Encouraging algorithmic thinking without a computer

Benjamin A. Burton*

Abstract

At the secondary school level, traditional programming competitions remain inaccessible to the vast majority of students. We describe the Australian Informatics Competition (AIC), a pen-and-paper event that is accessible to a much broader audience but still retains a core focus on algorithms. In addition to multiple choice questions, a unique feature of the AIC is its three-stage tasks that invite algorithmic thinking by posing similar problems of increasing size. In this paper we describe the AIC, the design decisions behind it, and the types of problems that it contains.

1 Introduction

Competitions and enrichment programmes in computer science have enjoyed a sharp rise in popularity over recent decades. Consider for instance the International Olympiad in Informatics (IOI)—although much younger than many of the science olympiads, the IOI has grown to become the second largest of these international events.[1] Non-competitive programmes have also enjoyed an enthusiastic reception, such as Australia's National Computer Science School (`http://www.ncss.edu.au/`).

Despite this popularity, many national informatics olympiad programmes struggle to find students at the secondary school level (Anido and Menderico, 2007; Boersen and Phillipps, 2006; Choijoovanchig et al., 2007; Pohl, 2007). In Australia the numbers are striking—the national entry-level mathematics competition enjoys several hundred thousand participants each year, whereas the entry-level programming competition for the informatics olympiad attracts just one or two hundred.

Several factors contribute to these extremely low rates of entry for national programming competitions:

(i) *Curriculum:* In many countries, computer programming and algorithm design receives very little attention in the secondary school curriculum (Verhoeff, 2009). In comparison, mathematics and other science olympiad disciplines (such as biology, chemistry and physics) are well-taught and widely studied. As a result, there are far fewer students with the necessary skills for a programming competition; moreover, it is difficult for teachers to identify who these students are.

(ii) *Technology:* Programming contests require students to have dedicated access to a computer in exam conditions, often for several hours during the school day. This can make scheduling and supervision difficult for teachers, and (depending on school resources) can severely limit the number of entrants from any given school.

*School of Mathematics and Physics, The University of Queensland, Brisbane QLD 4072, Australia (bab@maths.uq.edu.au)

[1] Measured by the number of attending countries in 2009.

(iii) *Grading:* For programming contests that employ traditional computer-based grading, a student cannot score any points at all if they cannot produce a working program that reads from an input file, writes to an output file and does some work in between. This is a significant barrier to scoring, particularly given the poor curriculum support mentioned above. The result is often a large number of low or zero scores, discouraging inexperienced students from participating and disheartening those who do.

If we are to make informatics competitions accessible to a significantly broader audience, we should strive to address all three of these difficulties. A natural solution is a pen-and-paper contest, with multiple choice or short answer tasks that are quick to solve (whether correctly or incorrectly), highly approachable without any prior knowledge, and ranging in difficulty from challenging to very easy.

Several countries have adopted such solutions in recent years, For example, South Africa has offered a pen-and-paper contest since 2003 with high rates of participation (Merry et al., 2008), and Australia introduced the pen-and-paper Australian Informatics Competition in 2005 (Clark, 2006). Lithuania introduced the Bebras (Beaver) contest in 2004 (Dagienė, 2006); this contest depends on computers but in a much more accessible way, and it has since spread to several European countries. Predating any of these events, Bulgaria has included informatics problems in their multiple choice mathematics contest Chernorizets Hrabar since 1992 (Tabov et al., 2003).

The greatest difficulty with such a format is retaining the focus on algorithms and algorithmic thinking. Particularly with multiple choice, it is challenging to find questions that do not rely on pre-assumed knowledge (such as programming languages or pseudocode), but which nevertheless have more of an algorithmic flavour than traditional mathematics problems and puzzles.

In this paper we describe how Australia has responded to these challenges through the Australian Informatics Competition (AIC). Held annually since 2005, the AIC has a clear focus on algorithms, yet maintains an informal pen-and-paper setting with no required knowledge. The contest employs a mix of multiple choice and integer answers, and incorporates unique "three-stage tasks" that encourage students to develop informal algorithms by posing similar problems of increasing size.

In Section 2 we outline the structure and design of the AIC. Section 3 describes how the AIC maintains its algorithmic focus in this informal setting, and offers examples of different question types including the three-stage tasks mentioned above. The concluding notes in Section 4 include statistics that show how well the AIC has been received.

2 The Australian Informatics Competition

The AIC is offered in three divisions that span all of secondary school (years 7–12 in Australia). Each division consists of 15 questions: six multiple choice questions requiring a single letter from A to E, and nine integer answer questions requiring a single integer from 0 to 999. The integer answers are used for the three-stage tasks (described in the following section), and also serve to limit the points that students can achieve through guessing (as discussed below).

The contest is designed to minimise the burden on both teachers and students:

- For teachers, the contest is easy to administer. It runs for just one hour, and students can sit the contest at their desks with nothing more than pencil and paper (calculators are optional but unnecessary). This make it possible for a teacher to give the contest to an entire class during a single lesson.

- For students, the contest is designed to be accessible and engaging regardless of their academic background. Questions are posed as puzzles in a real-world or fantasy setting, which students can approach simply by scribbling and following their intuition. Although the questions aim to stimulate algorithmic thinking, they do not rely on any knowledge of programming or computing, and they do not involve code or pseudocode.
- The simple format of the contest (A–E and 0–999) further reduces the burden on students, and makes the contest easy to grade for a large number of participants. Students record their solutions by colouring circles in pencil on a specially designed "mark sense" answer sheet, and teachers simply post the papers back to the central contest office where they are graded by machine.

One disadvantage of multiple choice contests is the ease and effectiveness of guessing. In a series of studies, Clark and Pollard measure the impact of guessing (Clark and Pollard, 2004*a*; Clark and Pollard, 2004*b*). The AIC has adopted some of their resulting suggestions, including integer answer questions and the effective use of distractors.

3 Creating algorithmic questions

In this section we show how the AIC maintains its algorithmic focus, despite the limitations of a puzzle-based setting, no assumed knowledge and a multiple choice / integer answer format. To do this, we walk through the different types of tasks that appear in the AIC and show how they stimulate thinking about algorithms in different ways. Section 3.1 begins with a discussion of the four major classes of multiple choice questions, and Section 3.2 introduces the AIC's distinctive three-stage integer answer tasks.

3.1 Multiple choice tasks

The multiple choice tasks in the AIC can be classified into four broad categories:

- *Algorithmic tasks*, which encourage students to develop an informal algorithm to solve a given puzzle;
- *Logic tasks*, which use non-algorithmic puzzles to encourage rigorous reasoning and case analysis;
- *Tracing tasks*, which are simple tasks that ask students to follow a well-defined set of instructions;
- *Analysis tasks*, where students probe the strengths and weaknesses of a given algorithm or problem.

This classification is of course rough—tasks may fit into more than one of these categories, and sometimes a task does not fit into any (such as the occasional pattern matching task).

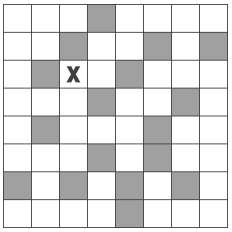
Algorithmic tasks

Algorithmic multiple choice tasks have the same aim as traditional informatics olympiad tasks: to encourage competitors to devise an efficient and correct algorithm to solve some problem.

In the multiple choice setting however, this aim is far less explicit. Tasks cannot ask for an algorithm directly—inexperienced students might not even know what an algorithm is. Instead, algorithmic tasks pose a puzzle where, if students are to solve it quickly and

Dungeon *(AIC 2005, Intermediate)*

A token (marked '`X`' in the diagram) is in a maze. You may move the token around according to the following rule: in each move the token may travel any distance either horizontally or vertically, but it cannot pass over or stop on a shaded square.



For example, from its starting position the token could travel either one square right, one square down, two squares down or three squares down in a single move. To reach any other square would require more than one move.

What is the minimum number of moves that you need to ensure that the token can reach any white square from its starting position?

(A) 8 (B) 9 (C) 10 (D) 11 (E) 12

Figure 1: The algorithmic task "Dungeon"

correctly, they must devise and follow some repeated systematic procedure—that is, an *algorithm*.

This is illustrated in the task *Dungeon* (Figure 1), taken from the first ever AIC. Although this problem can be solved in time through guesswork or trial and error, it is faster and more reliable to work systematically outwards from the token, identifying all the squares that are one move away, then two moves away, and so on. In other words, a student who has never learned about algorithms or programming may find themselves conducting an informal breadth-first search.

It is important to choose the size of the problem carefully. In a multiple choice setting, the only way a student can communicate their algorithm is to run it by hand, and then select the final output from amongst the five choices offered in the question. This means that the problem must be small enough for the student to manually step through their algorithm in a short period of time. On the other hand, if the problem is too small then an algorithm becomes unnecessary, and students can simply solve the puzzle through exhaustive (or educated) trial and error.

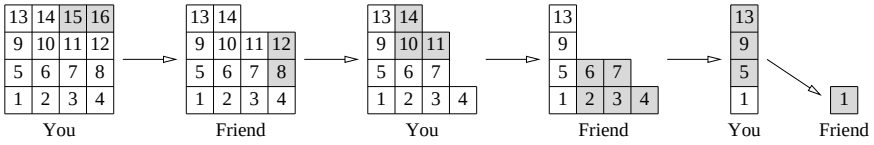
Another example of an algorithmic task is *Chocolate* (Figure 2). Again students can approach this problem through trial and error, and eventually they will succeed. However, a fast and accurate solution is to identify all "board shapes" from which you can "win" in one move, then all board shapes from which your opponent must lose in two moves, and so on. With only 19 possible board shapes, it is certainly feasible to iterate this by hand.

Like an informatics olympiad problem, algorithmic multiple choice tasks reward both correctness and efficiency. Correctness is rewarded directly by giving points for the correct answer. Efficiency is rewarded indirectly by leaving the student more time to finish the rest of the exam.

Chocolate *(AIC 2007, Intermediate & Senior)*

You and a friend are eating a block of chocolate by taking alternate bites — you take the first bite, your friend takes the second bite, you take the third bite, and so on.

Not all bites are the same size. To take a bite, you must choose a square and eat every square above it and/or to the right (including the square you chose). For example, consider the block illustrated below.



Here you take the first bite by choosing square 15 and eating everything above and to the right of it, which is just squares 15 and 16. Your friend then chooses square 8, thereby eating squares 8 and 12. You choose square 10, your friend chooses square 2, you choose square 5 and your friend finishes the block by choosing square 1.

As it happens, square 1 is a new experimental broccoli flavour that neither of you wants to eat. Suppose the chocolate has been reduced to the 3×3 block below, and it is your turn to take a bite. Only one of the following squares will allow you to force your friend to eventually take square 1 — which should you choose?

9	10	11
5	6	7
1	2	3

(A) Square 5 (B) Square 6 (C) Square 9

(D) Square 10 (E) Square 11

Figure 2: The algorithmic task "Chocolate"

Logic tasks

Logic tasks do not require an algorithm per se; instead they are typically solved through rigorous reasoning and case analysis. These skills, however, are essential for algorithm design, and so the AIC explicitly asks questions of this type.

An example is the task *Palindromes* (Figure 3), which is an exercise in rigorous case analysis. It is easy to show that the task is impossible using only one increment; with a little more work one can argue that it is impossible using two increments, and similarly for three. The task is finished by finding an explicit construction that solves it in four.

Another logic task is *Cities* (Figure 4). Here the focus is on creative logic rather than case analysis. A promising chain of reasoning might be to observe that the closest pair of cities must be adjacent; from here we can work outwards to more distant cities until the locations of all the cities are identified.

Tracing tasks

Tracing problems are simple tasks that ask students to step through a given procedure or follow a given set of well-specified instructions (which are presented in plain English, not code or pseudocode). This tests students' ability to understand and follow a given algorithm,

Palindromes *(AIC 2007, Intermediate & Senior)*

Given a sequence of digits, we can change it according to the following rules:

(i) If three consecutive digits are palindromic (i.e., the first is the same as the third), then all three digits can be removed. For instance, the sequence 163235 can be changed to 165.

(ii) Any digit except for 9 may be increased by one. For instance, 166725 could be changed to 176725 (thus allowing the 767 in the middle to be removed).

What is the least number of times that rule (ii) must be used in order to remove all the digits from the sequence 294563011 ?

(A) 1 (B) 2 (C) 3 (D) 4 (E) 5

Figure 3: The logic task "Palindromes"

Cities *(AIC 2007, Intermediate)*

The land of Pitopia is centred upon a large circular lake. Around this lake is a circular highway, with five cities placed along the highway. The distances between the cities are as follows:

Distance	City P	City Q	City R	City S	City T
City P		6 km	2 km	3 km	4 km
City Q	6 km		4 km	3 km	2 km
City R	2 km	4 km		5 km	6 km
City S	3 km	3 km	5 km		1 km
City T	4 km	2 km	6 km	1 km	

Note that there are always two different ways of travelling from one city to another (corresponding to the two different directions around the lake); the table above lists the shorter distance in each case.

You are travelling along the highway in a constant direction around the lake. In which order might you travel past the five cities?

(A) P, Q, S, T, R (B) P, R, S, T, Q (C) P, R, Q, T, S
(D) P, S, Q, T, R (E) P, T, S, Q, R

Figure 4: The logic task "Cities"

which is an important precursor to developing their own algorithms.

Tracing tasks are typically the easiest tasks on an AIC paper. An example is *Leet Speek* (Figure 5), which was the first question in the easiest AIC division in 2006.

Analysis tasks

The final class of multiple choice tasks is analysis tasks, which ask students to study the behaviour of a given algorithm and/or its underlying problem. Such a task might ask for an approximate number of steps or a worst case scenario, thereby encouraging thinking about

Leet Speek *(AIC 2006, Junior)*

To translate an English phrase to leet speek, the following rules are used:

1. Replace any occurrences of `two`, `tu`, `too` or `to` with the digit `2`;
2. Replace any occurrences of `four`, `fore` or `for` with the digit `4`;
3. Replace any occurrences of `eight`, `eat` or `ate` with the digit `8`;
4. Remove any remaining vowels (letters `a`, `e`, `i`, `o` or `u`).

Note that rules 1–3 cannot cross word boundaries. For instance, you can turn `foretold` into `42ld`, but you cannot turn `sleigh time` into `sl8ime`.

Consider the sentence "`They ate two great fortune cookies for tea.`" When converted to leet speek, how many letters and digits does the final version contain?

(A) 16 (B) 17 (C) 18 (D) 20 (E) 24

Figure 5: The tracing task "Leet Speek"

Pizza Delivery *(AIC 2008, Junior, Intermediate & Senior)*

A pizza delivery boy must deliver pizzas to 11 houses in a lane, with one pizza for each house. All houses are on the same side of the lane. His instructions read:

"Go to Mel's house, then go F3, B1, F6, B3, F1, B5, F1, F5, F1, B3",

where F3 means "go forward 3 houses", B1 means "go back 1 house", and so on.

Unfortunately, one of the instructions has been written down incorrectly. Where is the mistake?

(A) the 1st or 2nd instruction
(B) the 3rd or 4th instruction
(C) the 5th or 6th instruction
(D) the 7th or 8th instruction
(E) the 9th or 10th instruction

Figure 6: The analysis task "Pizza Delivery"

computational complexity and pathological cases. Other analysis tasks might ask students to find the error in a sequence of instructions (highlighting "debugging" skills) or to complete an exhaustive set of input scenarios (highlighting "testing" skills).

The purpose of analysis tasks is to encourage skills that complement algorithm design—it is important for programmers to be able to analyse the correctness, efficiency and robustness of their own code.

An example of an analysis task is *Pizza Delivery* (Figure 6). This is a simpler task—instead of an algorithm the user is presented with a straightforward sequence of steps, and the student must "debug" the sequence to locate the single error.

The Bulgarian contest Chernorizets Hrabar (Tabov et al., 2003) contains interesting analysis tasks of a different type. Given a low-level algorithm (expressed in code, pseudocode or as a flowchart), students are required to understand the algorithm and find out what high-level task it performs (typically some kind of numerical task, such as exponentiation, summation or root finding).

Spiders *(AIC 2006, Junior & Intermediate)*

As everybody knows, girl spiders are difficult to distinguish from boy spiders. You believe you can do this by using their size. For each spider colony you study, you declare that "all spiders larger than x millimetres are girls, and all spiders smaller than x millimetres are boys." Of course you will be wrong some of the time; your task is to choose a value for x so that you make as few errors as possible.

Each of the following scenarios describes a colony of spiders, giving the sizes of each boy and girl spider in millimetres. For each scenario, you must choose a value of x that gives you as few errors as possible (that is, the number of girl spiders of size $< x$ plus the number of boy spiders of size $> x$ is as small as possible). This value of x will be your answer to the question.

The size of every spider is even; each of your answers must be an *odd number*. There will never be more than one best possible answer.

1.

Sizes of boy spiders	12, 12, 14, 18, 22, 24
Sizes of girl spiders	16, 20, 26, 28, 28, 30, 30, 30, 32

2.

Sizes of boy spiders	14, 14, 14, 18, 18, 22, 26, 26
Sizes of girl spiders	16, 20, 20, 24, 24, 24, 28, 28, 28, 28

3.

Sizes of boy spiders	16, 18, 20, 22, 26, 28, 32, 34, 40, 42
Sizes of girl spiders	24, 26, 30, 36, 38, 42, 46, 48, 48, 50

Figure 7: The three-stage task "Spiders"

3.2 Three-stage tasks

Consider again the algorithmic tasks described in the previous section—tasks that encourage students to devise an algorithm that is both efficient and correct. Multiple choice algorithmic tasks suffer from two key difficulties:

- It is difficult to *encourage* students to find an algorithm. Students may simply attack a task with logic and/or educated guesswork and not realise that a more systematic procedure is necessary. Moreover, because tasks must be small enough to solve with pen and paper, logic and guesswork will often satisfy students that they have found the correct answer (regardless of whether this is true).
- It is difficult to *evaluate* whether students have found an algorithm. Because tasks are small, the right logic with some inspired guessing may well lead to the correct answer anyway, particularly when only five multiple choice options are available.

These difficulties stem from the facts that (i) multiple choice tasks are small, and (ii) they only contain a single "test case". To work around these difficulties, the AIC includes a series of *three-stage tasks*.

A three-stage task is a group of three related questions, each asking students to solve the same problem but with data sets of increasing size. Students will first be given the story and overall task description, followed by the three data sets that form the three questions. The answer for each data set is a single integer in the range 0–999. This is illustrated in the task *Spiders* (Figure 7).

Three-stage tasks are explicitly designed to entice students into formulating algorithms. The first data set is typically small, and can be solved using ad-hoc techniques. By the second data set the student should have a feel for the problem, and hopefully will be developing

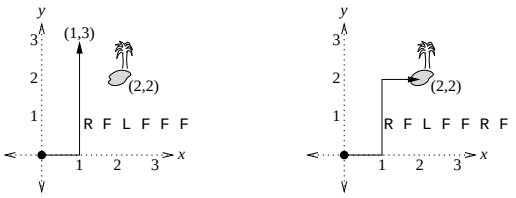
Lost *(AIC 2007, Intermediate & Senior)*

You are wandering through the desert with a map, which shows the desert as an (x, y) coordinate plane. You begin your journey at $(0, 0)$ facing north. In your hands are directions to an oasis, written as a sequence of letters. The possible letters are:

- `F`, indicating that you should walk forwards one kilometre in the direction you are currently facing;
- `L`, indicating that you should turn 90° to the left;
- `R`, indicating that you should turn 90° to the right.

Alas, the directions contain a critical mistake—one of the right hand turns has been deleted. Fortunately your map shows the coordinates of the oasis, and so you hope to use this information to work out where the missing right hand turn should be.

For example, suppose the directions are `R F L F F F` and the oasis is at $(2, 2)$. The first diagram below illustrates this path, which ends at the incorrect location $(1, 3)$.



With some thought it can be seen that the directions should be `R F L F F R F`. That is, the missing right hand turn takes place just before the final walk forwards, as shown in the second diagram above.

Each scenario below lists a series of directions, followed by the location of the oasis. For each scenario, how many letters appear before the missing `R` must be inserted?

1. `RFFLFLFFFRFF` $\longrightarrow (3, 3)$

2. `RFFLFRFFLFFLFLFRFFR`
`FFLFLFRFRFFFRFLFFLF` $\longrightarrow (5, 5)$

3. `RFFFLFFFRFFFFRFRFFRFFFF`
`FFLFFLFFFLFLFFFFFLFF` $\longrightarrow (8, 8)$

Figure 8: The three-stage task "Lost"

systematic techniques for manipulating the data. The third data set is larger again, and by this stage students should be able to apply their systematic techniques quickly and efficiently. The integer answers (0–999) serve to limit the value of guesswork, particularly for the larger data sets.

In summary, three-stage tasks aim to address the earlier difficulties as follows:

- By repeatedly asking students to solve similar tasks, they will be *encouraged* to develop algorithms;
- By successfully solving the larger data sets (which are less prone to ad-hoc techniques and guesswork), students can demonstrate that their algorithms work, allowing a clearer *evaluation* of algorithmic thinking.

The AIC includes three distinct three-stage tasks on every paper (questions 7–9, 10–12, and 13–15). Three-stage tasks need not be purely algorithmic; see for instance *Lost* (Figure 8), a task that includes aspects of algorithms, analysis and debugging.

4 Conclusion

In this paper we describe the Australian Informatics Competition, a short pen-and-paper contest that aims to make problems about algorithms accessible to a much broader range of students than traditional programming contests. The contest is designed to be manageable for teachers and schools even when students sit the contest in large numbers (such as entire classes). The tasks are puzzle-based with a strong focus on algorithmic thinking, and the unique three-stage tasks are designed to both encourage and reward the development of algorithms in ways that typical multiple choice tasks cannot.

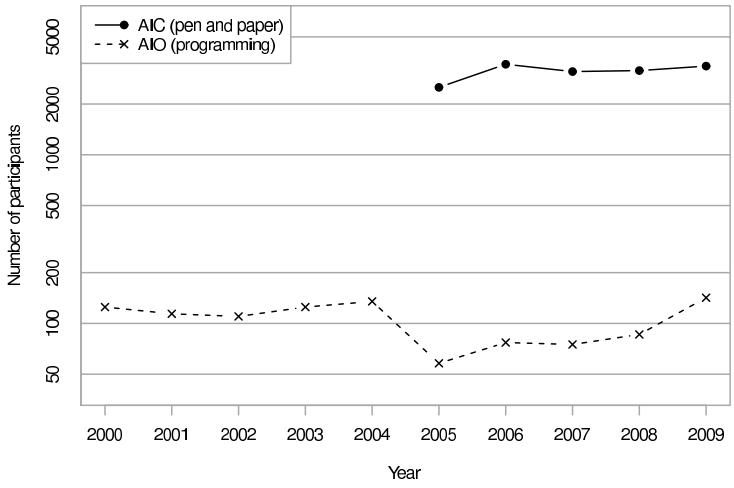


Figure 9: Participation in entry-level informatics contests over the past decade

The AIC has been extremely well received in Australia since its introduction in 2005. Figure 9 plots the number of entrants over the last decade for the AIC in comparison to the Australian Informatics Olympiad (AIO), which is the entry-level programming contest for the national olympiad programme. The first ever AIC attracted 2511 students, compared with a maximum of 142 students for the programming contest over its entire 12-year history. Current AIC entries number well over 3000, and the contest now attracts international entrants from New Zealand and Singapore.

It is interesting to note the sharp dip in entrants for the programming contest in the year when the AIC was introduced (58 students compared with 135 the year before). That year also marked a significant drop in the number of zero scores, suggesting perhaps that inexperienced students were moving from the programming contest to the more accessible written contest (understandable, given the respective aims of each contest). Happily, numbers for the programming contest have since grown again to beyond their pre-2005 levels.

The AIC now enjoys an important place within the Australian informatics olympiad programme. The reader is referred to (Burton, 2008) for a wider view of the national programme as a whole.

As noted in the introduction, other communities have responded to similar challenges in different ways. The Beaver (Bebras) competition retains a dependence on computers, which the AIC explicitly aims to avoid; however, this allows Beaver to offer innovative problems

such as interactive tasks that are impossible in a pen-and-paper setting. See (Dagienė and Futschek, 2008) for details, as well as a list of criteria that can help contest designers distinguish good tasks from bad.

Finally, readers are encouraged to try their hand at the sample tasks in this paper—solutions can be found in the table below.

Dungeon:	B	*Palindromes:*	D	*Leet Speek:*	A	*Spiders:*	25, 19, 35
Chocolate:	B	*Cities:*	C	*Pizza Delivery:*	D	*Lost:*	7, 20, 21

Acknowledgements

The author is grateful to the AIC problems committee for their hard work and enthusiasm over the last five years, and in particular to David Clark who chairs the committee and without whom the AIC might never have seen the light of day. The author is supported by the Australian Research Council's Discovery Projects funding scheme (project DP1094516).

References

Anido, R. d. O. and Menderico, R. M. (2007), 'Brazilian Olympiad in Informatics', *Olympiads in Informatics* **1**, 5–14.

Boersen, R. and Phillipps, M. (2006), Programming contests: Two innovative models from New Zealand. Available from `http://www.bwinf.de/competition-workshop/papers.html`.

Burton, B. A. (2008), Informatics olympiads: Challenges in programming and algorithm design, *in* G. Dobbie and B. Mans, eds, 'Thirty-First Australasian Computer Science Conference (ACSC 2008)', Vol. 74 of *CRPIT*, ACS, Wollongong, NSW, Australia, pp. 9–13.

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

Clark, D. (2006), 'The 2005 Australian Informatics Competition', *The Australian Mathematics Teacher* **62**(1), 30–35.

Clark, D. and Pollard, G. (2004*a*), 'A measure of the effectiveness of multiple choice tests in the presence of guessing: Part 1, crisp knowledge', *Mathematics Competitions* **17**(1), 17–33.

Clark, D. and Pollard, G. (2004*b*), 'A measure of the effectiveness of multiple choice tests in the presence of guessing: Part 2, partial knowledge', *Mathematics Competitions* **17**(1), 34–47.

Dagienė, V. (2006), 'Information technology contests—introduction to computer science in an attractive way', *Informatics in Education* **5**(1), 37–46.

Dagienė, V. and Futschek, G. (2008), Bebras international contest on informatics and computer literacy: Criteria for good tasks, *in* R. T. Mittermeir and M. M. Sysło, eds, 'Informatics Education – Supporting Computational Thinking', Vol. 5090 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, pp. 19–30.

Merry, B., Gallotta, M. and Hultquist, C. (2008), 'Challenges in running a computer olympiad in South Africa', *Olympiads in Informatics* **2**, 105–114.

Pohl, W. (2007), 'Computer science contests in Germany', *Olympiads in Informatics* **1**, 141–148.

Tabov, J., Kelevedzhiev, E. and Lazarov, B. (2003), 'Multiple choice style informatics', *Mathematics Competitions* **16**(2), 60–69.

Verhoeff, T. (2009), '20 years of IOI competition tasks', *Olympiads in Informatics* **3**, 149–166.

B.A. Burton was director of training for the Australian informatics olympiad programme from 1999–2008, and now sits on the IOI Scientific Committee. He has been involved in the Australian Informatics Competition since its inception in 2005, and edited the AIC papers for the first three years. He is presently a QEII Research Fellow at the University of Queensland, where he works on computational geometry and topology in three and four dimensions.