Mathematics and Education in Mathematics, Proc. of 38-th Spring Conference of UBM, Borovetz, 2009.

# GRADING SYSTEMS FOR COMPETITIONS IN PROGRAMMING*

**Krassimir Manev, Miloslav Sredkov, Tsvetan Bogdanov**

Competitions in programming appeared in the 70s of the past century for attracting talented young people to the profession of programmer. Soon after the start of organizing programming contests, grading systems become an inevitable part of the process of evaluation of contestants' results. Recently, different applications of the grading systems were announced that were going beyond the initial scope of these systems. The paper presents the main objectives and concepts in the domain, the architecture of a typical grading system and some examples. The perspectives and challenges of the grading system in their original usage, as well as in their usage in some related domains, are outlined.

**1. Introduction.** The positive role of the scientific contests, and especially of the contests in mathematical disciplines, was outlined long time ago (see for example [12]). Two of the main objectives for such estimation are that:

- it is more attractive for children when the education is organized in parallel with and for the purposes of the participating in competitions;
- many of the challenges that the children will meet in their adult life have a form of a competition, so it is necessary especially to teach students to compete. It is better to start the process as early as possible.

It was mentioned in the referred above paper that two years before restoring of the Olympic Games, in 1894, Eötvös University in Budapest, Hungary, organized the first national mathematical contest. It was mentioned also that William Lowell Putnam started in 1938 a mathematical competition for North-American college students. These national and regional contests eventually gave rise to the International Mathematics Olympiad (IMO) for school students [18]**.** The first IMO was hosted by Romania in 1959.

**1.1. Programming contests.** In 1977 the Association of Computing Machinery (ACM) starts its Collegiate Programming Contest for students in the American Universities. Very soon the contest was extended to an international event. Nowadays the International Collegiate Programming Contest of ACM (ICPC) attracts students from more than 1700 universities and more than 80 countries all over the world [17].

Bulgaria is among the pioneers of the programming contests for school students. First programming contests for school students in Bulgaria were organized in the early 80-th of the past century. From May 19 till May 23, 1987, an open international programming contest for school students was organized just before and in connection with the Second International Conference and Exhibition of IFIP "Children in the Information Age", which took place in Sofia, Bulgaria. Two years later, in May 1989, Bulgaria organized and hosted the First

* **Keywords.** Software testing, proving correctness of programs, operating systems, education in programming…

International Olympiad in Informatics (IOI) for school students [7, 19]. Soon some regional Olympiads in programming appeared too – Balkan OI, Central-European OI, Baltic OI, etc. In 2007 the first Olympiad for school students aged less than 15.5 years took place in Belgrad, Serbia – the Junior Balkan Olympiad in Informatics.

Recently, two kinds of programming contests, different from Olympiads, are organized also. On the first place, they are the programming contests of professional software companies, like "Top Coder" [22] and "Google Code Jam" [15] programming contests. Understanding well the role of the contests, some professional companies invest money and maintain their own teams for preparing tasks, for organizing contests and evaluating them.

The activities of second kind are the various web sites that are proposing a continuous on-line training process with competitive elements – each contestant could enter the site when she/he would like, to select a task of specific topic and hardness, to solve it, to submit solution and to obtain the evaluation of her/ his results, as many time as necessary and for as long period as necessary – till producing the correct solution. Very popular training sites nowadays are, for example, USACO Training Gateway [23], UVa Online Judge [24] and Russian Open Cup [25].

**1.2. Evaluating Mathematical Tasks.** Programming contests have many common features with the mathematical contests. Both are proposing to the contestant to solve (one or more) *tasks*. The statement of the task, in mathematical as well as in programming contests, usually contains an object (or set of objects) with given properties – *input*. Starting with the input, the contestant has to find (to calculate or to give reasons for validity of) some unknown property of the given object(s) or to construct a new object (or objects), being in prescribed relations with the input object(s). We will call the obtained result – found (calculated, reasoned) property or constructed object(s) – *output.*

From practical point of view, it seams enough to obtain the output of the task. But in mathematical contest (as well as in evaluation of the results of mathematical education) the Jury (respectively the teacher) would like to see how the contestant derives the output from the input. That is why the contestants are asked to submit an, as much formal as possible, description of the derivation process that starts from the input and lead to the output. We will call such formal description a *solution* of the task.

The main feature of the solution that is usually evaluated is its *correctness*. Contemporary mathematics has well developed mechanism for totally formal description of solutions of mathematical tasks. Nevertheless, nobody – neither the introductory text books nor the sophisticated research papers are using a totally formal description of the solutions. And the reason is very simple – the human being is not able to follow in depth complex formal descriptions, neither to validate its correctness. That is why the validation of the semi-formal descriptions is organized as bilateral interactive process – the one side (an author or a defender) is arguing about the correctness of some non formal part of the description and the other side (an independent opponent, a reviewer) have to decide whether it is really correct.

**1.3. Checking Correctness of Programs.** The programming tasks are in principle the same as the mathematical tasks. Some input is given and some output with prescribed properties or in prescribed relations with the input has to be produced. It is possible that the contestant is asked to supply only the output. But, in general case, as well as in the mathematical contests, the contestant has to supply a solution that, for given input, leads to the corresponding output.

First difference between mathematical and programming contests is that the solution submitted by the contestant has to be **pure formal** – a *program* written in some programming language. In order to be evaluated, the *source code* of the program has to pass successfully through the *compilation* process and to be translated to *executable code.*

As well as in mathematical contests, the first criterion for evaluation of the proposed by the contestant solution is its *correctness.* As it was mentioned above – checking the correctness of a complex formal description is extremely difficult even for very experienced professionals. Any way, not only in the beginning but even nowadays [8], the members of the evaluation teams of programming contests try to read and check in such way the correctness of the solutions of the contestants. With the increasing complexity of the proposed tasks such checking becomes harder and harder, practically impossible.

From mathematical point of view, it is possible to organize a pure formal checking of the solution when it is presented with a formal description – a program. The relatively well developed theory for *verification of programs* prescribes to assign to important points of the solution corresponding *verifying predicates* and the program is considered correct when all predicates are evaluated as true. Such approach for checking of programs is called *inner* or *transparent-box testing* [6]*.* It is intrinsic for the authors (the developers) of programs. There are even practical mechanisms for implementing such testing (ASSERT-mechanism of some C/C++ IDE, for example).

Unfortunately, such testing is not intrinsic for evaluating programming contests. The design of each verifying predicate is very dependent on the particularities of the solution. It is impossible for the members of the evaluation body to imagine what kind of solutions the contestants will supply and to construct in advance all necessary predicates. Something more, for applying inner testing the evaluator has to open the source code and to make changes in it – including the necessary predicates. Beside the slowness of such procedure it is considered unacceptable for programming contests – it could lead to unacceptable changes (accidentally or on purpose) in the programs, does not mater whether they are in favor or in harm of the contestants.

The software industry is using another kind of testing too – the so called *external* or *black-box testing*. It consists of execution of the tested program with a set of inputs, called *test cases,* and *comparing* the obtained results with supplied by the author *etalon outputs.* While inner testing is performed by the programmers during the development of the program, the external testing is performed by an independent testing team when the program or some important parts of it are ready. That is why such testing is more appropriate for programming contests.

The main problem of the external testing is the *selection of the test cases*. It is obvious that usually the size of the input space is enormously large and it is not possible to check the program with all possible inputs. That is why the author of the task usually tries to introduce some *relation of equivalence* in the space of inputs. The number of *classes of equivalence* of the relation has to be not very large. And more, it is expected that, if the program produces a correct output for one element of some class of equivalence, then it will produce correct output for each other elements of the same class. And the opposite – if the program produces wrong output for one element of some class of equivalence, then it will produce wrong output for each other element of the same class. It is obvious that defining such kind of relation is non trivial mathematical task [3, 5].

Another problem of the external testing is that some tasks' statements allow more than one correct output for some inputs and sometime – very large amount of correct outputs are possible. Reducing the number of the outputs by some additional rule (lexicographically first, for example) is possible but sometimes it is unacceptable because it could change the sense of the task. That is why in such cases the correctness of the output has to be checked by a special program – *checker*.

Writing a checker sometimes is more difficult then writing a solution of the task. The problem is that checking of the output is practically impossible when it is not *well formatted* – for example, because of an inadequate usage of blank characters (spaces and new lines). It is true that the efforts necessary for solving a task are much more then the efforts necessary for formatting the output. And there are appeals, checking to be more tolerant toward usage of blanks. But what has to do a checker, which is expecting an output with three integers and is receiving as output the string `123456`. That is why, the appeal for *more flexible toward formatting checking* remains one of the oldest unsolved problems of the checking.

**1.4. Checking Effectiveness of Programs.** Besides the correctness, there is one more criterion for the solutions quality of the programming tasks – their *efficiency*. Programming tasks are of algorithmic nature – for solving such task the contestants have to select or, sometimes, even to invent an *algorithm* that transforms the input to a correct output. In principle, each task could be solved by different algorithms, each of them with different *time* and *memory space complexity* (formal definitions of these basic notions could be found in each textbook on Algorithms (see, for example, [2, 4]). As well as with the case of correctness, the evaluator could try to estimate these characteristics of the used algorithm by reading the source code, but this is even more difficult then checking the correctness.

That is why the evaluators of programming contests use another approach. For each task a specific *time limit* and specific *memory limit* for the execution of the program are defined. Only when the program succeeds to produce a correct output inside the chosen time and memory limit, the contestant obtains the corresponding grading marks.

For example, suppose that the task is solvable by tree different algorithms with time complexity $\Theta(n^3)$, $\Theta(n^2)$ and $\Theta(n)$ respectively, where $n$ is some natural measure for the size of the input. Suppose that the author would like to assign only 20% of the grading points to contestants who are using the first algorithm, 50% of points to these using the second, and 100% of the points to these using the third. Then the author chooses some upper limit for the size of input, for example $2^m$ and generates 20% of the test cases with size of the input $n \leq 2^{m/3}$, 30% of the test cases with size $2^{m/3} < n \leq 2^{m/2}$, and the remaining 50% of test cases with size $2^{m/2} < n \leq 2^m$. The time limit in such case is defined by the time that is necessary for the algorithm with time complexity $\Theta(n)$ to solve a test case of size $n = 2^m$.

Limiting of the used memory space is not as important as the limiting of the execution time because of the following observation. The algorithm which is using memory space estimated by $\Theta(f(n))$ has execution time $\Omega(f(n))$ and so the time limit could be used for limiting the used memory too. Anyway, there are tasks for which make sense, and is important, a specific limitation of the memory to be declared and checked during the evaluation.

From the mentioned above it is obvious how important for the evaluation of the solution is the precise *measurement of the processor's time* elapsed by the program. Unfortunately, the universal operating systems have no built-in instrument for precise measurement of the *pure processor's time* consumed by the program. That is why the precise time measurement is still one of the challenges of the evaluation process.

**1.5. Automating of the Evaluation Process.** From the discussion above it is clear that the checking of correctness and effectiveness of computer programs is a complex set of activities. Many of these activities could be performed only with usage of appropriate software. Other could be performed manually but this is boring, time consuming and risky.

For example, let us consider a typical contest – IOI'1998 in Setubal, Portugal – with about 250 contestants, 3 tasks and 20 test cases average for a task. For each tested task one compilation is necessary, which means 750 compilations. Then for each test case 2 executions have to be performed – execution of the solution with the corresponding test case input and execution of the checker (typing the output of the contestant on the screen and comparing it, visually, with the etalon output is even more time consuming). That means a total amount of about 30000 executions.

Bearing in mind that just typing of a command on the keyboard takes about 10 seconds we will obtain 300000 seconds or more than 83 hours. Appending approximately 1/10 of this time for executions of programs, we will obtain more than 90 hours computing time for evaluating of one contest day, which even for a team of 10 evaluators is too much. That is why the grading in Setubal started at 3-4 P.M. and finished at 5-6 A.M. on the next day.

The solution of the outlined above problem is obvious – using a software system for organizing and implementing the evaluation process. We will call such system a *grading system* (GS). For example, the organizers of IOI'1999 in Antalya, Turkey, decided to use such system – not very sophisticated and far away from the complete possible functionality. With this imperfect system the time necessary for grading one day of the contests was reduced to 2 hours. Organizers of IOI'2000 (China), IOI'2002 (Republic of Korea), IOI'2003 (USA), IOI'2005 (Poland) and IOI'2007 (Croatia) developed their own grading systems. Organizers of IOI'2001 (Finland), IOI'2004 (Greece) and IOI'2006 (Mexico) used the "American" system and organizers of IOI'2008 (Egypt) – the "Polish".

In the popular ACM ICPC the system PC^2 (or PC2) is used for many years [20]. First version of PC^2 is issued in 1988 and used in Sacramento State University for evaluating local contests. In 1994 the version 4.1 is used for first time during the World Finals of ICPC. Since that PC^2 is the official GS of ICPC – not only for the World Finals but for all Regional Rounds of the contests.

Nowadays GS are inevitable part of the programming contests and training sites. In Section 2 we will describe in more details GS – their objectives, architectures and implementations. We will consider some GS developed or/and used in Bulgaria. We will stress some scientific and implementation challenges that stay in front of developers of such system. In Section 3 we will present some possible usage of the grading systems outside the scope of programming contests.

## 2. Grading Systems.

**2.1. Objectives.** There are many objectives of the grading systems that depend on kind of the contest, kind of the tasks, kind of the feedback answers that the system has to give to the contestant and so on. Let us consider these objectives.

**Types of answers.** As a result of the checking the evaluation system is producing different kind of answers. In the usual order of happening of the corresponding event the *negative answers* are:

- ***Compilation error*** – the source code of submitted by the contestant solution can not pass the compilation process. This event is relatively rare when the contestant and GS use the same compiler. But when different compilers are used, then such answer is not so rare;
- ***Run time error*** – the program was successfully compiled but during the execution it produced an exception on hardware or OS level;
- ***Time limit (or memory limit) exceeded*** – the program was successfully compiled but during the execution it did not finished inside defined time limit (or attempted to use more memory that it was limited);
- ***Wrong output formatting*** - the program was successfully compiled and executed inside the time and memory limits but the produced output has a wrong format and the system can not check its correctness;
- ***Wrong answer*** - the program was successfully compiled and executed inside the time and memory limits, produced output is well formatted but it

is not correct;

- And so on

If no one of the mentioned happened then the solution is accepted as correct.

**Types of contests.** There are two principle types of programming contest – let us call them *single test* and *multiple tests* contests. In the **single test contests** the solution of the contestant is executed as many times as the number of prepared by the author test cases is. For each test case the GS is checking the output correctness and when it is correct and the execution finished inside the time (and memory) limit – assigns the corresponding number of points. The final result is the sum of obtained for all test cases points. For test cases that are not solved by the program – the corresponding reason is announced, as mentioned above.

For this type of contests the evaluation is usually performed after the end of contest and it is not important when the contestant submitted the solution. This kind of evaluation is typical for the contests for school students – national olympiads, IOI and the regional olympiads for school students.

In the **multiple tests contests** the solution of the contestant has to be able to read many test cases during a single run and to produce an output for each of included in the input test case. The grading system runs the solution with an input that contains all prepared by the author test cases and checks the output. If it is correct and the execution of the program finished inside the defined time and memory limits, then the GS registers one solved task in the account of the contestant and informs immediately her/him for the success. If at least one test case is not solved – the task is considered not solved and the contestant is informed immediately for the reason, as mentioned above.

In this kind of contest, the time passed from the beginning of the contests to the moment of acceptance of the tasks is important. The contestants having the same number solved tasks are ranged by the sum of elapsed time of solved tasks. Something more, some time is appended to the sum for each submitted wrong solution. This kind of evaluation is typical for university students contests (ACM ICPC, for example).

Both kinds of evaluation have their advantages and disadvantages, as it was mentioned in [9]. For example, multiple test evaluation is more close to the real life (software industry) – the programmer is checking the program as many times as necessary in order to obtain relatively correct program. But this checking is bipolar and not appropriate for academic (school or university) grading where a large scale of grading marks is expected. Some convergence of the two kinds of evaluation is considered from both sides – recently IOI GS started giving to the contestant a partial feedback for the quality of submitted solutions.

**Types of tasks.** In GS different kinds of tasks have to be considered. Usually it is necessary to submit as a solution *source code* of a program that solves the task. The set of test cases for these tasks is unknown to the contestants. For some task it is not necessary to submit a code – *output-only* tasks. For this kind of tasks the set of test cases is given to the contestants and they have to supply the corresponding outputs. So, for solving output-only tasks the contestants could use pure mathematical reasoning as well as an appropriate program.

The tasks that have to be solved with a source code could be divided in two categories. For so called *batch tasks* the contestants have to supply a program that is able to read the input (from the standard input or from a named text file) and to write the output (to the standard output or to a named text file). For so called *interactive tasks* the contestant has to supply as a solution a part of a program – the *contestant module* that is able to communicate with a program part prepared by the author of the tasks – *author module*. During the evaluation process both modules are integrated in a program. All necessary data are received in contestant module from author module and all calculated results are passed back through preliminary specified *interface*. Lately an external binding between the modules (now being sole programs) is also considered as it allows a better evaluation of the elapsed time.

The tasks that have to be solved with a source code could be divided in two categories by another criterion too. For some of these tasks a *precise output* is expected**.** The contestant is obtaining corresponding points when the solution is able to produce the expected output or is obtaining zero points in the opposite case. For the other tasks, so called *relative*, it is very difficult or even impossible to find the precise output. That is why the solution of the contestant is trying to find some value as close as possible to the precise (usually some min or max). The best solutions are obtaining the full number of points and the other solutions are obtaining smaller number of points, relative to how far from the best solutions they are.

The programming contests community is continuously trying to include another form of tasks in order to make programming contests more adequate for the necessity of education of young programmers. For the moment, for example, the tasks leading to some graphical outputs are totally excluded from the programming contests.

**2.2. Architecture.** Modern GS do not share common history. In fact most are developed as silos from their beginning. The reason is that a simple GS is relatively easy to build, if not easier than understanding an already existing one. Moreover some of the best known GS are with *closed code* – their code is not publicly available and no guidance is provided for their setup. Thus most people that need to grade some programs outside of the software engineering process end up with creating their own GS.

That said most of them conform to a single architecture. Of course the concrete programming language, module names and the protocols used may vary. But most of them still seem to use the following types of modules:

- ***Sandbox:*** The main responsibility of the sandbox is the *integrity* of the GS. It ensures that the execution of the submitted program will not harm the host computer or GS itself. It also ensures that the solution does not go out of the allowed *restrictions* (memory space, time, number of open files, disallowed network access, etc.). This is the real “heart” of the system. It is also the most demanding module in terms of code quality. Because an error in this module usually exposes a serious vulnerability that easily propagates through the rest of the system. Fortunately it is also one of the least susceptible to changes.

- ***Grader:*** The grader deals with the compilation of the source code, manages the sandbox, and runs checkers and other similar functions. Ultimately the grader is the module *responsible to come up with a result* about whether the solution is correct or not. The grader is usually bundled with either the sandbox or the dispatcher. It also must be agile enough, because in the typical case it has to handle several different types of tasks and rare cases.
- ***Dispatcher:*** The dispatcher is the administrative server of the system. It handles *the communication* with the lightweight clients, the graders and any other modules there might be. It does not really make decisions on the correctness of the program. It has to collect the source code, to off-load it to a grader, to receive and store the results from the grader and propagate back any feedback necessary. Usually it also handles reporting. However in some rare case there might be a separate module for this purpose. As there are no real performance requirements for it but it might need to scale across instances and handle multiple protocols it is usually written in a high level language.
- ***Storage server:*** The storage server has the only obligations to store auditing records as a service to the dispatcher. That is why it is not rear to be implemented as part of the dispatcher but it is also equally likely to be just a RDBMS instance.
- ***Light-weight client:*** The competitor usually interacts with the GS via a thin client. In many cases it is simply a web browser. However a light java application [20, 22] or even a command line interface [8] is also a viable alternative. In any case it needs to stay simple as it is the most replicated instance of the system during a typical programming contest.
- ***Auxiliary modules:*** It is a common requirement that the GS also handles printing, backing up storage and files for the competitors or even games for bored jury members.

As for deployment, usually the dispatcher and the storage server share a common host (if they are separatea at all), the light-weight clients are spread on the competitors machines and few machines are used with preinstalled graders (and sandboxes on the same hosts). On the network level a firewall can be inserted between the dispatcher and the clients creating in that way at least two separate sub-networks.

### 2.3. Examples

**PC^2 System.** This is the oldest among the popular grading systems. It was designed and implemented for organizing programming contests in ICPC style. The architecture of the system is out of date because it was not changed seriously till the launching. The system has for example its own Light clients – for the administrator of the system, the judge, the contestant, the reporting and so on. It is impossible to use the system if these light clients are not installed on the user's computer.

The main shortcoming of the PC^2 system is the missing of a module that is corresponding to the described above Sandbox. As a result the system has not really adequate time measuring. Something more, the system can not stop automatically the checked program when it exceeded the time limit and this has to be done manually by the Judge. When the program is stopped by the Judge it produces an empty output and the Grader proposes the judgment *Wrong answer*, which is confusing.

**SMOC System.** System for Managing Online Contests (SMOC) is the GS used in all national programming competitions for high school students in Bulgaria. Originally it was modified from the grading system used in IOI'2002 (Republic of Korea) for the purposes of BOI'2004 (Bulgaria). It has since grown and the original system is virtually completely rewritten. Since then it was also used in BOI'2006 (Cyprus), BOI'2008 (FYRO Macedonia), JBOI'2008 (Bulgaria) and will be used for IOI'2009 (Bulgaria). Aside from the on-site contests it is also used to handle simultaneous online competitions.

For interfacing competitors SMOC uses web browsers. That is, it provides a simple HTML interface that the contestants are using to submit and test solutions, store and receive files and so on. This interface is of course provided by the administrative (web) server implemented in Java. As of the moment it also handles storage on the local file system. However it is already extended to allow RDBMS compatibility and is expected to move to such, external storage soon. The grader module is built for 32-bit Linux hosts. The package also encapsulates the Sandbox but the Grader calls it as an external process. This is a recent change introduced to improve both the security of the sandbox and the precision of the timing and allows greater convergence of the GS used in IOI. The latter is archived as the sandbox is owned by the IOI Technical Working Group. In the process of integrating the modules we found improvement opportunities on both sides.

The modern version of SMOC is able to support several types of tasks, including batch tasks, output only tasks and interactive tasks. However it is still closely coupled with IOI like contests in contrast to an ICPC-like contest.

Time measurement in SMOC has evolved in the process of usage. At the beginning the time was computed as **`user+sys`** time. That means only the time elapsed by the instructions of the program itself and the immediate system calls were aggregated. This way better (more precise) time measurements are achieved in a multi-process environment compared to "wall" time (the real time passed as it would be observed by a clock standing on the wall). However this advantage was not actually used until recently as the time limit was set as an **`ulimit`** restriction. Thus in a multi-process environment it was possible that the solution still had some time left when stopped. This problem was later resolved however the model of **`user+sys`** time does not account for the varying latencies due to disk cache. Recently we have moved to the model described in [8] which allows only the user time to be accounted since masking user instructions as system time is now disallowed.

**The system spoj0**. Spoj0 is a GS designed to work continuously. It is able to host several contests at a time, and allows online and offline submits. It is used for internal contests, exams, and homework assignments in several courses in Sofia University. Its first version, able to host a contest, was developed in one week, as an attempt to prove that such system can be developed for short time. After the first hosted contest, the system was evolving continuously, based on the feedback received from the hosted contests.

Since the quick development was the main initial goal, the whole system is written in Perl. The clients access the system with a web browser, using its web interface. The interface allows: browsing the contests, downloading problem statements, submitting solutions (to both active and past contests), browsing score-boards, accessing the source codes, results, and solutions from the past contests. System administration is performed by command line scripts on the hosting machine. A system daemon is continuously fetching waiting submits, and passes them to the grader. The grader uses tools provided by the operating system to allow secure execution of the submitted solutions. The system also has an installer for Debian GNU/Linux based systems and exporter for plagiarism checkers.

As grading is one of the most important activities, spoj0 uses operating system specific tools. Programs are executed by a user with limited permissions. CPU usage is taken to measure the execution time, but the total time is also limited to three times the time-limit. This is because sometimes programs may block waiting for input, or doing disk operations.

**2.4. Perspectives and challenges.** As it was already mentioned, the **precise time measurement** is a tricky thing. However the two main contributors seem to be the concurrent processes and the disk caches. That is why two consecutive executions of the same program on the same input data may produce the output data for a different perceived (wall) time. The disk caches can eventually be circumvented by using a **`ramfs`**. In such a way any test data is read from RAM memory, which on the other hand provides faster access and less deviation from the average access time. Simultaneously, the smaller access times will increase the speed at which the solution reads the input data and writes the output, thus decreasing the wall time for reaching either correct result or time limit. Of course this depends on the time limit and input and output sizes. But recently we tend to see more and more problems with input and output data for a single test case with a size over 1 MB. However it is observed that solutions with complexity $\Theta(n.log\ n)$ or less cannot be differentiated as the time it takes to read the input is more or at the best equal to the time actually taken by the algorithm. That is why marginally improving the input/output time, in the described way, is expected to produce significantly reduced execution times and time limits.

Another difficult problem is measuring the solution's time in the **interactive tasks**. There are two main approaches toward such tasks: to compile author module and contestant module in the solution or to compile both modules in separate programs and make them communicate via pipes. If we choose the first approach, aside from having to create a version of the author module for every available language, we also end up with no way to distinguish between the time spent by the solution and that spent by the author module. That is why recently the second alternative is becoming more popular. However we still need to address some blocking issues and there is no standardized protocol for those tasks. If a common library just for reading from and writing to the pipes was used in all competitions it might resolve the blocking problems.

The last part of a comprehensive time measurement is not only to achieve repeatability on a single host system but after **transferring the solution and GS** to different hardware also. In this case the problem is not only that the different architectures might execute the same instructions with different speed but that the proportions of the elapsed time of two different instructions might be different for the different architectures. This opportunity is worth to be considered as it would allow much easier transfer of tasks between grading systems, which will be discussed later. An idea for resolving the problem is to produce a statistic measurement of the times both on the original host and the destination and to search for such a coefficient $K$ that if the host system time limits are multiplied by $K$, the $\Sigma(|c_i - d_i|)$ is minimal, and $\min(c_i) = \min(d_i)$ and $\max(c_i) = \max(d_i)$ are satisfied. Here $c_i$ are the correctness of any solutions the jury has provided measured on the host system and $d_i$ are the correctness of the same solutions measured on the destination system. Unfortunately, up to our knowledge no grading system implements such an algorithm yet.

The reason that this has not been tried yet is that just now GS are starting to experience the need to communicate among themselves. The reason is that originally (and this still holds true) the GS are created as one-off systems that do not bear connection to any other system. However, as we start to explore alternative usages of grading systems the need to transfer tasks between them will grow. For example, if a task is given during the faculty training for ACM ICPC it might also be very suited as an exam task for a course in algorithms or programming. Or a task from a past national competition might be very useful in the internal training of a high school. In most of these cases the GS used will be different and virtually every time it will be on different hosts. Here the disconnection comes as the modern GS still do not support enough interoperability.

In order to gain such features two main problems must be solved. The first is to provide an equivalent, or at least adequate (as already described), time measurement. The second is to provide protocols for tasks description in place. The IOI community has been very active lately in this field (see for example [13] ) so we can expect some results in the near future.

**3. Usage of GS outside programming contest.** GS aroused in connection with the necessity of evaluation of programming contests. For their design and implementation knowledge and experience from many domains was used – education in programming, complexity of algorithms, operating systems, software testing, network programming, etc. As it happens frequently, development of such complex, interdisciplinary product has a direct impact on each of the concerned disciplines – not only by placing new interesting problems in of these disciplines, but helping them to solve some of the well known problems. Let us consider a fruitful impact that development of the GS could have on some of the mentioned above areas.

**3.1. Teaching of programming.** Teaching of programming is the closest area to programming contests because teaching of programming contestants includes teaching of programming. From the other side, teaching of programming includes evaluation of the results of the students and this is one of the most challenging parts of the educational process.

In many schools and universities the teachers still are checking manually program fragments written by the students, as homework or exam works, on paper. This is an activity which could be totally assigned to GS. The most natural usage of GS in education is during the heavy university courses of Algorithms. We are really using the mentioned above system spoj0 for many years in Faculty of Mathematics and Computer science of Sofia University for exams of the course of Algorithms.

It is amazing that GS could be used for evaluation in some very early stages of education – when the students have not the knowledge and experience to write complete program [10]. In such stage they are asked to write program fragments – functions, procedures, classes of simple objects, etc. – and the checking of the assignments could be done with GS if the statements of the tasks are prepared as interactive. The missing part, that students are not able to write have to be prepared as author module and to be integrated with the solution during the evaluation process. The authors of [10] are going further – they are using a GS not only for teaching classical courses of procedural programming but for courses of Functional and Logic Programming too.

Finally, it is possible to use the experience of the competitive training sites – GS together with a large archive of tasks could be published in Internet. All homework and exams could be made on-line by the students. And more, after expiring of the deadline of the homework or the exam, the students, that did not succeed to solve some of the tasks, could continue to work on them till obtaining a correct solution. In such way using the GS for education in programming make it more flexible, intensive and efficient.

**3.2. In software industry.** The main purposes of programming contest are discovering and development of talent in the domain of Informatics and Information technologies. This role of the programming contests is long years ago recognized by the software companies, which are searching contestants and enthusiastically hire them.

**Recruitment process.** A natural next step in this direction is to propose to software companies to use GS in the process of recruitment of software engineers. The company could prepare different kind of “contests” depending of the type of staff it needs – choosing the topics of the tasks, the programming languages, the development platforms, etc.

Because the individual qualities of the candidates have to be tested, IOIstyled contests are appropriate for the recruitment process. We know that different kind of competitions are used in recruitment process in some Bulgarian software companies as Musala Soft, Axway (formerly known as Tumbleweed), Telerik, as well as in the international giants Google [15], Microsoft [16], TopCoder Inc [10], AOL, etc. Applying GS in such contests will put new requirements to people who design and implement such systems. Involving super professionals from software companies in grading of contests, from the other side, will lead to significant amelioration of the qualities of GS.

**Team building.** Another possible domain of GS usage in software industry is the amelioration of the work of programming teams. It is very rare nowadays to develop software by a single developer. That is why, in the educational process, as well as in the software companies, developers have to be trained to work in teams. Programming contests for teams, i.e. in ICPC-stile, are a possible form of such training and the GS has its important place in such activities.

Internal contests can provide some moral boost and intrigue staff in new fields (algorithms, advanced abstract types and their efficient implementation in data structures, efficient usage of standard libraries, etc.). Such kind of contests is practiced, for example, in Google [15]. Recently an informal interest was expressed by the Bulgarian companies Axway and eBG.bg.

**4. Conclusions.** Nowadays Grading systems are inevitable part of the competitive programming – an extreme form of education of high level programmers. Being very complex and challenging products GS attract knowledge and experience from different domains of the science and the technologies – programming languages, operating system, DBMS, time and memory complexity of algorithms, software testing, etc. In such a way, the development of the GS is a large source of unsolved problems. Some of these problems are non trivial and have to be object of scientific research.

From the other side, GS could successfully and fruitfully be used in such related domains as education in programming and the software industry. Each usage of GS, different from the original, could outline disadvantages of the design or implementation, formulate new requirements and, finally, contribute to amelioration of the concepts and realization of GS.

## REFERENCES

[1] Sz. Acedacki. Polish grading system. Private communication.

[2] G. Brassard, P. Bratley. Fundamentals of Algorithms, Prentice-Hall, 1996.

[3] G. Cormack. Random factors in IOI 2005 test case scoring. *Informatics in Education,* **5**(1), 2006, 5-14.

[4] T. Cormen, Ch. Leiserson, R. Rivest. Introduction to Algorithms, MIT Press, 1990.

[5] M. Forišek, On the suitability of programming tasks for automated evaluation. *Informatics in Education,* **5**(1), 2006, 63-75.

[6] C. Kaner, J. Falk, H. Q. Nguyen. Testing Computer Software, John Wiley & Sons, 1999.

[7] P. Kenderov, N. Maneva (Eds.). International Olympiad in Informatics, Sofia, 1989.

[8] M. Mareš. Perspectives on grading systems. *Olympiads in Informatics,* **1**, 2007, 124-130.

[8] W. Pohl. Manual Grading in an Informatics Contest. *Olympiads in Informatics,* **2**, 2008, 122-130.

[9] M. A. Revilla, S. Manzoor, R. Liu. Competitive learning in informatics: the UVa on-line judge experience. *Olympiads in Informatics,* **2**, 2008, 131-148.

[10] P. Ribeiro, P. Guerreiro. Early Introduction of Competitive Programming. *Olympiads in Informatics,* **2**, 2008, 149-162.

[11] M. SREDKOV. A contest suporting system spoj0. Dioploma thesis for obtaining MSs degree, Faculty of Math. And Comp. Science, Sofia University, 2006 (in Bulgarian).

[12] T. VERHOEFF. The Role of Competitions in Education. Presented at *Future World: Educating for 21st Century*. A conference and exhibition at IOI'1997.

[13] T. VERHOEFF. Programming Task Packages: Peach Exchange Format. *Olympiads in Informatics*, **2**, 2008, 192-207.
http://olympiads.win.tue.nl/ioi/ioi97/ffutwrld/competit.html

[14] Bulgarian IOI-style grading system SMOC. http://judge.openfmi.net/SMOC/
http://openfmi.net/projects/pcms/

[15] Google Code Jam. http://code.google.com/intl/bg/codejam/

[16] Imagine Cup. http://imaginecup.com/

[17] International Collegiate Programming Contest. http://cm2prod.baylor.edu

[18] International Mathematical Olympiad. http://www.imo-official.org/

[19] International Olympiad in Informatics. http://ioinformatics.org/

[20] PC2 Home page. http://www.ecs.csus.edu/pc2/

[21] SPOJ0 training and grading system. http://judge.openfmi.net/spoj0

[22] Top Coder Competitions. http://www.topcoder.com/tc

[23] USACO Training Gateway. http://train.usaco.org/usacogate/

[24] UVa Online Judge. http://icpcres.ecs.baylor.edu/onlinejudge/

[25] Открытый кубок по програмированию. http://opencup.ru/

Krassimir Nedeltchev Manev
Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Oridski"
5, J. Bourchier blvd., 1164 Sofia, Bulgaria
and
Institute of Mathematics and Informatics,
Bulgarian Academy of Sciences
8, G. Bontchev str., 1113 Sofia, Bulgaria
e-mail: manev@fmi.uni-sofia.bg

Miloslav Antoniev Sredkov
Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Oridski"
5, J. Bourchier blvd., 1164 Sofia, Bulgaria

Tsvetan Krasimirov Bogdanov
Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Oridski"
5, J. Bourchier blvd., 1164 Sofia, Bulgaria
and
Tumblweed Communications,
Business Park Sofia, Mladost 4, Build. 11, fl. 2,
1715 Sofia, Bulgaria

# ОЦЕНЯВАЩИ СИСТЕМИ

Mathematics and Education in Mathematics, Proc. of 38-th Spring Conference of UBM, Borovetz, 2009.

# ЗА СЪСТЕЗАНИЯ ПО ПРОГРАМИРАНЕ

Красимир Манев, Милослав Средков, Цветан Богданов

Състезанията по програмиране възникват в 70-те години на миналия век за да привлекат талантливи млади хора към професията програмист. Скоро след организирането на първите състезания, системите за автоматична проверка на резултатите стават неразделна част от процеса на оценяване на състезателите. В последно време станаха известни различни приложения на оценяващите състезателни системи, които надхвърлят началната област на приложение на тези системи и повишават интереса към тях. В статията са разгледани целите и основните концепции на областта, архитектурата на типична оценяваща система и примери на такива системи. Очертани са перспективите и предизвикателствата, стоящи пред оценяващите системи, както при тяхното използване по предназначение, така и при използването им в други области.