

Structure, Scoring and Purpose of Computing Competition

Gordon Cormack, Graeme Kemkes, Ian Munro, Troy Vasiga

University of Waterloo

We identify aspects of computing competition formats as they relate to the purpose of these competitions, both stated and tacit. We consider the major international competitions – the International Olympiad for Informatics, the ACM International Collegiate Programming Contest, and top coder – and related contests whose format merits consideration. We consider the operational impact and possible outcomes of incorporating several of these aspects into scholastic competitions.

Introduction

Three international informatics contests – and national, institutional, and Internet contests modeled after them – dominate the landscape. The ACM International Collegiate Programming Contest [<http://icpc.baylor.edu>] (ICPC), in its thirtieth year, is a team competition for young post-secondary students. The International Olympiad for Informatics [<http://www.ioinformatics.org/>], half ICPC's age, targets high-school students who represent their country in individual competition. The TopCoder algorithm competition [<http://www.topcoder.com/tc>], established in 2001, is an open privately run on-line contest with a large electronic community. ICPC, IOI and TopCoder are all *algorithmic* contests in which students program solutions to word problems in traditional languages like Pascal, C++ and Java. There is substantial overlap in the style and subject matter of the problems addressed by participants in these contests; while none has a formal syllabus; a rough characterization might be:

programming problems involving college-level computing and mathematics, as well as associated fields such as operations research.

Skiena and Revilla's *Programming Challenges* [1] provides a good overview. Many individuals participate in (or are otherwise involved in) more than one; however, each has its own organization, community, and purpose. The stated purposes of IOI and ACM are to foster interest and community, and to promote achievement:

The ACM International Collegiate Programming Contest (ICPC) provides college students with opportunities to interact with students from other universities and to sharpen and demonstrate their problem-solving, programming, and teamwork skills. The contest provides a platform for ACM, industry, and academia to encourage and focus public attention on the next generation of computing professionals as they pursue excellence. [<http://icpc.baylor.edu/icpc/info/ppgs.pdf>]

The primary goal of the IOI is to stimulate interest in informatics (computing science) and information technology. Another important goal is to bring together exceptionally talented pupils from various countries and to have them share scientific and cultural experiences. [<http://olympiads.win.tue.nl/ioi/history.html>]

Only TopCoder has the explicit purpose of evaluating its participants:

TopCoder's mission is to create objective ratings that place high value on the

programming industries [sic] best and brightest, and build opportunity and community for programmers through ongoing programming tournaments and employer connections.

[\[http://www.topcoder.com/tc?module=Static&d1=about&d2=management\]](http://www.topcoder.com/tc?module=Static&d1=about&d2=management)

Contests in general serve a number of roles. They are sport, providing entertainment to spectators and participants. They recognize achievement. They illustrate, to students, educators and the public at large, the nature of the discipline. They serve as an incentive for students to enter the discipline, and to study and practice particular skills and techniques. They serve as a model for similar competitions, and drive preparatory contests and informatics curricula. They serve to build and support a community of participants, coaches and well-wishers.

The balance of roles in a contest involves many tradeoffs. Sports typically involve artificial rules and scoring methods that may be unreflective of any activity beyond the sport. Rules that involve time pressure or adversarial strategies may attract some participants and spectators while repelling others. Rules that introduce random or arbitrary components may aid *competitiveness* (in the American sense that any of a large number of competitors may win any given event) but compromise the recognition of true achievement. On the other hand such rules serve to counter the all-too-common inference that the winner is the best in the discipline; he or she is simply the winner of this particular event. Accolades and prizes for overall winners provide spectacle but little incentive or recognition for the majority who are not among the best. Similarly, tasks designed only to identify the best may fail to provide any valuable experience for the majority of participants. Lack of feedback may promote self-discipline and self-evaluation, reduce competitive pressure and introduce suspense as to the outcome; on the other hand it may increase frustration, diminish achievement and diminish spectator appeal.

The following sections describe the current structure of ICPC, IOI, and TopCoder competitions. Other competitions, computing and non-computing, are then summarized for contrast. Particular design issues are then elaborated, drawing examples from aspects of the various contests. Finally, several proposals are advanced for implementation within the context of existing structures. Each proposal is analyzed as to its compatibility with existing structures and the degree to which it might advance the purposes of scholastic computing contests.

Contest Structure and Scoring

ICPC. A team of three students shares one computer workstation in a five-hour effort to solve a number of problems, in English, given on paper. Each problem consists of a motivating application, a precise input and output specification, and some simple examples. The team may solve the problems in any order; a solution consists of a program written in a conventional programming language such as Pascal, C++, or Java, which must correctly handle all valid inputs. The source code for the solution is submitted electronically to the judges, who test it as exhaustively as possible on blind test data. If it runs in time and passes all test cases, it is accepted; otherwise it is rejected. No partial credit is given, but the team may resubmit a rejected program. When a program is accepted a penalty accrues – 1 point for every minute since the beginning of the contest, and 20 points for each prior rejected submission for the same problem. The final ranking is by number of problems solved, with penalty points as a tie-breaker. At the world finals, penalty points are recorded only for the top twelve teams; teams below the twelve with the same number of correct submissions are declared to be tied; teams below median are unranked. In addition, the following regional champions are declared, based on the same ranking, including penalty points (even for teams not in the top twelve overall): Africa and Middle East, Asia, Europe, Latin America, North America, South Pacific.

During the contest, an on-line scoreboard is available indicating the problems solved and penalty points for each team. Contestants and spectators (on-site and on-line) may see the scoreboard, but the scoreboard is not updated during the last hour of the contest, so as to preserve some uncertainty

pending the awards presentation. Balloons of different colours – one colour per problem – are affixed to the team's workstations as they solve problems. The balloons enhance the spectacle and allow observers to glean more information than shown on the scoreboard, especially during the last hour of competition.

Some regions release detailed final results, as well as the judge data and model solutions. The World Finals releases only summary scores consisting of number of problems solved and penalty points for the top twelve, number of problems solved for teams solving at least the median number of problems, and no ranking for teams solving fewer.

Recent ICPC World Finals have included a peripheral event – the Java Challenge or the Parallel Challenge. In this event, teams are given the specification for some sort of adversarial game (such as a war, demolition derby, etc.) and must, within a limited time, write a program to play the game. The various teams' entries then play one another in a live multi-round tournament that is projected during a reception on the evening preceding the contest proper. The winners of the challenge receive auxiliary awards; the result has no bearing on the ICPC standings.

IOI. Each country's delegation consists of up to four competitors. Individual competitors participate in two five-hour sessions on separate days. Each session involves solving three problems whose specifications are similar to those of the ICPC, but translated from English to the contestant's native language. Solutions must be written in Pascal or C++, and may be submitted electronically. However, no results are communicated to the contestant during the contest. He or she may submit or re-submit any of the problems at any time. Printed results are given to participants following each day's competition. In addition, judges' data is made available so that a participant may diagnose his or her errors, and appeal the score if he or she believes it is in error.

Unlike the ICPC, IOI awards marks for partially correct submissions. The majority of problems are judged by running the solutions on a series of test cases are graduated by perceived difficulty; the contestant scores ten points for each correctly-solved case. Typically, time limits are very tight, so that only the best-known algorithm to solve a problem will pass all test cases. (While the ICPC also enforces time limits, these limits are generous and a serious constraint for only a few of the problems; for most, any reasonable algorithm will do.) Some IOI problems are dubbed "output only." For these problems, like those of the IPSC, the input data is given to the contestants, and they submit the results rather than a program. Several distinct input files are given; the contestant must submit the result for each one. The inputs are chosen so as to present graduated levels of difficulty. Some IOI problems – whether "output only" are not, are open-ended, and scored on a continuous scale according to how well they solve the problem. These problems are dubbed "relative judging" because the scores are adjusted so that the best solution scores full marks.

No contestant ranking is disclosed until the final awards ceremony. However, more gregarious students are often able to deduce material portions of the ranking via the grapevine. At the awards ceremony, medals are given to those above median, in the order lowest-to-highest. About 1/12 of all contestants receive a gold medal; 1/6 receive silver; 1/4 receive bronze. In addition, the winner (or winners, in case of a tie) is traditionally recognized with a special prize from the host or a sponsor, but only the medal awards are elaborated in the rules.

TopCoder algorithm competition. Individual TopCoder members log in to an on-line development and judging environment. The environment includes "practice rooms" where a member may acquaint him or herself with past contest problems, and practice for the contest by solving them. The environment also provides "chat rooms" which allow for on-line discussion before, during, and after the contests. In addition, forums (formerly round tables) provide an archived bulletin board for discussing competitions, algorithms and related topics.

The contest itself takes place in three rounds – *coding*, *challenge*, and *system test*. During the coding phase, participants are presented with a menu of three problems that may be opened, each with a "point value" indicating its perceived difficulty. The problems resemble those of ICPC or IOI, but two of the problems are typically less algorithmically challenging than would be typical for these contests. Each solution consists of a Java, C++ or C# class which is embedded in a larger test program that invokes it. Unlike IOI or ICPC, solutions do not read or write files (or even standard input and output). Rather, they take their inputs as parameters and return their results. Extensive, but not necessarily exhaustive, test cases are available for testing. Contestants may also compose their own test cases using the contest environment. The TopCoder on-line system provides a complete environment for editing, compiling, and testing programs; however, competitors may paste in previously written code that they have on hand, or use "plug-ins" to the same effect. Contestants are proscribed from obfuscating their programs, or from including large amounts of unused code.

A contestant may submit a solution at any time. Contestants may open the problems in any order, but the point value for a problem begins to decay as soon as it is opened. When a solution is submitted, its point value is established, but the program is not evaluated until later. If it subsequently passes all tests, it receives the established point value, otherwise 0. If a contestant discovers that a submitted solution is wrong, it may be resubmitted, but its value will have decayed further in the interim, and an additional 10% penalty is applied.

The coding phase lasts 75 minutes. Following a short intermission, the challenge phase begins. Contestants can view their competitors' code, and may construct a test case which to challenge a specific submission. If the challenge succeeds (i.e. the submission fails on the test case) the challenger receives 50 points and the submission's score is set to 0. Otherwise the submitter receives 50 points.

Following the coding phase, system tests are run on all the submissions; those that survive all system tests are awarded the score that was determined at the time of submission.

Individual competition rankings are determined by the sum of submission and challenge scores. In addition TopCoder maintains an overall ranking of members based on formula that attempts to model the probability that a particular member will prevail over any other particular member in a given contest.

In addition to the on-line contests, TopCoder and its sponsors run several multi-level tournaments, with qualifying rounds done on-line and final rounds done on-site. On-line spectators may observe notices of the contestants activities – opening problems, compiling, testing, and submitting solutions, challenges and their results. The chat rooms provide a facility for spectators to observe and discuss these events, and to speculate and argue about their meaning. On-site spectators see a projected image of each contestant's computer screen.

Other computing contests. National and regional high-school contests are generally modelled after the IOI, with somewhat different emphasis. For example, the Canadian Computing Competition [<http://contest-cemc.uwaterloo.ca/ccc/>] (CCC) incorporates a distributed first-stage competition that is administered autonomously by individual schools. Its purposes include identifying several individuals for further competition (including the IOI), providing enriched education for gifted students, attracting students (girls in particular) to study informatics in high school and university, providing curriculum guidance for high schools, and fostering a community of computer science students and educators. The USA Computing Olympiad [<http://oldweb.uwp.edu/academic/mathematics/usaco/>] (USACO) administers its competition as an on-line contest. On-line derivatives of ICPC abound; the most well established is the Valladolid Programming Contest Site [<http://acm.uva.es/>]. The Internet Problem Solving Contest [<http://ipsc.ksp.sk/>] (IPSC) is an ICPC-style contest in which participants are given input data and must submit output data, as opposed to programmed solutions, thus allowing them to use any computing platform, tools and problem-solving strategy they choose.

Algorithmic programming contests in the style of ICPC, IOI and TopCoder emphasize a specific subset of the skills and techniques that comprise informatics. Are these the most important skills, or simply those that are easiest to measure? Are these timed contests attractive a broad population of potential participants, or do they select a narrow demographic? Are the contests appealing to an audience beyond the immediate participants and their acquaintances? Do the contests accurately convey – to students, educators, and the public at large – the nature of informatics? Bill Gates, for one, thinks not. In a recent visit to the University of Waterloo he said that contests were *hard core* and did a poor job of attracting a broad spectrum of talented individuals to study computer science [2].

Other informatics competitions and challenges diverge from the ICPC/IOI/TopCoder model. TopCoder's *component* competitions emphasize the process of designing and developing, over many days, reusable software components. Submissions are evaluated manually by filling in a structured *scoreboard* which is known to participants in advance. High-achieving designers are invited to serve as evaluators. The ICFP (International Conference on Functional Programming) Programming Contest [<http://icfpc.plt-scheme.org/>] promotes (but is not restricted to) the use of non-traditional programming languages to address open-ended problems. Participating teams prepare their submissions, in a language and development environment of their choosing, over several days. The relative performance of submissions, or their performance in a tournament, determines the winner. Various robotic and tournament competitions have been staged [<http://www.sumost.ca/steve/games/>]; for example, the RoShamBo Programming Competition [<http://www.cs.ualberta.ca/~darse/rsbpc.html>] is a tournament between computer programs, written in C, that play Rock-Paper-Scissors. The 24-Hour Programming Contest [<http://www.challenge24.org/>] requires participants to supply their own computing hardware and software, and to create several complex embedded applications in a 24-hour period. A number of companies have sponsored product- or application-specific programming challenges with a view to recruitment and promotion of their product.

Other competitions and challenges diverge still further, either in their format or in the skills required of participants. The J. W. Graham Computer Science Seminar [3] is an event in which 14-to-16-year-old girls with no prior programming experience are invited to attend an expense-paid week-long introduction to programming, computer hardware and other topics related to informatics. While these tasks involved tangible measures of accomplishment, they involved no *programming contest*, either before or during the seminar. Admission was, however, competitive; based on transcripts, teacher recommendations and written statements, 40 of 900 applicants were selected to attend in the first year. Some competitions and challenges use special-purpose languages to as to minimize complexity, to avoid requiring prior training with specific tools, and to integrate more readily into some tangible application. The J. W. Graham Seminar uses Tcl/tk so as to give immediate visual feedback for programming efforts. Statistical games, animation programming, robotic control scripting and games programming provide informatics challenges while avoiding programming *per se* [4-7]. Turing Machines - normally used only as an abstract model of computation - have been implemented and used as the basis for a programming contest [8]. Other competitions diverge in the opposite direction, emphasizing particular aspects of current methods in software development [9-11].

A particularly novel contest structure, which encourages collaboration over competition, is based on a wiki-like environment [12]. Participants post solutions to a shared wiki; each posting is evaluated when it is posted. Other participants are free to take others' submissions and improve upon them. In this way, the best overall solution constantly improves – sometimes incrementally and occasionally by leaps and bounds. Individuals' contributions to the process are visible but not explicitly rewarded

The Australian Informatics Competition (AIC) is a contest which requires no knowledge of programming. It is a paper-and-pencil contest with multiple-choice and short-answer questions that are designed to encourage algorithmic thinking. As an example, one question presents a game board together with rules for moving around the board. Students are asked to compute the minimum number of moves needed to reach any square.

Non-computing contests. Competitions in areas other than informatics are perhaps worthy of study. Verhoeff [13] describes the operation of the International Mathematics Olympiad (IMO) and contrasts it with the IOI. The IOI and IMO are two of the six International Science Olympiads [<http://olympiads.win.tue.nl/index.html>], the other four being Physics, Chemistry, Biology and Astronomy.

Mathematics is perhaps most closely associated with informatics, due as much to history as to common subject matter. Mathematics has a solid educational tradition; students from all countries have the opportunity to receive formal education in standard mathematical subjects. While the IMO has no prescribed syllabus, tasks are drawn from a de facto list which avoids post-secondary topics such as calculus. Six long-answer problems are posed over two competition days. Each is graded manually on a seven-point scale: 7 points are awarded for a perfect answer, 6 for an answer with some insubstantial flaw; 5 for a small mistake, 2 for significant progress, 1 for non-trivial progress, and 0 for busy work (including special cases). Intermediate scores are awarded only in special circumstances. This coarse scoring method (42 possible outcomes) ensures that there will be many tied scores among 200 or more participants; in particular ties for 0 and 42.

Elementary contests often use more structured formats. The CCC, for example, is administered by the Centre for Education in Mathematics and Computing (CEMC) [<http://www.cemc.uwaterloo.ca/>] – the same organization that has run the Canadian Mathematics Competitions for decades. These competitions are specifically targeted to students in grades 7 through 12. The early competitions consist of 25 multiple-choice problems, reflective of the curriculum, with graduated difficulty. Later contests include written proofs, in the style of the IMO, which are graded manually by CEMC. Prior to grade 9, students are recognized only in their schools; in later years, national results are published and the highest ranking contestants are invited to attend a seminar and second round of competition at the University of Waterloo. The CCC, on the other hand has a Junior Contest and a Senior Contest, with participants selected by experience not grade. Both are algorithmic programming contests in the IOI style. As for the mathematics competitions, winners are invited to a second round at the University of Waterloo. Although the CCC has occasionally included written questions, it is essentially a programming contest. It has nothing to offer grade 7 and 8 students, or for that matter, any student who has not learned to write a complete program from scratch in some algorithmic programming language.

The International Physics Olympiad (IPhO) [<http://www.jyu.fi/tdk/kastdk/olympiads/>] embodies theoretical and experimental tasks on separate competition days. A formal syllabus prescribes the subject areas from which tasks are drawn. The tasks, originally at a par with hard high school problems, now extend well beyond – comparable that might be tackled by a college major in physics. Tasks are posed as multi-component problems with graduated difficulty. The International Chemistry (IChO), [<http://www.icho.sk/>] like the IPhO, includes theoretical and practical tasks with graduated components and has a detailed syllabus. Topics in the syllabus are identified as one if (1) standard high-school curriculum material, (2) enriched high-school curriculum material, and (3) extra-curricular material not to be used unless introduced in preparatory tasks. The International Biology Olympiad (IBO) [<http://www.ibo-info.org/>] mirrors the format of the IPhO and IChO, but specifies proportions of each major subject area that must be represented in the tasks.

The International Astronomy Olympiad (IAO) [<http://www.issp.ac.ru/iao/>] is quite different from the others, occasioned by the fact that astronomy is rarely a component of high-school curricula. The competition therefore targets youth with a junior contest for 14-15 year-old and a senior contest for 16-17 year-old competitors. The rationale is to be the impetus to introduce and nurture interest in astronomy among a group who would otherwise see little opportunity. Accomplished older students of astronomy are expected to find other means, such as research symposia, of furthering their interests.

Finally, one may consider the nature of the appeal of cerebral contests outside the realm of science. Quiz shows and quiz games have enduring popularity. Reach for the Top

[\[http://www.reachforthetop.com/\]](http://www.reachforthetop.com/) has engaged Canadian high-school students (including the author) since 1961. Trivial Pursuit debuted as a smash hit, as did the National Trivia Network (NTN) online quiz contest [\[http://www.ntn.com/\]](http://www.ntn.com/). On-line poker, including NTN's version, is the current rage. Chess is widely played for recreation, and professional chess draws a wide audience. Ironically, chess provides a highly visible computer application – the 1996 ACM General Conference brought together Gary Kasparov, Deep Blue, and the ICPC world championship.

Contest Design Issues

We believe that contest designers would be well advised to consider contest design in light of specific objectives and constraints that are framed as independently as possible from the aspects of design that advance those objectives within the constraints. To this end, we offer the following points of discussion.

The need to identify a winner. The scientific olympiads in general identify sets of gold, silver, and bronze medal winners. However, they publish complete rankings (at least of the medalists) and there is some tendency to compare rankings of individuals and aggregate standings for the members of a particular country's delegation. Sponsors and promoters like to single out winners for prizes and attention. Complete rankings and the naming of a winner make serve to enhance the spectacle of the event; on the other hand, complete rankings belie the importance and accuracy of the contest scores. Excessive focus on a winner may detract from recognizing the achievements of others.

If one accepts that it is desirable to identify a (unique) winner, one should be careful in choosing the mechanism to do so. One approach is to make the tasks so difficult that it is nearly impossible for anyone to solve them all. If all the problems are at such a level (as they have been for several recent IOI competitions) they offer little to most participants by way of fostering achievement. Even the winners are expected to *fail* on several of the problems.

Pressure elements. A number of elements of competitions exacerbate the pressure felt by participants. These elements may enhance the entertainment value and appeal to some participants (primarily boys), but may discourage others. Such elements include onerous time limits, scoring by time, adversarial tasks and publication of total rankings.

Unpredictable evaluation and lack of feedback may also cause pressure, with little offsetting entertainment value. A participant's ability to guess how his or her performance will be evaluated should not be a determining factor; neither should a participant's ability to guess what subject matter might be covered, or participant's ability to *play the odds* to account for random factors.

Some elements of strategy may cause pressure but also foster valuable informatics skills. Blind testing, for example, emphasizes validation and testing. But all-or-nothing scoring (e.g. ACM and TopCoder), or after-contest testing (e.g. IOI and TopCoder) in which the participant has no opportunity to learn from and correct his or her errors, add considerable pressure and may not be the optimal method to foster the desired skills.

Prior knowledge vs. skill. Programming contests demand some combination of aptitude, prior knowledge and skills. The major competitions emphasize algorithmics – knowledge of existing algorithms, their application in new circumstances, their realization in some particular development environment, efficiency analysis, testing and debugging. Memorization of existing algorithms (both common and obscure) and their realization in a particular environment can, but should not, we suggest, dominate the other factors. Other aspects of informatics, such as databases, operating systems, user interfaces, software engineering and theory are addressed only insofar as they are embedded in programming tasks. Should they be included more explicitly?

The major contests use a standard development environment with conventional programming languages, editors, compilers and debuggers. Participants may be considerably advantaged or disadvantaged by prior experience with these tools. It is therefore important that all participants have equal access to the tools, exactly as used in the contest. It is also important to consider that the choice of tools constitutes an endorsement.

Similarly, the choice of tasks serves to illustrate the nature of informatics to contestants and to spectators. This purpose is well served when the applications and their solutions are realistic, and the inputs and outputs of the task can be readily seen to relate to the application.

Regardless of which skills and subjects are emphasized, participants' abilities will vary considerably. It is desirable that participants of all abilities are challenged and have an opportunity to demonstrate accomplishment.

Collaboration vs. competition. Team competition fosters collaboration to a certain extent. More general collaboration is traditionally confined to non-contest activities. For example, IOI 2001 included an activity in which countries' delegations paired in a scavenger hunt. Is it appropriate to project the image that social activities are collaborative whereas informatics is competitive?

ICPC's Java/Parallel challenge offers a chance for teams to write a software robot that competes on their behalf. Teams can discuss their strategy with coaches and use any other resources they please – they can even collude with other teams. However, the contest still involves extreme time pressure, the contestants teams do not have a chance to revise their submissions after observing tournament performance, and the animated activities in which the robots compete are modeled after the most combative of video games, in which attacking foes is an essential element for success.

Spectators, scoreboards and blackouts. Programming contests naturally draw spectators. Coaches, family members and peers are naturally attracted, regardless of the intrinsic sporting appeal of the contest. Those interested in computing may find the contest interesting in its own right. School, regional or national pride may elicit a following. A primary goal of scholastic contests – to attract interest – is well served by supporting and expanding the spectator community. Sponsors' interests are served as well.

Specific Proposals

Real-time feedback. Many major competitions use on-line submission and automatic judging. Within this context, real-time contestant feedback is readily implemented. This feedback may be an essential part of the scoring method, as in ICPC, or it may be strictly to assist contestants in assessing their own progress, thus reducing rather than exacerbating time and competitive pressures. In the context of the IOI, the following sources of feedback should be considered:

On-demand scoring. Contestant may submit his or her solution for testing at any time, in accordance with current IOI practice. In addition, contestant may request *final judgment* which runs the submission on the official judge data and reports summary results to the contestant. On-demand scoring by itself offers little advantage to contestants unless combined with incremental scoring or resubmission.

Incremental scoring. When contestant requests *judgment* his or her submission is run on the judge test cases in some predetermined order, stopping with the first unsatisfactory judgment. Summary results, and perhaps the test data and correct output, are returned to the contestant. Contestant may revise his or her solutions; further judgment requests apply the submission to

previously-untested cases.

Resubmission After receiving feedback, contestant's submission is re-scored on test cases for which an unsatisfactory judgment was previously returned, possibly for a reduced score. Resubmission requires that the test data – at least the correct output – not be returned to the contestant. So there's a tradeoff between two valuable opportunities – the opportunity to learn from incorrect submissions and the opportunity to repair mistakes.

"Buy" a test case. Contestants may, by forfeiting the right to resubmit, gain access to the input and output data for a particular case. This affords the contestant the opportunity to evaluate the tradeoff between learning and repairing; however, this very choice introduces a strategic element that is tangential to the principal focus of the competition.

Unscored mock judgment Contestant may request *mock judgment* which runs his or her solution on data resembling the judge data. Results are returned to contestant, but have no bearing on contestant's score. Mock judgment may be fairly exhaustive, in which case contestant may reasonably assume that satisfactory mock judgments predict a good score on the real test data. Or the mock judgment may omit certain difficult cases, in order to promote analysis. Some combination of exhaustive and non-exhaustive mock tests may be used for different tasks; however, the task descriptions should indicate which is provided. Mock judgment may use any of the feedback mechanisms outlined above.

Scoring of sample cases. Contestants are given a number of sample inputs and submit only the results, which may be derived using an ad hoc combination of manual and automatic computation. These serve to reinforce the contestant's understanding of the problem, independent of discovering and implementing a general efficient algorithm.

Test case creation task. Thorough analysis and testing may be encouraged by rewarding the composition of effective test data. This objective may be achieved with additional submission and scoring elements that complement existing evaluation methods, and the feedback mechanisms proposed above.

Test case submission. Contestant composes both input and output, which are run in the judge environment using a standard implementation. Validity of the submitted input and output is evaluated and reported to the contestant.

Test case scoring. Contestant scores could be based, in part, on successful test case submission. The intended effect is to encourage testing and the creation of good test cases. Evaluation criteria must support this effect. Of course, test cases should have valid input and output, but they should also expose special cases likely to expose errors. To this end, we suggest that test cases could be evaluated with respect their ability to correctly detect erroneous solutions – these erroneous solutions may be composed by the judges, or the actual submissions. Using the actual submissions need not be adversarial (as it is in TopCoder's challenge phase), as only statistics need to be reported, and only the scores of the test cases (not the scores of the program runs on the test cases) are affected. We suggest that an appropriate score would be the fraction of all incorrect submissions detected by a suite of test cases. If the effectiveness of test cases is evaluated on the basis of other all contestants' submission at the time of testing, final test case scoring could be computed only at the end of the contest. Feedback in this regard should be clearly labeled *provisional*.

Test case wiki. Collaboration may be encouraged by posting all submitted test data to a shared data space available to all contestants. Data sets may be anonymous, credited, or credited using a pseudonym. Data sets may be augmented by their current score – the fraction of incorrect

submissions (to date) that they correctly detect. Data sets may be ranked by their effectiveness; that is, the wiki forms a *ladder* tournament. Contestants are free to use any of the data sets, and may augment or enhance them to form better ones. Contestant's test cases may be scored in various ways, such as the incremental improvement of their submissions over previous ones, or the popularity of their test suites as measured by the number of other contestants that use them, or the popularity by vote, or simply the effectiveness of their final submission as detailed above. Careful consideration should be given to any scoring mechanism to ensure that anti-collaborative tactics, such as withholding submissions to the last minute, are discouraged.

Graduated difficulty. Of the major computer contests, only IOI presents contestants with a few very hard problems – very hard in that only a minority, generally a tiny minority solve them (i.e. provide a correct solution that meets all the specified constraints). This means that the vast majority of submissions are *partially correct*; in other words, *incorrect*. The overarching effect of scoring is to reward incorrect solutions with part marks. We believe that this situation has arisen partly due to the small number of tasks, and the fact that the tasks are not divided into subtasks of graduated difficulty. We believe that problem difficulty has risen in a quest to avoid (ties for) perfect scores, further exacerbating the problem. Even if we stipulate (which we do not) that ties for perfect scores are a problem, we advance measures of graduated scoring as being more appropriate than across-the-board increases in difficulty.

Larger problem sets covering a range of difficulty. ICPC uses a larger number of tasks, ranging in difficulty from straightforward to very challenging. Tasks have equal scoring weight and are not ordered or otherwise labeled by difficulty; therefore, difficulty assessment is an element of strategy. ICPC espouses the design principle that, in a good contest, every team should solve at least one problem, every problem should be solved by at least one team, and no team should solve all problems. The difficulty of IOI tasks is comparable to the hardest of ICPC tasks. Yet IOI medalists do not automatically succeed at ICPC, even though ICPC provides feedback, in large part because ICPC demands total correctness. TopCoder provides tasks with a range of difficulty; the assessed difficulty of each task is stated, and the score for each task depends on its difficulty.

Speed of submission as an element of score. Time pressure is a large element of the TopCoder algorithm competition. Contestants who do the easy problems quickly but cannot do the hard problem often outscore contestants who do the hard problem more slowly. We are not convinced that this emphasis on speed offers the right balance. ICPC's balance, which rewards speed but not nearly so heavily as TopCoder, may be appropriate for the college level, but we suggest that speed should not be an explicit factor in scoring high school competitions. Nevertheless, speed offers a more appropriate measure of accomplishment than scoring incorrect submissions.

Adjudication of near correctness or significant progress. IMO uses a small number of tasks and is therefore subject to some of the same criticism. However, IMO places no emphasis on identifying a unique winner, and scoring seeks to identify *nearly correct* solutions which, according to a human assessor, contain only a minor mistake. This judgment method is quite different from counting the number of cases for which the solution works (e.g. it may be nearly correct but work for no cases, or completely off track and work for several). Nevertheless, the format of the IMO has competitors in isolation, with no feedback, searching for solutions to a very small number of hard problems. We see no easy way to incorporate the adjudication of near correctness or significant progress within a fully automated judging system. A semi-automatic judging system like that used by ICPC, combined with scoring guidelines like those used by the IMO, might be an appropriate way to effect this approach. Task statements, for example, might encourage participants to submit as comments the algorithms chosen, rationale (or even proof) along with their submission. These could be the basis for assessing significant progress, which inspection or even modification and testing of the code might be the basis for

assessing near correctness. These activities are done routinely (but not formally or systematically) by ICPC (and, we believe, IOI and TopCoder) judges.

Multi-part problems. The other science olympiads tend to use multi-part problems in which the first parts are relatively straightforward, and successive parts develop the theme of the task. Each step is individually correct or not, so partial scores are based on number of correctly answered components. Most IOI tasks could be formulated as multi-part problems. Some parts might involve solving small cases by hand, or answering other questions relating to the task. We suggest that the now-infamous fifty-percent-rule, if applied, should be handled this way – the task should explicitly have one part which involves solving the problem with weaker constraints. The fifty-percent rule may be considered a very coarse and limited special case of the multi-part approach. But it introduces an extraneous element of strategy – the contestant must decide whether to try for a single implementation that solves both parts, or to solve them separately. Better to have different, if related, sub-tasks than simply to vary the size of the problem.

Speed of execution as an element of score. Efficiency is an important element of informatics, but so is correctness and any number of aspects. Execution speed is not a particularly good measure of efficiency, as it depends on the vagaries of the implementation of language and run-time features like input/output, maps and memory management subsystems. The specification of tight limits telegraphs information about the expected solution to the problem, potentially biasing the contestant's search for a solution. IOI uses very tight per-question time limits which are a challenge to meet for almost every task. We believe that this situation arose as the mechanism to award partial scores, a mechanism whose appropriateness we question. In contrast, ICPC and TopCoder use a fixed time limit – it is up to the contestant to determine, for each task, whether or not this limit is a serious constraint. For most problems, any reasonable algorithm will do. Plenty of challenge remains in designing and implementing a reasonable algorithm.

Open-ended tasks. Open-ended tasks provide a continuous or near-continuous score, which is attractive in avoiding ties and in rewarding a wide diversity of achievement. Such an approach – dubbed *relative scoring* – has been used occasionally for individual tasks in IOI competitions. The name is perhaps misleading, underscoring the adversarial rather than continuous aspect of scoring. There is no particular reason why such tasks must be graded *on the curve* as implied by the term. The important aspect is that there is no known optimal solution to a problem, and points are awarded for correct submissions in proportion to how well they do. Past relative scoring tasks have involved precisely defined, but intractable, optimization problems. A vast number of real-world applications, such as pattern recognition, information retrieval compiler optimization, admit no precise mathematical formulation, yet are suitable for this purpose. Although they may admit no mathematical specification, it is easy to evaluate submissions in an objective manner by applying them to data derived from real sources. It is also possible to describe a basic approach which, if implemented correctly, will yield a baseline score. Such tasks appear to be good illustrations of real informatics problems. A sample task, adapted from one used by the Canadian Computing Competition, appears in the appendix.

Tournament-based evaluation. Contestants write a program to play some game for two or more players. Contestant's entries are played in a round-robin tournament and scored by their win ratio. While there is an adversarial aspect to this method, it may be mitigated by the fact that proxies (i.e. the submitted programs) do battle rather than the contestants. Tournament scoring has been proposed but never used, to your knowledge, for IOI. It is easy enough to implement within the current environment. As with scoring test suites by effectiveness, feedback could provide at best provisional scores.

Practice contest. The practice contest introduces contestants and potential contestants to the contest,

and, in this capacity, should be as representative as possible.

Practice tasks are first-class citizens. All too often, practice problems are posed as an afterthought, and not subject to the same quality assurance process as the contest problems. Often the problems are utterly trivial, or relate to subject matter outside the scope of the contest. As an example of the former, the ICPC final every year uses for practice a task paraphrased as *given two positive integers a and b , compute the sum of the integers between a and b , inclusive.* The mathematics and programming are trivial, but the problem contains a devious trap: a may be greater than b . Every year this causes new teams untold fear and frustration for no reason – ICPC finals may be counted on to identify winners by their ability to solve difficult problems, not to avoid devious traps.

Tasks deemed inappropriate are inappropriate for practice. If a task is rejected as a competition task because it contains inappropriate subject matter, it should be rejected as a practice task as well.

Practice tasks may introduce or foreshadow new material. IPhO guidelines state explicitly that extra-curricular material must be introduced in the practice tasks. This approach provides a well-defined mechanism by which the subjects addressed by tasks may evolve. The practice serves notice that certain techniques may be required in the contest proper; sample solutions should provide a solid explanation of the new material.

Collaborative activities. The informatics aspects of the IOI emphasize individual accomplishment. The goal of providing shared scientific and cultural experience is largely addressed by ceremonies, field trips and leisure activities. Some of the leisure activities, like scavenger hunts and other games, foster interaction and collaboration. We are concerned here with informatics-oriented collaboration, both within and outside the contest proper.

Wiki-based activities. We have proposed above a scheme for wiki-based test data creation which has a high degree of collaboration within the context of the competition. Other wiki-based activities might be ongoing, with every participant having the opportunity to contribute to some grand challenge. Or the participants could be divided coarsely into, say 4 teams with each team having one member from each country. Tasks could be open-ended, like the spam filter task, or more design-oriented, like creating an interactive application to illustrate the IOI. Results from each year could be published on the web so that each year's participants would have an incentive to out-do the previous year.

Informatics-based leisure activities. Although competitors certainly need a rest between competition days, a change can be as good as a rest. Scavenger hunts may be adapted to solve informatics problems – perhaps of the sort pioneered by the AIC which require no computer. A software robot tournament might be appropriate – multi-country teams, perhaps with the aid of a wiki, strive to write robots. The tournaments are projected on a screen for the entertainment of all. Such a tournament could be modelled after the ICPC challenge, amended to remove time pressure and to make the tournament less resemble a boy's combat video game.

Entry-level contests. Programming contests modelled after the IOI competition require that students be familiar with a programming language and environment. These contests are accessible only to students who have already studied computer programming. But it may be wise to design additional contests that will be accessible to a broader audience.

Among the six science Olympiads, we find ourselves in a distinct, perhaps unenviable, position. Mathematics, Physics, Chemistry and Biology programs at the high school level are well established. Teachers of these subjects are generally well qualified; they understand what the subject is about and

can easily point to appropriate advanced material for the gifted student. Astronomy is different; students pick it up as a hobby, without school courses and in most countries will actually move toward the subject

at the university level by first studying physics. We are different. There are, in many, though not all, school systems, high school courses, typically taught by dedicated teachers with little formal training in the subject. Computer Science is a subject that offers a career to a very large number of students and hence there are large CS programs at many (most) universities. Unlike the older sciences, students not only lack solid courses and guidance for choosing it as a university program, but are also subjected to misinformation and a negative impression of its intellectual content. As a consequence, we often attract the wrong students at the university level. A goal of computing contests would seem to be in addressing this issue. Our contests have the potential to educate a broader audience about the intellectual challenge of computer science. This audience should include students who have not learned computer programming, as well as parents, teachers, and other mentors.

This brings us to our proposal: to study how to design a novice informatics contest with no programming prerequisites. For such a contest, no knowledge of a programming language or environment may be assumed. Students with no background in computer science could write this contest and learn about some of the techniques and challenges of our Computer Science.

The Australian Informatics Competition (described above) is an excellent example of an entry-level paper-and-pencil contest. Can we design a competition that still uses the computer, but in a way that requires no prior knowledge of programming? By retaining the computer in the contest, students can formally specify their procedures (using a self-contained environment defined within the contest) and enjoy interactivity and feedback.

Spectators are essential not peripheral. Spectator involvement should be an explicit objective of contest design and event organization. Media (traditional and non-traditional) should be used to communicate the key messages of the IOI and to involve spectators in following events that reinforce these key messages.

Spectator appeal depends on the sporting nature of the contest. The rules and elements of strategy must be transparent enough and interesting enough that the spectator generally understands and is engaged by what is going on. Spectator appeal depends on communication: the contest must be advertised, so that spectators know where and when it is taking place, how to follow it (by attending, following on-line or reading about it after the fact), and the basic theme, rules and elements of strategy. Announcements and coverage should take place at specific times and places, which should be well advertised in advance.

Live coverage can be an important element of spectator appeal. This can involve observing participants, or their progress, or both. The ICPC finals offers a gallery from which on-site spectators may observe participants, as well as balloons and an on-line scoreboard by which the teams' progress may be tracked. The scoreboard is available on-line, but its location (or, indeed, its existence) is undisclosed prior to the contest. The scoreboard is frozen one hour from the end of the contest, so as to maintain some uncertainty as to the final rankings pending the awards ceremony. The awards ceremony may itself could be a spectacle, but in recent years has been a private ceremony involving only immediate participants. It is not clear that spectator appeal is enhanced by deferring suspense from the contest to the awards ceremony. The IOI provides no live coverage, and no feedback to participants. Individuals are informed of their results, but collected results are withheld pending the awards ceremony. TopCoder's on-line competitions provide extensive live coverage to members who log in to the contest environment. Spectators can view participant's actions but not the problem sets. Spectators can discuss the action in chat rooms. The final results are available within the environment at an unpredictable time – the end of system testing. Public results are available much later, also at an unspecified time.

Analysis, commentary and personal interest may also contribute to spectator appeal. Technically savvy spectators may like to avail themselves of the task statements so as to *play along* by formulating solutions or to analyze them so as to speculate about possible competition outcomes. Other spectators may nevertheless find the task statements interesting and educational, and may benefit from expert commentary on the underlying problems that they expose, and possible strategies for their solution. Solutions, judging data and contestant submissions may provide fodder for additional analysis and commentary. Personal interest appeal draws from the lives and personal stories of those involved with the contest. An article by Sean Silcoff, a reporter who accompanied Waterloo's ICPC team to the 2000 world finals, captures some of the drama that might better be exposed to spectators.

Hanging over the Waterloo team like a dark cloud is Problem F. Lhoták has tried it twice and still hasn't cracked it. The problem should be the easiest of the contest, but it's confounding a lot of teams. What most of them don't realize is that there's something wrong with the data. But Lhoták doesn't give up. He figures a way around the discrepancy. At the 156-minute mark, the judges accept his submission, giving Waterloo its second correct answer. Relief. Twelve minutes later, Waterloo scores its third. Excitement. The team is back in the game, and suddenly in fifth place. Seventeen minutes later, Cheung solves the rubber-stopper stumper. The team vaults into second. Jubilation.
<http://www.canadianbusiness.com/article.jsp?content=10326>

References

- [1] Skiena S. and Revilla M., *Programming Challenges - the Programming Contest Training Manual*, Springer-Verlag, New York, 2003.
- [2] Gates, Bill, *spoken comment at faculty round table*, University of Waterloo, October 13, 2005.
- [3] Graham, S. and Latulipe, C. *CS Girls Rock: Sparking Interest in Computer Science and Debunking Stereotypes*, Proceedings of SIGCSE '03, Reno, February, 2003.
- [4] Rodger, S. and Walker, E., *Activities to attract high-school girls to computer science*, SIGCSE '96, Philadelphia, February, 1996.
- [5] Werner, L. et al, *Middle School Girls + Games Programming = Information Technology Fluency*, SIGITE '05, Newark, October, 2005.
- [6] Roberts, E., *Strategies for encouraging individual achievement in introductory computer science courses*, SIGCSE 2000, Austin, March 2000.
- [7] Ladd, B. and Harcourt, E., *Student competitions and Bots in an introductory programming course*, Consortium of Computing Sciences in Colleges, 2005.
- [8] Brogi, A. *A Turing Machine Contest for Introducing High School Students to Computer Science*, SIGCSE Bulletin 29:1, June 1997.
- [9] Adrianoff, S. et al., *Adding Objects to the Traditional ACM Programming Contest*, SIGCSE '04, Virginia, March, 2004.
- [10] Adrianoff, S. et al., *A Testing-Based Framework for Programming Contests*, OOPSLA 2003 Workshop on Eclipse Technology eXchange, Anaheim, 2003.
- [11] Sherrell, L. and McCauley, L., *A programming competition for high school students emphasizing process*, 2nd Mid-South College Computing Conference, Little Rock, 2004.
- [12] Gulley, N., *In praise of tweaking: a wiki-like programming contest*, Interactions 11:3, May/June, 2004.

Appendix – Sample Open-Ended Task – Spam or Not Spam?

Unwanted, unsolicited email (spam) is annoying and clutters your mailbox. You are to write a *spam filter* - a program that reads email messages of regular ASCII characters and tries to determine whether or not each message is spam.

How can we determine whether or not a message is spam? Spam contains words and phrases that are not common in genuine email messages. For example, the phrase

MAKE MONEY FAST, HONEY!!

is in all-uppercase, contains the word "money" and ends with a double exclamation mark.

One way to create a spam filter is to read through many spam and non-spam messages and to come up with a set of rules that will classify any particular message as spam or not. This process can be tedious and error prone to do manually. Instead you will write a program to automate the process.

A useful step in automatic classification is to split the text up into set of *trigrams*. A trigram is a sequence of three adjacent characters that appear in the message. A trigram is case sensitive. The example above is composed of the trigrams:

MAK
AKE
KE
E M
MO
MON
ONE
NEY
EY
Y F
FA
FAS
AST
ST,
T,
, H
HO
HON
ONE
NEY
EY!
Y!!

If we examine a sample of spam and non-spam messages we find that some trigrams are more common in spam; whereas others are more common in non-spam. This observation leads to a classification method:

- Examine a sample consisting of a large number of spam messages. Count the number of times that each trigram occurs. In the example above, there are 20 distinct trigrams; the trigrams ONE and NEY occur twice each and the remaining 18 trigrams occur once each. (Trigrams that do not occur are considered to occur 0 times.) More formally, for each trigram t we compute the frequency $f_{spam}(t)$ with which it occurs in the sample of spam.
- Examine a sample consisting of a large number of non-spam messages. Compute $f_{non-spam}(t)$, the frequency with which each trigram t appears in the sample of non-spam.
- For a each message to be filtered, compute $f_{message}(t)$ for each trigram t .
- If $f_{message}$ resembles f_{spam} more closely than it resembles $f_{non-spam}$ it is determined to be spam; otherwise it is determined to be non-spam.
- A *similarity measure* determines how closely f_1 and f_2 resemble one another. One of the simplest measures is the cosine measure:

$$\text{similarity}(f_1, f_2) = \frac{\sum_t f_1(t) \cdot f_2(t)}{\sqrt{\sum_t [f_1(t)]^2} \cdot \sqrt{\sum_t [f_2(t)]^2}}$$

Then we say that a message is spam if

$$\text{similarity}(f_{\text{message}}, f_{\text{spam}}) > \text{similarity}(f_{\text{message}}, f_{\text{non-spam}})$$

You are to use this method, or a better method of your own design, to filter spam.

The first line of input contains three integers: s the number of sample spam messages to follow; n the number of sample non-spam messages to follow; c the number of messages to be classified as spam or non-spam. Each message consists of several lines of text and is terminated by a line containing "ENDMESSAGE". This line will not appear elsewhere in the input, and is not considered part of the message. The messages will be real email messages, and will contain only printable characters, but will not all be English.

For each of the c messages, your program will output a single line giving the classification of the message ("spam" or "non-spam"). If your program runs without error and outputs c lines, each containing "spam" or "non-spam" it will receive a score proportional to the number of messages that it correctly classifies; otherwise it will receive a score of zero.

Sample Input 1

```
2 1 1
AAAA
BBBB CCCC
ENDMESSAGE
BBBB
ENDMESSAGE
AAAABBBB
ENDMESSAGE
AAABB
ENDMESSAGE
```

Output for Sample Input 1

```
non-spam
```

Sample Input 2

Found in the file [d1q1sample.txt](#)

Output for Sample Input 2

```
spam
non-spam
```