al.*, 1977).

Let K be the maximum number of qualifying stage points, N is the maximum number of the final stage points. Let's consider the random variables X and $Y^{(n)}$, $0 \le n \le \mathrm{N}$. The value x_k of random variable X is the number of participants with k points after the qualifying stage and n points in the final stage. Thus, the value of $(y_k^{(n)}, x_k)$ of two-dimensional random variable $(Y^{(n)}, X)$ is considered. To estimate the number of winners and awardees of the final stage, depending on the passing score when $p < p_{th}$ let us consider a linear regression of the random variable $Y^{(n)}$ to X (Krug *et al.*, 1977; Elfving, 1952). Let us use a linear regression equation in the form (Cramer, 1975)

$$y = m_Y + \frac{\rho_{XY}\sigma_Y}{\sigma_X}\left(x - m_X\right), \tag{1}$$

where m_X, m_Y – sample average values of random variables X и Y, σ_X, σ_Y – sample standard deviations of random variables X and Y, ρ_{XY} – sample correlation coefficient of the random variables X and Y. Equation (1) gives an estimate of the values of a random variable, determined best by the theoretical regression equation in terms of the principle of the least square (Cramer, 1975; Ayvazian *et al.*, 1983).

Like the formula (1), the expressions for the partial empirical equations of linear regression for estimation of the values $Y^{(n)}$ have the form:

$$y_k^{(n)} = m_{Y^{(n)}} + \frac{\rho_{XY}^{(n)}\sigma_{Y^{(n)}}}{\sigma_X}\left(x_k - m_X\right), \tag{2}$$

where m_X, $m_{Y^{(n)}}$ – sample average values of random variables X and Y, $\sigma_X, \sigma_{Y^{(n)}}$ – sample standard deviations of random variables X and Y, where $\rho_{XY}^{(n)}$ – sample partial correlation coefficient of the random variables X and $Y^{(n)}$ with fixed n.

Further, based on the calculated value of the random variables $Y^{(n)}$, let us construct a family $\{\Psi_p\}$ of distributions of the participants' number according to scored points in the final stage, depending on the passing score p. For each of the distributions Ψ_p random variable $Y^{(n)}$ values are calculated λ_p, μ_p. Here p is the passing score into the final stage, λ_p is the number of stage winners, μ_p is the number of stage winners and awardees.

3. Applying of Passing Score Estimation for Determining the Number of the Olympiads' Final Stage Winners

Every year in the series of Olympiads in Informatics, held by ITMO University, about 4,000 students from 11th form of all federal regions of the Russian Federation participate in them. Olympiads are divided into two stages: the qualifying (remote) and final (intramural). Qualifying stage is divided into three rounds and lasts about four months. During the qualifying stage the participants are offered problems of all relevant school curriculum of computer science. To take part in the final stage those participants are admitted who received the passing score, which is established by Olympiad's jury.

Methodical Olympiad committee develop problems for qualifying and final stages, according to the criteria of determining the winners and awardees approved by the organizing committee. Thus, in accordance with the criteria, winner is the participant who decides both creative problems of programming technologies with proper solutions for at least 9 out of 10 problems on general issues of computer science & ICT at reproduction and usage levels. Awardee is the participant who solves 9 out of 10 problems on general issues of computer science & ICT, and neither managed nor solved creative problems. Thus, it can be stated that Olympiad's awardee knows perfectly well Com-

puter Science & ICT at the levels of reproduction and use, but has not yet reached the level of creative mastery of the subject. Methodical committee and Olympiad's jury determine points for the final stage' problems, taking into account the above mentioned criteria. In determining the winners and awardees of the Olympiad's stage it is necessary to consider the following additional requirements: the winners' and awardees' number should not exceed 45% and the winners' number should not exceed 10% of the participants' number of that stage.

In a series of 2009–2010 academic year Olympiads, participants admitted to the final stage were those who scored 23 or more points out of 90 possible. Thus, the passing score is $p_{th} = 23$. The participants' number of the final stage is 931 students of 11th form. Virtually, less than a third of participants were admitted to participate in the final stage. Fig. 1 shows the distribution of participants according to scored points following the results of the qualifying stage among 11th form students in the series of 2009–2010 academic year Olympiads in Informatics.

Each problem of the final stage corresponds to a set of qualifying stage problems according to the theme and level of complexity. The average solvability index is estimated for this set of qualifying stage problems. The solvability indices of the final stage problems are compared with the average solvability indices of the corresponding sets of qualifying stage problems. The results are shown in Table 1.

The analysis shows that there is no direct correlation between the solvability index of the qualifying stage problems and solvability index of the corresponding types of the final stage problems. To estimate the number of final stage winners and prize-winners, depending on the value of the passing score when $p < 23$, it is necessary to determine the principles of assigning points for the final stage problems. Several approaches are

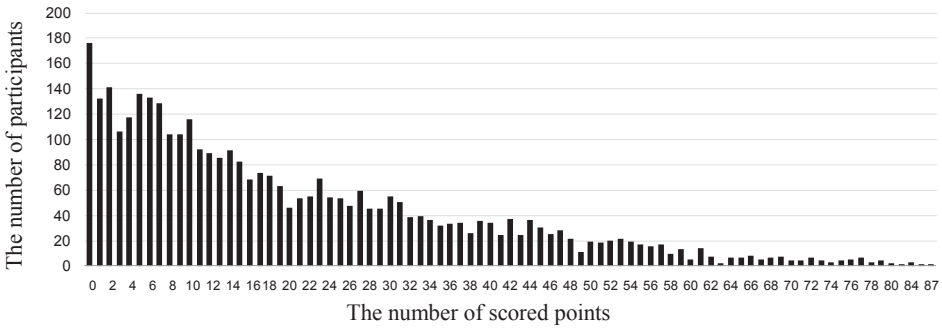


Fig. 1. Histogram of qualifying stage participants' distribution according to scored points.

Table 1

Problems' solvability indices according to Olympiad's stages

Problem №	1	2	3	4	5	6	7	8	9	10	11	12
Final stage	45,86	43,39	35,23	44,68	44,36	68,85	29,43	41,03	40,28	53,38	26,42	27,71
Qualifying stage	29,43	39,42	33,30	33,51	8,92	40,49	29,32	34,59	32,01	46,62	15,79	17,08

considered: expert evaluation (option "A"), evaluation according to solvability index of corresponding problems of the qualifying stage (Option "B").

Using the data of Table 2 and the actual results of the participants, participants' distribution of the final stage is made according to scored points for options "A" and "B" points' distribution according to problems (Fig. 2 and Fig. 3).

Besides, the error estimation of the options «A» and «B» in terms of principle ranking participants according to scored points, rather than types of solved problems. The

Table 2

Scores distribution according to problems

Problem №	1	2	3	4	5	6	7	8	9	10	11	12
	Number of points											
Option "A"	1	2	1	1	2	2	2	1	2	1	3	3
Option "B"	5	3	4	4	10	2	5	4	4	1	8	8
Option "C"	5	6	7	6	7	1	9	6	7	4	10	10

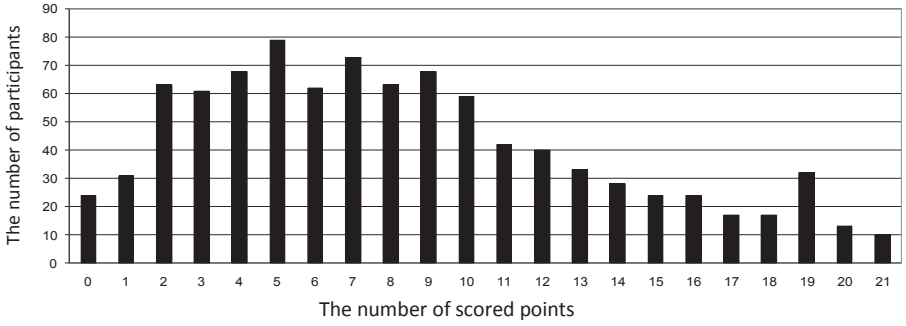


Fig. 2. Histogram of the final stage participants' distribution according to the number of scored points. The participants' number – 931. Option "A" of distribution points according to the problems (see: Table 2).

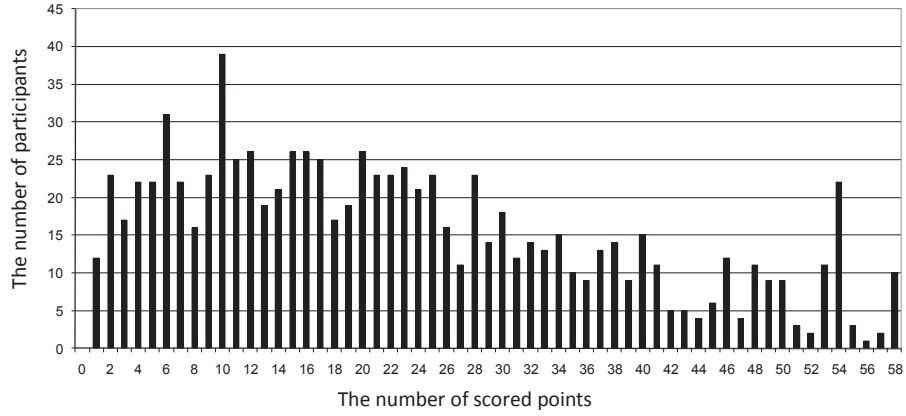


Fig. 3. Histogram of the final stage participants' distribution according to the number of scored points. The participants' number – 931. Option "B" of distribution points according to the problems (see: Table 2).

error/ inaccuracy occurs when the winner or the awardee is determined by the number of scored points, rather than composition of solved problems and means that the participant actually decided the set of problems meeting the winner criteria, but because of scored points attributed to the awardees, and vice versa.

"B" is preferred option of the two options «A» and «B», because it gives a smaller amount of error in comparison with the option «A» (see: Table 3).

The calculations of the values of the random variable allow with regard for the results of the qualifying stage – problems' solvability index and participants' distribution according to scored points – to estimate when $p < 23$ the number of the final stage winners

Data in Fig. 4 show the steady increase in the number of the final stage winners with a decrease in the values of a passing score.

The histogram in Fig. 5 shows the calculated values of the final stage winners' percentage depending on the passing score value obtained with the help of the distribution of

Table 3

The error/inaccuracy of the method of identifying winners and awardees' groups composition

	Error: winner-awardee	Overall error
Option "A"	89 participants	144 participants
Option "B"	35 participants	52 participants

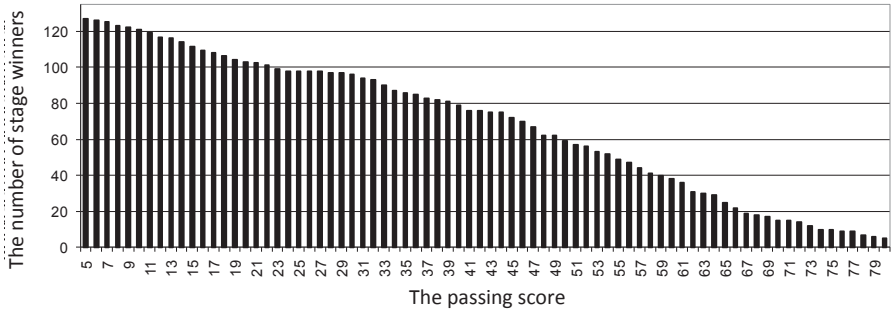


Fig. 4. The calculated number of winners depending on the value of the passing score.

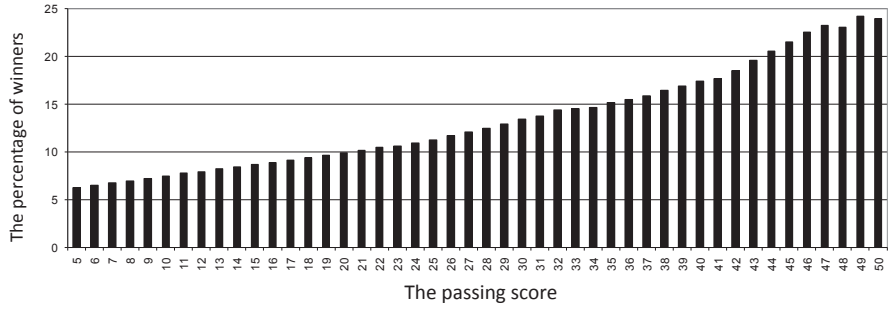


Fig. 5. Percentage of stage winners.

qualifying round participants according to points (see: Fig. 1), problems 'solvability indices (see: Table 1) and option "B" points distribution according to problems (see: Table 3).

4. Conclusion

In this paper a method of estimating the amount of the final stage winners of the Olympiad is proposed, depending on the value of passing score p in the final stage of the Olympiad.

It is obvious that there is a correlation between distribution of final stage participants according to scored points and distribution of qualifying stage participants according to scored points and an additional parameter – passing score in the final stage.

The results can be used to create methods for the development and estimation of problems' complexity of mass competition and Olympiad's finals.

References

Ayvazian S.A., Enyukov I.S., Meshalkin L.D. (1983). *Applied Statistics. Fundamentals of Modeling and Primary Data Processing*. Moscow, Finance and Statistics.

Bespalko V.P. (2002). *Education and Training with Computers* (*Pedagogy of the Third Millennium*). Moscow, Publishing House of the Moscow Psychological and Social Institute.

Cramer G. (1975). *Mathematical Methods of Statistics*. Moscow, Mir.

Elfving G. (1952). Optimum allocation in linear regression theory. *Ann. Math. Statist*, 23, 255–262. http://projecteuclid.org/euclid.aoms/1177729442

Krug G.K., Sosulin Y.A., Fatuev V.A. (1977). *The Planning of Experiments in Problems Identification and Extrapolation*. Moscow, Science.

A. Maiatin holds a Ph.D. in Pedagogical Sciences. He has been a deputy chairman of the methodical committee of open Olympiad in Informatics since 2008. His research interests other than education in computer science include an IT-infrastructure management based on technologies.

P. Mavrin is ACM ICPC World Champion in 2004, Silver medalist in Informatics of IOI in 2002, a member of the Technical Committee of IOI and ACM ICPC, an academic of Computer Technology chair of ITMO University where he teaches gifted school and university students.

V. Parfenov is a professor and dean of the faculty of information technologies and software engineering of ITMO University, a member of international organizing committee of the world programming championship, director of semifinal competitions in North-Eastern European region. He is one of the main organizers and creators of national and international Olympiads in computer science and programming for students and pupils in Russia. He has contributed a lot to the formation of educational system of search and training gifted in mathematics, physics, computer science and programming students since the year 2000.

O. Pavlova is an academic of Computer Technology chair of ITMO University. She has been teaching students having talent for mathematics, physics, computer science and programming since the year 2000. Nowadays her main research interest is the development of adolescents by investing in their education, in particular, by participating in Olympiads which considered one of the greatest contributions to children and adolescents' development.

D. Zubok holds a Ph.D. in physics and mathematics. He has been the deputy dean of information technology and software engineering department since 2006. Since 2008, the Executive secretary of the organizing committee of open Olympiad in Informatics and mathematics for school students. His current interests include computer science and data analysis.

Olympiads in Informatics, 2015, Vol. 9, 147–161
DOI: http://dx.doi.org/10.15388/ioi.2015.12

Math Contests: Solutions without Solving

Mārtiņš OPMANIS
Institute of Mathematics and Computer Science, University of Latvia
29 Raina Boulevard, Riga, LV-1459, Latvia
e-mail: martins.opmanis@lumii.lv

Abstract. The paper gives an insight in the possibility to use tools and methods usually not allowed at mathematical olympiads and contests for finding correct answers for original problems. Lot of problem examples from the various math contests are given. Possible effects and risks of competition format change are discussed, caused by usage of additional tools and Internet resources.

Keywords. Olympiads in Informatics, Math contests, competition tasks, grading, online tools.

1. Introduction

There are lot of popular math contests or their rounds where just a short answer like some number (listed together with few other possible options or without them) must be provided instead of a full "classic" solution with an adequate level of reasoning. Among such contests are, for example, Kangaroo [Kangaroo], MAA American Mathematics Competitions – American Junior High School Mathematics Examination, AMC 8 [AJHSME], AMC 10, AMC 12 and AIME [AMC], Sri Lankan Mathematics Competition [SLMC], Schools Maths Olympics held by University of Melbourne [MUMS SMO] and others. The famous resource "Project Euler" which "is a series of challenging mathematical/computer programming problems that will require ... programming skills ... to solve most problems" [Project Euler] also requires single numbers as answers to problems.

In this sense "finding a correct answer" usually is used as a synonym to "properly solve the task in the usual way as consecutive steps of correct reasoning and conclusions". And most probably, the intention of problem setters is that there is practically impossible to guess the correct answer or find it in any other way, and the solving process must be completed anyway. And the only difference is that it is not required to describe this process in a "polished" form, but only the final result must be provided. Such "lightweight" competition format is appealing due to the fast and simple answer grading

process. If additional tools are not used, then this assumption is almost correct, and the only way is to complete whole way from the given in a task statement till the correct answer, usually described in written form and named as "solution" and corresponding process as "solving". So we get "solution with solving".

Nowadays at the competitions in mathematics from the very basic level till the top – International Mathematics Olympiad (IMO) [IMO] only a limited number of tools are allowed in the process of problem solving. These tools are discussed in Section 2.

However, outside of competitions there are plenty of tools available which can help to find problem solutions. Some of them are described in Section 3.

With additional tools it is possible to obtain correct answers without the usual reasoning process sometimes essentially faster than in the "classical" way. Therefore "solution phase" (in its old meaning) may be omitted or, more correctly, a completely different way how to obtain the correct answer may be found. Further in this paper we will draw clear distinction between "solving" in its classical meaning and "finding correct answer" by using tools, methods, information sources, completely different from the solving process. In general, by "solution without solving" we denote the process of finding a correct answer by using an approach essentially different from the intended one. More precise description of this "alternative process" is given in Section 4.

By rise of computers programming in general became a powerful tool for rapid calculations and even for proving of propositions. As Petar S.Kenderov pointed out: "The nature of the mathematical research has changed significantly since considerable computing power came to the desk of almost every researcher and student. Mathematicians today can conduct complicated numerical experiments, use software for complex algebraic and analytic transformations, find patterns in huge data sets." [Kenderov]

By wide dissemination of computers and by the appearance of appropriate software (like as spreadsheets – MS Excel, Google Spreadsheets), appearing of Internet with enormously powerful information search capabilities (Google) and web-based tools (like as WolframAlpha [WA]), the possibility to find answers to particular mathematical problems increased dramatically. The main threat here is the possibility to obtain a correct answer having no clue about the solving process or even without understanding of the task statement.

Retaining of the old fashioned style of mathematical competitions makes bigger and bigger the breach between everyday life and math contests. Similarly, we could ask to perform communication tasks without usage of mobile phones and social networks, or to perform routine calculations without using electronic devices. At some point, such limitations becomes too orthodox and this may decrease the interest to participate in such competitions. This represents a serious problem also for problem setters – should avoid offering problems for which one can find straightforward solution in the "world of computer aided tools". For example, routine tasks like as multiplication of two long integers is not interesting as a contest problem.

At the same time Math olympiads are not alone in testing traditional mental skills. For example, at the Latvian national and Baltic regional level quite popular is a version of the Estonian mental calculation competition Miksike MentalMath[Miksike] where students are asked to perform fast calculations without additional tools.

Further in Section 5, problems from various mathematics competitions will be analyzed. Problems described in the present paper may be considered as outcomes of the experiment "What could happen if at real math contests additional tools would be allowed?". Although most of the examples below come from various math competitions, similar problems are observed also in informatics(programming) competitions [IOI, BOI] where solutions are mainly programs written in a programming language. The so-called "open input" tasks where problem solving strategy is not clearly defined are formally allowed but are rarely used. For solving problems in "Project Euler", it is intended that some programming must be involved. If correct answer will be found without programming, this also can serve as illustration of "solution without solving".

2. Tools Allowed at Mathematical Contests

Till now, in the classical mathematics competitions up to IMO only Euclidean tools – straightedge and compass [MathWorld] allowed. These tools allow creating of the basic geometric constructions known since Euclid's "Elements". Without irony it can be noted that the same tools are used in math competitions for centuries (even if we start counting from the first typeset of these books in 1482). These two simple tools can illustrate the difference between tools used in classrooms and mathematics competition.

In the classroom, rulers (with centimeter or inches marks) of finite length are used – in opposite to math contests where straightedge has no marks and is assumed to possess infinite length. The story with compass is even more interesting. According to Euclid's third postulate "Given any straight line segment, a circle can be drawn having the segment as radius and one endpoint as center", compass still is quite a limited tool. More precisely – this tool is "collapsing compass" or "compass without memory" – i. e. after drawing a circle, it loses any reference to the provided circle and the user must find a new segment and choose its endpoint to draw new circle. Quite a natural construction – drawing of two circles with the same radius with the centers at two given different points may not be performed by simple movement to another destination without changing the compass aperture. Luckily, Euclid himself (second proposition in Book I of "Elements") showed that also such compass can be used for transfer of distance to a given point and therefore its capacity is the same as the one of a "real" compass, and people must not worry about such limitations. Strictly speaking, when using a "real" compass, its equivalence to the abstract one must always be mentioned. Moreover, this is not case with straightedge and ruler (it is impossible to model ruler marks by simple straightedge).

3. Tools Available

One obvious tool which could be included in math contests is calculator, which became widely accessible for the general public since 1970-s. By the end of this decade

calculators became common in schools. However, till now in a lot of competitions the usage of calculators is prohibited without excuses – IMO, AMC, FERMAT and Math Bowl [UT contests], Kettering Mathematics Olympiad [KMO], Schools Maths Olympics [MUMS SMO].

At the same time other competitions are not so strict – a limited use of calculators is allowed in contests organized by University of Waterloo: "Calculators are allowed, with the following restriction: you may not use a device that has internet access, that can communicate with other devices, or that contains previously stored information. For example, you may not use a smartphone or a tablet." and (for Contests with Full Solution Questions): „While calculators may be used for numerical calculations, other mathematical steps must be shown and justified in your written solutions and specific marks may be allocated for these steps. For example, while your calculator might be able to find the x-intercepts of the graph of an equation like $y = x^3 - x$, you should show the algebraic steps that you used to find these numbers, rather than simply writing these numbers down." [Waterloo]

In Australian Mathematical Competition, calculators are banned starting from the year 7 and are allowed for younger participants as one of "aids normally available to them in the classroom" [Australian MC]. In the Math competition for students who are deaf or hard of hearing organized by Rochester Institute of Technology, only calculators with four basic arithmetic operations and square, square root and percent calculations are allowed. Scientific and graphing calculators are not permitted [RIT competition].

However, there are competition focused on usage of calculators – Calculator Applications Contest [CAC] or particular rounds of MATHCOUNTS Competition [MATHCOUNTS] and PA Math League and Atlantic Pacific Math League Contests [PAML].

Completely different "tool" which could be mentioned are sheets of limited number of formulas distributed for secondary school students at final exams in mathematics in Latvia. Similar supporting material are books together with printed materials of limited volume which teams are allowed to prepare in advance in secondary school students team competition in programming VKOSHP[VKOSHP].

Today, Internet together with personal computers, tablets, smartphones and other electronic devices may be qualified as a "tool available for secondary school students".

There are several competitions designed especially by assuming the free usage of Internet resources and making sometimes impossible to solve tasks without additional resources. Characteristic representatives of such competitions are The Internet Problem Solving Contest[IPSC] and MIT Mystery Hunt[MIT], there are a lot of competitions aimed at finding facts in the Internet.

There are only a few contests and competitions where the usage of tools and Internet resources is allowed but not always required for finding correct answer. Project Euler [Euler] and Latvian team competition in Mathematics and Informatics "Ugāle" [Opmanis] are among the few which can be named.

An interesting topic would be the classification of problem statements, solving strategies, etc. at the various existing competitions, but this is out of the scope of the present paper.

4. Description of "Finding Correct Answer"

It is very easy to argue against usage of Internet or sophisticated electronic devices: "You always may ask for help from aside by using Skype or e-mail and nobody can be sure that these answers are obtained you at all."

If we look from the perspective of a problem setter, several simple "fair play" rules must be stated:

1. No help from outside is allowed – no other people may be involved in the process of finding the answer directly or indirectly (like direct conversation or asking for advice in a forum or a newsgroup).
2. Only free, accessible for all, resources and tools may be used.
3. The goal is to speed up process of finding correct answer – i. e. the total time of finding answer must be less if compared with the classical approach where no additional tools are involved.
4. It shouldn't be possible to find a correct answer simply by using the task attribution. For example, by searching for an answer or solution of a particular competition in some published source because the title of competition or book is known. As an example here may serve IMO and its satellite resource – special section in website "Art of Problem Solving" [AoPS]. Solutions of "Project Euler" [Project Euler] tasks are also widely discussed on the Internet, so knowing that some particular task comes from the project, you will find a lot of hints, program examples or even correct answer.
5. Tasks must be non-trivial and non-routine, different from the well known tasks like "find the least possible Pythagorean triangle" or "calculate the fifth Fibonacci number". There must be no specialized tools (websites) for solving of general tasks of this class. For example, for tasks where a particular member of series must be found, an excellent resource is The On-Line Encyclopedia of Integer Sequences® founded by N.J.A.Sloane [OEIS®] – if few first members of sequence are provided, all the possible candidates together with vivid supporting information and references are reported back.
6. Clairvoyance like "I just saw the answer in my mind and can't explain why it is correct." or lucky guessing is not counted. Process must be repeatable and determined.
7. And the last but not least – there MUST exist verifiable correct answer of the task. For example, tasks like "What is the next integer in the sequence 2,0,2,1,2,2,2,2,2,2,3,2,4,...?" doesn't belong to this class, because without additional comments it is impossible to prove that the answer is correct (even if there exists a simple deterministic algorithm as in the above example).

Some remarks may be added:

As in the classical problem solving, impossibility to find an appropriate answer doesn't guarantee that answer cannot be found in a different way, by different tool or by different people.

There must be gained some experience to get maximum profit from the available tools and to see that some resource may be helpful. Even when final answer can't be found directly, usage of tools may help in finding new ideas or ways how the problem may be solved.

English (in general – any "big" language) speakers have advantage due to a larger number of useful resources.

From the problem setter's perspective, if almost unlimited usage of tools is allowed as described above, the invention of new tasks becomes a real challenge. Even if the task inventor is confident that the task is created by himself, "stress test" must be provided to be sure that there are no obvious workarounds or alternative approaches that could making the finding of the correct answer too easy. From the other side – if usage of some tools is assumed, the task may become too hard for solvers not familiar with particular tool and therefore the task becomes too narrow-focused.

Alternative way how to use old tasks is to "obfuscate" the original formulation so that keywords cannot be used in search engines to get on right track. Or "wrap" simple equations in the form of text problems and force contestants to correctly "unwrap" the task before solving.

5. Finding Answers for Tasks from the Previous Math Contests

To justify the approach with almost unrestricted usage of additional tools allowed, a lot of problems (tasks) from the previous math contests of various levels were investigated. This class of tasks is chosen because: 1) they are short in form, 2) they are clear and self-contained (no additional knowledge like foreign language is necessary to comprehend the task statement), 3) there is a unique answer. Of course, these tasks are not designed as tool usage exercises and some of them were created even before computers became widely used. These examples represent just a brief sketch of what are the options today if we want to obtain correct answers without solving. Original task formulations are given below together with competition title and problem number, brief description of tool(s) used, and how the usage of these tools helped in finding correct answer. Examples are denoted by "E" followed by the consecutive example number.

E1. USA AMC 8 1999, Problem 24

When 1999^{2000} is divided by 5, the remainder is
(A) 4 (B) 3 (C) 2 (D) 1 (E) 0

Tool: WA. Input expression "`1999^2000 mod 5`" and obtain the answer `1` immediately. Therefore the correct answer is answer D. Only basic knowledge about expression syntax in WA is necessary. Almost the same approach works for USA AJHSME 1996, task 15.

E2. USA AMC 8 2000, Problem 14

What is the unit digit of $19^{19}+99^{99}$?
(A) 0 (B) 1 (C) 2 (D) 8 (E) 9

Tool: WA. Input expression "19^19+99^99" and obtain a 198 digit integer with the last digit 8. Therefore the correct answer is answer D. Only basic knowledge about expression syntax in WA is necessary.

E3. USA AMC 8 2010, Problem 24

What is the correct ordering of the three numbers, 10^8, 5^{12} , and 2^{24}?
(A) $2^{24}<10^8<5^{12}$ (B) $2^{24}<5^{12}<10^8$ (C) $5^{12}<2^{24}<10^8$ (D) $10^8<5^{12}<2^{24}$
(E) $10^8<2^{24}<5^{12}$

Tool: WA. Step 1: Input expression "10^8−5^12" and obtain the answer –144140625. Therefore $10^8<5^{12}$ and possible valid candidates are just (A), (D) and (E).

Step 2: Input expression "10^8−2^24" and obtain the answer 83222784. Therefore $10^8 > 2^{24}$ and the correct answer is answer (A). It was just coincidence that the correct answer was obtained by two queries. However, even if more queries would be used, it is still possible to obtain the correct answer without any clue about comparison of powers in general. Only basic knowledge about expression syntax in WA and simple logical reasoning is necessary.

E4. USA AMC 8 2011, Problem 24

In how many ways can 10001 be written as the sum of two primes?
(A) 0 (B) 1 (C) 2 (D) 3 (E) 4

Tool: WA. Input query EXACTLY as it is formulated "In how many ways can 10001 be written as the sum of two primes?". Despite the fact that WA responds "WA doesn't understand your query", it gives valuable information about the number 10001 itself and theoretically from "10001 is an odd number." you can derive a correct answer. If this doesn't help, the initial textual query may be modified: "10001 as the sum of two primes". Now, WA is processing this query and gives the result "no such prime numbers exists". Therefore, the answer (A) is the correct one. It must be emphasized that the only skill necessary was a correct rephrasing of query. Even the knowledge of what does "prime number" means was not necessary at all.

E5. USA AJHSME 1995, Problem 3

Which of the following operations has the same effect on a number as multiplying by 3/4 and then dividing by 3/5?
(A) dividing by 4/3 (B) dividing by 9/20 (C) multiplying by 9/20
(D) dividing by 5/4 (E) multiplying by 5/4

Tool: WA. Input the expression "a*(3/4)/(3/5)" and obtain the result "5a/4", from which you can decide that (E) is the correct answer. In this query a general variable "a" is used. However, even without it you can obtain the answer 5/4 and by using some basic knowledge can transform it to a correct test answer.

E6. USA AJHSME 1995, Problem 15

> What is the 100th digit to the right of the decimal point in the decimal form of 4/37?
> (A) 0 (B) 1 (C) 2 (D) 7 (E) 8

Tool: WA. Input the expression "(floor((4/37)*10^100)) mod 10" and obtain the result 1, from which you can decide that (B) is the correct answer. Skills necessary to formulate a correct query are above the basic level and you must know how to use arithmetical functions. However, if you know that, you need not to know anything about the character of infinite decimal fractions like 4/37 – just translate the query and obtain the answer.

E7. International Mathematical Talent Search [IMTS] – Round 9 – Problem 4

> A triangle is called Heronian if its sides and area are integers. Determine all five Heronian triangles whose perimeter is numerically the same as its area.

Tool: Google search for "Heronian triangle". One of the first resources is Wikipedia topic [http://en.wikipedia.org/wiki/Heronian_triangle] having special paragraph „Equable triangles", where it is said: „A shape is called equable if its area equals its perimeter. There are exactly five equable Heronian triangles: the ones with side lengths (5,12,13), (6,8,10), (6,25,29), (7,15,20), and (9,10,17)." So you can obtain the correct answer without any calculations, reasoning, etc. Key to the success was a mathematically correct naming of terms in the problem statement and careful reading of source found. If „Heronian" would be replaced by other word, the described approach would not work. Of course, from the description of „Equable triangles" you must be wise enough to understand that these are the triangles you are searching for.

E8. Project Euler – Problem 21 „Amicable numbers"

> Let d(n) be defined as the sum of proper divisors of n (numbers less than n which divide evenly into n). If d(a) = b and d(b) = a, where $a \neq b$, then a and b are an amicable pair and each of a and b are called amicable numbers.For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore d(220) = 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; so d(284) = 220. Evaluate the sum of all the amicable numbers under 10000.

Tool: Google search for "Amicable numbers". One of the top resources is a reference to sequence A063990 [OEIS]. In this resource, the first members of the sequence of amicable numbers are listed: 220, 284, 1184, 1210, 2620, 2924, 5020, 5564, 6232, 6368, 10744, 10856,...

You can observe that only the first ten of them are under 10000 and you can simply sum them up obtain the correct answer. Essential again was a consequent usage of the term „amicable numbers". Otherwise, it would be impossible to obtain the answer so quickly, or (in case if this term would be used for denoting different concept) the answer would be incorrect.

E9. Project Euler – Problem 19 „Counting Sundays"

> You are given the following information, but you may prefer to do some research for yourself.
>
> - 1 Jan 1900 was a Monday.
> - Thirty days has September,
>   April, June and November.
>   All the rest have thirty-one,
>   Saving February alone,
>   Which has twenty-eight, rain or shine.
>   And on leap years, twenty-nine.
> - A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400.

How many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

Tool: An old notebook was received as a gift during the 9th Baltic Olympiad in Informatics 2003 in Estonia (in Estonian!) containing the so-called calendar of the century (fragment is shown in the Fig. 1).

Even without any knowledge of Estonian language it is possible to guess that 0 in the table denotes that the first day in the chosen month of a particular year is Sunday. For example, 1st of October of 1961 was Sunday. To get correct answer it is enough to process necessary rows and calculate total number of zeroes.

KALENDER 1901....2100

Aastad	Kuud											
	Jaan.	Veeb.	Mär.	Apr.	Mai	Juuni	Juuli	Aug.	Sept.	Okt.	Nov.	Dets.
1925 1953 1981 2020 2048 2075	4	0	0	3	5	1	3	6	2	4	0	2
1926 1954 1982 2021 2049 2076	5	1	1	4	6	2	4	0	3	5	1	3
1927 1955 1983 2022 2050 2077	6	2	2	5	0	3	5	1	4	6	3	4
1928 1956 1984 2023 2051 2078	0	3	4	0	2	5	0	3	6	1	4	6
1901 1929 1957 1985 2024 2052 2079	2	5	5	1	3	6	1	4	0	2	5	0
1902 1930 1958 1986 2025 2053 2080	3	6	6	2	4	0	2	5	1	3	6	1
1903 1931 1959 1987 2026 2054 2081	4	0	0	3	5	1	3	6	2	4	0	2
1904 1932 1960 1988 2027 2055 2082	5	1	2	5	0	3	5	1	4	6	2	4
1905 1933 1961 1989 2028 2056 2083	0	3	3	6	1	4	6	2	5	0	3	5
1906 1934 1962 1990 2001 2029 2057 2084	1	4	4	0	2	5	0	3	6	1	4	6
1907 1935 1963 1991 2002 2030 2058 2085	2	5	5	1	3	6	1	4	0	2	5	0
1908 1936 1964 1992 2003 2031 2059 2086	3	6	0	3	5	1	3	6	2	4	0	2

Fig. 1. Calendar of Centuries – Estonian version.

E10.

Prove that from the five different tetrominoes it is not possible to create solid rectangle (without holes and overlapping).

Tool. This may be called „proof found by accident", because when searching for „Tetromino", as one of top results you obtain Wikipedia page with a brief proof of the stated problem (second picture at the right hand side).

E11. British Mathematical Olympiad 2007 – Problem 2.

Find all solutions in positive integers x, y, z to the simultaneous equations
$x + y - z = 12$
$x^2 + y^2 - z^2 = 12$.

Tool. WA. Input "x+y-z=12, x^2+y^2-z^2=12, x>0, y>0, z>0" and obtain eight possible answers (x,y,z): (13,78,79), (14,45,47), (15,34,37), (18,23,29), (23,18,29), (34,15,37), (45,14,47), (78,13,79). The only necessary skill was correct reformulation of initial problem in WA syntax. On this particular example, two side effects could be observed: a) one can easy write incorrect equations just by omitting comma separating both equations. This leads to misinterpretation of equations and wrong answers, b) WA is evolving and the interpretation of an incorrect set of equation in different tool versions may change over the time.

For this particular problem, the input "$x + y - z = 12\ x^2 + y^2 - z^2 = 12$" in different WA versions was interpreted as "$x + y - z = 12,\ 12x^2 + y^2 - z^2 = 12$" (lot of „valid" integer answers having nothing in common with the solutions of the original equations) or „$x + y - z = 12x^2 + y^2 - z^2$" (no solutions in integers).

E12. International Mathematical Talent Search Round 10 – Problem 1.

Find $x^2 + y^2 + z^2$ if x, y and z are positive integers such that $7x^2 - 3y^2 + 4z^2 = 8$ and $16x^2 - 7y^2 + 9z^2 = -3$.

Tool. WA. Input "p=x^2+y^2+z^2,x>0,y>0,z>0,7x^2-3y^2+4z^2=8, 16x^2-7y^2+9z^2=-3" and obtain the answer p=165, x=4, y=10, z=7 immediately. It must be pointed out that in the input set of directives integers are not required.

E13. British Mathematical Olympiad 2007 – Problem 1

Find the value of $(1^4 + 2007^4 + 2008^4)/(1^2 + 2007^2 + 2008^2)$.

Tool. WA. Expression in almost original form may be given as input: „(1^4+2007^4+2008^4)/(1^2+2007^2+2008^2)" and answer 4030057 is obtained straightforwardly

E14. Pan African Mathematical Competition 2004 – Problem 2

`Is 4 sqrt(4 – 2 sqrt(3)) + sqrt(97 – 56 sqrt(3)) an integer?`

Tool. WA. Simple "Copy-Paste" from the original problem formulation into WA and as a result the value 3 is obtained. So, answer to the original question is "yes".

E15. British Mathematical Olympiad 2001 – Problem 1

`Find all positive integers m, n, where n is odd, that satisfy 1/m + 4/n = 1/12.`

Tool. WA. Query must be slightly modified to fulfill the "n is odd" requirement: "`solve{1/m+4/n=1/12,n=2k+1,m>0,n>0} over the integers`". Execution gives three pairs of answers (n, m): `(49, 588)`, `(51, 204)` and `(57, 76)`.

E16. 1960 IMO – Problem 1

`Determine all three-digit numbers N having the property that N is divisible by 11, and N/11 is equal to the sum of the squares of the digits of N.`

Tool. WA. Nontrivial translation of the original statement in terms of WA is necessary. The corresponding query is "`solve (100a+10b+c-11(a^2+b^2+c^2)=0; a>0, b>=0, c>=0) over the integers`". Two answers (a, b, c): `(5,5,0)` and `(8,0,3)` , which must be interpreted as three-digit numbers 550 and 803. You may consult the resource [AoPS] to see that these answers are the same as in the official solution.

E17. 1962 IMO – Problem 4

`Solve the equation` $\cos^2 x+\cos^2 2x+\cos^2 3x=1$.

Tool. WA. Input query "`solve (cos(x)^2+cos(2x)^2+cos(3x)^2=1)`" and get answer: $x = (8\pi n + \pi)/4$, $x = \pi(8n - 1)/4$, $x = \pi(8n - 3)/4$, $x = \pi(8n + 3)/4$, $x = \pi(4n - 1)/2$, where $n \in Z$.

E18. 1962 IMO – Problem 2

`Determine all real numbers x which satisfy the inequality:` $\sqrt{(\sqrt{(3-x)}-\sqrt{(x+1)})}>1/2$

Tool. WA. The query "`solve (sqrt(sqrt(3-x)-sqrt(x+1))>1/2)`" gives straight answer $-1 \le x < (32-\sqrt{127})/32$.

E19. 1965 IMO – Problem 4

`Find all sets of four real numbers` x_1, x_2, x_3, x_4 `such that the sum of any one and the product of the other three is equal to 2.`

Tool. WA. Input the query "`solve(x+y*z*q=2;y+x*z*q=2;z+x*y*q=2;q+x*y*z=2) over the real numbers`" and obtain the answers shown in Fig. 2.

Surprisingly, a simplified query "`solve(x+y*z*q=2;y+x*z*q=2;z+x*y*q=2;q+x*y*z=2)`" gives more appealing values of (x,y,z,q) (however, they are tailed with complex values and you must be able to distinguish them): `(-1,-1,3,-1)`, `(-1,3,-1,-1)`, `(3,-1,-1,-1)`, `(1,1,1,1)` and `(-1,-1,-1,3)`.

The same answers could be obtained by substitution in the expressions for z in Fig.2.

E20. International Mathematical Talent Search [IMTS]– Round 44– Problem 2

Let $f(x)=x\lfloor x\lfloor x\lfloor x\rfloor\rfloor\rfloor$ for all positive real numbers x, where $\lfloor y\rfloor$ denotes the greatest integer less than or equal to y.

1. Determine x so that $f(x)=2001$.
2. Prove that $f(x)=2002$ has no solutions.

Let's split this problem into two sub-problems – (1) and (2) and as the first step let's try to find an answer for the sub-problem 1.

Tool: WA. Execution of the query "`x*floor(x*floor(x*floor(x)))=2001`" gives the answer `x=2001/286`.

To solve E20 spreadsheets may be used as well. Let's denote p=floor(x floor(x floor(x))). Then the solution of equation x = 2001/p must be found. We can use the exhaustive search by filling six columns of spreadsheet with values of p (by definition, p is integer), x, floor(x), floor(x floor(x)), floor(x floor(x floor(x))) and floor(x floor(x floor(x)))-p. It is easy to go through all the values from 1 to 2001 looking for a value of p leading to 0 in the sixth column. Excerpt of the spreadsheet is given in the Fig. 3.

Let's consider solving of the second sub-problem of E20 by using the same approach as in the first sub-problem.

Tool: WA. Execution of the query "`x*floor(x*floor(x*floor(x)))=2002`" gives the answer x = 7 (with the remark that standard computation time exceeded). This answer is incorrect, because $7^4 = 2401 \neq 2002$.

Spreadsheet approach is better. Excerpt is given in the Fig. 4.

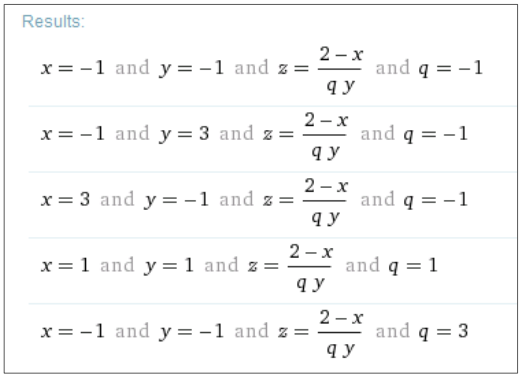


Fig. 2. Results of query for E19.

p	*x=2001/p*	*floor(x)*	*floor(x·floor(x))*	*floor(x·floor(x·floor(x)))*	*floor(x·floor(x·floor(x))) - p*
283	7,070671378	7	49	346	63
284	7,045774648	7	49	345	61
285	7,021052632	7	49	344	59
286	**6,996503497**	**6**	**41**	**286**	**0**
287	6,972125436	6	41	285	-2
288	6,947916667	6	41	284	-4
289	6,923875433	6	41	283	-6

Fig. 3 Excerpt from the Google spreadsheet for finding answer of the first sub-problem of E20 (formulas are shown as column headers).

p	*x=2002/p*	*floor(x)*	*floor(x·floor(x))*	*floor(x·floor(x·floor(x)))*	*floor(x·floor(x·floor(x))) - p*
285	7,024561404	7	49	344	59
286	7	7	49	343	57
287	6,975609756	6	41	286	-1
288	6,951388889	6	41	285	-3

Fig. 4 Excerpt from the Google spreadsheet for finding answer of the second sub-problem of E20 (formulas are shown as column headers).

From the spreadsheet you can conclude that there is no solution – a value of p for which difference in the sixth column is 0.

Are the above examples "good examples" (in almost all cases we succeed in finding a correct answer) or "bad examples" (in some cases it was too easy to obtain the right answer without proper evidence of adequate thinking process)? It depends whether you are a problem setter and are worried about the possibility of cheating by using modern gadgets with Internet access, or just want to check your solution of the task, or represent different group of people.

Will the future be so simple? Just input your equation and obtain a correct answer without any clue and without the need to understand how it works? We already observed problems with WA when tried to find answer to the sub-problem 2 of E20. Is this just an accident? Unfortunately, NO.

E21.

Find all solutions in integers of equation $(x+y)^2=x^3-y^5$.

In the original problem statement it was necessary to find all solutions in prime numbers. Obviously this would be a subset of the problem stated there.

Tool: WA. Input the equation "`(x+y)^2=x^3-y^5`" and you will obtain two solutions in integers (x, y): `(0,-1)` and `(0,0)`. However, there exists at least one more obvious solution: `(1,0)` not reported there. Even worse, there is a nontrivial answer `(7,3)` not

given without any sign that the answers found are not all possible or warning that it was not possible to calculate all solutions due to lack of resources or any other reason.

The general observation is as follows: "Tools may be not perfect and their produced answers may be incomplete or wrong even if you use them properly". For complicated tools like WA it is hardly possible to justify that tool is working properly for ALL correct inputs. If tools are used, you always must to justify the answers found. However, this justification task may as complicated as solving of the initial problem.

6. Conclusions

Computer and Internet based tools are very powerful and may be successfully used for finding answers for different math problems. However, their unlimited usage may be dangerous in a sense that young people may use them without necessary criticism like today calculators are used for routine calculations ignoring the possibility of computational errors. Tasks from the previous contests can't be given in such contests without careful examination as well. In the case when tools and resources are allowed a shift in trained skills is necessary – more attention must be paid to proper selection of the most appropriate tool, its usage and validation of the received answers. If we observe traditional topics in math contests – Algebra, Number theory, Combinatory and Geometry, it seems that problems from algebra and number theory are more suitable for tool usage in comparison with problems from geometry.

More tools, their usage for different problem classes and competitions, possible side effects and drawbacks must be further investigated.

Acknowledgements

I would like to thank Prof. Kārlis Podnieks for valuable comments.

References

AMC. *Mathematical Association of America (MAA) American Mathematics Competitions.*
http://www.maa.org/math-competitions/amc-contests
AJHSME. *American Junior High School Mathematics Examination.*
http://artofproblemsolving.com/wiki/index.php/AMC_8_Problems_and_Solutions
AoPS. *Art of Problem Solving. Community-Contest collections-IMO.*
http://artofproblemsolving.com/community/c3222_imo
Australian MC. *Australiam Mathematics Competition*. http://www.amt.edu.au/amcfact.html
CAC. *Calculator Applications Contest.*
http://www.uiltexas.org/academics/calculator-applications
IMO. *International Mathematics Olympiad.* http://www.imo-official.org/
IMTS. I*nternational Mathematical Talent Search.*
https://cms.math.ca/Competitions/IMTS, https://cms.math.ca/Concours/IMTS/
IPSC. *The Internet Problem Solving Contest.* http://ipsc.ksp.sk/

Kangaroo. *International Mathematical Kangaroo, Kangourou sans frontières.*
http://en.wikipedia.org/wiki/Mathematical_Kangaroo
Kenderov, P.S. (2006) Competitions and mathematics education. In: *Proceedings of the International Congress of Mathematicians, Madrid, Spain, 2006*. 1583–1598.
KMO. *Kettering Mathematics Olympiad.* http://paws.kettering.edu//~acheng/Olympiad/
MATHCOUNTS. *Competition.*
http://www.artofproblemsolving.com/Wiki/index.php/MATHCOUNTS
MathWorld. http://mathworld.wolfram.com/EuclideanTools.html
Miksike. *MentalMath.* http://en.wikipedia.org/wiki/Miksike_MentalMath
MIT. *Mystery Hunt.* http://www.mit.edu/~puzzle/
MUMS SMO. *Schools Maths Olympics, Melbourne University Mathematics and Statistics Society.*
http://www.ms.unimelb.edu.au/~mums/olympics/smo.html
OEIS®. *The On-Line Encyclopedia of Integer Sequences®.* https://oeis.org/
PAML. *PA Math League and Atlantic Pacific Math League Contests.*
http://www.npenn.org/page/1437
Project Euler. https://projecteuler.net/
SLMC. *Sri Lankan Mathematics Competition.*
http://www.slmathsolympiad.org/slmcPast.jsp
Opmanis, M. (2009). Team competition in mathematics and informatics "Ugāle" – finding new task types. *Olympiads in Informatics*. 3, 80–100.
RIT competition.
http://www.rit.edu/ntid/mathcompetition/what-kind-calculator-allowed
UT contests – FERMAT (Fundamental Exams of Remarkable Mathematical Ability and Talent) and Math Bowl.
http://www.math.utk.edu/MathContest/ContestInfo.html
VKOSHP. http://neerc.ifmo.ru/school/russia-team/rules.html
Waterloo. *University of Waterloo, Mathematics and Computing Contests.*
http://www.cemc.uwaterloo.ca/contests/calculators.html
WA. *WolframAlpha*. http://www.wolframalpha.com/

M. Opmanis is researcher at the Institute of Mathematics and Computer Science of University of Latvia. He is one of the main organizers of Latvian Olympiad in Informatics, was deputy or team leader of Latvian IOI and Baltic OI teams. M.Opmanis was head of jury of Baltic Olympiad in Informatics at BOI 1996, 1999, 2004 and 2012. Since 2012 he is member of IOI International Committee.

Olympiads in Informatics, 2015, Vol. 9, 163–172
DOI: http://dx.doi.org/10.15388/ioi.2015.13

Conducting Complex Competitions in Informatics with Individual Tasks

Pavel S. PANKOV, Jyldyz R. JANALIEVA
International University of Kyrgyzstan
e-mail: pps50@rambler.ru, noledi@yandex.ru

Abstract. Most olympiads in informatics are actually programming contests, sometimes it is stressed by the rules „tasks must be of algorithmical nature". Meanwhile, now informatics covers almost all branches of human activity and participation in competitions with wide spectrum would be of interest to young people. To prevent participants from copying answers from others, we propose to generate individual tasks from personal data to achieve uniqueness (all contestants would receive different versions of tasks). Also, we draw attention to tasks of "output-only" and "open-ended" types (which can be also called "semi-algorithmical" because the contestant combines intuition with using algorithms, which may be known in advance or devised while solving) and optimization tasks with unknown exact answers. The main problem is automated assessment of solutions of various tasks, some techniques are proposed.

Keywords: complex competition, informatics, individual tasks.

1. Introduction

Let us retell general items of Calls for tasks, 2011–2015: "IOI tasks are typically focused on the design of efficient, correct algorithms, etc. … the submission of novel task types not yet seen in IOIs, tasks whose basic rules (if not optimal strategy) are accessible to a *wide audience* … tasks that illustrate algorithms and computational problems that arise in *a variety of human endeavours* … open-ended tasks, ones that do not necessarily have a known efficient or optimal solution … tasks that go beyond the typical format in which a program collects input, performs some computation, and returns output … tasks with some measure of solution effectiveness other than CPU time consumption … are encouraged" (emphasis by us).

But "Examples include "reactive" and "output only" tasks which have been used occasionally in previous IOIs…", that is, most IOI tasks were of the only type: to write an effective (discrete) algorithm.

We may add that informatics now covers all directions of human activity. The aim of this paper is an attempt to evolve a variety of tasks meeting more its branches and its applications.

One of such scopes was proclaimed in well-known and widely conducted *Bebras* under V. Dagiene‘s leadership:

- Information: conception of information, its representation (symbolic, numerical, graphical), encoding, encrypting.
- Algorithms: action formalization, action description according to certain rules.
- Computer systems and their application: interaction of computer components, development, common principles of program functionality, search engines…
- Structures and patterns … of discrete mathematics … combinatorics and actions with them;
- Social effect … cognitive, legal, ethical, cultural, integral aspects…
- … puzzles: logical games, mind maps, used to develop technology-based skills.

Also, another of such attempts is our conducting competitions on “applied mathematics” (new genre, as we hope) in Kyrgyzstan, examples of tasks proposed to them are marked (*) below.

To avoid cribbing and make such competitions more interesting we propose to use individually generated tasks.

Section 2 contains definitions of extended tasks and individually generated tasks.

Section 3 contains examples of tasks with simple discrete response.

Section 4 describes various types of tasks with objects outside of the examination program.

Some non-standard tasks on programming to be solved without translators are proposed in Section 5.

Tasks on interactive search of a “hidden object” are considered in Section 6.

Sections 7 and 8 apply the ideas of output-only and open-ended tasks on approximate calculations and optimization to proposed competitions.

2. Definitions of Extended and Individually Generated Tasks

In our observation, almost all young persons who could master programming begin to write test programs with standard “multiple choice” of some preliminary written answers. Diversification is almost always reduced to a random choice of data base with ready tasks and to a random permutation of answers. It does not give opportunity to evince programming skills and knowledge of subjects previously studied. Meanwhile, since the eighties, we as well as others proposed more flexible methods to develop software involving various peculiarities of the subjects studied, and examining various abilities of persons tested.

Besides of common requirements (Validity, Objectivity and Reliability), to improve efficiency of evaluation of knowledge and skills, we offered the following ones:

Generativity: the complete text (content) of a task must not exist before examination (competition) and must be generated randomly just before it.

Uniqueness: all examinees (contestants) must receive different versions of tasks.

Concreteness: the examinee’s (contestant’s) response must be in the form of number, word, action.

Definition 1. (Pankov and Janalieva, 1995). An extended task is an algorithm generating different logically correct and methodically proper tasks (of the same level of difficulty) and corresponding right answers from the initial data (randomly) chosen from finite but sufficiently large sets (ranges). Another term for such algorithms is "parameterized questions".

Definition 2. The number of ranges being used for generating the task in sufficient different ways is said to be dimension of the extended task.

Definition 3. (Janalieva, 2009). An algorithm permitting one (a teacher) to choose subsets of sets of initial data and generating different logically correct and methodically proper tasks by initial data, (randomly) chosen from these subsets is said to be an adjustable extended task.

Definition 4. (Janalieva, 2012). An individually generated extended task consists of two algorithms. The first algorithm composes different, logically and methodically correct tasks using personal information about a contestant. The second algorithm generates corresponding right answers using the same information.

Correspondingly, at distant competitions the contestant is to enter his/her personal data and his/her answers and the second algorithm generates right answers for checking using the same data. And same data are printed into the certificate.

We give examples of both easy and difficult tasks to present the scope of possibilities. Also, weaker participants will spend more time to solve easy tasks and obtain non-zero points; stronger participants will spend less time to solve easy tasks and have more time to solve difficult tasks.

Simple tasks in different sciences (see, for example, Pankov, 2010) are constituents of such examination too, because "…algorithms and computational problems that arise in a *variety of human endeavours*."

All the tasks in the following sections are extended ones. Randomly generated elements are denoted by italics (some of them are "primary"; others are calculated from them).

3. Tasks with Discrete Response

These tasks are intended for "Objective evaluation of understanding by means of discrete response" instead of traditional "Evaluation of understanding by means of multiple choices". They are relatively easy because their aim is not checking of skills, but checking of knowledge of notions.

3.1. *Tasks with Simple Calculations on Digits*

Task 1. (Notion of symmetry). What is the minimal total amount to be added to the digit(s) to obtain a symmetric pattern:

1.1. [Axial symmetry] 65299756
14388342 [answer: 6]

Remark. Traditional multiple-choice version of this task is: choose the symmetrical:

A) 65299756	B) 65299256	C) 65299756	D) 65299256
14388342	14388341	14388341	65299252

1.2. [Transitional symmetry] 65296526
14381436 [answer: 5]

1.3. [Rotational symmetry] *6*5298381
14389256 [answer: 4]

Task 2. (Notion of numeration). How many asterisks are here in *binary/hexadecimal* system?

******* **** ****
******** **** *

Remark. Random initial data are presented in such a way that it would be easier to write out the answer immediately than to count in decimal and convert.

Task 3. (Notion of graphical information). How many *bits/bytes/kilobytes (rounded)* are necessary to encode a 400 × 560 pixels image with *the colours: white; light grey; dark grey; black /* 16 *colours /* 256 *colours*?

Task 4. (Notion of error correction). 25 digits, 5 checksums (mod 10) by rows and 5 checksums (mod 10) by columns were transmitted, but one of the 35 digits was changed. Which one?

652980
143895
256014
558273
087623
47617

Write the answer in the form: <changed digit>#<restored digit> [answer: 2#8]

Task 5. (Notion of directed rounding). Given: $7 \le X \le 20$; $50 \le Y \le 70$. Find the width of the narrowest closed interval with integer boundaries containing all quotients Y/X: [answer: 8]

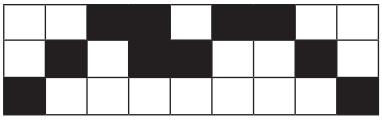
Task 6. (Notion of chemistry). How many whole molecules of $A_4B_3C_2$ can be made of 45 atoms of A, 40 atoms of B and 30 atoms of C? [answer: 11]

3.2. *Tasks with Simple Calculations on "Big Pixels"*

Tasks like Task 1 also can be presented as follows:

Task 7. (Notion of symmetry). How many black squares, at least, are to be *added/*

erased/shifted to obtain a symmetric pattern (in real task, the table should be about 5 × 10 pixels):



Task 8. (Notion of connectedness). (Also, given pattern of white and black squares).

A) How many, at least, black squares are to be *added/shifted* to obtain a connected set [from non-connected one]?
B) … to be erased to make the set non-connected [from connecting one]?

3.3. *Interactive Tasks with a Simple Discrete Response*

Tasks of the above types can be also presented as follows:

"Change as few digits as possible to (obtain the result)";
"*Add/Erase/Shift* as few black squares as possible to (obtain the result)"
(An interactive interface is to be programmed for such actions).

4. Tasks with Objects Outside of the Examination Program

4.1. *Tasks with Real Objects Outside of the Examination Program*

General task 9. *(Measuring a real object). The real object contains many marks (numbered points); their coordinates are in the examination program; it is prepared and copied by the jury in advance.*

Each contestant is given a copy of the object and a ruler.

"What is the (quantitative characteristic) of the object formed by … (*random*) marks (points)?"

The examination program calculates the result by means of coordinates of these marks and compares it with the contestant's answer; some error is permitted.

Examples:

Task 10. (Area of a real triangle). Given a sheet with some dozens of numbered points and a ruler.

"What is the area (in square cm) of the triangle formed by point 32, point 14 and point 56?"

Remark. If the area is too small then the examination program picks another triple.

Task 11. (Geodesy). The same sheet.

"Denote the imaginary intersection of straight lines point 32 – point 14 and point 12 – point 65 as point 100. What is the length (in cm) of the segment point 32 – point 100?"

4.2. *Tasks with Extracting Information of Files Outside of the Examination Program*

These tasks are intended to evaluate skills in applying various existing software.

Task 12. (Geography). Given a file with geographical maps and there is a vast list of names of geographical points (with their coordinates) in the examination program.

"Find (very approximately) the distance (km) between *point*1 and *point*2 along the earth's surface."

The examination program calculates the distance by means of formulas of spherical trigonometry.

For the following tasks, files are formed in advance such that there is a functional relation between a file's name and its content but this relation is too complex to be guessed by the contestant by two-three examples. For instance, let the file file*N*.txt, $N = 10 .. 90$ contain the number $3980 - N * (N - 44)$.

General task 13. (Searching for and extracting information outside). (Find and) open the file(s) with *name(s)* and extract *information* (a *number* or a *word*).

Examples:

Task 14. (Search in text). Find the distinguished record (the least number, the longest word, etc.) in the announced (short) files, for instance d:\dir12\file5*.txt .

Task 15. (Search in text). Open the (vast) file Task15.doc and input the least one among all the numbers written immediately before numbers 16.

Task 16. (Excel). Open the file Task16.xls and find the sum of all positive numbers in the rectangular area *B*4–*F*29.

4.3. *Tasks with Treating Files Outside of the Examination Program*

General task 17. (Transforming files). Open the file …, make the following *transformation,* extract *information* and input it.

The idea is the following. Any *transformation* can be made by several means (by mouse, by keyboard, by hotkeys, etc.) and the examination program, certainly, cannot check correctness of all of them. But the file is composed in such a way that the result of *transformation* defines *information.*

A simple example:

Extended task 18. (Deleting symbols). The file Task18.txt contains

E214F5F325H43H54345G5G56H7G8G9

"Open the file Task18.txt, delete all $N = 1 .. 5$ and input the position of $M = 6 .. 9$."
The answer is $24 - N + 2\,(M - 6)$.

For example: "Open the file Task18.txt, delete all "3" and input the position of "7".
The answer is 23.

4.4. *Tasks with Creating Files*

General task 19. *(Creating images). Given (verbal) description of an (random) object. "Find its (continuous quantitative characteristic rounded to integer) or (discrete quantitative characteristic)."*

[Calculation of such characteristic is absent in standard software.]

The contestant is to guess that the only possible way to solve the task in given time is creating an image of the object by means of standard software and looking at it.

Examples:

Task 20. How many zeros has the function $F(X) = \sin X + 0.2\,X - 0.01\,X^2$, $0.8 \le X \le 5.7$?

Task 21. What is the value of the least local maximum of (such function) rounded to integer?

The examination program itself solves such tasks by means of (written in advance) procedure where random values are input.

5. Non-Standard Tasks in Programming

In contrast to subsections 4.2, 4.3 and 4.4, tasks below are intended to be solved without existing software. One of the techniques is involving too large numbers.

Tasks are given in a conventional Pascal-like language, the contestant is to guess the (very simple) syntax of this language.

General task 22. (Analyzing a program). Given a program. "What Input will yield given Output?"

A simple example:

Task 23. Given the program

```
M := 1000^1000; Input X; X := 2 * X + M;
If X < 102 + M then X := X + 50 - M else X := X + 80 - M;
Output X.
```

What Input will yield Output 146? [answer: 48]

General task 24. (Graphical operations). Given some graphical operations involving objects that do not fit on the screen. How many pixels will be coloured?

A simple example:

Task 25. Given a piece of program:

Line (20, 2000)–(20, 5000), red.
Line (15, 2100)–(15, 2200), green.
Line (15, 2100)–(15, 2150), red.

How many pixels will be red? How many pixels will be green?
The following types are well-known. We mention them for completeness.

General task 26. (Manual execution). Given a program without input and with large intermediate results. What will the Output be?

General task 27. (Acceleration of program). Given a program without input working too slowly (with some embedded cycles which can be eliminated). What will the Output be?

6. Interactive Tasks with "Black Box"

For reactive tasks, the contestant is to write a program which will call given procedure(s) and find the answer by their responses. We propose the contestant to make queries him/herself.

The simplest example is:

Task 28. You are to detect an unknown integer between 120 and 1100. For this purpose you may guess numbers and obtain responses "Guess higher", "Guess lower", "Exactly!" If you solve the task in less than 11 queries you will get the full score.

Of interest for students waere:

General task 29*. You are to reach the goal in a (may be unusual, see Borubaev *et al.*, 2003) space. For this purpose you may choose the directions of your steps and obtain distances to the goal as responses.

7. Output-Only Tasks – Approximate Calculations

General task 30*. A *random* geometrical object is described. To get full score calculate its (*quantitative characteristic*) with the accuracy …

Remark 1. Such tasks are more interesting than logically equivalent "calculate the definite integral".
Remark 2. A task about the length of an arc of parabola gave the paradoxical, although predictable result. All students who knew the formula for arc length got into a tangle, but most of students, who did not know this formula, have solved the task correctly.

8. Output-Only and Open-Ended Tasks – Optimization

A *random* function to be optimized may be constructed in such a way that the exact answer is known but it is practically impossible to find it. Then the task is "Find the *maximum/minimum* of the function ... with the accuracy..." or "as precisely as possible".

For example, the function $F(X) = A(X - C)^4 + B(X - C)^2 + D$ after opening the brackets by the examination program where random (individual) real numbers

$A \in [5.1, 5.2]$; $B \in [2.1, 2.2]$; $C \in [5, 10]$; $D \in [10, 30]$ (argmin $F(X) = C$; min $F(X) = D$).

Tasks with an unknown exact answer should be used in the final competition only.

Difficulty of obtaining results for different *random* functions must be equal, as in the example.

If the contestant submits an approximate answer $M < D$ then it is incorrect; else the smaller the difference $(M - D)$, the better is the answer.

(We recall the traditional procedure.) During the competition the examination program checks correctness of submitted answers only. After the competition the submitter of the best result earns the full score and other contestants who submitted correct answers earn corresponding parts of the full score.

8. Conclusion

We hope that implementation of tasks of types listed above will enrich competitions like *Bebras,* will distinguish young persons who have good command in informatics as whole, but are not experts in programming, and will attract more young people to informatics. Developing software for such competitions will be of interest for programmers.

References

Pankov, P.S., Janalieva, J.R. (1995). Experience and perspectives of using UNIQTEST complex of unique tests in the learning process. *Theses of Reports of Scientific-Practical Conference Education and Science in New Geopolitical Space*. Bishkek, International University of Kyrgyzstan. (In Russian).

Borubaev, A.A., Pankov, P.S., Chekeev, A.A. (2003). Chapter 4. Constructive and computer presentations of uniform spaces. In: Borubaev, A.A. *et al.*, *Spaces Uniformed by Coverings*. Budapest. Hungarian-Kyrgyz Friendship Society.

Janalieva, J.R. (2009). Development of software for the examination on the basis of extended tasks. In: *Proceedings of the VI International Scientific-Practical Conference Intellectual Technologies in Education, Economics, Management.* Voronezh, Russia, 330–335. (In Russian).

Pankov, P.S. (2010). Real processes as sources for tasks in informatics. *Olympiads in Informatics*, 4, 95–103.

Janalieva, J.R. (2012). Conducting offline informatics olympiads with individual tasks. *Olympiads in Informatics*, 6, 170–177.

P.S. Pankov (1950), doctor of physical-math. sciences, prof., corr. member of Kyrgyzstan National Academy of Sciences (KR NAS), was the chairman of the jury of Bishkek City OIs (1985–2013), of Republican OIs (1987–2012), the leader of Kyrgyzstan teams at IOIs (2002–2013). 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.

J.R. Janalieva (1969), candidate of pedagogical sciences, conducts various competitions on mathematics, informatics and languages, including collective ones for students of Bishkek and Internet ones for students of Kyrgyzstan. Graduated from the Kyrgyz State University in 1991, she works as a docent of the International University of Kyrgyzstan.

Olympiads in Informatics, 2015, Vol. 9, 173–191
DOI: http://dx.doi.org/10.15388/ioi.2015.14

Multiple Criteria Decision Methods in Informatics Olympiads

Jūratė SKŪPIENĖ
Vilnius University Institute of Mathematics and Informatics
Akademijos 4, LT-08663, Vilnius, Lithuania
e-mail: jurate.skupiene@mii.vu.lt

Abstract. Multiple Criteria Decision Making (MCDM) is applied in a variety of areas, including education. Informatics Olympiads, problem solving contests for high school students, is the area where MCDM methods can also be applied. The case of the Lithuanian Informatics Olympiad is analysed in this paper. There are several aspects occuring while maintaining the contest that requires decision making. The work of each contestant is evaluated in terms of several criteria, where each criterion is measured according to its own scale (but the same scale for each contestant). Several jury members are involved in the evaluation. Thus we get a problem: how to calculate the aggregated score for whole submission in the above mentioned situation. Another similar problem is making decision on national team selection for other international contests where each candidate is evaluated in terms of several criteria. The chosen methodology for solving this problem is multiple criteria decision analysis (MCDA). The outcome of this paper is the score aggregation method proposed to be applied in LitIO developed using MCDA approaches.

Keywords: Informatics Olympiads, programming contests, evaluation, grading, multiple criteria decision analysis.

1. Introduction

The field of multiple criteria decision analysis (MCDA) is also termed as a multiple criteria decision aid or multiple criteria decision making (MCDM). Its target is to help reach a consensus and compromises between conflicting goals (i.e., multiple criteria) in complex problems.

In real life it is unusual that the problem is presented to the analyst in a form of a clearly defined set of alternatives and criteria (Belton and Stewart, 2003). Problems might be complex and confusing and they typically involve a wide range of criteria that need to be considered. They might involve conflicting criteria, the conflicts between different stakeholders about the importance of criteria in making a decision. It might even be required to define criteria as they are not clear at the initial stage of the problem. The general goal of MCDA is to assist individual or groups of decision makers to choose the best alternative.

MCDA is defined as a collection of formal approaches which seek to take into account multiple criteria in order to help decision makers to explore different decision alternatives (Belton and Stewart, 2003).

Potential problems that MCDA can be applied to solve come from a variety of areas like business, medicine, banking, marine industry, bioinformatics, public policies or education (Aruldoss *et al*., 2013).

Education is one of the areas where MCDA can be widely applied. These are learning content and learning software evaluation problems (Kurilovas and Serikoviene, 2010), higher education decision making problems, (resource allocation, performance management, budgeting and scheduling) (Ho *et al*., 2006), using MCDA for accreditation in order to evaluate IT skills and qualifications (Siskosa *et al*., 2007), evaluating factors that determine the quality of higher education (Tsinidou *et al*., 2010), evaluation of the quality of e-learning systems (Tzenga *et al*., 2007) and educational websites (Shee and Wang, 2008), pedagogical evaluation of teachers (Filipe *et al*., 2015), evaluating quality of learning objects (Kurilovas *et al*., 2011).

The majority of research of application of MCDA in education that we discovered was related to the evaluation of quality of various educational factors or tools. However, we noticed that the choice of MCDA approaches highly depends upon the category of the problem under consideration.

Four broad categories of MCDA problems have been proposed (Roy, 1996):

- **The choice problematique.** Problems fall into this category if there is a need to make a choice from a set of alternatives. However the set of alternatives might be either finite or infinite.
- **The sorting problematique.** In this case the given alternatives have to be sorted into several categories, such as “definitely acceptable”, “possibly acceptable”, “definitely unacceptable”.
- **The ranking problematique.** The alternatives have to be ranked in some order of preference.
- **The description problematique.** Possible alternatives and their consequences have to be described formally in a systematic way so that the decision makers could evaluate the alternatives.

Variations or amendments to this classification are also possible (Belton and Stewart, 2003).

Another classification of MCDA problems is *one-off* versus *repeated* problems. In some cases, a decision has to be made only once as the problem is unique. This is a one-off problem and the process is oriented towards arriving at a specific decision. In the case of repeated problems the same problem is recurring a few times or periodically. Then MCDA is oriented towards creating a procedure to be used in decision making.

An MCDA problem can also be classified either as *a single decision making* or *group decision making* problem. In the case of a group decision making problem, several decision makers are involved and they can have different values and opinions how to address the problem. In order to approve the decision, the consensus and compromise among different decision makers has to be reached.

Different authors suggest different stages of the MCDA process. (Val, 2002) proposes a scheme consisting of four stages in particular, problem structuring (decomposed into five sub-stages), preference elicitation, recommended decision, and sensitivity analysis. (Oberti, 2004) suggests four stages of the MCDA process, i.e., beginning of the study, evaluation of actions, multiple criteria modelling, multiple criteria processing, and recommendations.

Each stage consists of two or three sub-stages. (Belton and Stewart, 2003) offer three stages: problem identification and structuring, model building, and using a model to inform and challenge thinking. The scheme based on (Belton and Stewart, 2003) is presented in Fig. 1.

These stages reflect a variety of approaches to MCDA, however, they confirm that an extensive problem analysis and structuring are vital before mathematical algorithms can be applied. In all those approaches the stages are iterative and interactive, i.e., they foresee a return to previous stage, review and update its outcome.

Even though mathematical MCDA algorithms help to arrive at some acceptable alternative, many authors emphasize that MCDA cannot be used to arrive at the "right" answer and it cannot provide a fully objective analysis and totally eliminate subjectivity (Belton and Stewart, 2003). The process of MCDA is emphasised more than the decision it helps to arrive at (Keeney and Raiffa, 1976; Roy, 1996; Zeleny, 1982). The process involves not only the application of mathematical algorithms to come up to the final decision, but also learning about the problem, identifying the key concerns, priorities, uncertainties, values, exploring and generating different alternatives. This should lead to better explainable and justifiable decisions.

2. Submission (Contestant) Ranking in LitIO as an MCDA Problem

The Lithuanian Informatics Olympiad (LitIO) is a state supported algorithmic problem solving competition for students in secondary education. The contestants are given algorithmic tasks and have to solve them in four or five hour contest sessions. They have to design and implement the algorithm in order to solve the task. The task may also require to submit reasoning for algorithm design or a set of test cases. The material

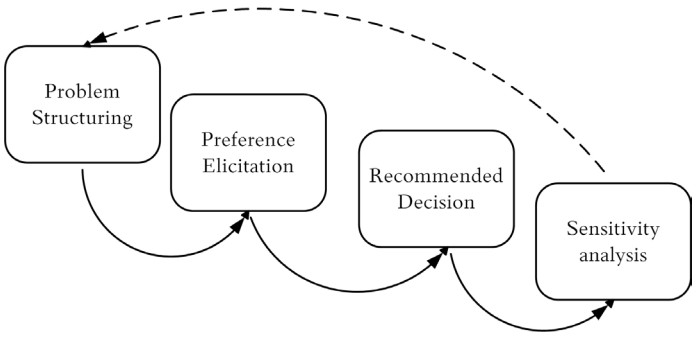


Fig. 1. Basic stages of the MCDA process.

submitted for evaluation by a contestant is called *a submission*. After the submission has been evaluated in terms of separate criteria where each criterion is measured according to its own scale (but the same scale for each contestant), the aggregated score has to be calculated so that the submissions can be ranked with respect to other submissions for the same task. Measuring the distance between contestants is also important.

If the contest also serves as team selection event, then the decision has to be made which contest participants will be invited to represent country in a regional or international contest. In the long term practice several criteria for this selection were used and the decision is made by the Scientific Commitee of LitIO.

Evaluation in LitIO as such can be treated as an MCDA problem, and the work presented in (Skūpienė, 2010) corresponds to the first stage of the MCDA process, i.e., problem structuring. The outcome of problem structuring is an explicit list of criteria and alternatives. The task that has to be explored in this paper is ranking of submissions once the submisions have been evaluated in terms of separate criteria. Note, that in practice, the overall ranking has to be based on the scores of several tasks. However, in this paper, we limit our research to determining a score for one task only.

Team selection in LitIO can also be treated as an MCDA problem. The list of selection criteria, as well as the procedure how to evaluate a contestant aginst each criterion has been decided many years ago. The criterion are: score of the final round of LitIO in the current year; score of the regional contest in the current year (if applicable); awards received in regional and international competitions (there's been approved a concrete list of such competitions); competitor's grade in the current year. There is a consensus over that among the decision makers.

However the criterion are not directly comparable and the understading of the goals of the team selection varies. Therefore the discussions and the search for the algorithm for the team selection continues. The overall problem - determining the ranking based on the criteria can also be treated as as MCDA problem.

Three major roles can be identified in MCDA. They are *decision maker*, *decision analyst* and *stakeholder* (Val, 2002).

The scientific part of LitIO is managed by the scientific committee. The scientific committee is responsible for all the scientific decisions, i.e., approving the syllabus of the contest, designing tasks and tests, approving the evaluation procedure, performing the evaluation, approving ranking and declaring winners and selecting teams to represent Lithuanian in regional and international contests. In 2015 the scientific committee of LitIO consisted of 22 members (Sci, 2015). The scientific committee is the only decision maker in this context. The role of decision analyst is played by the author of this paper.

The most important stakeholders are interested programming and algorithmics students in secondary education from all over Lithuania, as well as the community of informatics teachers. This community of stakeholders is affected directly by each decision or change in the evaluation scheme. The scientific committee of LitIO is also a stakeholder, because possible changes in the evaluation scheme might change their working procedures, time spend on task design and evaluation.

The model of relationship between the different roles in the decision analysis process in the problem under consideration is presented in Fig. 2.

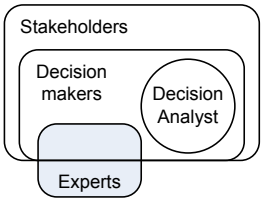


Fig. 2. Model of relationship between different roles in decision analysis in the analysed problem.

There were suggested several ways how to classify MCDA problems (Belton and Stewart, 2003; Roy, 1996). Submission (contestant) ranking problem is *the ranking problematique* as the final outcome of the evaluation is a ranked list of contestants based on which the awards will be distributed or team selected.

Based on another type of classification, the submission ranking problem is *a repeated* problem, therefore the focus of this research is on constructing the ranking procedure which could be applied annually in LitIO.

It is a *group decision making* problem, because the role of decision maker is played by the members of the LitIO scientific committee and the opinions of all of those members who are involved in the evaluation of submissions of a particular task has to be taken into account.

3. Decision Matrix

Once the list of alternatives and criteria for an MCDA problem is determined, the following step is to construct a decision matrix. In this section we present the decision matrix constructed both previously described situation, i.e. for the submission ranking problem and for the contestant ranking problem.

Let $A = \{A_1, A_2, \cdots A_n\}$ be a finite set of alternatives (i.e., submissions or contestants), $C = \{C_1, C_2, \cdots C_m\}$ be a finite set of criteria (i.e., evaluation criteria or team selection criteria), and $P = \{P_1, P_2, \cdots, P_q)$ be a finite set of decision makers (i.e., scientific commitee members).

Assume that all the decision makers are involved into defining relative weights and determining performance of each alternative in terms of each criterion.

Let w_j^k be a relative weight of criteria j $(j = 1, 2, \ldots m)$ given by decision maker P_k $(k = 1, 2, \ldots q)$

Let x_{ij}^k be the performance of alternative A_i $(i = 1, 2, \cdots, n)$ in terms of criteria C_j $(j = 1, 2, \cdots, m)$ assigned by decision maker P_k $(k = 1, 2, \ldots q)$.

Note that if the criterion is measured subjectively, i.e. the decision makers assess the performance of an alternative i in terms of criterion j manually, then the value of x_{ij}^k is linguistic. Otherwise the value of x_{ij}^k is numeric and $x_{ij}^1 = x_{ij}^2 = \cdots = x_{ij}^q$.

In the case of team selection problem the value of x_{ij}^k is always numeric.

Then the submission ranking problem can be expressed by the following decision matrix:

$$D^k = (x_{ij}^k)_{n\times m\times q} = \begin{array}{c|cccc} & C_1 & C_2 & \cdots & C_m \\ & w_1^k & w_2^k & \cdots & w_m^k \\ \hline A_1 & x_{11}^k & x_{12}^k & \cdots & x_{1m}^k \\ A_2 & x_{21}^k & x_{22}^k & \cdots & x_{2m}^k \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ A_n & x_{n1}^k & x_{n2}^k & \cdots & x_{nm}^k \end{array} \tag{1}$$

where $k = 1, 2, \ldots q$.

Note, that the classical MCDA algorithms, assume the single decision maker problem, i.e. they assume that $q = 1$.

4. Application of Fuzzy Numbers for Quantifying Linguistic Variables

Some of the proposed evaluation in LitIO criteria are measured manually using linguistic variables. Linguistic variables are variables whose values are linguistic terms and not numbers. They are used to express results of subjective qualitative evaluation. Linguistic variables were introduced and described by (Zadeh, 1975a,b,c). Triangular and trapezoidal fuzzy numbers are used for quantifying linguistic variables.

Next we shortly present the related concepts of fuzzy logic based on (Lee, 2005; Triantaphyllou, 2000).

Fuzzy set. A Fuzzy set is any set that allows its members to have different grades of membership (membership function) in the interval $[0, 1]$, i.e. for any subset $\widetilde{A}$ of a universe X it is possible to define a membership function of a fuzzy set: $\mu_{\widetilde{A}} : X \to [0, 1]$.

A crisp set is a separate case of fuzzy set and to make distinctions between crisp and fuzzy sets we will use $\widetilde{A}$ notation for fuzzy sets.

Operations on Fuzzy sets (Lee, 2005; Triantaphyllou, 2000; Zadeh, 1965):

- Negation: $\mu_{\widetilde{\bar{A}}}(x) = 1 - \mu_{\widetilde{A}}(x), \forall x \in X$.
- Union: $\mu_{\widetilde{A}\cup\widetilde{B}}(x) = Max\ [\mu_{\widetilde{A}}(x),\ \mu_{\widetilde{B}}(x)], \forall x \in X$.
- Intersection: $\mu_{\widetilde{A}\cap\widetilde{B}}(x) = Min\ [\mu_{\widetilde{A}}(x),\ \mu_{\widetilde{B}}(x)], \forall x \in X$.

Fuzzy number. A fuzzy set is called a fuzzy number if the fuzzy set is convex, normalised, its membership function is defined in $\mathbb{R}$ and is piecewise continuous.

Trapezoidal fuzzy number. A trapezoidal fuzzy number is a fuzzy number represented with four points as follows: $\widetilde{A} = (a_1, a_2, a_3, a_4)$ and this representation is interpreted in the following way:

$$\mu_{\widetilde{A}}(x) = \begin{cases} 0, & x < a_1 \\ \frac{x-a_1}{a_2-a_1}, & a_1 \le x \le a_2 \\ 1, & a_2 \le x \le a_3 \\ \frac{a_4-x}{a_4-a_3}, & a_3 \le x \le a_4 \\ 0, & x > a_4 \end{cases} \tag{2}$$

When $a_2 = a_3$, the trapezoidal number coincides with a *triangular fuzzy number*.

Many conversion scales have been created for transforming linguistic terms into fuzzy numbers. (Chen *et al.*, 1992) proposed eight conversion scales with different numbers of linguistic terms which are commonly used. An example pretty standard fuzzy set theory nine-item scale is presented in Table 1 and Fig. 3 (Sule, 2001). The choice of a concrete scale from the available ones is intuitive and left to the responsibility of the decision maker. Note that the same linguistic term in different conversion scales can have different crisp values.

Thus, all the linguistic terms, are converted to fuzzy numbers using the chosen scales, and after this the decision matrix will contain only numeric (crisp or fuzzy) values. In this paper we will not suggest the concrete scales, because the scales are chosen intuitively and we believe that the jury members also have to be involved in the decision. Only after the piloting of the evaluation scheme it might be possible to make a final decision about the scales.

Table 1

Weights of a trapezoidal distribution of a linguistic scale (Sule, 2001)

Item of linguistic scale	Numerical weights
Very poor (VP)	(0, 0, 0, 0.2)
Between poor and very poor (BPV)	(0, 0.2, 0.2, 0.4)
Poor (P)	(0, 0.2, 0.2, 0.4)
Between poor and fair (BPF)	(0, 0.2, 0.5, 0.7)
Fair (F)	(0.3, 0.5, 0.5, 0.7)
Between fair and good (BFG)	(0.3, 0.5, 0.8, 1)
Good (G)	(0.6, 0.8, 0.8, 1)
Between good and very good (BGV)	(0.6, 0.8, 0.8, 1)
Very good (VG)	(0.8, 1, 1, 1)

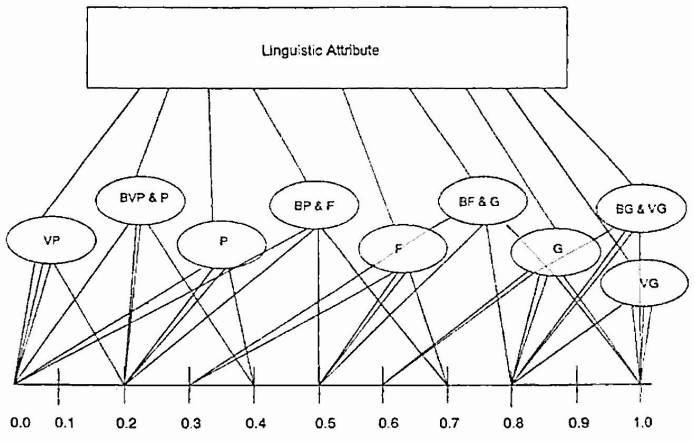


Fig. 3. Trapezoidal fuzzy numbers are used to quantify nine-item linguistic scale (Sule, 2001).

5. Submission Ranking Problem Constraints

The decision context of our problem is rather specific. The problem belongs to the ranking problematique category and is a group decision making problem. Moreover, the chosen method will have to be applied in an educational informatics contest situation. Therefore it is highly important that the approach would be *accepted by the community* of informatics contests. (Belton and Stewart, 2003) emphasize that the ability to explain the chosen to approach to a variety of backgrounds is an important factor in the choice of MCDA approach.

The score aggregation and team selection procedures contains parts which are revealed to the contestants, but it also contains the hidden parts. For example, the scores assigned by individual jury members during manual evaluation are not revealed to the contestants, only the aggregated score is. We emphasise that the parts of the scoring function which are revealed to the contestants must be easily understandable and transparent. More complicated techniques could be applied to the hidden parts.

It must be noted that our problem is a repeated problem. This means that the process of ranking submissions and contestants will have to be repeated each time a LitIO contest session takes place, though on different submissions possibly of different contestants. Therefore it is very important to achieve that the stakeholders would accept and understand the method.

Even though the problem is described as ranking problematique, it is not enough to present ranking to the contestants. The jury (during medal allocation procedure) and the contestants are interested not only in the position in the ranking table, but also in the score differences among a group of contestants.

It is commonly accepted in LitIO that a score aggregation function mapping the performances for separate attributes (groups of criteria) into real numbers is defined and announced to the contestants in advance.

Therefore we will focus on MCDA approaches which foresee defining score aggregation function and partial score functions, inducing a ratio scale, and the ranking is made after the aggregated scores for each alternative have been calculated.

6. Choice of MCDA Approach

Many different MCDA approaches are presented and categorised in (Belton and Stewart, 2003; Carlsson and Fullér, 1996; Chen *et al*., 1992; Kahraman, 2008; Triantaphyllou, 2000). Instead of focusing of separate MCDA methods, we will first look at the major families of MCDA methods. (Belton and Stewart, 2003) distinguish three major families of MCDA approaches:

- **Value measurement theory** (Keeney and Raiffa, 1976). The main idea of this approach is to construct a value function which would associate each alternative with a real number in order to produce ranking of alternatives. The main idea of this

theory correspond the intentions and reasoning for our problem. Therefore we will include it for further consideration.

- **Satisficing (or Goal programming)** (Simon, 1976). This approach instead of creating one value function operates on partial value functions. By partial value function we mean a value function which maps performance of alternatives in terms of a certain criteria to real number. The main idea of the approach is that the most important criterion is identified and the acceptable level of it is determined. Then the alternatives are eliminated until all the remaining alternatives achieve the acceptable level. At this point the second most important alternative together with its satisfactory level is identified. The alternatives which do not reach satisfactory level of the second criteria are eliminated again.

  This approach is not suitable for our problem as it does not assume score aggregation at all.
- **Outranking** (Roy, 1996). Outranking methods also operate with partial value functions and involve pair-wise comparisons of alternatives. An alternative is dominated by another alternative if the other alternative performs better in terms of one or more criteria and equals in the remaining criteria. The concept of *outranking* is introduced.

  The outranking relationship of two alternatives describes that even though the two alternatives do not dominate each other mathematically, the decision maker accepts the risk of regarding one alternative almost surely better than the other.

  We consider this approach also unacceptable in our situation because it again deals with preferences in terms of separate criteria and does not foresee score aggregation using single value function. The concept of outranking, i.e. allowing the decision maker to take the risk of considering one alternative better than the other is not acceptable in a contest community where scoring is a sensitive issue.

Out of three major MCDA families, only one foresees a construction value function, which is required in ranking submissions problem as well. Therefore further we focus on algorithms of value measurement theory.

Besides the main families of MCDA approaches, fuzzy logic is often considered to be applied for MCDA problems. Fuzzy logic is used in group decision making which is our case. However, fuzzy logic is not a separate methodology, but a tool that can be applied within other MCDA approaches including the ones described above. Therefore we assume that fuzzy logic might be applicable in case of this problem and we will look at the concepts of fuzzy logic as well.

7. Choice of Value Measurement Theory Method

Value measurement theory was mainly started by Keeney and Raiffa (Keeney and Raiffa, 1976). More on it can be found in (French, 1988; Roberts, 1979).

The main idea of this theory is that a real number ("value") is associated with each alternative in order to produce ranking of alternatives. *The value function* is defined as a

function assigning a non-negative number to each alternative indicating the desirability (or preference) of the alternative.

The value function has to satisfy the following requirements: an alternative A_{i1} is preferred to alternative A_{i2} $(A_{i1} \succ A_{i2})$ if and only if $V(A_{i1}) > V(A_{i2})$; the alternatives are indifferent $(A_{i1} \succ A_{i2})$ if and only if $V(A_{i1}) = V(A_{i2})$. Note, that the value function must induce complete order.

Value measurement approach introduces partial value functions $v_j(A_i)$. They are constructed for separate criteria and partial value functions hold the essential features (i.e. induces complete order) of value functions in terms of separate criteria.

There were developed several value measurement theory algorithms and the most popular ones are *Weighted Sum Model* and *Weighted Product Model*. We would also assign *Topsis* algorithm (we will present it later) to the same category of algorithms. Note that those algorithms are constructed for single decision maker problems.

Weighted Sum Model (WSM) is the most commonly used method for single decision maker problems (Triantaphyllou, 2000). It can be described using the following formula:

$$V(A_i) = \sum_{j=1}^{m} w_j v_j(A_i) \tag{3}$$

where $i = 1, 2, \cdots n$ and $j = 1, 2, \cdots m$.

One of the reasons of wide acceptance of this model is its simplicity, i.e. it can be easily explained by the decision makers to a variety of backgrounds (Belton and Stewart, 2003).

Note that the requirement *preferential independence* has to be satisfied so that the WSM model could be applied (Belton and Stewart, 2003). Suppose that two alternatives A_{i1} and A_{i2} differ only on a set of criteria $R \subset C$ (R is a proper subset of C) and values of partial functions are equal on all other criteria. Then it is possible to decide the relationship of A_{i1} and A_{i2} (i.e. $A_{i2} \succ A_{i1}$ or $A_{i1} \succ A_{i2}$ or $A_{i1} \sim A_{i2}$) knowing their performances on criteria from $\mathbb{R}$ only, i.e. irrespective of values of their performances on all the other criteria.

However among the submission evaluation criteria there are several dependent criteria, e.g. quality of programming style is related either to the performance of an algorithm-code complex or to its efforts to solve the task (Skūpienė, 2010). Thus the partial independence of criteria is violated.

We suggest that this does not eliminate WSM from applying it for score aggregation in LitIO. WSM still can be applied for aggregating those criteria that are preferentially independent. Special functions have to be introduced for aggregating scores of dependent criteria.

In the team selection problem the partial independence requirement is fully satisfied.

Another requirement is to use the same scale of measurement for all the criteria. Performance of submissions as well as performance of contestants are measured using

different scales. However we intend to unify the scales by constructing the corresponding partial value functions.

WSM can be potentially applied for score aggregation in LitIO, though the above mentioned condition has to be observed.

Weighted Product Model (WPM). WPM can be described using the following formula:

$$V(A_i) = \prod_{j=1}^{m} [v_j(A_i)]^{w_j} \tag{4}$$

where $i = 1, 2, \cdots n$ and $j = 1, 2, \cdots m$.

There have been suggested arguments that preferences are often perceived in ratio scale terms therefore product is more natural than sum (Lootsma, 1997; Triantaphyllou, 2000). The consequence of trade an additive approach into multiplicative approach is that partial value functions have to satisfy ratio scale properties instead of interval scale properties.

Simplicity of the approach is a high priority in the choice of score aggregation algorithm. We conclude that the WSM algorithm would be more suitable than WPM for the submission ranking problem as it is simpler and better understandable to the wide audience. Otherwise they seem to be identical in terms of the problem under consideration.

For the team selection process WPM can be considered as an option, because the overall number of stakeholder is much smaller. Usually there are not more than 10 candidates among which the selection is performed.

Topsis *(Technique for Order Preference by Similarity to Ideal Solution)* (Saghafian and Hejazi, 2005; Triantaphyllou, 2000). We did not find it explicitly stated that Topsis belongs to Value measurement theory approaches, nor to other specific family of MCDA approaches. However, as it involves calculating value of closeness coefficient and ranking based on the values of the coefficient, we suggest that it is appropriate to consider it here.

Topsis introduces concepts of hypothetical solutions, i.e. positive ideal solution and negative ideal solution. The positive ideal solution is calculated as a function from the best performance values of the concrete decision matrix in terms of each criteria: $A^* = \{v_{1*}, v_{2*}, \cdots, v_{m*}\}$, where $v_{j*} = \max\limits_{i}(v_j(A_i))$, $i = 1, 2, \cdots n$.

The negative ideal solution is calculated as a function from the worst performance values in terms of each criteria: $A^- = \{v_{1-}, v_{2-}, \cdots, v_{m-}\}$, where $v_{j-} = \min\limits_{i}(v_j(A_i))$, $i = 1, 2, \cdots n$.

For each alternative the Euclidean distance from ideal positive solution and ideal negative solution is calculated: $S_{i*} = \sqrt{\sum_{j=1}^{m}(v_j(A_i) - v_{j*})}$, and $S_{i-} = \sqrt{\sum_{j=1}^{m}(v_j(A_i) - v_{j-})}$.

Finally, the relative closeness coefficient to the ideal positive solution is calculated: $F_{i*} = \frac{S_{i*}}{S_{i*} + S_{i-}}$.

The alternatives are ranked based on the value of the relative closeness coefficient to the ideal solution of each alternative. This method from mathematical point of view is interesting and appealing however, it gives in to WSM due to the simplicity of the latter.

Moreover, the score of one submission is dependant upon the quality of the submissions. There were cases where such approach was applied in large informatics contests. However, LitIO contestants also compete in small groups and there are cases where just few (i.e. fewer than 10) submissions per task are presented. If the score is dependant upon the submissions in such case, then it might become too biased.

In case of team selection, the number of alternatives is allways low (i.e. 5 to 12), however we might consider this method, because the contestant evaluation scores play different role in team selection than in submission evaluation. In submission evaluation the contestants are interested in the absolute score, i.e. how good the contestant is performing in terms of a concrete task. However in the team selection process the main question that matters is how good the contestant is in comparison to other contestants.

After looking at several value measurement theory associated methods, we came to the conclusion, that as simplicity and the ability of wide audience to accept the evaluation scheme plays significant role in the choice of approaches, the WSM approach suits best to solving evaluation in LitIO problem. Though certain requirements have to be observed. We did not find any evidence that other methods would be more suitable than WSM.

The situation is different with team selection problem. The overall number of stakeholders is much lower (not more than 12 contestants, their teachers and parents), relative ranking is more important than performance in terms of a separate criterion, hidden and revealed parts of the selection process differ form that of submission evaluation. Therefore at this stage all the three methods, i.e. WSM, WPM and Topsis, can be presented for further consideration.

We decided on the score aggregation methods, however they assume a single decision maker and therefore we have to look for the extension to be applicable for group decision problems.

8. Group Decision Making

Group decision making (GDM) can be defined as decision making process based on the opinions of several individuals. The goal of GDM is to arrive at a satisfactory group solution, rather than to the best solution which almost does not exist (Lu *et al*., 2007). There are available various methods for group decision making from mathematical to psychological and social.

Among MCDA approaches explicitly meant for solving group decision making problems there are techniques which foresee negotiation theory, working with group dynamics, etc. References to that can be found in (Carlsson and Fullér, 1996; Lu *et al*., 2007). Those approaches have been experienced in LitIO many times. Investigation of their suitability in the LitIO evaluation problem would require much investigations from other sciences, in particular management and psychology. For example, most

meetings are conducted online (as members of the scientific committee are associated with different universities in different cities and even countries), some members are reluctant to discuss issues on-line, less experienced tend to vote as more experienced members, etc. These aspects should have been investigated if the above mentioned direction was taken.

Our choice is to focus on mathematical group decision making methods which assume eliciting concrete information from decision makers and using it in a mathematical algorithm, but do not require interaction and negotiation between decision makers.

There are different ways to implement group decision making. Many references can be found at (Lu *et al*., 2007; Rao, 2007). Many common GDM methods (e.g. *authority rule*, *majority rule*, *negative minority rule*) are not suitable because they are intended for *the choice problematique* (i.e. determining the best alternative), but not for *the ranking problematique* problems.

(Lu *et al*., 2007) distinguishes three factors which influence GDM:

The weights of the decision makers. Among the decision makers there might be those who play more important roles in the decision making. In this case, the decision makers should be assigned different weights and that should be reflected in the group decision making process.

Weights of criteria. The decision makers may have different views, attitudes, experience and therefore propose different weights to the criteria.

Preferences of decision makers for alternatives. If the performance of an alternative is evaluated subjectively, then different decision makers can have different understanding, different experiences and can evaluate performance of the same alternative in a different way.

It is common in GDM that the weight of a decision maker, the suggested weights for evaluation criteria, and the performances of alternatives suggested by the decision makers are expressed by linguistic terms, since linguistic terms reflect uncertainty, inaccuracy, and fuzziness of the decision makers (Lu *et al*., 2007). We also assume that the information, provided by each decision maker is consistent and non-conflicting.

The linguistic scale presented in presented in Table 1 and Fig. 3 is not suitable for assigning weights to the decision makers or to the criteria. For that purpose we will use the importance scale presented in Table 2 (Lu *et al*., 2007).

Table 2

Weights of a triangular distribution of a linguistic importance scale (Lu *et al*., 2007)

Item of the Importance degrees scale	Numerical weights $\widetilde{A}$
Absolutely unimportant	(0, 0, 1/6)
Unimportant	(0, 1/6, 1/3)
Less important	(1/6, 1/3, 1/2)
Important	(1/3, 1/2, 2/3)
More important	(1/2, 2/3, 5/6)
Strongly important	(2/3, 5/6, 1)
Absolutely important	(5/6, 1, 1)

9. Score Aggregation Method for Submission Ranking and Team Selection

We already concluded that the WSM approach best suits the submission ranking problem. All three analysed methods (WSM, WPM and Topsis) can be considered for team selection. However in this paper we focus on WSM, leaving application of the other two methods for a separate research.

We were looking for an extension of WSM to GDM, such that it would allow fuzzy numbers in the decision marix, but would use crisp number for partial scores for the attributes and for the final ranking, i.e. its public parts would be acceptable by the community of LitIO.

Many fuzzy GDM algorithms (e.g. *an intelligent FMCGDM method* (Lu *et al*., 2007) or the one described in (Sule, 2001)) assume aggregating fuzzy numbers and only then deriving the final ranking. There was performed a systematic and critical study of the existing fuzzy MCDA methods. It arrived at the conclusion, that the majority of currently existing fuzzy MCDA approaches involve complicated calculations, require all the elements of decision matrix to be presented in a fuzzy format (though some of them might be crisp), and are not suitable for solving problems with more than ten alternatives associated with more than ten criteria (Chen *et al*., 1992; Rao, 2007).

The method presented by Chen *et al*., 1992 is considered to be the one which avoids the above mentioned problems (Rao, 2007; Zhang, 2004). It consists of the following phases:

- Linguistic terms (if such are used) are converted to fuzzy numbers.
- Fuzzy numbers are converted into crisp scores.
- Classical MCDA approaches, which assume crisp values, are applied.

Now we have to find a classical GDM method which assumes crisp matrix. Such method is *The group decision support* algorithm suggested by Csáki *et al*., 1995.

Therefore we have to combining the group decision support algorithm with the approach of (Chen *et al*., 1992). Thus we obtain a GDM algorithm suitable to apply in LitIO evaluation and the team selection problem. Further we use the notations introduced in the third section.

The linguistic terms are converted to fuzzy numbers as it was previously described. The crisp score of a fuzzy number $\widetilde{A}$ is calculated in the following way. First there are defined two functions $\mu_{max}(x)$ and $\mu_{min}(x)$:

$$\mu_{max}(x) = \begin{cases} x, & 0 \le x \le 1 \\ 0, & otherwise \end{cases} \tag{5}$$

$$\mu_{min}(x) = \begin{cases} 1-x, & 0 \le x \le 1 \\ 0, & otherwise \end{cases} \tag{6}$$

Then the left and the right scores of $\widetilde{A}$ and the two functions are defined as:

$$\mu_R(\widetilde{A}) = \underset{x}{Sup}[\mu_{\widetilde{A}}(x) \cap \mu_{min}(x)] \tag{7}$$

$$\mu_L(\widetilde{A}) = \underset{x}{Sup}[\mu_{\widetilde{A}}(x) \cap \mu_{max}(x)] \tag{8}$$

Here *Sup* stands for the least upper bound. The total score crisp score of the fuzzy number $\widetilde{A}$ is defined as:

$$\mu_T(\widetilde{A}) = (\mu_L(\widetilde{A}) + 1 - \mu_R(\widetilde{A}))/2 \tag{9}$$

Conversion of a fuzzy number to crisp value is illustrated in Fig. 4.

The values of the nine-item linguistic scale presented in Table 1 converted to crisp values are presented in Table 3. The values of the seven-item linguistic importance scale presented in Table 2 converted to crisp values are presented in Table 4. Note that the same linguistic term in different conversion scales can have different crisp values.

The algorithm for converting fuzzy numbers to crisp values might be hardly understandable to the wide audience, however its application will remain invisible for the contestants. It will only be applied for dealing with group decisions and linguistic evaluation. If a criterion requires manual evaluation, the linguistic scores and the scores of individual jury members are never revealed to the contestants, just the aggregated score

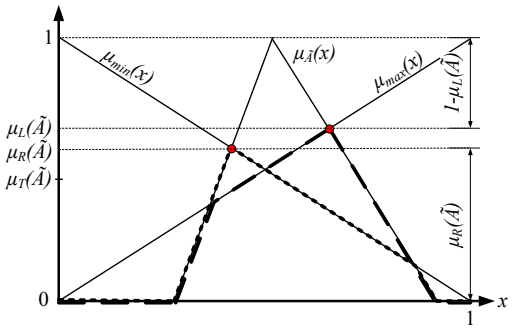


Fig. 4. Conversion of triangular fuzzy number to crisp value.

Table 3

Calculating crisp values of the nine-item linguistic scale given in Table 1

Item of linguistic scale	Fuzzy number $\widetilde{A}$	$\mu_R(\widetilde{A})$	$\mu_L(\widetilde{A})$	$\mu_T(\widetilde{A})$
Very poor	(0, 0, 0, 0.2)	1	0.17	0.08
Between poor and very poor	(0, 0.2, 0.2, 0.4)	0.83	0.33	0.25
Poor	(0, 0.2, 0.2, 0.4)	0.83	0.33	0.25
Between poor and fair	(0, 0.2, 0.5, 0.7)	0.83	0.58	0.38
Fair	(0.3, 0.5, 0.5, 0.7)	0.58	0.58	0.50
Between fair and good	(0.3, 0.5, 0.8, 1)	0.58	0.83	0.63
Good	(0.6, 0.8, 0.8, 1)	0.33	0.83	0.75
Between good and very good	(0.6, 0.8, 0.8, 1)	0.33	0.83	0.75
Very good	(0.8, 1, 1, 1)	0.17	1	0.92

Table 4
Calculating crisp values of the seven-item linguistic importance scale

The Importance Degrees	Fuzzy number $\widetilde{A}$	$\mu_R(\widetilde{A})$	$\mu_L(\widetilde{A})$	$\mu_T(\widetilde{A})$
Absolutely unimportant	(0, 0, 1/6)	1	0.14	0.07
Unimportant	(0, 1/6, 1/3)	0.86	0.29	0.21
Less important	(1/6, 1/3, 1/2)	0.71	0.43	0.36
Important	(1/3, 1/2, 2/3)	0.57	0.57	0.50
More important	(1/2, 2/3, 5/6)	0.43	0.71	0.64
Strongly important	(2/3, 5/6, 1)	0.29	0.86	0.79
Absolutely important	(5/6, 1, 1)	0.14	1	0.93

for the criterion is announced. Thus, if fuzzy techniques are used to aggregate scores of several jury members, they remain behind the curtains and do not become the source of discussions and doubts for the contestants.

The final step is to apply the *group decision support* algorithm (Csáki *et al*., 1995) to the crisp decision matrix.

The application slightly differs for the evaluation and for the team selection problems. We will start from the evaluation problem.

Let $A = \{A_1, A_2, \cdots A_n\}$ be a finite set of alternatives, $C = \{C_1, C_2, \cdots C_m\}$ be a finite set of criteria. Let $D = \{D_1, D_2, \cdots, D_q\}$, $q \geq 2$ be a finite set of decision makers.

Each decision maker is assigned a linguistic weight of his/her importance and the linguistic value has been tranformed to a fuzzy number and to its crisp value: $p = \{p_1, p_2, \cdots, p_q\}$.

Each criterion is assigned a linguistic weight of its importance by each decision maker and transformed to a fuzzy number and then to its crisp value: $w_j = \{w_j^1, w_j^2, \cdots, w_j^q\}$, $(j = 1, 2, \cdots, m)$.

Let $v_j^k(A_i)$ be the values of partial value functions of the performance of alternative A_i in terms of each criterion C_j by the decision maker D_k, where $i = 1, 2, \cdots n$, $j = 1, 2, \cdots m$, and $k = 1, 2, \ldots q$.

First the aggregated group weights for each criterion are calculated:

$$w_j = \frac{\sum_{k=1}^{t} w_j^k p_k}{\sum_{k=1}^{t} p_k}, j = 1, 2, \cdots, m \tag{10}$$

The values of partial value functions of performance of each alternative in terms of each criterion are calculated in a similar way:

$$v_j(A_i) = \frac{\sum_{k=1}^{t} v_j^k(A_i) p_k}{\sum_{k=1}^{t} p_k} \tag{11}$$

The total aggregated values for each alternative are calculated in the following way:

$$v(A_i) = \frac{\sum_{j=1}^{m} v_j(A_i) w_j}{\sum_{j=1}^{m} w_j} \tag{12}$$

Based on the calculated values, the ranking of the alternatives is performed. The above holds for submission evaluation problem.

For the team selection problem the values of the alternatives against each criterion are numeric, pre-calculated and given to the decision makers, i.e. $v_j(A_i) = x_{ij}$. x_{ij} was defined when defining the decision matrix 1). The total aggregated values for the team selection problem are calculated using this formula:

$$v(A_i) = \frac{\sum_{j=1}^{m} x_{ij} w_j}{\sum_{j=1}^{m} w_j} \tag{13}$$

Conclusions

In this paper we proposed to combine the group decision support algorithm combined with score aggregation method to be applied during evaluation and team selection process in the Lithuanian Informatics Olympiad. The method takes into account linguistic values (outcome of manual evaluation) and multiple decision makers (members of the scientific committee).

Even though MCDA theory is acceptable from the scientific point of view, there arise many difficulties with its application in practice because the stakeholders feel reluctant and sensitive about the application of complicated formula to sensitive issues (in this case score aggregation).

The most important requirements to the score aggregation method were the understandability and acceptability of parts of it (i.e. those disclosed to the contestants) to the wider audience. Another important requirement was use of a value function. As a result of these requirements, we spent time on looking for a suitable method that would fulfill all the problem specific requirements, rather than analysing several equally possible options. The paper reveals how we arrived to the suggested score aggregation method for the evaluation problem.

There is much more potential for the MCDA application for team selection problem. One reason is that the number of stakeholder is very small (from 5 to 12 contestants involved) and they are top students with good mathematical and algorithmical skills, which makes it easier to explain for them to accept mathematical decision making methods. Therefore several methods were chosen as possible for consideration for this problem. Before proposing to apply any of those methods in practice, the intermediate step would to model the problem with data from previous years and to analyse differences between the models for the team selection problem.

Even though this paper presented the case of the Lithuanian Informatics Olympiad, the proposed solutions can be considered in other educational contexts as long as similar constraints are valid. The constrains include that the problem under consideration is a ranking, repeated, group decision making problem involving decision makers with a different level of their expertise skills as well as the need to present the decision making process to the stakeholders.

References

Aruldoss, M., Lakshmi, T.M., Venkatesan, V.P. (2013). A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1), 31–43.

Belton, V. Stewart, T.J. (2003). *Multiple Criteria Decision Analysis: An Integrated Approach*. Boston, Kluwer Academic Publishing.

Carlsson, C. Fullér, R. (1996). Fuzzy multiple criteria decision making: recent developments. *Fuzzy Sets and Systems*, 78(2), 139–153.

Chen, S.J., Hwang, C.L., Hwang, F.P. (1992). Fuzzy multiple attribute decision making: methods and applications. In: *Lecture Notes in Economics and Mathematical Systems*, vol. 375. Berlin, Germany, Springer-Verlag.

Csáki, P., Rapcsák, T., Turchányi, P., Vermes, M. (1995). Research and development for group decision aid in Hungary by WINGDSS, a Microsoft Windows based group decision support system. *Decision Support Systems*, 14, 205–221.

Filipe, M., Ferreira, F., Santos, S. (2015). A multiple criteria information system for pedagogical evaluation and professional development of teachers. *Journal of the Operational Research Society*.

French, S. (1988). *Decision Theory: an Introduction to the Mathematics of Rationality*. Chichester, Ellis Horwood.

Ho, W., Dey, P.K., Higson, H.E. (2006). Multiple criteria decision making techniques in higher education. *International Journal of Educational Management*, 20(5), 319–337.

Kahraman, C. (2008). *Fuzzy Multi Criteria Decision Making. Theory and Applications with Recent Developments* (vol. 16 of Optimization and its Applications). Springer.

Keeney, R.L. Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons.

Kurilovas, E. Serikovienė, S. (2010). Learning content and software evaluation and personalisation problems. *Informatics in Education*, 9(1), 91–114.

Kurilovas, E., Vinogradova, I., Serikovienė, S. (2011). Application of multiple criteria decision analysis and optimisation methods in evaluation of quality of learning objects. *International Journal of Online Pedagogy and Course Design*, 1(4), 62–76.

Lee, K.H. (2005). *First Course on Fuzzy Theory and Applications*. Springer.

Lootsma, F.A. (1997). *Fuzzy logic for planning and decision making*. Kluwer.

Lu, J., Zhang, G., Ruan, D. (2007). *Multi-Objective Group Decision Making: Methods*, *Software and Application with Fuzzy Set Techniques* (Series in Electrical and Computer Engineering).

Oberti, P. (2004). Décision publique et recherche procédurale : illustration d'une démarche multicritère à la localisation participative d'un parc éolien en région corse. In: *Actes des Journées de l'Association Française de Science Economique, Economie : aide à la décision publique*. Université de Rennes.
`http://crereg.eco.univ-rennes1.fr/afse/TEXTES-PAR-SESS/A2/OBERTI.P.75.pdf`

Rao, R.V. (2007). *Decision making in the manufactoring environment*. Springer.

Roberts, F.S. (1979). *Measurement Theory with Applications to Decision Making*, *Utility and the Social Sciences*. London, Addison-Wesley.

Roy, B. (1996). *Multicriteria methodology for decision aiding*. Dordrecht, Kluwer Academic Publishers.

Saghafian, S., Hejazi, S.R. (2005). Multi-criteria group decision making using a modified fuzzy *topsis* procedure. In: *Computational Intelligence for Modeling*, *Control and Automation* (IEEE Proceedings, vol. 15).

Scientific Committee of Lithuanian Informatics Olympiads. (2015).

Shee, D., Wang, Y. (2008). Multi-criteria evaluation of the web-based e-learning system: a methodology based on learner satisfaction and its applications. *Computers and Education*, 50(3), 894–905.

Simon, H.A. (1976). *Administrative Behavior*. New York, The Free Press.

Siskosa, Y., Grigoroudisb, E., Krassadakib, E., Matsatsinisb, N. (2007). A multicriteria accreditation system for information technology skills and qualifications. *European Journal of Operational Research*, 182(2), 867–885.

Skūpienė, J. (2010). Improving the evaluation model for the lithuanian informatics olympiads. *Informatics in Education*, 9(1), 141–158.

Sule, D.R. (2001). *Logistics of Facility Location and Allocation*. New York, Basel, Marcel Dekker.

Triantaphyllou, E. (2000). *Multi-Criteria Decision Making Methods: a Comparative Study*. Kluwer Academic Publishers.

Tsinidou, M., Gerogiannis, V., Fitsilis, P. (2010). Evaluation of the factors that determine quality in higher education: an empirical study. *Quality Assurance in Education*, 18(3), 227–244.

Tzenga, G., Chiangb, C., Lia, C. (2007). Evaluating intertwined effects in e-learning programs: a novel hybrid mcdm model based on factor analysis and dematel. *Expert Systems with Applications*, 32(4), 1028–1044.

Value Tree Analysis. (2002). Multiple Criteria Decision Analysis E-learning site created in the EU project OR-World by System Analysis Laboratory of Helsinki University of Technology. `http://www.mcda.hut.fi/value_tree/theory`

Zadeh, L.A. (1965). Fuzzy sets. *Information and Control*.

Zadeh, L.A. (1975a). The concept of a linguistic variable and its application to approximate reasoning – I. *Information Sciences*, 8(3), 199–249.

Zadeh, L.A. (1975b). The concept of a linguistic variable and its application to approximate reasoning – II. *Information Sciences*, 8(43), 301–357.

Zadeh, L.A. (1975c). The concept of a linguistic variable and its application to approximate reasoning – III. *Information Sciences*, 9(1), 43–80.

Zeleny, M. (1982). *Multiple Criteria Decision Making*. New York, McGraw-Hill Book Company.

Zhang, W. (2004). Handover decision using fuzzy MADM in heterogeneous networks. In: *Wireless Communications and Networking Conference*, *WCNC*, *IEEE* (vol. 2). 653–658.

dr. **J. Skūpienė** works at Informatics Methodology Department in Vilnius University Institute of Mathematics & Informatics. She has published about 20 scientific papers. She is a member of the Scientific Committee of National Olympiads in Informatics. Sha ehas been involved in many projects related to identifying, attracting and working with secondary and high school students gifted in IT. For a few years she was director of studies of Young Programmers School, since 2004 she has been a coordinator of Informatics division in the Lithuanian National Academy of Students. She is author/co-author of four books on algorithms and algorithmic problems.

Olympiads in Informatics, 2015, Vol. 9, 193–203
DOI: http://dx.doi.org/10.15388/ioi.2015.15

Ant Colony Optimisation Applied to Non-Slicing Floorplanning

Mirzakhmet SYZDYKOV[1], Madi UZBEKOV[2]

[1] *Kazakh National Technical University named after K.I. Satpayev*
Satpayev Str. 22a, Almaty, Kazakhstan 050013
[2] *Kazakh Economical University named after T.Ryskulov*
Zhandosov Str. 55, Almaty, Kazakhstan 050035
e-mail: rbtinf@gmail.com, *uzbekm7@gmail.com*

Abstract. In this article experimental results are provided for a very-large-scale integration (VLSI) floorplan design problem. Given is a set of modules to be placed non-overlapping on a 2-dimensional rectangular plane. We use ant system simulation as a heuristics to produce feasible layouts in order to minimize the total unused area. The algorithm differs from many others in that fact that it produces non-slicing floorplan. Our experimental results show comparable results of previous methods using ant colony optimization (ACO) in VLSI design. For this purpose we define the "interior" structure for a geometrical computation of module positions.

Keywords: algorithm, ant system, interior, optimization, VLSI, floorplanning.

1. Introduction

At the present time there are several known methods to solve the floor planning problem in VLSI circuit design using ACO heuristics (the main algorithm's described in Dorigo *et al.*, 1996):

1) With a temperature-aware constraint (Luo and Sun, 2007).
2) With a clustering constraint (Chiang, 2009).
3) With a non-overlapping constraint (Alupoaei and Katkoori, 2004).

The last method also provides a solution that removes overlaps of placed modules and reduces the total area and wire length. Our method uses a similarly incremental approach to build constraint graphs and place modules in vertical (to the bottom) or horizontal direction (to the right). This method utilizes the interior structure in order to find a relative placement of the module. In our problem the placement has a single constraint: modules do not overlap each other. We will use a mathematical notation to represent a target function in order to minimize the total unoccupied empty space, which is further defined as a *dead space*.

The optimization methods using ACO heuristics are widely discussed in recent publications:

For circuit partitioning in VLSI design (Arora and Lall, 2013).

For routing optimization with *tabu* search(Yoshikawa and Otani, 2010).

Our method mainly differs in the definition of *visibility* and *distance* functions used in original algorithm (it is better described in Section 3) from the algorithms above which in fact are driven models with modified *core* functions.

1.1. *Problem Definition*

The floor planning problem consists of a set of modules on an integral circuit to be arranged on a planar area in such a way that they will not overlap each other while the occupied areas' measurements, which are given by their formulas, are to be optimized. We solve the problem where the total space unoccupied by the modules is minimized with a non-overlapping constraint by an experimental algorithm. The minimization function is given as a ratio. This can be better defined with an equation:

$$\frac{\sum_{m \in M} Area(m)}{Area(R)} \rightarrow min \tag{1}$$

where M is a set of placed modules and R is a rectangle bounding the placement. The function $Area\ (m)$ is the total area occupied by the module m, whereas the $Area\ (R)$ is the total area of the rectangular board. In our problem the module is given by its bounding box, here it does not actually matter what is the physical shape of the element. We also consider the total area to be the area of the bounding box containing all the placed modules. The minimization of the function (1) is achieved by minimizing the total dead space, which in experimental purposes is measured as a percentage ratio of the part of the bounding box containing all the modules. In can be better represented as:

$$Area\ (R) : R = Bounding\ Box\ (\ Union\ \{each\ m\ in\ M\})\ ), \tag{2}$$

$$Target\ function\ (1) \rightarrow min, iff\ "Dead\ space"\ (\%) \rightarrow min, \tag{3}$$

1.2. *Known Solutions*

There are number of methods to generate a feasible placement for the given set of modules. Most of the methods use specific structures like a B-Tree (Sivaranjani and Kawya, 2013), polish notation or Corner Intersection Sequence (CIS) (Hoo *et al.*, 2013) to internally represent a valid placement. These structures can represent a *slicing floorplan* where the rectangular area of placement can be recursively divided into two parts by a horizontal or vertical line, while each of the modules is within the bounds of the final rectangles produced by an algorithm (Fig. 1).

Our method differs from listed above in fact that it produces non-slicing floorplan. The main difference is also in type of structures used in algorithm. They will be discussed in the next section.

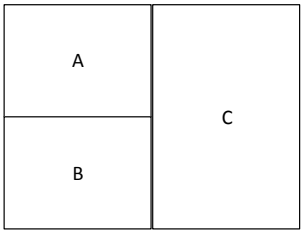


Fig. 1. An example of a slicing floorplan.

2. Data Structures Used in Algorithm

The algorithm uses two types of data structures in order to solve the optimization task – interior and constraint graph. These structures are of planar geometric (interior) and abstract (constraint graph) type. We present simple algorithms to construct them. This generally does not limit the variety of type of data structures which could replace interior and constraint graph for their main purpose to place the module and minimize the value of function (1).

The purpose of interior structure is to store the modules' placement. The non-overlapping condition is to hold true. Using this structure we have to answer queries to find the coordinates of the side projections on vertical or horizontal axis.

The constraint graph structure is to represent the abstract order of module placement relative to the horizontal or vertical axis. The graph is to be acyclic. Using this structure we put all the modules in the placement in abstract topological order. This is necessary to pack the modules after the new module is placed. This structure is to answer the queries to find the placement coordinates of the leftmost bottom corner (x- and y-coordinate for side projections) of the module as if they would be packed without overlapping each other by a physical power vector coming from outermost space on a plane (i.e. the most upper right area).

On the Fig. 2, the packing scheme is presented, the vectors are denoted as P(X) and P(Y), for vertical and horizontal direction respectively.

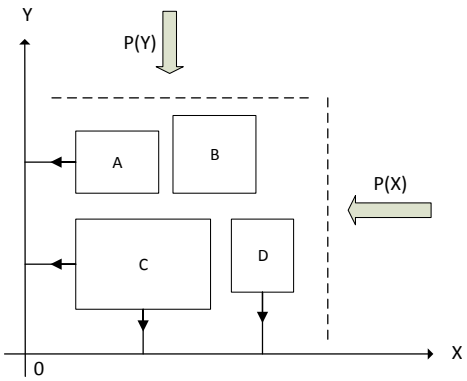


Fig. 2. The packing scheme.

2.1. *Constraint Graph*

A constraint graph is the structure we use to represent the floorplan as an acyclic directed graph. The constraint graph represent the order of modules' placement relatively to the horizontal or vertical axis. Thus the constraint graph can be either horizontal or vertical. This can be better illustrated if we would draw these graphs for a module placement on Fig. 1 (Fig. 3).

Here the "0"-mark stands for the artificial starting element which in fact is a parental node having no incoming edges. Physically on a plane this means that the leftmost or the most bottom modules are to be connected to this parental node as it can be seen on the example diagrams (Fig. 3).

2.2. *Interior Structure*

The interior of the current feasible placement may be described as a set of vertical or horizontal ranges representing the projections of modules taking into account their relative order. To better understand the structure of interior study the example in Fig. 4.

This structure can be effectively used to build constraint graphs or to detect the relative position of the placing module.

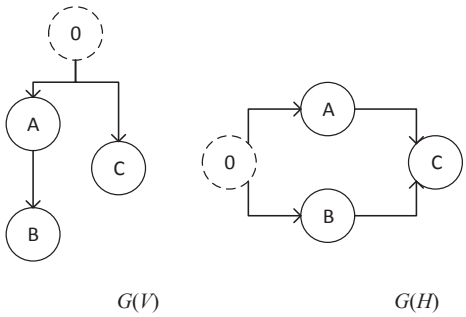


Fig. 3. An example horizontal and vertical constraint graphs.

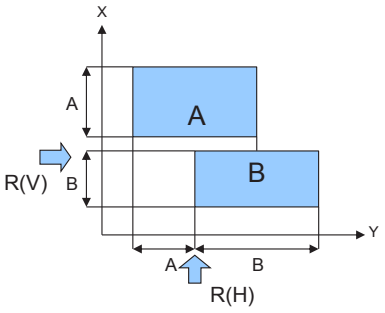


Fig. 4. Example of placement modules and their vertical R(V) and horizontal R(H) interiors.

The interior structure is an ordered list of ranges which may also be used in constraint graph detection. This is mainly because of that fact that each element of this list is a segment on the *X*- or *Y*-plane with *Z*-axis as the other (minor) coordinate. This can best be seen in Fig. 4 in the red line. To build the graph we have to detect if the interior's "red line" intersects the next item which is to the left or at the top according to parameter *Z*. If yes, then there would be a relation between the modules represented by segments in a constraint graph. This relation has direction according to the *Z*-axis. The axis may be either vertical or horizontal according to the constraint graph type. These types split the process of extracting the *x*- and *y*-coordinate for the module.

Mathematical Description of Interior Structure

The interior structure was designed to find a position of a module $m_i \in M / M_{Final}$ to be added to the right or bottom without overlapping an arbitrary element which is already included in the final placement M_{Final}. More precisely, the interior can be viewed as an outermost horizontal or vertical line lying on the edges of modules in placement, viewed from right or bottom side on a plane. Because the algorithm iteratively produces the feasible placement, the interior structure needs to be updated. The interior I can be represented as a set of segments given by a vector of four values:

$$I = \{(L_i, R_i, Z_i, m_i) : I = 1..n, m_i \in M\}, \tag{4}$$

where L_i, R_i are left and right coordinates of the segment on a linear vertical or horizontal axis (this depends on the type of interior which can be either horizontal or vertical),

Z_i is a distance between the segment and parallel axis,

m_i is a module which covers the segment by its right or bottom edge.

Fig. 5, as is, gives an example of this structure on a geometric plane.

Below is an algorithm to update the interior structure according to the new module to be placed.

Please note, Z_i is a pre-determined value which does not change. For the *X*-interior it is obviously a *Y*-value of the bottom corner of the module and vice versa for the *Y*-interior.

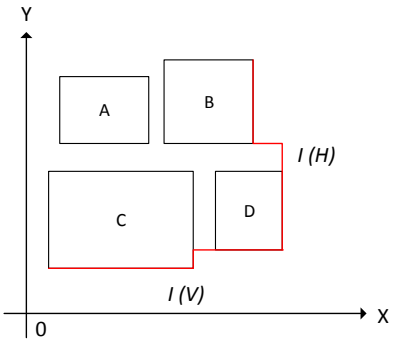


Fig. 5. View of an interior on a geometric plane.

Algorithm 1. Update interior structure.

Initial values: $I = \{\}$.
Input: The new module m_{New}.
Output: Interior I.
for each segment $a \in I$:
  if projection of a intersects edge of m_{New}:
    b = intersection result of a and m_{New};
    if $a.L < b.L$ then $I = I \cup \{(a.L, b.L, a.Z, a.m)\}$;
    if $b.R < a.R$ then $I = I \cup \{(b.R, a.R, a.Z, a.m)\}$;
    $I = I/\{a\}$;
  end if
end for

3. Ant System

In (Dorigo *et al.*, 1996) there is a proposed solution for a *Traveller-Salesman Problem* (TSP) problem using ant agents' simulation. This algorithm uses the measurement functions in order to get the probabilities of transitions between towns given on a planar area:

$tau_{i,j}(t)$ is an *intensity of trail* on edge (i, j) at time t.

The trail intensity is to be updated according to the following formula:

$$tau_{i,j}(t+n) = rho * tau_{i,j}(t) + \Delta tau_{i,j}, \quad (5)$$

where *rho* is a coefficient such that (1 - *rho*) represents the *evaporation* of trail between time t and $t + n$.

$$\Delta tau_{i,j} = SUM \{\Delta tau^{k}_{i,j} \mid k = 1..[Ants]\}, \quad (6)$$

where $\Delta tau^{k}_{i,j}$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on edge (i, j) by the k-th ant between time t and $t + n$; these values is non-zero if ant uses edge (i, j) on his tour and equals value:

$$\Delta tau^{k}_{i,j} = Q / L_k, \quad (7)$$

where Q is a constant and L_k is the tour length of the k-th ant.

The visibility $eta_{i,j}$ is defined as a quantity $1 / d_{i,j}$, where $d_{i,j}$ is a distance between towns i and j.

The transition probability from town i to town j for the k-th ant is defined as:

$$p^{k}_{i,j}(t) = (tau_{i,j}(t) * eta_{i,j}) / SUM \{ tau_{i,k}(t) * eta_{i,k} \mid k \text{ is allowed to be used in a tour}\}, \quad (8)$$

Please note: the functions (5)–(8) are *core* functions required to create a base for simulation model which uses results of computation to make a decision.

Our method differs only in the definition of *visibility* and *distance* functions:

$$Viz\ (a, b) = \text{“Total Module Area”} / \text{“Total Area”}; \tag{9}$$

$$Distance\ (a, b) = Beta / Viz\ (a, b), \tag{10}$$

where the total module area is a cumulative sum of corresponding modules and total area represents the rectangular placement bounding box. The visibility function *Viz* (*a*, *b*) between modules *a* and *b* is a “visibility” used in the TSP algorithm. The distance function *Distance* (*a*, *b*) is a measure of divergence between modules *a* and *b*. This function is also used in this algorithm. *Beta* is defined as “Q” in Dorigo *et al.*, 1996 (equation (3)), in this paper it is equation (7). It is used as a constant of any positive value. In our algorithm it always equals one.

4. Basic Algorithm

The matrix of trained ants’ probability values to search the best placement is represented as a product of dimensions $2N \times 2N \times P$, where P is a set of values – {BOTTOM, RIGHT}. We use this notation in order to include the possible flipped (sides of a module rotated 90 degrees) orientation of the corresponding module. In this case the module index is multiplied by two. The set P represents the possible relative placement of modules. Thus, the pheromone matrix (Dorigo *et al.*, 1996) in our algorithm is a multi-dimensional array. In order to take into account that fact that the new module can be placed to the right or to the bottom, the dimension degree is increased by using a set P accordingly. The dimension of the rectangular matrix is also increased (in fact it is multiplied by two) with respect to that fact that the modules can be rotated. For the indexes 1..2N the following is assumed:

(1) Indexes in form of $2k + 1$ ($2k + 1 <= 2N$) are original modules.
Example: 1, 3, 5, ...
(2) Indexes in form of $2k$ ($2k <= 2N$) are rotated modules.
Example: 2, 4, 6, ...

The algorithm is similar to the ant colony best path search simulation described in (Dorigo *et al.*, 1996). On every step the new module can be placed either to the right or bottom relatively to any module from the set built using ant simulation. The new module is to be placed according to the minimal value of the distance function. I.e., from all the distance values we choose the module corresponding to the minimal value. To solve the problem of placement on the plane the interior structure is used which on every step determines the position of the new module. This can be done in $log\ (N)$ number of operations using a binary search.

When the placement is created, an additional operation is applied. We call it *packing* and it uses the constraint graphs of the placement to rebuild it according to the topological structure of the graph. More precisely, the constraint graph is used to rebuild the placement in order to pack it. This is achieved by computing the values of *X*- and *Y*-coordinates of the modules according to the graph structure (horizontal or vertical). This can be done using a *Breadth-first search* (BFS). Thus the possible residual dead space is excluded from area occupied by the newly placed modules.

The complexity of the solution is $O\ (NC\ N_{Ant}\ N^3\ log\ (N))$, where NC is the number of outer iterations, N_{Ant} – number of artificial ants and N – the number of modules in final placement.

5. Experimental Results

In this section we give the graphical plot of the obtained results using the described method of ant colony optimization of modules to be arranged with no overlaps. The benchmark tests included test cases from *CompaSS* software package (CompaSS, 2004–2005). Below are graphical plots of the algorithm results for the cases *AMI33* and *AMI49* presented on Fig. 6 and Fig. 7 respectively.

The practical observations show that algorithm gives better results if the number of artificial ants and number of outer iterations is increased. This can be better analysed from the results presented in Table 1.

On the diagram below (Fig. 8) the results are visualized for each iteration (one line for each value). The ants count values are on horizontal axis by 5 ants per unit and dead space percentage values are on vertical axis.

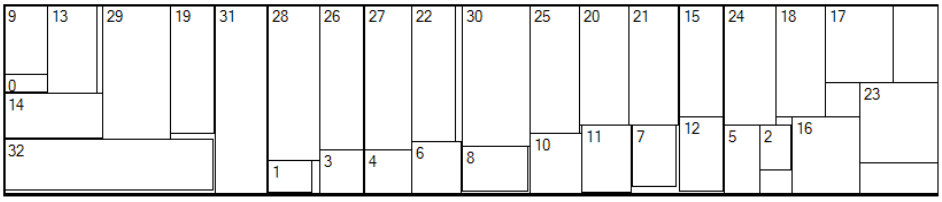


Fig. 6. Physical placement of AMI33, Unused area = 6.888%.

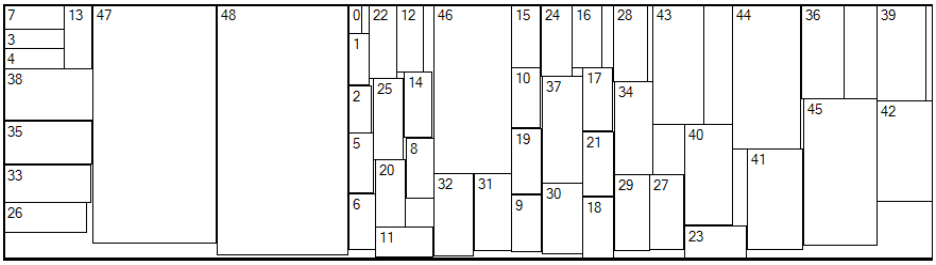


Fig. 7. Physical placement of AMI49, Unused area = 10.621%.

Table 1
Experimental results for AMI33 with varying parameters

Module	Number of iterations	Number of ants	Dead Space (%)	Running time
AMI33	5	5	32,24%	890 ms
		10	12,20%	1 sec. 389 ms
		15	6,89%	1 sec. 988 ms
		20	17,25%	2 sec. 645 ms
		25	6,89%	3 sec. 145 ms
	10	5	28,72%	1 sec. 795 ms
		10	12,20%	2 sec. 948 ms
		15	6,89%	4 sec. 260 ms
		20	6,89%	5 sec. 184 ms
		25	6,89%	6 sec. 392 ms
	15	5	21,00%	2 sec. 473 ms
		10	17,25%	4 sec. 643 ms
		15	6,89%	6 sec. 303 ms
		20		7 sec.
		25	6,89%	9 sec. 811 ms
	20	5	12,20%	3 sec. 567 ms
		10	6,89%	6 sec. 21 ms
		15	12,20%	8 sec. 177 ms
		20	6,89%	10 sec. 831 ms
		25	6,89%	12 sec. 513 ms
	25	5	19,86%	4 sec. 407 ms
		10	13,66%	7 sec. 447 ms
		15	12,20%	9 sec. 682 ms
		20	6,89%	13 sec. 118 ms
		25	6,89%	16 sec. 321 ms

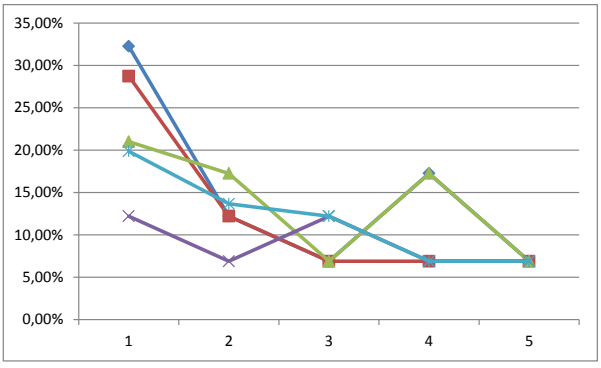


Fig. 8. The visualization of results in Table 1

5.1. *Generalizing Algorithm*

By the generalization of the described algorithm we mean the application of heuristics plan for a large data set (estimated as more than 10K nodes). These data sets were proposed for the contest held at the *International Symposium on Physical Design* (ISPD, 2005). Practically, the algorithm can be maintained effectively even for large data sets if we apply the clustering paradigm. This paradigm includes the following steps to be applied:

1. Select a set of clusters to divide the entire list of modules independently:

$$C\text{: } Union\ \{each\ c\ in\ C\} = C\ \&\ Intersection\ (each\ a\ in\ C, each\ b\ in\ C \mid a\ !=\ b) = \{\}; \tag{11}$$

2. Apply locally the ACO algorithm for each cluster using the set of modules in cluster as an input data:

$$Local\ Placement = Union\ \{ACO\ (each\ c\ in\ C)\}; \tag{12}$$

3. For the list of placements obtained from step 2 create a list of bounding boxes:

$$Global\ List = Union\ \{\text{“}Bounding\ Box\text{”}\ (each\ p\ in\ \text{“}Local\ Placement\text{”})\}; \tag{13}$$

4. Apply the ACO algorithm globally:

$$Global\ Placement = Union\ \{ACO\ (each\ p\ in\ \text{“}Global\ List\text{”})\}; \tag{14}$$

These steps can be applied recursively to large data sets, if we would use the algorithm for the clusters as the input data, which in turn may be a result of ACO algorithm for the other clusters. These clusters at their finite hierarchy are modules representing the input data for the global algorithm.

5.2. *Conclusion and Further Work*

The working algorithm produces better results when the number of ants is increased. The further work includes the study of the application of clustering method to handle large amounts of data. This is not limited to the experiments when the list of constraints is extended as well as the list of semantic rules, for which the placement satisfies (for example, the final placement rectangle’s size and shape constraint).

Acknowledgements

We are glad to mention the editor of this article – prof. V. Dagienė (Vilnius University Institute of Mathematics and Informatics, Lithuania). We are grateful for the review due to which the article describing a novel algorithm became more readable and understandable including all the necessary and important information.

References

Alupoaei, S., Katkoori, S. (2004). Ant colony system application to macrocell overlap removal. *IEEE Transactions on VLSI Systems*, 12.

Arora, M., Lall, G.C. (2013). Circuit partitioning in VLSI design: an ant colony optimization approach. *International Journal of Advances in Engineering & Technology*, 6(1), 536–541.

Chiang C.-W. (2009). Ant colony optimization for VLSI floorplanning with clustering constraints. *Journal of the Chinese Institute of Industrial Engineers*, 26(6), 440–448.

CompaSS: *Compacting Soft and Slicing Packings* (2004–2005). `http://vlsicad.eecs.umich.edu/BK/CompaSS/`

Dorigo, M., Maniezzo, V., Colorni, A. (1996). The ant system: optimization by a colony of cooperating angents. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part B*, 7–8.

Hoo, C.-S., Jeevan, K., Ganapathy, V., Ramiah, H. (2013). Ant system-corner insertion sequence: an efficient VLSI hard module placer. *Advances in Electrical and Computer Engineering*, 13(1).

ISPD: *The International Symposium on Physical Design* (2005). `http://archive.sigda.org/ispd2005/contest.htm`

Luo, R., Sun, P. (2007). A novel ant colony optimization based temperature-aware floorplanning algorithm. In: *Proceedings. Third International Conference on Natural Computation, ICNC* 2007. IEEE, 4, 751–755.

Sivaranjani, P., Kawya, K.K. (2013). Performance analysis of VLSI floor planning using evolutionary algorithm. *International Journal of Computer Applications*, *International Conference on Innovations in Intelligent Instrumentation*, *Optimization and Signal Processing* "*ICIIIOSP-2013*", 9, 42–46. `http://research.ijcaonline.org/iciiioes/number9/iciiioes1662.pdf`

Yoshikawa, M., Otani, K. (2010). Ant colony optimization routing algorithm with tabu search. In*: Proceedings of the International MultiConference of Engineers and Computer Scientists, IMECS 2010, March 17–19, 2010, Hong Kong.* III. 2104–2107.

M. Syzdykov currently works as a consultant in a small firm, graduated from Kazakh National Technical University named after K.I. Satpayev receiving a degree of engineer in systemotechnics. He's a participant of ACM ICPC 2004–2006 (NEERC subregion) and a CBOSS programming contest. He's also a certified specialist in such technologies like Oracle and Informatica Power Center. He has a working experience with data staging process optimization.

M. Uzbekov – currently works as a specialist in IT industry, Kazakhstan. Graduated from Kazakh Economical University named after T. Ryskulov. He's a certified specialist in such technologies like SAP and Informatica Power Center. He has a working experience with enterprise database and ETL systems like TeraData. He's also interested in programming challenges

Olympiads in Informatics, 2015, Vol. 9, 205–212
DOI: http://dx.doi.org/10.15388/ioi.2015.16

REPORTS

Report of the IOI Workshop "Creating an International Informatics Curriculum for Primary and High School Education"

Nevena ACKOVSKA[1], Ágnes ERDŐSNÉ NÉMETH[2],
Emil STANKOV[1], Mile JOVANOV[1]

[1] *Faculty of Computer Science and Engineering, University Ss. Cyril and Methodius st. Rugjer Boshkovikj 16 Skopje, Macedonia*
[2] *ELTE IK, Budapest*
Batthyány Lajos Gimnázium, Nagykanizsa
e-mail: nevena.ackovska@finki.ukim.mk, agi@microprof.hu
emil.stankov@gmail.com, mile.jovanov@gmail.com

Abstract. This report entangles the endeavors undertaken during the IOI Workshop "Creating an International Informatics Curriculum for Primary and High School Education". Considering the need to discuss the role of informatics in the primary and secondary education, the Workshop participants tried to encapsulate several activities that might give insight on how to treat this issue with success. An overview of the current situation with the informatics education in thirteen countries was presented. Further, a group work took place considering relevant topics in creating informatics curricula and computational thinking. A fruitful discussion that considered establishing guidelines and further steps in creating informatics curricula and some ways to promote informatics concluded the Workshop.

Keywords. informatics curriculum, promoting informatics.

1. Introduction

The IOI Workshop "Creating an International Informatics Curriculum for Primary and High School Education" took place from 19–24.04.2015 in Bitola, Macedonia. It gath-

ered 16 people coming from the following countries: Belgium, Brazil, Bulgaria, Croatia, Estonia, Hungary, Lithuania, Macedonia, Serbia and Slovenia. It also included 2 Skype presentations coming from New Zealand and Bolivia, and a submission coming from India. There were three different sets of activities that took place on this Workshop:

1) Country presentations, including tutorials on Tools used in some of the countries.
2) Workgroup activities.
3) Conclusions, results, recommendations and further steps that should be taken.

In the sequel of this document an elaboration on all of the activities is presented. The next chapter gives an overview of the details of the primary and the secondary school curricula in the countries whose representatives presented on this IOI Workshop. The effort done in four different workgroups considering relevant topics in creating informatics curricula and computational thinking is presented in chapter three. The recommendations, conclusions and the possible further steps are given at the end of this report.

2. Participating Countries' Specifics

During the IOI Workshop each country representative presented the specifics of the primary and the secondary school curricula in his own country. Some of the characteristics and best practices that each country presented from their specific experience in informatics education are given in the sequel:

- **Belgium**: Neither Informatics nor ICT is in the official curriculum in Belgium. Some efforts are made through informal teaching: 1) Encourage participation to first stages of contests (IOI, Bebras...) for everyone, making them easier. 2) Organise workshops and fairs to promote informatics at large.
- **Bolivia**: There is not enough staff for teaching Competitive Programming; however there is a community of ICPC contestants that helps the school students in order to prepare for national and international contests. There isn't any standard about CS in schools, but the schools start to use as a standard the Syllabus of IOI Bolivia which is used for the local contests.

  The most known and used path is the syllabus that starts from 10 years old children (or 5th of primary) and is divided in 4 level of contests (Level 0: Operating Systems (Windows and Linux), Level 1: Introduction to Programming with videogames (Scratch, Kodu or other), Level 2: Basic Programming (until Arrays), Level 3: Advanced Programming (Data Structures including Graphs)).
- **Brazil**: In Brazil there is no ICT or Programming in the official curriculum, except at Professional Education Courses. A new program started, but only in a few schools, to include Informatics as "complementary activities" at Primary and Secondary levels, involving only Digital Literacy and ICT.
- **Bulgaria**: There are two curricula in parallel: IT curriculum (Computer system, Organization of data and information carriers, Image processing...) and Informatics curriculum (Math foundation – binary system, propositional logic, formal lan-

guages, computer architecture, operating system, introduction to algorithms and data structures, programming). Most of the regular schools have no teachers in Informatics and do not really teach Informatics.

- **Croatia**: Informatics in primary schools is an elective subject, attended only by a small number of pupils. The teaching plan consists of basics of using the computer, using Office applications and principles of programming. Because of the enthusiasm of the teachers, there are groups where something more on programming can be learned. In secondary schools, informatics is mandatory but in most of the schools only for one year. Only in math-science gymnasiums, the programming and teaching of logical thinking are present through all four years. Preparing for programming competitions is part of additional elective informatics. In the creation of the new curriculum that has started recently, there is a plan to have informatics as mandatory subject in primary schools.
- **Estonia**: Some of the recommendations are: 1) Teachers should have some freedom in the curriculum to have the option to choose what and how they want to teach; 2) Various learning and teaching materials should be created systematically and made publicly available online for free; 3) Constant teacher trainings should be provided to keep teachers aware of the currently most suitable software for these activities and able to search for suitable tools; 4) Informatics should be viewed as an independent scientific subject including elements of programming; 5) Society (pupils, teachers, parents, municipality, stakeholders, researchers, etc.) should be included in raising the awareness of the importance of informatics; 6) Extra curriculum activities and competitions should be supported to increase the motivation of pupils learning IT.
- **Hungary**: 1) It is very late to teach the fundamentals of informatics as a science in secondary school, it is a must to begin it in an early primaries; 2) Informatics means Information and Communication Technology (ICT), Computer Science (CS) and Digital Literacy (DL), altogether.
- **India**: Due to the lack of infrastructural facilities like electricity (43%), broadband connectivity, computers (76%), and qualified teachers, the teaching of computer programming is currently restricted to urban cities and towns of India.
- **Lithuania**: 1) Learning by contests (introducing Bebras tasks); 2) Involving various players in informatics education: pupils, teachers, parents, municipality, stakeholders, researchers…) creating resources for teaching and learning informatics available to everybody (description of methods, exercises, learning objects…).
- **Macedonia**: 1) Introducing programming in primary school as an elective subject in one of the last two years of study; 2) Most of the topics covered in the informatics courses are related to programming; 3) Intention to introduce programming and algorithmic thinking starting from the first grade, and also in the lower grades as a part of the other subjects; 4) Most of the pupils interested in programming are attracted through the competitions.
- **New Zealand**: There is no compulsory Computational Thinking or Computer Science in years 1 to 10. In years 11 to 13 there have been elective standards in Programming and Computer Science, phased in since 2011. So far numbers are

small. Teachers have lacked Ministry funded professional development to learn Programming although there has been support for "the big ideas" of Computer Science through CS4HS workshops and a student and teacher website. School and parental awareness of the need for CT in the curriculum at all year levels is slowly gathering strength.

- **Serbia**: In the elementary school there are elective IT courses in grades 5 and 6 (36hrs), oriented to computer use (including modeling with Scratch), and also in grades 7 and 8 (34hrs). Plus additional hours of ICT are "integrated into other disciplines". In Gymnasiums, informatics is taught with one or two classes per week, which is really low compared to other important courses. The aim of the course is the acquisition of basic computing literacy and training students to use computers in their further education and work. Unfortunately, the emphasis is far from the algorithmic nature.
- **Slovenia**: The informatics curriculum in Slovenia is on a satisfactory level in the first few years (primary school), where students have the option to learn the basics of algorithmic thinking and programming, but it declines rapidly from then on. Currently, informatics is mandatory only in the secondary schools, where only 70 hours are assigned. A big problem is the lack of qualified teachers, which is (probably) due to low salaries and the lack of interest for the profession. "Best" practices: Competitions are excellent for attracting young pupils into the field and they motivate them to refine their skills. Due to the currently minuscule curriculum, the best competitors think they already learned everything and very often decide to continue their university studies in some other field.

3. Work Group Reports

Additional work has been done in 4 work groups. The specifics of the analyses done by every workgroup are given below.

3.1. *Methodology of Creating an International School Curriculum for Informatics and Information Technologies (ISCIIT)*

This group's first objective was to study the available curricula for computer science and informatics technologies for undergraduate students. The recommendations from IEEE and ACM were overviewed, as well as the learning standards recommended by the Computer Science Teachers Association, Computing at School Working Group and the Australian Curriculum. The main goal was to provide a set of recommendations for the creation of programs for informatics in primary and secondary schools.

The Curriculum Guidelines should identify a **body of knowledge**, set of **learning outcomes**, **core** and **curriculum models**. The body of knowledge needs to be organized by knowledge **areas** that are broken down into **units**. Each unit is further subdivided into a set of **topics**. The age when a set of topics should be introduced must be specified.

3.2. *Analysis of the Joint Report from Informatics Europe and ACM Europe*

The report analyzed by this group was developed by a group of experts from academia and industry representing the two principal scientific societies in the field, Informatics Europe and ACM Europe. More resources: (Barr and Stephenson, 2011), (Delors, 1996), (Kingfield, 2012), (Snyder, 1999), (Snyder, 2005).

Based on the analysis of the current situation and of experiences in many countries across Europe, this report makes **four key recommendations**:

- All students should benefit from education in **digital literacy**, starting from an early age and mastering the basic concepts by the age of 12. Digital literacy education should emphasize not only skills, but also the principles and practices of using them effectively and ethically.
- All students should benefit from education in **informatics as an independent scientific subject**, studied both for its intrinsic intellectual and educational value and for its applications to other disciplines.
- A **large-scale teacher training** program should urgently be started. To bootstrap the process in the short term, creative solutions should be developed involving school teachers paired with experts from academia and industry.
- The definition of **informatics curricula** should rely on the considerable body of existing work on the topic and the specific recommendations of the present report.

3.3. *Computational Thinking*

One of the main goals for computer science should be to teach computational thinking, just as the mathematics' main goal is to teach logical thinking and to increase problem solving skills. This should be achieved by incorporating the concepts of computational thinking into most/all courses as the implementation of these concepts is interdisciplinary. The process can benefit from computer scientists, who can promote understanding of how to bring computational processes to bear on problems in other fields. Also this can help students understand processes as algorithmic. In addition to incorporating these skills formally in the classroom, research regarding the implementation of computational thinking skills in informal education also provides valuable insights.

However, embedding computational thinking in primary and secondary education requires a practical approach, for example:

- What would computational thinking look like in the classroom?
- What are the skills that students would demonstrate?
- What would a teacher need in order to put computational thinking into practice?
- What are teachers already doing that could be modified and extended?

More resources: (Barr and Stephenson, 2011), (Brennan and Resnick, 2012), (Lee *et al*., 2011), (Mannila *et al*., 2014), (Wing, 2006), (Wing, 2011).

3.4. *Preparing a Template for Gathering Data for a Catalogue of Experiences*

The purpose of this sub-group was to provide a process and a framework for producing a "Catalogue of Existing Experiences" in Informatics and Computer Science at Schools, in different countries.

Some documents have been reviewed, such as (Guerra *et al*., 2012).

After some discussion it was decided that the best way to produce such a catalogue was to use the IOI community as an input, preparing a questionnaire to be sent to IOI leaders and deputy leaders. It was decided that the questionnaire should be divided into two sections: one with more general information (for example, about the organization of education and of schools), and one with more specific information (for example, which topics are taught at each grade).

The agreement has been made to prepare a questionnaire which should be disseminated to 5–10 valuable teachers in IOI participating countries. It is expected that this questionnaire will gather valuable data that will be used as a basis for building future informatics curricula.

4. Conclusions and Recommendations

The activities that took place in the IOI Workshop "Creating an International Informatics Curriculum for Primary and High School Education" lead to several conclusions and recommendations. They are sublimated below:

1. Primary and secondary levels of education (ages 6 through 19) need to incorporate informatics. As the experience in many European countries has shown, pupils by the age of 12 can be educated, and the education must cover the technical usage of IT tools as well as the rules on how to use them safely, effectively and ethically.
2. Informatics should become a mandatory subject in schools.
3. Combining formal and informal education should be supported.
4. Various methods of teaching and learning informatics fundamentals should be used at primary school: tools that use visual block based programming, learning through games, game creation, robotics, CS unplugged etc.
5. For advanced informatics curricula, modular design is highly recommended.
6. Informatics ought to be thought by teachers trained in informatics.
7. Constant teacher training should be carried out due to the constant changes in the field of informatics.
8. Country specifics should be considered when developing informatics curriculum.

Acknowledgement

The authors would like to express their sincere gratitude to all the participants of the IOI Workshop 2015, for the hard work done and the efforts made in order to define

sound directions and construct a proposal that the IOI community could use for the purpose of establishment of an international informatics curriculum for primary and secondary school education. Here the full list of participants is presented: Sébastien Combéfis – Belgium, Willmar Pimentel – Bolivia, Ricardo Anido – Brazil, Krassimir Manev – Bulgaria, Krešimir Malnar – Croatia, Maria Gaiduk and Tauno Palts – Estonia, Ágnes Erdősné Németh – Hungary, Narayen Ugar – India, Valentina Dagienė – Lithuania, Mile Jovanov, Emil Stankov, Marija Mihova, Nevena Ackovska and Bojan Kostadinov – Macedonia, Margot Phillipps – New Zeland, Jelena Hadzi-Puric – Serbia, and Darko Pevec – Slovenia.

References

Barr, V., Stephenson, C. (2011). Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54. http://dl.acm.org/citation.cfm?id=1929905

Bebras – International Contest on Informatics and Computer Fluency (2007–2015). http://bebras.org

Brennan, K., Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In: *Proceedings of the 2012 annual meeting of the American Educational Research Association*. Vancouver, Canada. http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

CECE – Committee on European Computing Education (2013–2015). http://www.uni-muenster.de/CECE/

Delors, J. (1996). *Learning: the Treasure Within, Report to UNESCO of the International Commission on Education for the Twenty-first Century*. UNESCO publishing. http://www.unesco.org/delors/

Exploring Computational Thinking. https://www.google.com/edu/resources/programs/exploring-computational-thinking/

Guerra, V., Kuhnt, B., Blöchliger, I. (2012). Informatics at School – Worldwide: An International Exploratory Study about Informatics as a Subject at Different School Levels. http://fit-in-it.ch/sites/default/files/small_box/study_informatics_at_school_-_worldwide.pdf

Informatics Education: Europe Cannot Afford to Miss the Boat (2013). Report of the joint Informatics Europe & ACM Europe Working Group on Informatics Education. http://europe.acm.org/iereport/ACMandIEreport.pdf

ISTE Computational Thinking for All. http://www.iste.org/learn/computational-thinking

Kingfield, N. (2012). Fostering tech talent in schools. *New York Times*, 30 September 2012. http://nyti.ms/Sudld4

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2(1), 32–37. http://dl.acm.org/citation.cfm?id=1929902

Mannila, L., Dagienė, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A. (2014). Computational thinking in K-9 education. In: *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR '14)*. 1–29. http://dl.acm.org/citation.cfm?id=2713610

Snyder, L. (1999). *Being Fluent with Information Technology*. Washington, D.C., The National Academies Press (NAP) http://www.nap.edu/openbook.php?record_id=6482&page=1

Snyder, L. (2005). Bringing fluency with information technology to high schools. *CSTA Voice*, 1(3). http://bit.ly/cCPUFR

Royal Society (2012). *Shut Down or Restart? The Way Forward for Computing in UK Schools*. January 2012. http://bit.ly/zDqu7F

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf

Wing, J.M. (2011). *Computational Thinking: What and Why*. http://www.cs.cmu.edu/link/researchnotebook-computational-thinking-what-and-why

N. Ackovska holds B.Sc. in Computer Engineering, Informatics and Automation (2000), M.Sc. in Intelligent Systems (2003), and Ph.D. in the field of Intelligent Systems (2008) from "Sts. Cyril and Methodius University" in Skopje, Macedonia. She is Associate Professor at the Faculty of Computer Science and Engineering at UKIM. She is author of five books (in Macedonian) and more than 60 research articles in the field of Intelligent Systems and hardware education. Her research interest are Beings, both living and artificial. Dr. Ackovska is a member of the Computer Society of Macedonia. She has actively participated in the organization and realization of the Macedonian national competitions and Olympiads in informatics since 2001.

Á. Erdősné Németh teaches mathematics and informatics at Batthyány Lajos High School in Nagykanizsa. A lot of her students are in the final rounds of the national informatics tournaments. She is a Ph.D. student in the Doctoral School of Faculty of Informatics at the University of Eötvös Loránd in Budapest. Her current research interest is teaching computer science for talented pupils.

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 a Ph.D. student at the Faculty of Computer Science and Engineering. His research includes analysis of program code correctness using different techniques, and its application to e-learning.

M. Jovanov is an assistant professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. As the President of the Computer Society of Macedonia, he 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 at International Olympiads in Informatics since 2006. His research interests include development of new algorithms, future web, and e-education, and he has authored more than 40 research peer reviewed papers.

Olympiads in Informatics, 2015, Vol. 9, 213–224
DOI: http://dx.doi.org/10.15388/ioi.2015.17

Selecting and Training Students with No Suitable Informatics Background for Informatics Olympiads – The Case of Syrian Olympiad in Informatics

Ammar ALNAHHAS[1], Emad ALAZAB[2]
[1] *Damascus University, Syria*
[2] *National Commission for the Syrian Science Olympiad, Syria*
e-mail: eng.a.alnahhas@gmail.com, emadalazab@gmail.com

Abstract. Many countries suffer from a bad informatics syllabus in their educational systems, which leaves the students with no skills in computer programming or algorithms. This fact imposes a big challenge to the process of selecting the best secondary school students for the informatics Olympiad; it also makes training them a harder process.

In this paper, we present our vision to tackle this issue in Syria, where the educational system lacks the principles of computer algorithms; moreover, the educational system tends to target student skills that do not include creativity nor innovation. We show the process of gradual selection of students along with the scientific materials of contests in each step, we present the training process in all stages as well. As the process evolved in the last ten years, the statistics of the enrolled students as well as the practical results were provided at last.

Keywords: informatics olympiad, Syria, Syrian olympiad in informatics, programming contest.

1. Introduction

It is now 11 years since the first Syrian Olympiad in Informatics was held. Syria started participating in the International Olympiad in informatics in 2004, it has not been easy for us in Syria to organize a local competition. It was still very difficult for us to train students that are going to participate in IOI; this is mainly because of many reasons, which vary from social, educational and logistical ones.

Syrian universities had no computer science major before 2001, moreover the school syllabus had no real informatics content before 2005; hence the informatics education was almost absent from the society. However, the informatics education has been improved significantly in the last decade; we have been suffering from many problems:

- School informatics syllabus focuses on computer usage principles, such as using the operating system, the internet and some famous applications; it has no serious content related to computer programming and algorithms. The informatics subject

does not exist in the curriculum of general secondary education certificate, which dissuades students from learning informatics.
- Informatics schoolteachers are not qualified, professional teachers tend to not to teach at schools because of bad syllabus content, therefore anyone with an ICDL certificate is allowed to teach informatics in schools.
- General school educational concepts tend to make students memorize rather than think about problem solving, this makes students much more far from learning computer algorithms and programming and deepens the gap between students and informatics.

Because of the problems mentioned earlier selecting and training students for informatics Olympiad is a very tough mission, as it is hard to find suitable students, it is hard to convince them to participate, and it is hard to train them in a manner that they can challenge international students.

In the last ten years, we have tried to find solutions to these problems, by inventing selection and training models. At first we tried a model based on the concept of "Training first", while starting from 2012 we started a new model based on "selection first". In this paper we discuss the details of these two models, the first model is viewed in section 2, the second model is presented in section 3, section 4 shows the comparison between the two models, we conclude the paper in section 5.

2. "Training First" Model

When the first Syrian national Olympiad in informatics was held in 2004, it was very tough to find students that can be trained and selected for the IOI. So two strategies were introduced at that time: the first strategy was a "long term training" which is based on training young children so that they can be qualified to learn computer programming and algorithms after years of training, the second strategy is to select students that should be trained and prepared to participate in IOI in the same year.

To achieve the two mentioned strategies, students are classified into divisions according to their ages; the final distribution of divisions was to position students whose age is below 12 in the first division, students with age below 15 in the second division and students of the secondary school who were younger than 20 are considered in the third group. The training materials of the first and second groups were chosen to improve the mental capabilities of students, the content was about logic games, mathematics and simple algorithms, later "Scratch" programming language was used to train students in these two groups, details of using Scratch was published earlier (IDLBI, 2009).

The students of the third division were chosen and trained to participate in IOI, it is obvious that students of the second division were being moved automatically when they became older to be members of the third division so they can participate in IOI. But here we faced a problem that the number of students in the second division who are eligible to move to the third division was very low, which makes it hard to create a challenging environment for students, and the candidates of Syrian national team was very limited. Therefore, we had to start a new selection and training process with three stages:

selection, training and qualification, whereas another process of preparing students for national competition was held, the winners of the national competitions are prepared to participate in the next year's IOI.

2.1. *Selecting Students*

The first stage of selecting students is held in each Syrian province separately, each province has a separate team who is responsible for selecting and training their students.

The selection process is based on IQ tests, because at that point it is impossible to test students' abilities in computer programming and algorithms, since they lack the minimum amount of knowledge to test their abilities. The aim of these IQ tests is to find the best students that are suitable to be taught IOI materials, we believed that we should look for students that had talent, creativity, intelligence and a quick-wit.

The first problem we faced here is the general society view of the informatics Olympiad, the school teachers have no background nor are interested to know, that is because of the large gap between the school informatics definition and the one intended in IOI. The solution was to hold the selective IQ tests centrally by the organizers of the national informatics Olympiad.

The first problem causes another problem of organizing a selection test that should accept a huge number of students, we insisted on making this selection on site test so that we can ensure the up most integrity.

This selection process was just elementary we wanted to choose about approximately 1000 students that would be trained and then the last teams would be selected from this student group.

2.2. *Training Stage*

This is the toughest part; we had to teach students the principles of computer programming and algorithms in about eight months.

As we had selected students with high IQ we expected that they were able to learn algorithms and programming easier and faster than usual people.

The selected students had to attend two concurrent courses: one for computer programming and one for computer algorithms. The programming course was based on learning "Scratch" programming language which was a good starting point into programming, the rest of the course focuses on real programming languages concepts. We noticed that there was a general acceptance of learning programming; the students who were not able to adapt to the materials left the course, so it was a process of natural selection along with the students' education. The algorithm course focuses on the principles of simple computer algorithms, we could not present advanced algorithms concepts as the experience in computer algorithms can be gained by practicing and solving problems rather than just getting theoretical knowledge, which will not be useful until it is applied in a real experience.

The training process of this stage was not that useful because of the following reasons:

- The training was held in 14 provinces of Syria where each province training team is responsible for training their own students, the training teams were not fully qualified, and differ in experience.
- The number of students was very large for the trainer to follow up with all of them, so that the theoretical knowledge was not backed by any practical training, this caused a loss of the efforts of trainers.

We tried to solve the first problem by conducting a training camp for trainers, the goal of these camps was to try to improve the trainers' experience and direct their efforts in the correct direction, these camps proved their effectiveness for first and second division group. However, the third group trainers were not benefiting through them, as gaining the experience to train computer algorithms needs a long time practicing which cannot be gained in short term camps.

The second problem was not solvable in this training model and was one of the reasons to move to a new model, which is going to be presented later in this paper.

At the end of the training two groups of students were chosen for each province, the first group is chosen to proceed to the IOI team selection, where the other group is chosen to participate in the national Olympiad as the team of this province.

2.3. *Selection*

The IOI team selection was based on series of contests, the selection is based on the average of these contests, the students allowed to participate in these contests are the groups of students selected by each province training team, along with the winners of the past national Olympiad, the team members of the last years are also allowed to participate.

A special training program was prepared for the selected team, training camps were also held to support students in the team.

2.4. *This Model in Brief*

The model we had used for five year (from 2006 to 2011) was based on selecting an initial group of possible suitable students, training all of them for the whole year then choosing the teams. The team members still participated in the selection competitions for the next years if they still meet the age of IOI. The winners of the national competition were also allowed to participate in the next year's selection competitions. Fig. 1 visualize this process.

This model has many drawbacks; the selection process does not guarantee attracting suitable students. The training process does not cover the most important practical part, and the most important is that the whole system does not create the motive for the

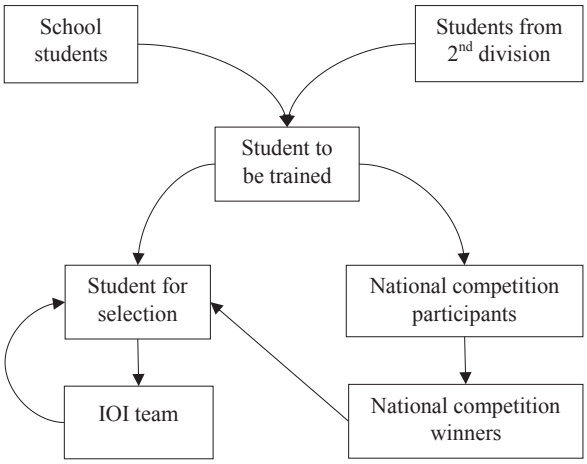


Fig. 1. the "training first" model.

students to start training by themselves, trying to improve their own problem solving skills, which needs a long time of practice, and is important for any novice programmer (Lahtinen *et al.*, 2005).

3. "Selection First" Model

This new model was adopted in 2012, the main goal of this model is to select students first then concentrate on training the selected small group of students, we thought that enhancing the selection process and making it more effective would help us increase the ratio of suitable students as a result of the selection.

We had to build the selection process to avoid selecting students according to their school marks, which is not suitable for evaluation in this case. The student's experience in computer programming is useless for the evaluation as well, as they have no previous knowledge because of syllabus issues as mentioned in the introduction. Moreover, the only way to reach the maximum number of students is to reach their schools, unfortunately, we cannot rely on schoolteachers to help us choose students as teachers are not qualified.

We decided to start from schools, we had to reach the maximum number of students and we cannot conduct an online contest, because the students will have no motive to participate and we wanted to build a general society acceptance of the Olympiad as a good activity for students. Besides, the integrity of online contests is not guaranteed and proctoring about 5000 students is not possible at all.

Each school selects its team, then teams from the same area participate in a competition to select the area team, the area teams participate in a competition to select the province team and the teams of provinces participate in the final national competition.

3.1. *Teachers' Olympiad*

As mentioned earlier the starting point is the schools, and as the teachers of school are in charge of selecting students in this important stage, we had to qualify and train teachers so they can help us choose the most suitable students.

The teachers' Olympiad is a competition that informatics schoolteachers can participate in. The completion tasks are similar to those presented to students during selection process. The goal of this completion is to achieve the following:

- Spread the informatics Olympiad among teachers, school administrators and the whole society.
- Let the teachers be introduced to what informatics Olympiad is about.
- Get good statistics about teachers' qualifications and find the good teachers who can be trained and prepared to be a trainer.

Teachers of informatics in Syrian schools are not specialists; they have a good knowledge in computer general information, computer usage such as using the operating system and applications, some teachers know the principles of computer programming but they constitute no more than 5% of total number of teachers. Hence, we faced a new challenge, which is inventing tasks that can be solvable by teachers as well as targeting the goal that the competition is all about.

The tasks used in the teachers' competition are based on computer algorithms without programming, that is, the solution of the questions is based on finding the correct algorithm without the need to program it, these types of tasks are used later in the student selection process, but using them here has some advantages:

- The type of tasks is algorithmic, that reflects the correct image of informatics Olympiad.
- These tasks can help discover teachers with good ability to be prepared to train students later as it reflects their creativity, innovation and intelligence.

The competition had a good impact on participants, the type of tasks had a general acceptance among teachers, and as we observed it almost achieved the goal it was made for.

3.2. *Student Selection – School Phase*

To reach almost all secondary school students in the country, the selection process starts from the school itself. As there is still no general overview of informatics Olympiad in the society, starting from schools creates a good challenging atmosphere that help motivating students to participate.

Each school chooses its selection criteria, which depends usually on a quick IQ and math tests, the team of each school consists of five students that should participate in the next selection phase.

3.3. Student Selection – Area Phase

The students of schools at the same area competes in the same site, at this moment we have students with no programming or algorithmic background, but they are chosen as the best of their colleagues. The target of this selection phase is to choose students that are eligible to go on, we believe that students with creativity, innovation and high ability to understand simple tasks and find a solution to sophisticated riddles are best suitable for being trained on computer programming and algorithms. IQ tests are a good measure for intelligence scores are closely associated with creativity scores for secondary school students (Kim, 2005) . So the competition in this phase consists of a number of IQ questions, some of which are shaped like a real life problem that the student should investigate, analyze and find a good method to reach the solution.

Both students and teachers generally accepted this type of questions, it seemed familiar to the students, as it is not related to any school syllabus content, it also encouraged the qualified students to practice preparing for the next selection phase. Five students are chosen from each area to participate in the next phase.

3.4. *Student Selection – Province Phase*

In this selection phase, students still have no programming skills, but they are talented. We have to prepare tasks that are both solvable by students and are related to IOI content.

The solution was to prepare tasks that have an algorithmic nature, but students should solve them by hand, this can attract students that know nothing about programming (Marcin KUBICA, 2010) especially innovative students with good thinking abilities.

The tasks of this phase are chosen to meet the following requirements:

- It has the same programming problem structure (input, output).
- It requires some algorithm to be invented for the given test case be solvable by feasible time when solved by hand.
- It is not multiple-choice task; students should find the output.

The task statement is similar to programming tasks, the student is given one or two inputs, and he/she should find a good algorithm to get to the correct output, Table 1 shows some task samples

The tasks achieved many benefits to the students such as:

- Make the students familiar with programming problems.
- Teach students that finding the correct result as well as the algorithm is important.
- Select students that have talent and have a good algorithmic thinking and good problem solving abilities.

In addition to the mentioned type of tasks, another type is used to introduce students to algorithm analysis, this type of tasks presents an algorithm to students then asks some questions that encourages them to understand and analyze the algorithm. This type of

Table 1

Sample tasks for province selection phase

Given a sequence of letters (ABACBCDBCD): • Find the number of increasing subsequences. • Find the longest increasing subsequence.
We have towers of coins where the number of coins in each tower is given: 6 10 4 2 3 Find the minimum number of coins that should be moved to make the number of coins equal in all towers.
We want to buy 1000 liter of milk, there are 10 salesmen, and each has a quantity of milk and has a specific price, find the minimum money to be paid to buy the desired amount of milk.

tasks helps selecting students with good abilities in self-learning and problem solving; Table 2 shows a sample of this task.

Although the competition is held in all province centers at the same time; where each student compete in the province center he lives in, the result of all students is merged, then the best sixty students are chosen to move on to the national competition.

3.5. *National Competition*

The qualified students to the finals are trained for programming for the first time after the province selection phase. The students showed good and fast training skills, most of them accepted programming concepts easily, and they learned C++ programming language principles and simple algorithm concepts in a small period.

Table 2

Sample of second type province selection tasks

Given the following grid of number where all rows and columns are sorted:

				0
		0	2	3
	2	4	5	6
1	3	6	7	0

To search for a number in this grid we start from the square in the lower-left corner, if the number we are searching for is greater than the number in this square, we remove the first column of the grid, otherwise we remove the last row of grid, and we do the same method with the remaining grid.

- How many comparison operations are needed to find number 10?
- How many comparison operations are needed to make sure the number does not exists in the worst case?
- Answer the last question with another grid with n rows and m columns.

The students attend an intensive training program in the period between province selection competition and national finals; training includes conducting lectures in training centers, online lectures, recorded sessions. Students are encouraged to learn by themselves, they are provided by books, introduced to online resources and online training websites. The usage of social networks to establish communication between students and trainers proved a good efficiency in improving student skills as they are motivated by teamwork and challenging other students, feeling in direct and continuous contact with trainer pushes them to train more and get help when needed. In usual people are interested in learning using social networking (Acharya *et al.*, 2013).

The national competition is an IOI-like competition with algorithms and computer programming tasks, the system used in Syrian national Olympiad since 2012 is the well-known CMS system (Maggiolo and Mascellani, 2012).

The best ten students wins the competition and join the so-called national team; which is the group of the winners of national competitions in last few years.

3.6. *IOI Team Selection*

The new students in the national team start training after the national competition, where old students have been training since they had joined the national team. All the national team students participate in a selection competition. The competition tasks are tough and are selected to be similar to IOI tasks level, the four winners of this competition are chosen to be the IOI team of Syria.

The IOI team selection is the last phase of the selection process of this model, Fig. 2 visualize the whole process.

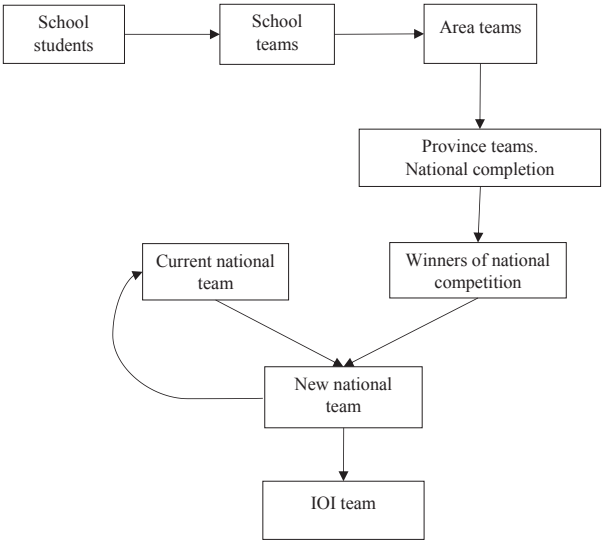


Fig. 2. The “Selection first” model.

3.7. *Training*

Training is the most important part of the whole process, to achieve good results selected students should be trained properly.

The training is important in three periods:

Between province selection competition and national competition: in this phase the training starts by presenting basic programming and algorithms as described earlier in this paper.

Between the national competition and the IOI selection competition: In this period the training is based mainly on online lectures and online competitions, students are encouraged to solve as many problems as possible, they are followed up daily and directed to any resources and materials needed.

After the IOI team selection: The training continues for all students in the national team, the training is continuous and is based on the following:

- Online lectures that are provided via a special e-learning server.
- Training camps are conducted two times in the summer, where students meets for a month to training.
- Online competitions: Online competition has proved a high efficiency in training students, an online contest is prepared almost every week, which has a good acceptance among students, where they find it a good place to show their evolution and improvements. It provided a good challenging environment that motivate students to make advantages to beat each other; it also provides a good tool to introduce new ideas to students by presenting it inside problems. Moreover, when using problems from old competitions student can estimate their levels internationally and make more improvements. Our observations conforms to researches shows that online programming contests can be used to build programming trainings (Combéfis and Wautelet, 2014)

4. Comparing "Training First" Model to "Selection First Model"

The first model is based on training a large number of students, then selecting best students after training, this strategy did not suites the case of Syrian Olympiad in informatics because of many reasons including:

- Failing to attract many talented students, thus losing them in Olympiad.
- Large number of students makes training need a lot of stuff, resources and time, lacking professional people and needed resources makes this model not suitable.
- Training students with different abilities with the same training material is not suitable, good students may not benefit at all, where bad students fail in the selection.

This model was used for six years, the average participating students count at the first initial selection is 500, which is relatively a small number, the average students participating in the final selection count is about 20, the average non-zero marks rate in

the last selection is 75%, and the average non-zero marks rate in the national competition is 70%.

The second model is used since 2012; it is based on choosing best students gradually using suitable selection criteria, the selected students are then trained and prepared for IOI participation.

The process of this model has proved good advantages, start selecting students from schools makes it possible to target almost all secondary school students, spreading out the culture of Olympiad among society. Providing students with algorithmic and programming content gradually has proved good results as well, focusing efforts to train small group of good students and improving the training process has a good impact on students in general.

Table 3 shows statistics of participating students since 2012, we consider non-zero mark in the final competition as a criterion to measure the success of student selection and training in the small period between province selection contest and final competition, the average non-zero marks count is 80%.

The national team students prove good abilities in learning new concepts, training and competing. In the last four years, they started to have good ranks in famous online contests and training platforms, like Codeforces.

5. Conclusion

In this paper, we viewed the Syrian Olympiad in informatics experience in the last 11 years, we talked about the difficulties we faced, including poor informatics teaching and lack of professional trainers, and we presented two different models that were used in Syria. The first model depends on training a large number of students in order to select teams from them; this model proved that it is not that suitable. The second model is based on choosing good students according to a special criteria, then training them.

The feedback of the whole process is analyzed each year, and many improvements are planned to be added, merging the bright side of the first model with the second model is going to help improving our strategy in the next years, which will improve the student training process and make national informatics Olympiad a more successful story that will help improving the society.

Table 3
Student participation statistics

	2012	2013	2014
Schools	7550	7020	7980
Areas	2334	2940	3250
Provinces	356	542	620
Finals	62	51	72

References

Acharya, V., Patel, A., Jethava, S. (2013). A survey on social networking to enhance the teaching and learning process. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6).

Combéfis, S., Wautelet, J. (2014). Programming trainings and informatics teaching through online contests. *Olympiads in Informatics*, 8, 21–34.

Idlbi, A. (2009). Taking kids into programming (contests) with. *Olympiads in Informatics*, 3, 17–25.

Kim, K. H. (2005). Can only intelligent piople be creative? The Jouranl Of Secondary Gifted Education, XVI, 57–66.

Lahtinen, E., Ala-Mutka, K., Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In: *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education.* New York, ACM SIGCSE Bulletin. 14–18

Maggiolo, S., Mascellani, G. (2012). Introducing CMS: a contest management system. *Olympiads in Informatics*, 6, 86–99.

Marcin KUBICA, J. R. (2010). Algorithms without programming. *Olympiads in Informatics*, 4, 52–66.

A. Alnahhas – Holds M.Sc. in Computer science from Damascus University and is preparing for PhD, he was a former IOI contestant and has been involved in coordinating Syrian Olympiad in Informatics and training students since 2005, he has many contribution for the national Olympiad including preparing tasks and development grading systems.

E. Alazab – The head of the National commission for the Syrian Science Olympiad.

Olympiads in Informatics, 2015, Vol. 9, 225–232
DOI: http://dx.doi.org/10.15388/ioi.2015.18

Informatics Olympiads in Turkey: Team Selection and Training

Tolga CAN[1], İ. Onur SIĞIRCI[2], Osman ABUL[3], M. Fatih DEMİRCİ[3]

[1] *Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*
[2] *Department of Computer Engineering, Yıldız Technical University, Istanbul, Turkey*
[3] *Department of Computer Engineering, TOBB University of Economics and Technology, Ankara, Turkey*
e-mail: tcan@metu.edu.tr, ionur@yildiz.edu.tr, {osmanabul,mfdemirci}@etu.edu.tr

Abstract. In this country report, we present the yearly activities related to Turkish team selection and training for national and international olympiads in informatics. First, we outline the organizational structure and describe the scientific and administrative components. We, then, describe the several stages of team selection, which start with a nation-wide written test administered in selected cities in Turkey. Each stage is followed by two week long training camps. In these camps, students are taught the IOI curriculum and they engage in programming competition practices. In this report, we give the details of curriculum organization and test preparation. We conclude with medal statistics covering the last 22 years of the Turkish IOI team.

Keywords: IOI Country Report, Turkey.

1. Introduction

Turkey has been actively participating in the International Olympiads in Informatics (IOI) since the IOI'93 in Argentina. The continued support of the government funding agency TÜBİTAK, the Scientific and Technological Research Council of Turkey[1], the universities, and the IOI alumni help ensure that IOI remains a tradition among algorithms and programming enthusiasts in high schools. The first curriculum, training materials, and team selection tasks were developed by a team of faculty members lead by professors Göktürk Üçoluk, İsmail Hakkı Toroslu and Faruk Polat from the Department of Computer Engineering, Middle East Technical Univerity (METU).[2] Today, the preparation, team selection, and organization for IOI participation of Turkey are conducted via a well-established work-flow (Fig. 1).

[1] http://www.tubitak.gov.tr/en
[2] http://www.ceng.metu.edu.tr

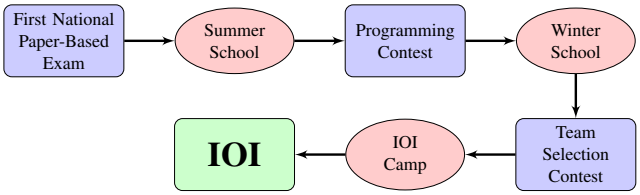


Fig. 1. Overview of the team selection and IOI organization in Turkey.

TÜBİTAK is a critical stakeholder, which provides the finances and maintains the organizational structure of the olympiads in Turkey, not only in Informatics, but also in other fields, such as Mathematics, Chemistry, Physics, and Biology. An official scientific executive committee, consisting of three academics from universities, is formed by TÜBİTAK each year. This committee is reponsible for all the scientific matters from task preparation and training to team selection. The scientific committee works in collaboration with TÜBİTAK for administration of a nation-wide first-stage selection exam, for organization of training camps, and for all administrative tasks regarding participation in that year's IOI. The scientific executive committee also recruits faculty members, interested graduate students, and alumni from previous years to teach the IOI curriculum (IOIS, 2013) effectively and help students practice for the competition during the training camps.

Being organized by an offical body, the participants of the National Olympiads in Turkey and the member of the Turkish IOI Team are also recognized officially by the Ministry of Education and the Higher Education Council of Turkey. Successful students are awarded additional points at the nation-wide university entrance exam and medal winners are granted acceptance to any Computer Science/Engineering Department of their choice at a state university in Turkey.

In the following sections of this national report, we give details of the entire workflow of Informatics Olympiads organization and preparation in Turkey. We also give the medal statistics covering the last 22 years of the Turkish IOI team. Finally, we conclude with a brief summary and an outlook for the future IOIs.

2. Team Selection and Training Camps

The national paper-based exam is held to select students that will participate in the summer school. Once the programming contest is done after the summer school, students with high scores are invited to the winter school, after which the final team selection is done. The IOI camp is held one week before the actual IOI to get the team prepared for IOI . TÜBİTAK organizes each stage (Fig. 1).

The team is selected after 3 contests and 2 training camps. After the final training camp, the students attend the final team selection contest. In this section, information will be given about contests and training camps.

2.1. *The First Stage Exam*

The first stage exam is a written exam in which nearly 1300 students from all over Turkey participate. It is paper-based, takes 2.5 hours, and consists of 50 multiple choice questions (QFSE). The exam is generally held in May.

The point distribution of the questions in the exam is as follows: 15% intelligence skills, 30% basic algorithm knowledge, 25% maths and 30% C programming language.

A sample programming question and its solution is given below:

Code

```
int ffffff(int x, int y) {
    int z;
    while (y!=0) {
        z=y;
        y=x%y;
        x=z;
    }
return x;
}

void main() {
    int a=360, b=75, c=15;
    b = a*b/ffffff(a, b);
    c = b*c/ffffff(b, c);
    printf("%d", c);
}
```

Question

What would be the output of the code?

A) 25
B) 225
C) 1125
D) 1800
E) None

Answer

`ffffff` function is to calculate the greatest common divisor of x and y. $LCM(a,b) = \frac{a \times b}{GCD(a,b)}$, so the 13^{th} line indicates that $LCM(360,75) = 1800$ and the 14^{th} line indicates that $LCM(1800,15) = 1800$. In conclusion, the correct answer is D.

After the first stage exam, the top 55 students qualify for the second stage. In genereal, higher participation from the more populated cities such as İstanbul, Ankara, and İzmir is observed. In Fig. 2, the location of these cities on the geographical map of Turkey is shown. In addition to high level participation, these cities are also the most successful ones. Since 1993, 88 students from Turkey have participated in IOIs. 85 of these students have been from these three cities.

2.2. *The Summer Training Camp*

Approximately, 55 top students from the first stage exam attend the scientific camp that lasts for two weeks, beginning in late August and ending in early September. The students and lecturers are accomodated at a five-star hotel in Afyon (as shown in Fig. 2).

The lecturers in this scientific camp are usually academics from the most respected universities in Turkey. Additionally, graduates who have participated in the past IOIs in recent years also help the new students in the training camps.

The curriculum of the camp is in parallel with the training curriculum adopted in other countries such as Russia (Kiryukhin, 2007) and Serbia (Ilic and Ilic, 2012) and includes all the content of the second exam, which is listed below. Note that, due to the relatively heavier schedule of the first training camp, the subject of Graph Theory is divided among

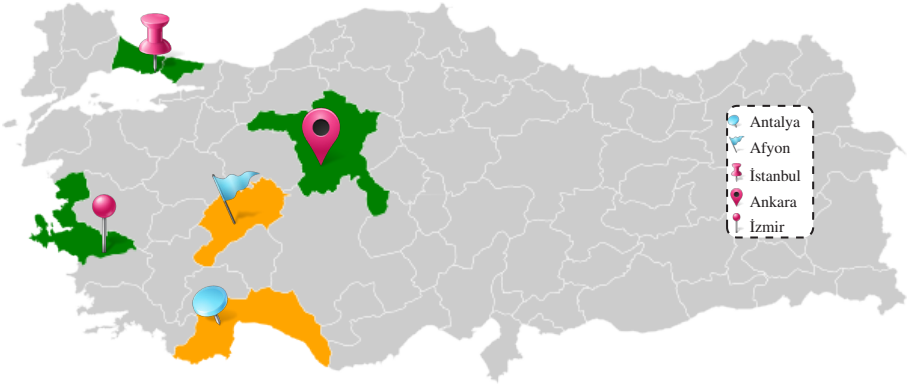


Fig. 2. Although the first stage exam is nation wide, a higher participation in this exam is observed from more populated cities such as Ankara, İstanbul, and İzmir. The summer and winter camps are held in Afyon and Antalya, respectively. These cities are colored in the map.

the two training camps and some basic graph algorithms such as "finding the connected components in graphs" are left for the second training camp (see Section 2.4).

- **Basic Data Structures**: Linked lists, stacks, queues, trees, binary trees, heaps.
- **Sorting and Searching**: Bubble sort, insertion sort, selection sort, quick sort, merge sort, heap sort, counting sort, radix sort, sequential search, binary search, basic space search, hashing.
- **Standard Template Library** (STL).
- **Dynamic Programming**: Longest common substring, knapsack problem, matrix chain product, longest increasing subsequence, Kadane's algorithm, greedy algorithms.
- **Graphs and Basic Graph Algorithms**: Graph traversal, Dijkstra's shortest path, Floyd-Warshall algorithm, Prim's and Kruskal's algorithms for minimum spanning tree, topological sort.
- **Maths and Number Theory**: Modular arithmetic, GCD-LCM, Euler-Fermat-Euclid.

Theorems, primarity test, integer factorization, Chinese remainder theorem, fast exponential, matrix exponential, Fibonacci numbers.

2.3. *The Second Stage Exam*

The second stage exam is prepared according to IOI standards and taken by the 55 students who participated in the summer training camp. The exam is performed in two days of November and its questions are developed according to the contents of the summer school.

In each of the exam days, five hours are given to solve three tasks. The top 18 successful students are awarded national medals. The distribution of the medals is 3 gold, 6 silver and 9 bronze.

In addition to these medals, TÜBİTAK awards students with monetary values of approximately $1500, $1300 and $1100 for each gold, silver and bronze medal respectively.

More than this one-time monetary award, each medal winner is offered monthly scholarship, enough to cover basic living costs, extending to the end of PhD study for the recepient.

Besides monetary/scholarhip supports, students are entitled to additional points at the national university entrance examination.

2.4. *The Winter Training Camp*

Approximately 18 students who have passed the second stage exam attend the winter training camp that lasts two weeks in February. This time, the students and lecturers stay at a five-star hotel in Antalya (as shown in Fig. 2).

In contrast to the summer training camp, the remaining subjects of the IOI curriculum are taught in the winter training camp. Due to the fewer number of students attending the camp, the lecturers have a chance to spend more time with the students. The curriculum of the camp covers:

- **Advanced Data Structures**: Range minimum query, segment tree, binary indexed tree, lowest common ancestor.
- **Analytic and Computational Geometry**: Vectors, line, segment, circle, polygons, convex hull, plane sweep, Voronoi diagrams.
- **String search**: Trie, suffix array, suffix tree, Aho-Corasick algorithm, Knuth-Morris- Pratt algorithm.
- **Graph Algorithms**: Bellman-Ford shortest path, longest path, connected components, articulation points, bridge edges, Eulerian path, Hamilton cycle.
- **Game Theory**: Nim game, Grundy numbers, game on graphs.

2.5. *The Team Selection Exam*

Students who have attended the winter camp take another two day long exam, which includes the content of all of the IOI curriculum, in April. Similar to the second stage exam, five hours are given to solve three tasks in each of the exam days. The best four successful students form the final team that represents Turkey in both IOI and BOI (Balkan Olympiad in Informatics).

2.6. *IOI Camp*

The national IOI team prepares for the IOI for two weeks just before the actual IOI. This preperation is usually held in Ankara, the capital of Turkey (see Fig. 2). Since

2011, the Computer Engineering Department of TOBB University of Economics and Technology[3], located in Ankara, has hosted IOI camps as well as second stage and team selection examinations. Students take exams in each day of the camp and the solutions of the questions are discussed in detail with the students in a post-exam problem solving session.

A student who wins a medal in the IOI is given an opportunity to enter a computer engineering department of any state university, regardles of his/her achievement in the national university entrance examination. In additon, some private universities offer considerable scholarships for medal-winner students. TÜBİTAK offers one-time monetary awards to IOI medal winners: approximately $8000 for gold, $6000 for silver and $4000 for bronze.

Starting in 2011, we have developed `www.ubilo.org` (TOITS), written from scratch by a few Turkish IOI alumni, to serve as the national online judge during the three camps. Like Codeforces (www.codeforces.com), it has two main interfaces: the training interface and the contest interface.With the former, the students are assigned online tasks and their submissions are graded. The latter is more like the CMS system employed in IOI exams.

The system's language, from user interfaces to task descriptions, is in Turkish since many high-schools in the country offer only Turkish curriculum.

3. Results and Statistics for the Last Ten Years

Table 1 shows the medal distribution of the Turkish IOI team since the first participation in 1993 in Argentina. The first gold medal was achieved by Barı¸s Kaya in 2011 in Thailand and the second gold medal was achieved two years ago in IOI Australia by Yusuf Hakan Kalaycı. The Turkish IOI team was able to get at least one medal in all of the 22 years of attendance since 1993.

4. Conclusions and Outlook

Turkey has been actively participating in IOI since 1993 and a total of 55 medals have been won by the Turkish IOI teams so far.With the help of the government funding agency TÜBİTAK, the universities, and the IOI alumni, being selected as a national IOI team member attracts many high school students. Earning additional points for the national university entrance examination and even obtaining a chance to the exam-free entrance to the computer engineering department at any state university along with financial aids are the top motivations for the high school students for participating in the IOI activities in Turkey. While the medal distribution in each year varies, the gold medals were achieved within the last 4 years, showing a growing interest in IOI among Turkish students.

[3] `http://www.etu.edu.tr`

Table 1

Medal distribution of the Turkish IOI team since 1993

Year	IOI Host	Medals			
		Gold	Silver	Bronze	Total
1993	Argentina			3	3
1994	Sweden			2	2
1995	Netherlands			1	1
1996	Hungary			2	2
1997	South Africa		1	2	3
1998	Portugal		2	1	3
1999	Turkey		1	1	2
2000	China		1	1	2
2001	Finland			3	3
2002	South Korea		2	2	4
2003	USA		1	1	2
2004	Greece		1	1	2
2005	Poland			1	1
2006	Mexico			1	1
2007	Croatia		1	1	2
2008	Egypt			2	2
2009	Bulgaria		1	3	4
2010	Canada		1	3	4
2011	Thailand	1	1	2	4
2012	Italy		1		1
2013	Australia	1	1	1	3
2014	Taiwan		2	2	4
Total		**2**	**17**	**36**	**55**

References

IOI. *International Olympiad in Informatics*. http://www.ioinformatics.org

IOIS (2013). *The International Olympiad in Informatics Syllabus*. http://people.ksp.sk/~misof/ioi-syllabus/ioi-syllabus.pdf

QFSE. *Questions of First Stage Exam (in Turkish)*. http://www.tubitak.gov.tr/tr/olimpiyatlar/ulusal-bilim-olimpiyatlari/icerik-bilgisayar

Kiryukhin, V. M. (2007). The modern contents of the Russian national olympiads in informatics. *Olympiads in Informatics*, 1, 90–104.

Ilic, A., Ilic, A (2012). IOI training and Serbian competitions in informatics. *Olympiads in Informatics*, 6, 158–169.

TOITS. *Turkish Olympiads in Informatics Training System*. http://kamp.ubilo.org

T. Can received his BSc degree in computer engineering from Middle East Technical University, Turkey, in 1998, and his PhD degree in computer science from the University of California at Santa Barbara in 2004. He is currently an associate professor in the Computer Engineering Department, Middle East Technical University, Turkey. His main research interests are bioinformatics, graph theory, and algorithms. He has worked on protein structure visualization and alignment and on construction and analysis of large-scale protein protein interaction networks. He has been teaching computational geometry in the training camps for Informatics Olympiads since 2007 and he has served on the scientific executive committee of the Informatics Olympiads in Turkey since 2011.

İ.O. Sığırcı is a PhD student at the Computer Engineering at the Yıldız Technical University in Turkey. He is a teaching and research assistant at the Department of Computer Engineering. His main research interests are computer vision, machine learning and algorithms. He was the technical committee member at the Balkan Olympiad in Informatics 2014 in Turkey.

O. Abul received his PhD in computer science from Middle East Technical University, Turkey, in 2005. He is currently an associate professor of computer science at TOBB University of Economics and Technology, Turkey. His research interests include artificial intelligence, databases, data mining and privacy. He has been the chair of the scientific executive committee of the Informatics Olympiads in Turkey and has lead the Turkish IOI team since 2011.

M.F. Demirci received his PhD in computer science from Drexel University in 2005. After working as a postdoctoral research scientist at Utrecht University, The Netherlands, he joined TOBB University of Economics and Technology, Turkey as a faculy member at computer engineering department. His research interests include structural pattern recognition in computer vision, shape indexing, and applied graph theory. He was the recipient of the best PhD dissertation award in engineering and physical sciences, Drexel University in 2006 and TUBITAK Career project award in 2010. He has been a member on the scientific executive committee of the Informatics Olympiads in Turkey since 2011.

Olympiads in Informatics, 2015, Vol. 9, 233–242
DOI: http://dx.doi.org/10.15388/ioi.2015.19

Technical Report of Baltic Olympiad Informatics 2014

Motiejus JAKŠTYS
Technical Committee, National Olympiad Informatics, Lithuania
e-mail: *motiejus@jakstys.lt*

Abstract. In this article we present the technical and organizational details of Baltic Olympiad Informatics (BOI[1]) 2014 held in Palanga, Lithuania.
BOI has been held since 1995. During that years, we have accumulated a lot of experience and polished the process. The article presents some of the organizational and technical issues and how we dealt with them. In conclusions, we highlight some recommendations for preparation for the next years. Other countries organizing national or international Olympiads might benefit from the recommendations and experiences.

Keywords: informatics olympiads, linux, desktop preparation.

1. Introduction

Baltic Olympiad Informatics (BOI) is an annual informatics competition by high school students started in 1995 by Lithuania, Latvia and Estonia. Later, more countries joined BOI: Finland, Sweden, Norway, Denmark, Germany and Poland. The goal of the Olympiad is to bring gifted students to international environment as a preparation for IOI.

The key part of the Olympiad is preparing high-quality tasks, tests and checkers. The evaluation system must be robust, the evaluation criteria must be clear. These responsibilities are called ”Scientific part”, and have a Scientific Committee dedicated to them.

Besides Scientific committee, there are two supporting committees: Organizational and Technical. Organizational part is something that is generic for every Olympiad, be it for informatics, physics, mathematics or of the Old Prussian Literature. Responsibilities

[1] BOI is also used as abbreviation for the Balkan Olympiad in Informatics. Within this article, BOI refers only to the Baltic Olympiad.

include finding the acommodation venue, organizing arrivals and departures, meals and leisure activities.

Technical committee is responsible for evaluation servers, student workstations, public internet, private network, student workstation network configuration, and technical support of the Organizational Committee.

In this article we will briefly remind the history of Baltic Olympiad Informatics (section 2). After historical introduction, we will present work of Organizational Commitee (section 3) and of Technical Committee (section 4) which was carried out in BOI2014. The article describes some challenges we faced in BOI2014, and how each of the challenge was handled. The cases will be followed by related work, concluded by suggestions and recommendations for the following events.

2. Short History

Baltic Olympiad Informatics increased from three countries in 1995 to nine countries in 2014.

Since 1996, each participating country arrives with 6 students and 2 deputy leaders. On rare occasions, a country might be permitted to arrive with more than 6 students; a situation where 7 students are needed might happen when the performance between 6'th and 7'th in the national contest is very similar.

Up to this day, the style of BOI is kept similar to IOI. We will further elaborate on the similarities and differences in organization between the two. Detailed history of Baltic Olympiad Informatics are covered in (Bulotaitė *et al.*, 1997, Poranen *et al.*, 2009).

3. Organizing BOI2014

The very first work organizing the Olympiad is carried out by Organizational Committee. The dates have to be agreed, venues picked, countries invited any many other things. We will cover them in this section.

3.1. *Schedule*

Compared to IOI, BOI schedule is more compressed and fully fits to four-five days.

First day: arrivals, opening ceremony. At night, first set of tasks is selected and translated.

Second day: competition day 1. At night, second set of tasks is selected and translated.

Third day: competition day 2.

Fourth day: closing ceremony, party.

Fifth day: departures.

Short duration brings in many advantages. First, the organizers, which are usually university students or full-time employees, need to take less days off from their primary activities. Secondly, it is cheaper for the organizers: less nights to pay for accommodation means more money can be spent on other things like prizes and leisure activities.

Why can BOI allow short duration, but not IOI? That is mainly possible because of close distance: all delegations are able to arrive in a short time (usually, all delegations arrive in the allocated half-day period). What is more, there are no major time zone differences which would require long adaptations.

3.2. *The Venue*

In BOI2014, participating countries take turns when hosting the Olympiad. BOI2013 was hosted in Rostock, Germany. BOI2014 was Lithuania's turn.

During National finals, Lithuania is hosting each Olympiad in a different location every year. Olympiads are usually hosted at schools in different towns through out the country. Given a similar number of participants and team leaders to the national contest (50 contestants and 20 organizers in National Olympiad compared to 60 contestants 40 team leaders and organizers in BOI), it was decided to use the same style for BOI2014: host the Olympiad in an interesting venue (e.g. health resort, school in an interesting town), rather than a technically convenient one (e.g. University campus).

After evaluating possible venue options, a health resort in a sea resort Palanga was chosen. The health resort could comfortably accommodate the staff, students and guests. For leisure activities, besides being close to the sea, surroundings of Palanga are rich with notable natural points of interest and historical landmarks.

Organizing the Olympiad in a health resort gave advantages for organizational matters (accommodation, meal, activities), but raised technical challenges. Computer and power networks had to be created from scratch. Computers needed to be rented before the Olympiad and fully set-up with tables, chairs and cabling.

In the following section, we will discuss the technical setup in the health resort.

4. Technical Infrastructure

Like mentioned in section 3.2, the technical infrastructure needed to be created from scratch. At the venue, the following were readily available:

- Uplink internet connection (DSL).
- Power sockets.
- Large hall.

Technical and Organizational Committees needed to rent tables, chairs and computers, set them all up, create and connect to a power grid.

4.1. *Physical Infrastructure*

All student workstations were in a large hall. See Fig. 1 for schematics of the layout. Red lines and circles show power grids. Black crossed circles are WiFi access points. White squares are desks with computers.

Since all the infrastructure needed to be created from scratch, we tried to make it as simple as possible. For example, to avoid cutting and pulling many Ethernet cables to set up wired networking, we opted for wireless. This could have been done because the workstations were laptop computers, and thus all had ability for wireless networking.

During planning stages, there were concerns about interference of many devices using wireless network in such a small area. To avoid that, we created 3 non-overlapping channels (1, 6, 11) and configured two access points per channel. The workstations were

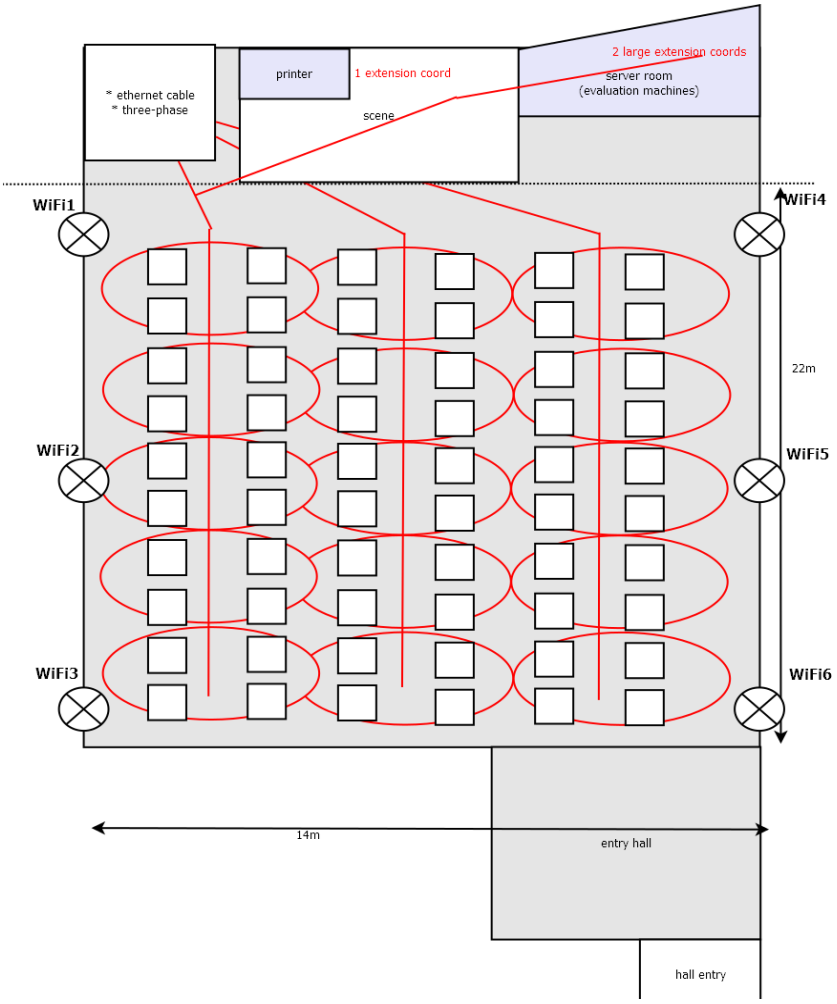


Fig. 1. Hall schematics.

configured to pick all access points at random (the same priority in `wpa_supplicant` configuration file). Networking was used only for accessing the Contest Management System's web interface, thus low traffic, and worked perfectly.

According to Power Supply Unit specifications, a laptop can draw up to 200 W of power. 60 laptops can draw up to 12 kW. Including network equipment, contest management system and evaluation servers, this figure goes up to 15 kW. A single commodity power socket can provide up to 8 kW. To accommodate our needs, we were provided with a three-phase commodity power network, each phase capable delivering 8 kW each, giving 24 kW of total capacity.

4.2. *Student Workstations*

Student workstations are providing students a comfortable environment to develop the contest tasks. Workstation software has to be recent and comfortable. It also should restrict contestants from cheating, e.g. sharing solutions with each other.

4.2.1. *Software*

Until 2001, DOS programming environment was used in the IOI contest. That changed in 2005 when it was replaced to Linux (Jyrki Nummenmaa, 2005). Like in IOI, in BOI we permit and support only Linux operating system. As a task to Technical Committee, an Operating System image with necessary configuration and software needs to be created.

When thinking what to base the Linux distribution for BOI2014 on, we had the following requirements in mind:

- Should have all the necessary packages included (compilers, IDEs, documentation, ...).
- Contestants should be able to test the environment easily, so it needs to be easy to run it outside of the Olympiad. Ideally, a downloadable ISO file which can be booted from a CD or USB memory stick.
- The distribution might be reused for other contest (or) in the following years. Since the requirements in the following contests might change, we want to make the distribution configuration and creation as automatic as possible.

Because of the following requirements, we chose Debian-Live. Debian-Live is a framework for creating Debian-based bootable system images. It works as follows: given some configuration (e.g. list of packages to be installed, users, passwords and other settings), the Debian-Live machinery creates a bootable ISO image. The bootable ISO image is a live image which can be booted and is perfectly suitable for playing around.

However, the system on the live image does not have a permanent file system. This is a problem during the contest, because, in case of accidental reboot, the contents of user's home directory will be lost, including all the produced solution files. To avoid that, the live ISO image is installed to a hard drive, which permits permanent storage, and the hard-drive image is created for usage during the contest. Hard-drive based installation from the generated ISO file is very similar to the live ISO image, just a few extra hard-drive specific tweaks are necessary.

After some preparation by Technical Committee, now ISO generation process is single-click[2]. Normally, installing the ISO to a (virtual) hard drive automatically is a bit more involved because, by default, the Debian Installer asks questions. Luckily, these questions can be pre-answered in a *preseed* file, and the installation to the (virtual) hard-drive can be one-click too[3].

Like mentioned before, a few tweaks are still needed to bring use the installed hard-drive image in the Olympiad. First, security: there are many things we want to restrict users to do:

- Connect to unauthorized networks.
- Change network configuration.
- Start and stop services.
- Use external media.

To support the restrictions above, we created scripts which modify the resulting hard-drive image to accommodate these requirements.

4.2.2. *Booting the Machines*

We had two major challenges for imaging and booting the machines: we had no wired connectivity to transfer the images, and we were not permitted to alter the on-disk Windows installation. After the contest, the laptops must be sent back in the condition they were received.

Since we cannot alter Windows (and change the Master Boot Record, MBR), it brings up a challenge of booting the Linux image. Booting from WiFi network is not an option, because it will overload the wireless network, and even to bootstrap PXE is not an option (since motherboards do not support netboot over wireless). Therefore we devised a way to store the operating system and user files on the hard drive without modifying the existing Windows installation. Here's how: the full disk image was placed on `C:\` drive of the Windows partition. The MBR stays untouched. That way the partition table could stay intact, and it is easy to clean up a single file afterwards.

Placing a hard-drive image on NTFS drive still requires some work to boot it. Motherboard must find stage0 boot loader in the MBR, which should eventually start booting the hard-drive image on the NTFS partition.

Here is the sequence we devised to reach the partition on disk:

- Computer boots kernel and initrd from the USB stick.
- Initrd has a script which finds `C:\`.
- `C:\` is mounted on `/mnt/looproot`. `/mnt/looproot/liox.raw` is the full Linux system image.
- Root partition on `/mnt/looproot/liox.raw` is bound to `/dev/loop0`.
- initrd binds `/dev/mapper/loop0p1` as a root partition and passes the command to `/init` of the root partition. The booting continues as normal as if root filesystem were on `/dev/mapper/loop0p1` with the kernal command-line argument `root=/dev/mapper/loop0p1`.

[2] or rather, a `make iso` invocation
[3] `make vm`

Here is the snippet of the script from initrd which executes the actions above:

```
mkdir -p /mnt/looproot
mount -t ntfs-3g ${LOOPROOT} /mnt/looproot
losetup /dev/loop0 /mnt/looproot/${LOOPSRC}
kpartx -a /dev/loop0
```

`${LOOPROOT}` and `${LOOPSRC}` can be passed via cmdline. For example:

```
initrd initrd.img
kernel vmlinuz looproot=/dev/sda2 \
loopsrc=liox.raw \
root=/dev/mapper/loop0p1
```

It is still necessary to bootstrap the kernel and initrd so they can do the work described here. Small USB sticks were chosen for this: when a (re)boot is necessary, the operators plug in a USB stick. After kernel and initrd are loaded, the stick can be taken out and the operator can proceed to the next machine. For 60 laptops on-site, with this approach, it takes around five minutes to boot all machines for three trained operators.

4.2.3. *Uploading the Image*

Copying the disk image to the machines is different every year. In other words, we did not find the best solution yet which would prove its value for the long-term.

In BOI2014 we were walking around with external hard drives and USB sticks having the copying of the image scripted. It is laborious and it takes time, but is very simple. Because its simplicity and reliability, this copying solution has been used for many years. Over the years organizing the National Olympiad, the Technical committee tried different network-based approaches, but, if the approaches work in the labs, they usually fail in real-life due to misbehaving overloaded networks.

4.2.4. *Networking*

Main server was running the following services:

- Contest Management System – all services except Workers.
- HTTP proxy – nginx.
- DHCP and DNS services – dnsmasq.

DHCP was serving the IP addresses to student workstations, which were querying for IP addresses over WiFi.

Since the workstations were laptops and physical network infrastructure did not exist, we opted for using WiFi for networking. We used 6 access points in the room. Each AP had a different ESSID, but were connected to the same bridged L2 network.

Students' network access to other public networks (which can also be WiFi Access Points from the cell phones in their pockets) was restricted by pre-configuring the network and not granting the permissions to change network settings. This is done in `wpa_supplicant.conf`.

This network setup is simple: there was only one DHCP server running in the network, and all the IP addresses were dynamically configured. Dynamic configuration

(DHCP) brings a nice side-effect that all the network configuration is in a single place (in our case, `dnsmasq.conf`). What is more, having the same L2 network does not require any switch configuration.

Having everything in the same L2 network has security drawbacks: students are blocked from accessing each other only in their workstations. If a student manages to get superuser privileges on the machine, it would be only a small step away to communicate with other machines on the network, and, for example, to share and discuss the problems and solutions.

Another disadvantage is contest server reliability. If the server goes down due to hardware failure, it will take non-trivial amount of time to recover. Taking the hard drives out, plugging them to another machine, reconnecting the network would all take time and students' ability to interact with the contest management system would be impaired. This risk could be mitigated by having a standby server ready to take over in case the first one goes down.

4.3. *Internet Access*

Providing internet access for Scientific Committee and Public proved to be quite a challenge on the facilities. In the next sections, we describe our experiences in setting it up.

4.3.1. *First Attempt*

About one month before the contest, we arrived at the planned contest location to inspect the facilities. Among other things, we tested internet connectivity. The uplink was a DSL line with the following properties: small latency (up to 20ms to known servers) with 10Mb/2Mb of bandwidth. Since internet should have been primarily used for accessing public internet by staff and students, we agreed these will be sufficient.

After arrival to the contest location to do the preparation and preliminary network setup, internet connection quality significantly degraded. Degradation manifested by IP packet loss of up to 80%. Internet connection is critical for the Scientific Commitee, since the tasks and tests are shared via a third-party internet-accessible service GitHub. The packet loss was so significant that initial cloning of the task repositories was failing. This needed to be fixed.

While investigating, we could not relate the problem with anything from our side. After testing various options to reduce the packet loss, all in vain (like putting all our infrastructure under a single a NAT host), we decided to look for the cause of the problem in the facilities. Late in the afternoon, we discovered an Ethernet hub which we could not identify the purpose of. Disconnecting the hub fixed the packet losses and removed jitter, and did not seem to break anything: we received no complaints from the staff, nor internet access was broken any on-premise machines we had access to.

Later at night, a staff member contacted us with a complaint that credit card reader does not work. The night before the Olympiad, the facilities were hosting a wedding party, and they were unable to service the customers' credit cards for the extra drinks. We turned the hub back on, and the staff reported the card problem fixed. Further throughout

the Olympiad, we did not dare to disconnect the hub any more and had to resort to alternative means of internet connectivity.

4.3.2. *Second Attempt*

We connected a 3G USB dongle to a netbook, and connected that netbook to the main server. With this setup, all ”internal” network access (contestants’ machines and WiFi of the Scientific Committee) went through that dongle. As long as nobody were downloading large files, latency and throughput were acceptable.

The 3G dongle remained in use throughout the duration of the Olympiad.

5. Related Work

(Poranen *et al.*, 2009) is an accurate article describing the history and scientific part of Baltic Olympiad Informatics. The article was a joint effort of a number of hosting and participating BOI countries. Besides details about BOI tasks, history and organization, each country gives a brief overview of their national Olympiad.

(Dagienė and Skūpienė, 2007) describes experiences of National Olympiad of Lithuania. Like mentioned in section 3.2, challenges in organizing Baltic Olympiad Infor matics and National Olympiad Informatics are very similar. This article highlights the reasons for creating the Olympiad, its history, challenges and organization.

(Imajo, 2011) describes a case study running 60 laptop computers under a single wireless station. The article describes Japanese National Olympiad Experience using a single Apple AirPort Extreme wireless. According to the article, it was a success. At BOI2014 we used six commodity routers instead of one: ”This time we relied on AirPort Extreme wireless routers because they can officially manage 50 wireless connections at the same time according to Apple Inc., although ordinary wireless routers can manage about 20 at the most.”

(Blackham, 2013) describes the technical experiences of running IOI. Due to the number of participant workstations, setup and characteristics in IOI are quite different. Interestingly, (Blackham, 2013) pushes the images using UDP multicast. BOI and IOI are quite different in scale and requirements for safety:

- Even though the Olympiad was hosted on ”on an extremely reliable power grid”, the there was backup power supply prepared nevertheless. At BOI, we were also on an extremely reliable power grid, however, did not invest in renting and setting up the generators.
- From the contest system side, all of the critical pieces of infrastructure had replicas or hot backups: the database for Contest Management System had a was replicated and in hot-standby mode. Core servers and network equipment were backed up by UPSes to help survive the power outage. At BOI, the main server had a software RAID-1 array with a prepared machine to put the disks to in case of breakage.
- Each participant’s workstation was connected to their own L2 network. Since this requires support from the networking gear, we went for a simpler approach – each participant got their own L3 network instead of L2. Being simpler, the approach

is less safe: if two participants are able to gain superuser privileges on the workstation, they would be able to configure the network so they could see each other. However, exploiting this fact requires both gaining the elevated rights and coordination, which we deem an acceptable risk.

6. Further Recommendations

Hosting the event in a sea resort proved to be a very good choice. Contestants lived only a few hundred meters away from the sea, which provided very good opportunities for relaxation close by. Good location had the price of running the contest in a health resort without technical infrastructure (only electricity granted), which, in the end, was fully sorted out. The laptops and desks were rented, and, thanks to sufficient manpower and time, assembled on time. As long electricity and basic internet facilities are granted, it is sensible to prioritize accommodation, leisure and attraction facilities rather than technical infrastructure.

For the next BOI, we would recommend to keep researching for an efficient way to distribute the distribution images over the network. It might be worth looking at udpcast proposed by (Blackham, 2013).

It should go without saying that upon arrival to the contest location, the systems should be preconfigured as much as possible. However, it is important to leave flexibility for ad-hoc changes on every stage. Things that are prepared one day or the night before the contest should be simple rather than perfect. It is acceptable for things to be dirty, as long as they are documented, can be properly revised after the contest.

References

Blackham, B. (2013). A behind the scenes look at IOI 2013. *Technical Report*, University of Queensland. http://tiny.cc/ioi2013

Bulotaitė, J., Diks, K., Opmanis, M., Prank, R. (1997). *Baltic Olympiads in Informatics*.

Dagienė, V., Skūpienė, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics*, 1, 37–49.

Imajo, K. (2011). Contest environment using wireless networks: a case study from Japan *Olympiads in Informatics*, 5, 26–31.

Jyrki Nummenmaa, C.I. (2005*). Linux Programming Environment on CD*. http://www.ioinformatics.org/newsletters/html/ioinews5.htm.

Poranen, T., Dagienė, V., Eldhuset, A., Hyyro, H., Kubica, M., Laaksonen, A., Opmanis, M., Pohl,W., Skupienė, J., Soderhjelm, P., Truu, A. (2009). Baltic olympiads in informatics: challenges for training together. *Olympiads in Informatics*, 3, 112–131.

M. Jakštys is a software engineer at Amazon.com, Inc. He is a chairman of the Technical Committee in National Olympiad Informatics (Lithuania) since 2014 and was a chairman of Technical Committee in Baltic Olympiad Informatics 2014 in Palanga. Motiejus was a Deputy Leader of the Lithuanian team in IOI 2014.

Olympiads in Informatics, 2015, Vol. 9, 243–245
DOI: http://dx.doi.org/10.15388/ioi.2015.20

VisuAlgo – Visualising Data Structures and Algorithms Through Animation

Steven HALIM
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, 117417, Singapore
e-mail: dcssh@nus.edu.sg

VisuAlgo (Fig. 1, `http://visualgo.net`) is the continuation of the work that was presented in IOI conference 3 years ago[1] (Halim *et al.* 2012). VisuAlgo retains all the strong points of its predecessor:

- A web-based algorithm visualization tool without the need to install any additional software.
- It uses the latest web technology: HTML5, CSS3, JavaScript.
- It allows users to specify their own algorithm inputs and the visualization will work with that inputs.
- It is a collection of algorithm visualizations with unified interface.

VisuAlgo is a major improvement over its predecessor with ~2000 sessions daily from worldwide visitors:

- It has significantly many more algorithm visualizations in the collection – all with the same unified look and feel. Almost all visualize-able data structures and algorithms covered in the author's Competitive Programming book 3rd ed (Halim and Halim, 2013) have been included in VisuAlgo.
- It has an improved User Interface and more detailed algorithm animation steps.
- More importantly, we have added an important learning component: An *Online Quiz* tool (Fig. 2). It currently has hundreds of questions (and growing) with *randomized inputs and/or question parameters that can be graded instantly*. A Computer Science instructor can assess the *basic* data structure and algorithm knowledge of his/her students with much less effort. Computer Science students can also *self-assess* their proficiency of the basic material and they can always go back to corresponding visualization tool to restudy the concepts if they need to do so.

[1] The old URL: `http://www.comp.nus.edu.sg/~stevenha/visualization/` is now no longer used.

References

Halim, S., Koh, Z.C., Loh, B.H.V., Halim, F. (2012). Learning algorithms with unified and interactive web-based visualization. *Olympiads in Informatics*, 6, 53–68.

Halim, S., Halim, F. (2013). *Competitive Programming 3: The New Lower Bound of Programming Contests.* `http://cpbook.net`

Fig. 1. VisuAlgo landing page.

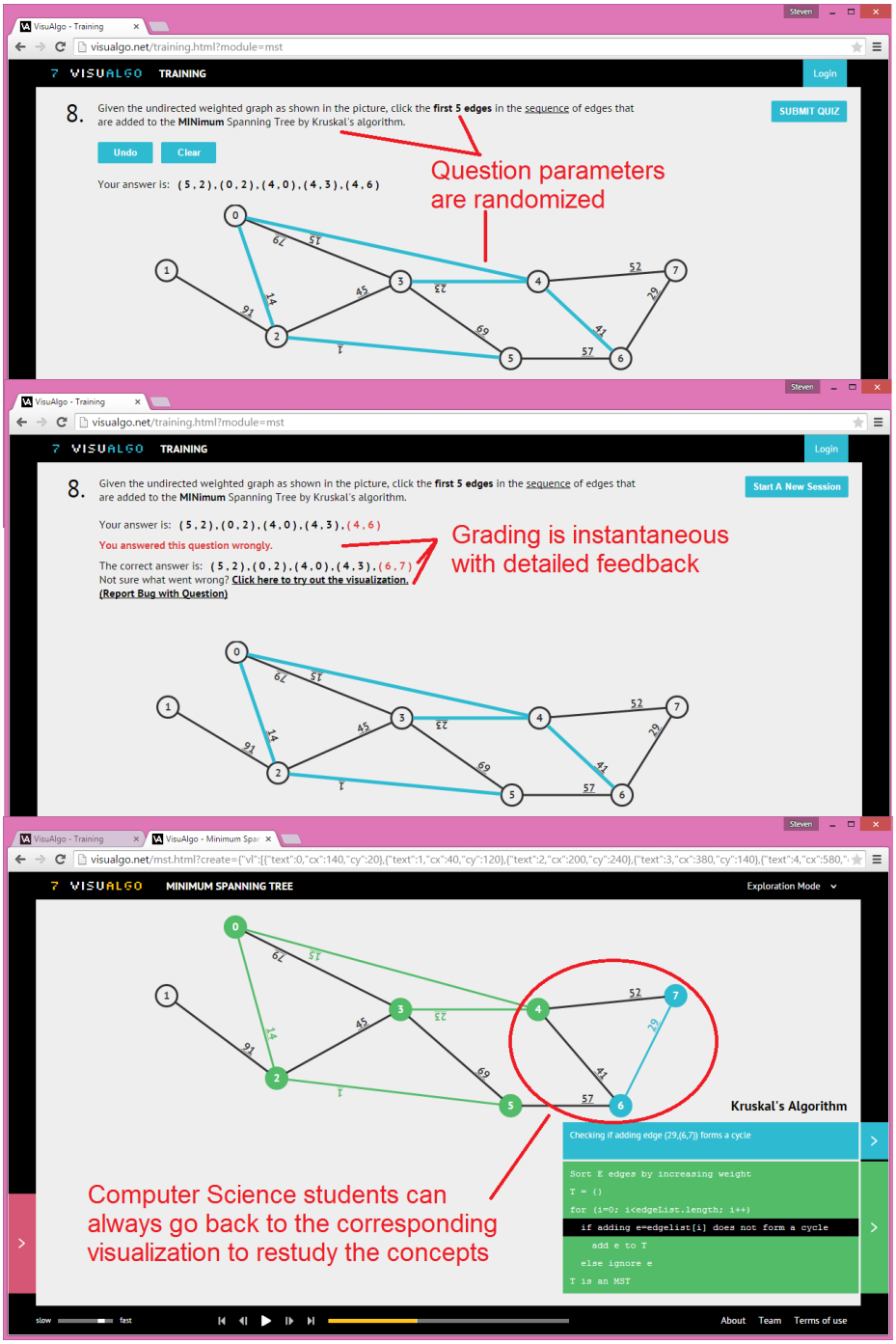


Fig. 2. The newest Online Quiz feature of VisuAlgo.

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)
- Elsevier Bibliographic Databases (SCOPUS)
- ERIC
- 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.

The references cited in the text should be indicated in brackets:

- for one author – (Johnson, 1999)
- for two authors – (Johnson and Peterson, 2002)
- for three or more authors – (Johnson *et al.*, 2002)
- the page number can be indicated as (Hubwieser, 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

All the information about the journal can be found at:

`http://ioinformatics.org/oi_index.shtml`

Vilnius University
Institute of Mathematics and Informatics
Akademijos str. 4, LT-08663 Vilnius, Lithuania

Olympiads in Informatics

Volume 9, 2015

ISSN 1822-7732