The Electronic Journal of Mathematics and Technology, Volume 12, Number 3, ISSN 1933-2823

A Proposed Method for Designing Diagnostic Mathematics Tests

Kevin K. H. Cheung
kevin.cheung@carleton.ca
School of Mathematics and Statistics
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6
Canada

Brett Stevens
brett@math.carleton.ca
School of Mathematics and Statistics
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6
Canada

Yunkai Wang
YunkaiWang@cmail.carleton.ca
School of Computer Science
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6
Canada

Abstract

A method based on mathematical results from non-adaptive group testing and combinatorial design theory for designing diagnostic mathematics tests containing multi-skill questions is described. The goal is to minimize the number of questions asked while covering a large list of skills without sacrificing the ability to pinpoint deficiencies in individual skills provided that the number of deficiencies to identify is low. A set of tools developed in Python 2.7 demonstrating the method is available as a free download for real-life testing needs or research in mathematical knowledge assessment.

1 Introduction

To be successful in a first-year university calculus course, competence in high school mathematics is required. Developing such competence requires the mastery of many skills such as laws of arithmetic, manipulating algebraic expressions, solving equations, graphing, etc. Having a robust way to test mastery of these skills and identify deficiences can help improve student support initiatives and retention.

A straightforward approach to test mastery of skills is to assess each skill individually. However, it is conceivable that the ability to perform a skill in isolation does not necessarily mean that the skill can be performed in a broader context. For example, a student who is able to evaluate $0.5 - \frac{1}{3}$ and $(2^6)^{\frac{1}{6}}$ individually might fail to evaluate $(2^6)^{0.5-\frac{1}{3}}$. Unfortunately, once questions are allowed to contain multiple skills, it is not clear how individual deficient skills can still be identified. To see the difficulty, say there are nine skills to be tested. If there are only three questions, each of which tests three different skills such that no skill is tested twice, then any wrong answer to any of the questions will leave us uncertain as to which of the three skills in the question caused difficulty. The only way to tease out the deficient skill(s) is to add additional test questions.

The central issue that is explored in this paper is the following: Given a set of skills such that every small subset of these can be tested in a single question, how does one minimize the number of multi-skill questions asked so that each skill is tested at least once and that there is a way to identify deficient skills provided that the number of deficient skills is small? As stated, the question contains certain ambiguities. For instance, it does not specify how many skills each question can contain. It also makes no mention of what is meant by "the number of deficient skills is small." We will discuss these technical aspects after considering a detailed motivating example. We will also give a brief description of the tools that we have developed and made available for the interested readers to use to design their own tests. In the meantime, we point out that our approach uses results in non-adaptive group testing from combinatorial design theory. The possibility of using results in adaptive group testing is left for future investigation. The reason for focussing on the non-adaptive setting first is that non-adaptive tests can be administered on paper and at scale.

In a nutshell, group testing uses results in combinatorial designs to divide the work of identifying "defects" into tests on groups of items rather than on individual ones. It was first studied by Robert Dorfman [3] who proposed a way to perform blood tests on army recruits during World War II though his method was never implemented. Over the years, many constructions with good asymptotic bounds have been obtained with immediate applications in large-scale problems such as software testing and the human genome project. In addition to the successes in large-scale applications, group testing results could appear in small-scale applications such as skills development and knowledge diagnostics as demonstrated by the work described in this paper. For more details on the history, the mathematics and applications of group testing, see [4] and the references therein.

2 Motivating example

Suppose that we want to design a test covering the following skills:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Absolute value
6. Exponentiation
7. Logarithm
8. Factorial
9. Fraction-to-decimal conversion

We would like the test to contain questions that test multiple skills each. As an illustration, a question that combines the second, the fifth, and the seventh skills could be the following:

Evaluate $\log_3 |2 - 5|$.

If a test-taker gives the right answer, then we regard the test-taker to have no deficiency in subtraction, absolute value, and logarithm. What happens if the test-taker gives an incorrect answer? From just this question, there is no way of knowing which of the three skills in the question is causing difficulty unless we have the luxury of looking at the written work of the test-taker. If we want to be able to pinpoint deficiencies, we cannot simply partition the skills into questions.

Consider the following test design consisting of ten questions, each of which tests a combination of three of the nine skills listed above:

Question	Skills tested
1	1, 4, 7
2	1, 5, 9
3	1, 6, 8
4	2, 4, 9
5	2, 5, 8
6	2, 6, 7
7	3, 4, 8
8	3, 5, 7
9	3, 6, 9
10	4, 6, 8

As an example, the following test conforms to the design:

1. Evaluate $\log_2\left(\frac{1+3}{4}\right)$.
2. Evaluate $\left|3+\left(-\frac{5}{4}\right)\right|$. Express your answer as a decimal.
3. Evaluate $(2!+3)^2$
4. Evaluate $(6-2)/5$. Express your answer as a decimal.
5. Evaluate $|3-4!|$.
6. Evaluate $\log_2(3^2-1)$.
7. Evaluate $3! \times 20/5$.
8. Evaluate $\log_8|(-2)\times 4|$.
9. Evaluate $\left(\frac{1}{10}\right)^{2\times 3}$. Express your answer as a decimal.
10. Evaluate $(-2)^{\frac{4!}{6}}$.

Suppose that a test-taker answered all the questions except questions 1, 6, and 8 correctly. Since question 2 was answered correctly, we know that skills 1, 5, 9 are not deficient; from question 3, skills 6 and 8 are not deficient; from question 5, skill 2 is not deficient; from question 7, skills 3 and 4 are not deficient. Note that skill 7 cannot be eliminated. Hence, we conclude that the test-taker was deficient in skill 7.

Notice that questions 1, 6 and 8, the questions that the test-taker answered incorrectly, are the only questions that contain skill 7. In reality, the test-taker might not necessarily answer all questions containing skill 7 incorrectly even with a deficiency in that skill. Now, what if question 6 was answered correctly? Using the method just described would eliminate all the skills, leading to the conclusion that the test-taker has no deficiency in any of the skills. However, the fact that questions 1 and 8 were answered incorrectly tells us that there probably was a deficiency somewhere. We will need a more sophisticated method for decoding test results which we will discuss in a later section. In the meantime, we articulate some practical considerations when combining skills and describe the mathematical tools we used for designing tests.

3 Considerations for combining skills

One simplifying assumption that we make in this paper is that all the skills that one wants to test can be arbitrarily combined. In practice, it may not make sense to combine certain skills. For example, how would one combine curve-sketching and multiplication of quarternions? A mechanism for encoding valid skill combinations is to use hypergraphs. However, except for some easy special cases, designing tests taking into account the structure of such a hypergraph is non-trivial and remains a direction for further research.

Even in the case when there is no inherent incompatibility of the topics of the skills to be combined, one must exercise care in designing questions that test multiple skills. For example, consider the following questions:

1. $(3 + 1!) \times 5$
2. $(3 + 1)! \times 5$
3. $((3 + 1) \times 5)!$

All of these questions test multiplication, addition, and factorial. However, they do not have the same degree of difficulty. One can argue that the first question tests factorial only superficially. The third question is unsuitable as a test question because the answer is too large for a typical diagnostic test. This example illustrates the need to exercise care and even ingenuity in constructing pedagogically sound questions that combine specific skills. The conceptual framework that we describe in the next section only provides a listing of which skills to combine for each question. The actual design of test questions that are suitable for tests in real life is left to test creators. Nevertheless, we show that it is possible to automate test-generation satisfying some basic feasibility constraints in Section 5.

4 d-disjunct matrices

For our test designs, we used results from non-adaptive group testing in combinatorial design theory. Before we go into the technical details, let us reframe the process of designing a test as constructing a binary matrix with some desired properties.

Consider the test design described in the previous section. We can construct a 10×9 matrix with rows representing the questions and columns representing the skills. The entries of the matrix are determined as follows: If skill j is tested in question i, then the (i, j)-entry of the matrix is 1; otherwise, it is 0. Thus, we have the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

In general, designing a test involving n skills amounts to constructing an appropriate binary matrix of dimension $t \times n$ where t denotes the number of questions. For practical purpose, we want t to be as small as possible subject to the following constraints:

- there cannot be a column of zeros (every skill is tested at least once);
- the number of ones in each row cannot be too large because there is a practical limit on how many skills can be effectively combined into a single question;
- there is a positive integer d such that if a test-taker is deficient in no more than d skills and answers incorrectly only questions that contain one or more of the deficient skills, there is an efficient way to identify the deficient skills by looking at the test results.

It turns out that a good starting point for our construction is given by d-disjunct matrices defined as follows:

Let t, n, d be positive integers. A $t \times n$ matrix $\mathbf{M}$ is said to be *d-disjunct* if for every subset $S \subseteq \{1, \ldots, n\}$ having at most d elements, for every $j \notin S$, there exists $i \in \{1, \ldots, t\}$ such that the (i, j)-entry of $\mathbf{M}$ is 1 but the (i, k)-entry of $\mathbf{M}$ is 0 for all $k \in S$.

Constructing d-disjunct matrices with small t given n is non-trivial. One method that we used with reasonable success is described in [6]. It is beyond the scope of this paper to describe the technical details of our construction method. The interested reader is referred to a forthcoming paper that explains the various techniques of obtaining d-disjunct matrices for values of n that we are interested in. Note that the value d here is the number of deficient skills that we want to be able to identify. To keep t to a reasonable number, d cannot be too large. Based on our experiments, d should be chosen to be no more than 6. Otherwise, we will end up with too many questions that make the tests impractical for real-life usage.

5 Experimental implementation

As a demonstration that automatic test-generation can be implemented based on the ideas presented above, we developed an experimental GUI application written in Python 2.7 on Ubuntu Linux 16.04 for generating tests from a predefined set of arithmetic skills. In this section, we give only a brief description of the application. The reader is refered to [1] for a detailed description and the source code of the application.

The GUI application allows the user to select a subset of these skills to be tested and specify the maximum number of skills that can appear per question and the maximum number of deficient skills to be identified. Figure 1 is a screenshot for the skill selection window. The application automatically determines the number of questions to include in the test by generating an appropriate d-disjunct matrix and then generates questions with random numbers subject to some intelligent choices that ensure intermediate and final answers (especially when the skills involve certain operations such as exponentiation and factorial) do not become unwieldy. After administrating the test, the user can obtain an assessment of the test results by using another application to process a user-created CSV file containing the test results. A sample test generated by the application that tests addition, subtraction, multiplication, exponentiating, and absolute value with at most four skills per question is included in the appendix.

In addition to the GUI application, we have developed a simple web server in Python using the Flask[1] web development framework for users who simply wish to obtain test designs (that is,

[1] `http://flask.pocoo.org/`

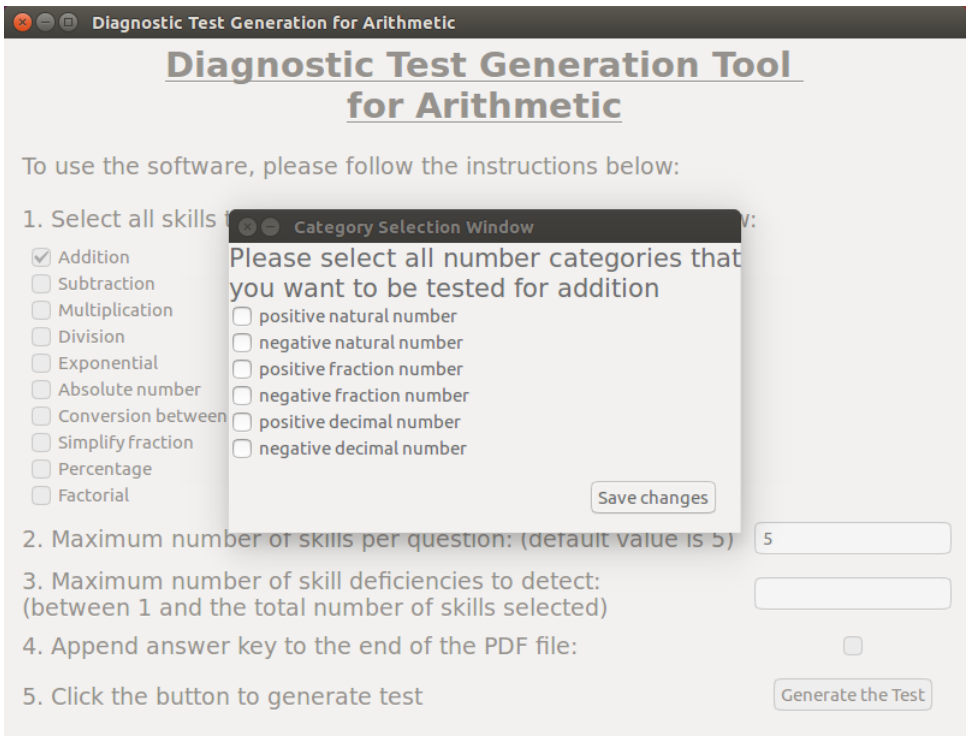

Figure 1: Screenshot of skill selection window

d-disjunct matrices) instead of generated tests for predefined skills. The server can be deployed locally or remotely in a server. It allows users to access a user interface using a web browser to generate a number of different constructs which include d-disjunct matrices. As a result, users can apply the test designs to domains other than math.

Installation and usage details about the test-generation application and the server can be found in [1].

6 Discussion

We now consider a number of issues that cannot be ignored if the ideas presented so far are to be used in real-life testing. There are two big assumptions that we have made:

1. Skills can be arbitrarily combined.

2. Test-takers always answer questions that contain a deficient skill incorrectly and questions that do not contain a deficient skill correctly.

The first of these assumptions has already been discussed in Section 3. We now address the second assumption.

It is possible that a question is answered correctly because of luck rather than competence. It is also possible that a question is answered incorrectly because of carelessness rather than deficiency in one or more of the skills tested. Though the former is less likely than the latter, it

cannot be completely disregarded because using the naive method of removing skills in questions that have been answered correctly from the list of possible deficient skills could lead to overly optimistic conclusion. We now illustrate a more sophisticated method for decoding test results using a standard idea from error-correcting codes. Consider again the d-disjunct matrix that we constructed earlier:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

This matrix is in fact 1-disjunct. That means we can identify up to 1 deficient skill with a test designed using this matrix.

For $i = 1, \ldots, 9$, let $\mathbf{c}^{(i)}$ denote the ith column of this matrix. Then $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(9)}$ are binary vectors of size ten. To decode a test result, form a binary vector $\mathbf{r}$ of size ten as follows: Set the ith component of $\mathbf{r}$ to 1 if question i is answered incorrectly and 0 otherwise. We call $\mathbf{r}$ the *result vector*. For example, if questions 1 and 6 are the only questions that are incorrectly answered, then

$$\mathbf{r} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Given two binary vectors $\mathbf{u}$ and $\mathbf{v}$ having the same size, the *Hamming distance* between $\mathbf{u}$ and $\mathbf{v}$ is defined as the number of components in which $\mathbf{u}$ and $\mathbf{v}$ differ. For example, the Hamming distance between $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$ is 6 whereas the Hamming distance between $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(4)}$ is 5. To decode the test results, we simply search for a $k \in \{1, \ldots, 9\}$ that minimizes the Hamming distance between $\mathbf{r}$ and $\mathbf{c}^{(k)}$. In this case, there is a unique choice for k which is 7. Hence, we report skill 7 as the deficient skill.

The reasoning behind this decoding method is that in the ideal situation where every question containing skill 7 is answered incorrectly and every question not containing skill 7 is answered correctly, $\mathbf{c}^{(7)}$ will be the result vector. Since $\mathbf{c}^{(7)}$ is closest to our result vector, it is most likely that the deficiency is in skill 7. Note that this approach works only when $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(9)}$ are pairwise sufficiently apart in terms of Hamming distance. For our example, one can check

that the smallest Hamming distance between $\mathbf{c}^{(i)}$ and $\mathbf{c}^{(j)}$ for $i \neq j$ is 4. Thus, we can uniquely decode any result vector that is of Hamming distance no more than 1 from some $\mathbf{c}^{(i)}$.

We have illustrated decoding for the case when $d = 1$. When $d > 1$, we would need to do the following. Suppose that $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(n)}$ are the columns of the d-disjunct matrix for a test design. Let $\mathcal{S}$ denote the set of all nonempty subsets of $\{1, \ldots, n\}$ of size at most d. For example, when $n = 4$ and $d = 2$, the elements of $\mathcal{S}$ are

$$\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}.$$

For each $A \in \mathcal{S}$, form the binary vector $\mathbf{x}^{(A)}$ such that the ith component equals 0 when the ith component of $\mathbf{c}^{(i)}$ is 0 for all $i \in A$ and 1 otherwise. To decode the test result, we find $A \in \mathcal{S}$ that minimizes the Hamming distance between $\mathbf{x}^{(A)}$ and the result vector. We then report skill i for each $i \in A$ as deficient.

There are methods for constructing d-disjunct matrices that are robust to small deviations from ideal result vectors. Such methods involve the construction of what are known as d^e-disjunct matrices which allow one to have result vectors of Hamming distance at most e from the ideal result vector. The details for the construction are beyond the scope of this paper. The interested reader is referred to [5].

7 Final remarks

One problem that we set out to study was the design of tests that can identify pairs of skills that only show up as deficient when they are tested in the same question but not when they are tested separately. What we have described in this paper will not provide us with sufficient information to say that skill i and skill j together cause difficulty only when tested together. To determine such deficiency pair requires a notion of *separable matrices* described in [2]. We have not yet succeeded in implementing the method for constructing such matrices. However, even if we have a proper implementation with current mathematical techniques, it is doubtful that the tests obtained can be practical because the number of questions required is estimated to be rather high relative to the number of skills to be tested. A careful analysis of the definitions of d-disjunct and seperable matrices shows that they have more constraints than is strictly necessary, especially if one is willing to trade extra cost to analyse diagnostic test results for smaller matrices. Additionally, as mentioned in Section 6 in the discussion of assumption 1, when we combine two skills, for example, we will not have to consider all the combinations of pairs of skills, only a restricted set which is reasonable. Taking advantage of these points, however, requires more mathematical research in combinatorial group testing.

Acknowledgement

The research of this paper was supported and funded by eCampusOntario. The authors would like to thank the anonymous referees for their comments and suggestions for improving the exposition of the paper.

References

[1] Accompanying software and documentation. `http://people.math.carleton.ca/~kcheung/math/apps/gt4mcode.zip`.

[2] Chin, F.Y.L., Leung, H.C.N., and Yiu, S.M. (2013): Non-adaptive complex group testing with multiple positive sets, *Thoery and Applications of Models of Computation*, 505:11–18.

[3] Dorfman, R. (1943): The Detection of Defective Members of Large Populations, *The Annals of Mathematical Statistics*, 14(4):436–440.

[4] Du D. and Hwang, F.K. (1993): *Combinatorial group testing and its applications*, Singapore: World Scientific.

[5] Ngo, H.Q., Porat, El, and Rudra, A. (2011): Efficiently decodable error-correcting list disjunct matrices and applications, in *Proceedings of the 38th international colloquium conference on Automata, languages and programming*, Volume Part I, 557–568.

[6] Porat, E. and Rothschild, A. (2011): Explicit nonadpative combinatorial group testing schemes, *IEEE Transactions on Information Theory*, 57(12):7982–7989.

Appendix

Diagnostic test:

Data file is saved as /home/kcheung/Desktop/diagnostic_test_data.json, please keep this file for obtain the result using the retrieve result program.

1. $\Big(\Big((-2)^{(3-(-4))}\Big) - (-10)\Big) + (-7)$
2. $\big(|(7^1)|\big) + (-7-9)$
3. $((|-3|+2)-8)+4$
4. $-2+\Big((-8-(-9))^7\Big)$
5. $-6-(|-7|-0)$
6. $\big(|(0^5)|\big)+7$
7. $((4+(-4))-4)+(-2)$