A SERIES OF PAPERS ON DETECTING EXAMINEES WHO
USED A FLAWED ANSWER KEY

by

Marcus W. Scott

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Mathematics

Approved:

____________________
Joseph V. Koebbe, Ph.D.
Major Professor

__________
____________________
James S. Cangelosi, Ph.D.
Committee Member

____________________
Guifang Fu, Ph.D.
Committee Member

____________________
Mimi M. Recker, Ph.D.
Committee Member

____________________
Kady Schneiter, Ph.D.
Committee Member

____________________
Mark R. McLellan, Ph.D.
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018
ABSTRACT

A SERIES OF PAPERS ON DETECTING EXAMINEES WHO USED A FLAWED ANSWER KEY

by

Marcus W. Scott, Doctor of Philosophy
Utah State University, 2018

Preknowledge of test content is a threat to exam validity, and can be difficult to detect. However, detection can be easier if the examinees with preknowledge are using an answer key that has errors, known as a flawed answer key. Three papers on this subject were written.

The first paper presents a case study in which a testing program found a flawed answer key with incorrect answers to 37 of the 65 items. Three methods for detecting examinees who used the flawed key were studied: a modified version of Angoff’s B index, an outlier detection method based on a principal components analysis (PCA), and a Bayesian classifier based on the PCA. The Bayesian classifier detected the most examinees, and the other two methods had very similar results.

The second paper investigated using the answer-copying statistic, $\omega$, to estimate the response pattern of an unknown flawed answer key. Four estimation methods were
developed and applied to real-life and simulated test data. One method, denoted Common_Max, had near-perfect performance on the real-life data. Another method, denoted Most_Flagged, had the best performance in the simulation study.

Because answer-copying analysis can be a lengthy process as the number of examinees increases, the third paper presents a case study in parallelizing this task. Two directive-based application programming interfaces, OpenMP and OpenACC, were used to parallelize an answer-copying analysis of a population of examinees. The goal of the paper was to demonstrate the utility of OpenMP and OpenACC for parallelizing computer programs that analyzes test data. For the answer-copying analysis case study, the code was estimated to be 6.4 times faster after using OpenMP to enable parallelization across the central processing unit, and 19.0 times faster after using OpenACC to accelerate the code on the graphics processing unit.

(121 pages)
PUBLIC ABSTRACT

A Series of Papers on Detecting Examinees who Used a Flawed Answer Key

Marcus W. Scott

One way that examinees can gain an unfair advantage on a test is by having prior access to the test questions and their answers, known as preknowledge. Determining which examinees had preknowledge can be a difficult task. Sometimes, the compromised test content that examinees use to get preknowledge has mistakes in the answer key. Examinees who had preknowledge can be identified by determining whether they used this flawed answer key. This research consisted of three papers aimed at helping testing programs detect examinees who used a flawed answer key.

The first paper developed three methods for detecting examinees who used a flawed answer key. These methods were applied to a real data set with a flawed answer key for which 37 of the 65 answers were incorrect. One requirement for these three methods was that the flawed answer key had to be known. The second paper studied the problem of estimating an unknown flawed answer key. Four methods of estimating the unknown flawed key were developed and applied to real and simulated data. Two of the methods had promising results. The methods of estimating an unknown flawed answer key required comparing examinees’ response patterns, which was a time-consuming process. In the third paper, OpenMP and OpenACC were used to parallelize this process, which allowed for larger data sets to be analyzed in less time.
CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>CASE STUDY: USING FLAWED ANSWER KEY ANALYSIS TO DETECT BRAINDUMP USERS</td>
<td>4</td>
</tr>
<tr>
<td>III</td>
<td>USING THE ω STATISTIC TO ESTIMATE AN UNKNOWN FLAWED ANSWER KEY</td>
<td>35</td>
</tr>
<tr>
<td>IV</td>
<td>CASE STUDY: PARALLELIZING ANSWER-COPYING ANALYSIS</td>
<td>61</td>
</tr>
<tr>
<td>V</td>
<td>ADDITIONAL STUDIES</td>
<td>89</td>
</tr>
<tr>
<td>VI</td>
<td>SUMMARY AND CONCLUSIONS</td>
<td>106</td>
</tr>
</tbody>
</table>

REFERENCES..............................................................................................................109

CURRICULUM VITAE......................................................................................................112
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Accuracies of the Estimated Flawed Answer Keys for the Real-Life Data Sets</td>
<td>47</td>
</tr>
<tr>
<td>3-2</td>
<td>Average Accuracies of the Estimated Flawed Answer Keys for the Simulation Study</td>
<td>50</td>
</tr>
<tr>
<td>3-3</td>
<td>Simulation Results for Common_Source</td>
<td>51</td>
</tr>
<tr>
<td>3-4</td>
<td>Simulation Results for Max_Omega</td>
<td>53</td>
</tr>
<tr>
<td>3-5</td>
<td>Simulation Results for Most_Flagged</td>
<td>55</td>
</tr>
<tr>
<td>3-6</td>
<td>Simulation Results for Common_Max</td>
<td>57</td>
</tr>
<tr>
<td>5-1</td>
<td>Type I Error Rates for Simulations with a 50-Item Exam</td>
<td>94</td>
</tr>
<tr>
<td>5-2</td>
<td>Type I Error Rates for Simulations with a 75-Item Exam</td>
<td>94</td>
</tr>
<tr>
<td>5-3</td>
<td>Type I Error Rates for Simulations with a 100-Item Exam</td>
<td>95</td>
</tr>
<tr>
<td>5-4</td>
<td>Detection Rates for Simulations with a 50-Item Exam</td>
<td>97</td>
</tr>
<tr>
<td>5-5</td>
<td>Detection Rates for Simulations with a 75-Item Exam</td>
<td>97</td>
</tr>
<tr>
<td>5-6</td>
<td>Detection Rates for Simulations with a 100-Item Exam</td>
<td>98</td>
</tr>
<tr>
<td>5-7</td>
<td>Average Accuracies of the Estimated Flawed Answer Keys for the Simulation with 500 Examinees</td>
<td>102</td>
</tr>
<tr>
<td>5-8</td>
<td>Results for the Most_Flagged Method in the Simulation with 3,000 Examinees</td>
<td>103</td>
</tr>
<tr>
<td>5-9</td>
<td>Results for the Common_Max Method in the Simulation with 3,000 Examinees</td>
<td>104</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>2-1</td>
<td>Incorrect matches with the flawed key for tests taken before and after the date of theft</td>
<td>7</td>
</tr>
<tr>
<td>2-2</td>
<td>Flawed scores plotted against keyed scores</td>
<td>8</td>
</tr>
<tr>
<td>2-3</td>
<td>Standardized residuals plotted against the keyed scores</td>
<td>18</td>
</tr>
<tr>
<td>2-4</td>
<td>Standardized residuals plotted against the predicted number of incorrect matches</td>
<td>18</td>
</tr>
<tr>
<td>2-5</td>
<td>Uncorrelated data after performing the principal components analysis</td>
<td>20</td>
</tr>
<tr>
<td>2-6</td>
<td>Histograms of baseline data and the fitted normal distribution</td>
<td>21</td>
</tr>
<tr>
<td>2-7</td>
<td>Histogram of the second principal components with the 5σ threshold</td>
<td>22</td>
</tr>
<tr>
<td>2-8</td>
<td>Histograms of the second principal components for the initial group of likely braindump users and the fitted beta distribution</td>
<td>23</td>
</tr>
<tr>
<td>2-9</td>
<td>Probability of membership in the group of likely braindump users as a function of the second principal component</td>
<td>27</td>
</tr>
<tr>
<td>2-10</td>
<td>Plot of (keyed score, flawed score) ordered pairs categorized by the number of methods that flagged the examinee</td>
<td>30</td>
</tr>
<tr>
<td>4-1</td>
<td>Sequential pseudocode for answer-copying analysis with ω</td>
<td>64</td>
</tr>
<tr>
<td>4-2</td>
<td>Answer-copying analysis pseudocode with the outer loop parallelized</td>
<td>69</td>
</tr>
<tr>
<td>4-3</td>
<td>Answer-copying analysis pseudocode with the outer and inner loops parallelized</td>
<td>70</td>
</tr>
<tr>
<td>4-4</td>
<td>Answer-copying analysis pseudocode using the sections directive</td>
<td>72</td>
</tr>
<tr>
<td>4-5</td>
<td>Pseudocode for computing ω for a given source-copier pair</td>
<td>73</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4-6</td>
<td>Vectorized pseudocode for computing ω for a given source-copier pair</td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td>Pseudocode parallelized for the GPU using the kernels directive</td>
<td></td>
</tr>
<tr>
<td>4-8</td>
<td>Pseudocode parallelized for the GPU using the parallel directive</td>
<td></td>
</tr>
<tr>
<td>4-9</td>
<td>Sample output when compiling with the -Minfo compiler option</td>
<td></td>
</tr>
<tr>
<td>4-10</td>
<td>Pseudocode with the collapse clause</td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>Simulated data and a fitted normal distribution</td>
<td></td>
</tr>
<tr>
<td>5-2</td>
<td>Upper tails of the distributions in Figure 5-1</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

In the field of test security, test fraud is defined as any action that is intended to jeopardize the fairness and validity of a test. The International Testing Commission (ITC) identifies two types of test fraud: test theft and cheating (2014). The ITC further identifies six types of cheating, one of which is preknowledge of test content. One method for examinees to get preknowledge of test content is to download compromised test content from the Internet. This compromised test content is commonly referred to as a “braindump.”

Determining which examinees obtained preknowledge from a braindump can be a difficult task, and the quality of the braindump is a factor. The quality of the braindump is determined by how much of the item pool is found in the braindump and whether the braindump items are exact copies of the live items or approximations of them. With a high-quality braindump, braindump users are difficult to distinguish from high-ability examinees. The imperfections found in low to medium-quality braindumps may be used to determine which examinees used them. Of particular interest is the case in which the answer key in the braindump has some incorrect responses listed as answers, resulting in a flawed answer key. This dissertation presents three papers related to determining which examinees used a flawed answer key found in a braindump.

The first paper presents a case study in which a testing program purchased a braindump with an answer key that gave incorrect responses to 37 of the 65 items on the exam. Additionally, the testing program was able to determine the identity of the test
thief and the date of the theft by matching the order of the items in the braindump to the order they were administered to a specific examinee. This allowed the testing program to identify a baseline group of tests that were not taken with the assistance of the braindump. The tests were scored using the real answer key and the flawed answer key. Using these two sets of the scores and the existence of the baseline examinees, the case study in the first paper presents and compares three methods for determining which examinees likely used the flawed answer key in the braindump.

The second paper addresses one shortcoming of the methods from the first paper for identifying examinees who used a flawed answer key, which is that the flawed answer key must be known so the tests can be scored using the real key and the flawed key. There are situations where a testing program may believe that a flawed answer key is being used by some members of the testing population, but this flawed key may not be possible to obtain. An example is if a braindump is being circulated among members of the testing population, but is not available for download anywhere.

Thus, the second paper addresses the problem of estimating an unknown flawed answer key from the examinees’ responses to the items. Because examinees who use a flawed answer key are essentially copying a response pattern that is external to the testing session, the answer-copying statistic $\omega$ (Wollack, 1997) is used to try to estimate the unknown flawed key. The paper presents four methods of estimating the unknown flawed key using the results from computing $\omega$ for every possible source-copier pair. Real-life data sets involving flawed answer keys and a simulation study are used to validate the four methods.
Computing $\omega$ for a population of examinees is computationally-intensive, and can have a long runtime as the numbers of examinees and items increase. This was experienced firsthand in the second paper, as the full simulation took several days to run. Therefore, the third paper explores methods of decreasing the time needed to compute $\omega$ for a population of examinees. Because $\omega$ asymptotically follows a known distribution (Wollack, 1997), there is no need to fit a distribution to the results. Therefore, the computation of $\omega$ for one source-copier pair is independent of the computation for another source-copier pair. This means the task of computing $\omega$ for a population of examinees can be parallelized.

In the third paper, two application programming interfaces (APIs) known as OpenMP (Open Multi-Processing) and OpenACC (Open Accelerators) are used to parallelize computation of $\omega$ across the computer’s central processing unit (CPU) and graphics processing unit (GPU), respectively. Doing so allows for some significant decreases in computation time, which makes answer-copying analysis using $\omega$ more feasible. Although the third paper focuses on parallelizing $\omega$, these APIs can be used to parallelize computationally-intensive tasks involving test data. Therefore, the goal of the third paper is to demonstrate the utility of OpenMP and OpenACC to individuals who analyze test data.

A fourth chapter presents some supplementary results for the first and second papers that were beyond their original scope.
CHAPTER II

CASE STUDY: USING FLAWED ANSWER KEY ANALYSIS TO DETECT BRAINDUMP USERS

Preknowledge of test content poses a serious threat to the validity of a test. One way that examinees acquire preknowledge of test content is by downloading a set of stolen test items, commonly known as a “braindump,” from the Internet. Examinees who use a braindump gain an unfair advantage, regardless of whether the braindump includes a portion of the item bank or the entire item bank. The validity of the test is compromised because a high score could result from knowledge of the material or preknowledge of the test (Qian, Staniewska, Reckase, & Woo, 2016).

An investigation of the contents of one braindump site illustrates how much damage one can cause. In 2005, the braindump site CertExperts.com went out of business and offered to sell all of its braindump content for $199 (Foster & Zervos, 2006). This one braindump site had live test questions from 498 exams being offered by 50 different certification programs. Altogether, 58,000 test questions were obtained. The replacement cost for this many items was estimated by Foster and Zervos to be in the tens of millions of dollars.

Because braindump usage threatens the validity of a potentially large proportion of test results, decreases the worth of certifications and licensures, and undermines the financial investment that testing programs put into their tests, methods to detect individuals who used a braindump are an important part of test security research. This case study analyzes a situation in which a braindump from an actual certification exam
was obtained and evaluated (Scott, Cooper, & Maynes, 2015). The case study discusses three applications of statistical methods to determine which examinees used it.

**Case Description**

A testing program purchased a braindump for one of its exams and discovered that the braindump contained high-quality copies of all 65 live items. The answer key included in the braindump had incorrect answers to 37 of the 65 items, so it was a “flawed answer key.” The testing program noticed that the 37 items with incorrect keys were some of the more difficult items. This exam was administered to 599 examinees over the course of a year.

One unique quality about this braindump is that its structure allowed the testing program to determine the date that the items were stolen. Computers were used to administer the tests, and the items were presented in a random order for each test instance. Because the testing program’s database included a record of item presentation orders for every exam that had been administered, it was possible to compare the presentation order for each test instance to the order of the items in the braindump. One test instance administered on July 7, 2014 had the exact same item order as the braindump. The probability of a random ordering of 65 items exactly matching the braindump ordering by chance alone is $1/(65!) \approx 1.2 \times 10^{-91}$. Therefore, it could be said with confidence that the test items were stolen on July 7, 2014. For comparison, the next highest number of position matches observed among the 599 test instances was eight.
Nearly half (285 out of 599) of the test instances had no test items in the same positions as the braindump ordering.

There were 236 test instances administered before July 7, 2014, and 363 were administered on or after that date. For each test instance, the testing program provided the score using the actual answer key, the score using the flawed answer key, and the number of times the examinee chose one of the incorrect answers listed in the flawed answer key. Item-level data, such as item parameters and p-values, were not provided. Examinee responses and individual item scores also were not provided.

Initial examination of the data suggested that the flawed answer key was indeed being used by some members of the testing population after July 7, 2014. Prior to the date of the theft, the average number of incorrect matches with the flawed answer key was 6.1, with a maximum value of 14. After the date of the theft, the average number of incorrect matches with the flawed answer key increased to 16.9, with a maximum value of 37, which was the number of incorrect answers in the flawed key. Figure 2-1 shows the counts of incorrect matches with the braindump key for tests taken before the theft date and tests taken after the theft date.
In Figure 2-1, it can be seen that the agreement with the incorrect answers in the braindump key for tests administered before July 7, 2014 was more consistent than the agreement with the incorrect answers in the braindump key for tests administered after that date. Tests taken before the date of the theft typically had fewer incorrect matches with the braindump key than those taken after the date of the theft. In this figure, it also can be seen that some examinees had response patterns that exactly matched or closely matched the flawed key. Figure 2-2 shows a scatter plot of the scores using the flawed answer key, referred to as “flawed scores,” against the scores using the actual answer key, referred to as “keyed scores.” To prevent data points from stacking on top of each other, the data were “jittered” by adding a random number in the interval [-0.5, 0.5] to each coordinate.
Figure 2-2. Flawed scores plotted against keyed scores.

There is some separation in the data points. Data points in the upper group had a negative correlation between the two sets of scores, indicating that these examinees had better performance when their tests were scored using the flawed key, as opposed to the actual answer key. Test instances in this group likely used the braindump. No test instances from before the date of the theft were found in this upper group. Data points in the lower group had a positive correlation between the two sets of scores, so these examinees likely did not use the braindump.

Because the empirical data presented in Figures 2-1 and 2-2 have a distinct pattern of separation, it is reasonable to infer that some examinees had preknowledge of the test questions and responded to them using the flawed answer key from the purchased braindump. The next section reviews techniques that have been proposed and published for detecting examinees with preknowledge of test content.
Review of Traditional Techniques for Detecting Preknowledge

Person-Fit Statistics

Person-fit statistics have been studied as a way to identify aberrant response patterns belonging to examinees who may have had preknowledge of the test items. Karabatsos (2003) compared the ability of 36 person-fit statistics to detect five different types of aberrant response patterns, one of which resulted from cheating (answer copying). He found that aberrant response patterns resulting from cheating were difficult for the person-fit statistics to detect. He also observed that non-parametric person-fit statistics generally had better performance than those based on item response theory (IRT). The answer-copying research by Karabatsos is relevant because answer copying can be viewed as a form of preknowledge. Indeed, the present case study can be viewed as a situation in which examinees potentially copied answers from the braindump answer key.

Tendeiro and Meijer (2014) compared the performance of five non-parametric person-fit statistics that were based on the Guttman model. Their research expanded on that of Karabatsos by also studying the performance of statistics based on cumulative sum procedures. Like Karabatsos (2003), they found that $H^T$ (Sijtsma, 1986) had the highest detection rate of invalid test scores, which can result from preknowledge, among the five statistics. They also found that the item discrimination factors had a large effect on detection rates.

McLeod, Lewis, and Thissen (2003) developed a Bayesian index that computes a posterior probability that the examinee had item preknowledge. The method was
designed for use in a computerized adaptive test, because the probability that an examinee had preknowledge is updated after each item response is given. A final log odds ratio (FLOR) is computed from the final posterior probability and the initial prior probability. The FLOR value indicates how much more or less suspicious the testing program should be that an examinee had preknowledge of test content.

Belov (2012) used Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) to determine which examinees had preknowledge of a compromised set of items. His method was designed to be used in a CAT environment. For each examinee, the posterior distribution of ability on the items that were compromised is compared to the posterior distribution of ability on the non-compromised items using KL divergence. A significance level is obtained from the empirical distribution of KL divergence values, and examinees with KL divergence values above this significance level are flagged as aberrant.

**Answer-Copying Statistics**

Preknowledge of test content can be thought of as a form of answer copying because the examinees are essentially copying from a response pattern that is external to the testing session. Flawed or inaccurate preknowledge by a group of examinees, such as that present in the data for this case study, may result in examinees having multiple matching incorrect responses, which generally improve the detection power of answer-copying statistics.

Angoff (1974; also see Frary, 1993) developed eight indices, denoted A-H, for detecting answer copying. The eight indices depended on quantities such as the number
of matching correct responses between a pair of examinees, the number of items answered incorrectly by both examinees, the number of items with matching incorrect responses, common omits, and runs of incorrect and omitted items. Indices B and H were determined by Angoff to be the best for detecting answer copying. Index B analyzed the relationship between the product of the numbers of incorrect responses for the two examinees and the number of matching incorrect responses between them. Index H analyzed the relationship between the sum of matching incorrect responses and common omitted items for two examinees and the length of the longest “run” of identical incorrect responses between them.

Another early answer-copying statistic is $g_2$, which was developed by Frary, Tideman, & Watts (1977). After designating a source and a copier, the probability that the copier would choose the source’s answer is computed for each item. Summing these probabilities gives the expected number of matches, and the $g_2$ statistic is the standardized value of the observed number of matches. The $g_2$ statistic is modeled by the standard normal distribution, meaning it has a mean of zero and a standard deviation of one. Wollack (1997) developed an IRT-based statistic called $\omega$ that is similar in form to $g_2$. Wollack also demonstrated that $\omega$ maintained the nominal type I error rate, but $g_2$ did not.

Holland (1996) documented the $K$ index (used at ETS) to compute the probability that two examinees would agree on their incorrect answers. To account for the copier’s ability, the probability is computed using response data only from those examinees with
the same number of incorrect responses as the potential copier. The $K$ index used the binomial distribution for the probability computation.

However, Sotaridona and Meijer (2002) found that the $K$ index loses power when the binomial distribution is used, and when the source and the copier have many matching correct responses. They improved on the $K$ index by computing the binomial probability parameter using a quadratic regression that incorporated more data than the approach by Holland. This version of the $K$ index was denoted $\overline{K}_2$, and it had an improved detection rate over the $K$ index.

Further research by Sotaridona and Meijer (2006) introduced two more versions of the $K$ index: $S_1$ and $S_2$. The primary difference between $K$ and $S_1$ is that the Poisson distribution is used instead of the binomial distribution. A log-linear model is used to estimate the Poisson parameter. The $S_2$ index utilizes information from matching correct and matching incorrect responses. Simulations showed that the $S_1$ and $S_2$ indexes had higher detection rates than $\overline{K}_2$, with $S_2$ typically outperforming $S_1$.

**Score Differences**

Roberts (1987) studied score differences based on two keys as a method of determining whether an examinee copied from another examinee with a different form. He considered a worst-case scenario with examinees randomly guessing on all items, so any matches between the true answer key and the alternative key would be random. The score difference was modeled by a normal distribution with a mean of 0 and a standard deviation of $\sqrt{2NP(1 - P)}$, where $N$ was the number of test items and $P$ was the probability of correctly answering an item. For the case of the random guesser, $P = 1/A,$
where $A$ was the number of response options. Roberts calculated score difference values corresponding to alpha values of .05, .01, and .001 for three different test lengths (25, 50, and 100 items) and three different response option counts (2, 4, and 5 options). As an example, a 100-item test with 5 response options has a score difference of 17 at the $\alpha = .001$ level.

Based on his calculations, Roberts (1987) observed that large score differences could be observed by chance alone, and that larger score differences were more likely to be observed among examinees with low scores. He then concluded that the method’s type I error rate was too high to be used in practice. However, it must be noted that Roberts never actually determined the type I error rate of the score difference method. No thresholds for flagging an examinee as a cheater were set, no simulations were performed, and no measures to control the error rate were discussed.

**Trojan Horse Items**

For tests delivered using a computer, some test administration systems required downloading the exam and its corresponding answer key to a server at the test site. This architecture created a test security vulnerability because employees at the test site who illicitly obtained a copy of the exam (e.g., by hacking) would have the answer key as well (International Test Commission, 2014). For this reason, Radwin (2008) developed the “Trojan Horse” item to detect examinees who used stolen answer keys. Trojan Horse (TH) items are non-scored items that have been purposefully mis-keyed for the intent of identifying examinees who used a stolen copy of the answer key. By design, TH items are easy so it is unlikely for knowledgeable examinees to choose the incorrectly-keyed
responses. Examinees who did well on the exam but chose the incorrectly-keyed responses to the TH items likely used a stolen key. On the other hand, examinees who did well on the exam and chose the correct answers to the TH items (which are marked as incorrect) likely did not use a stolen key.

Advances in technology have allowed computer-based testing (CBT) to evolve into internet-based testing (IBT). With IBT, test items are delivered over the Internet, so the items and their answer key are never downloaded to the test site (Gibson & Mulkey, 2016). Although the items can still be stolen (e.g. using a small camera to capture items as they are displayed on the screen), test thieves are unable to steal the answer key. Because the answer key is not stolen, TH items cannot be used to detect examinees with preknowledge. However, because the thieves must create their own answer key, it is often the case that answer keys produced by test thieves contain errors. The differences between the braindump key and the actual answer key can be used to determine which examinees likely used the braindump, as will be demonstrated in this case study.

Limitations of the Traditional Methods for the Case Study Data

Although the previously-discussed methods have merit, most of them cannot be utilized for the case study data because the testing program did not provide item-level data. Most of the person-fit statistics and answer-copying statistics require knowing the examinee’s item responses and item scores. Even if the required data were available, the person-fit statistics would still be of little use because the entire item bank was compromised. Person-fit statistics typically detect examinees with preknowledge by
identifying aberrant response patterns caused by the examinees having better performance on the compromised items than on the non-compromised items.

An even more restrictive limitation on the case study was the small sample size. Typical IRT-based analysis methods will require a larger sample size than 236 in order to create the necessary models.

Because the case study data involve examinees using a flawed answer key, answer-copying statistics seem to be more appropriate than person-fit statistics for determining which examinees used it. However, most of the previously-discussed answer-copying statistics require item-level data, which were not available. There is enough information in the case study data to apply Angoff’s B index, so a modified version of that method is used for analyzing the case study data.

Finally, Trojan Horse items cannot be used because none were included on the exam in the case study. Even if they were included, TH items only work when the answer key is stolen, which did not occur for the exam in the case study.

**Three Methods for Detecting Examinees with Preknowledge**

Because the only available data were each examinee’s keyed score, flawed score, and number of incorrect matches with the flawed key, the following methods were used to analyze the case study data:

1. Angoff’s B index,
2. An outlier detection method based on a principal components analysis (PCA) of the data, and
3. A Bayesian classifier based on the PCA.

The next three subsections discuss each method in greater detail. Results of the three methods are presented later.

**Angoff’s B Index**

Angoff (1974; also see Frary, 1993) designed the B index to analyze the relationship between the product of the numbers of items answered incorrectly by a pair of examinees (the independent variable) and the number of items for which the examinees chose the same incorrect response (the indicator variable). Angoff obtained conditional distributions of the indicator variable by stratifying the data based on the independent variable. He then used these conditional distributions to determine whether an observed value of the indicator variable was statistically extreme. Angoff’s stratification method was viable due to working with large samples. Because of the large samples, Angoff was able to model heterogeneity in the data.

Some modification of Angoff’s B index was necessary to apply it to the case study data. With data from only 599 tests, stratification in the independent variable was not feasible. Therefore, the B index was modified into a regression of the matching incorrect responses on the product of incorrectly-answered items. This was the approach employed by Saupe (1960), whose work influenced Angoff’s (1974), although Saupe used different sets of variables. Also, because the flawed answer key had 37 incorrect answers, one of the terms in the product of incorrectly-answered items was held constant at 37. Thus, the product was $37 \times (65 - \text{score})$, which is a linear transformation of the test score. In regression analysis, linear transformations of the predictor variables can be
absorbed into the constants, so the regression could be performed on the test scores, instead of the product of incorrectly-answered items.

Because a baseline of test data for known non-cheaters was available, it made sense to use only these data when constructing the regression equation. In Figure 2-2, the baseline data, shown using black circles, seemed to exhibit a linear relationship between the keyed score and the flawed score. The correlation between the keyed scores and flawed scores for the baseline data was .62. Therefore, it was determined that a linear regression analysis was appropriate for these data.

Denoting the keyed score as $x_K$ and the number of incorrect matches with the flawed key as $y_{IM}$, the equation of the regression line fitted to the baseline data was:

$$y_{IM} = -0.17x_K + 12.55.$$  \hspace{1cm} (2-1)

Before applying Equation 2-1 to the test data from after the theft, it is important to verify that the assumptions of linearity and homoscedasticity of the residuals were held. Figures 2-3 and 2-4 show the standardized residuals plotted against the keyed scores and predicted numbers of incorrect matches, respectively. The data in these figures have been jittered to prevent data points stacking on top of each other.
Figure 2-3. Standardized residuals plotted against the keyed scores.

Figure 2-4. Standardized residuals plotted against the predicted number of incorrect matches.

Figure 2-3 shows that there was no relationship between the residuals and the keyed scores, which were the independent variables. Therefore, the assumption of linearity of the regression was held. Similarly, Figure 2-4 shows that there was no
relationship between the residuals and the predicted numbers of incorrect matches. The “fanning” pattern commonly seen in heteroscedastic data was not observed, so it appeared that the assumption of homoscedastic residuals was held.

These results provided additional evidence that Equation 2-1 was an appropriate model of the relationship between the keyed scores and the numbers of incorrect matches for the baseline data. Therefore, this model was deemed suitable to apply to the test data from after the date of theft as a method of detecting examinees who likely used the braindump. For each test instance, the residual between the actual number of incorrect matches with the flawed key and the value predicted from Equation 2-1 was computed. Examinees with extreme residuals were flagged as likely braindump users.

**Outlier Detection**

When the testing program requested the data analysis, it was emphasized that the technique used to detect examinees with preknowledge must be simple to explain in layman’s terms so that it could be presented to upper management and corporate lawyers. Hence, the method of outlier detection presented here was developed.

Correlation between the two sets of scores complicated the task of determining which examinees used the braindump. By removing the correlation between the two sets of scores for the baseline examinees, a threshold could be established for detecting likely braindump users. Correlation between the two scores was removed by performing a principal components analysis (PCA) of the baseline data (i.e., test instances before July 7, 2014). The transformation obtained from the PCA was then applied to the data from after the theft. Figure 2-5 shows the transformed data.
Figure 2-5. Uncorrelated data after performing the principal components analysis.

Figure 2-5 shows that two distinct groups formed with respect to the second principal component. Therefore, a specific value of the second principal component could be used as a threshold for determining whether or not an examinee likely used the flawed key. To establish this threshold, a distribution had to be fit to the second principal component values for the baseline data. It was found that these data could be modeled by a normal distribution with a mean of 0 and a standard deviation of 2.65. For the goodness-of-fit test, the $\chi^2$ statistic had a value of 7.14 with 12 degrees of freedom, which was not significant ($p = .85$). Figure 2-6 shows histograms of the second principal components of the baseline data and the fitted normal distribution.
In consultation with lawyers for the testing program, the threshold was set at five standard deviations above the mean, which was 13.24 for the case study data. By using a five standard deviation threshold, the probability of a test instance being erroneously flagged for using braindump content was about 1 in 3.5 million. Figure 2-7 shows a histogram of the second principal components for all of the data. A black line indicates the 13.24 threshold for flagging a test.

*Figure 2-6*. Histograms of baseline data and the fitted normal distribution.
Bayesian Classifier

The Bayesian classifier developed here was an extension of the outlier detection method based on the PCA, which was previously discussed. The method involved determining whether the second principal component for an examinee likely came from the distribution for baseline examinees who did not use the braindump, or from the distribution for examinees who likely did use it. The expectation-maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977) was used to iteratively refine the classifier.

The initial distribution of non-braindump users was the normal distribution for the baseline data, which was constructed previously. Likely braindump users were initially identified by the following criteria:

1. The test was taken after the theft date (July 7, 2014), and
2. The value of the second principal component was more than three standard deviations above the mean of the normal distribution for the baseline data.
These data were not normally distributed, but could be modeled by a beta distribution. The principal component values for the initial group of likely braindump users ranged between 8.23 and 42.38. However, all principal component values would be tested with the fitted beta distribution, so the interval [-7, 42.5] was used for fitting the distribution. The fitted beta distribution had shape parameters $\alpha = 2.00$ and $\beta = 0.64$. For the goodness-of-fit test, the $\chi^2$ statistic had a value of 42.41 with 13 degrees of freedom, which was significant ($p = 5.61 \times 10^{-5}$). However, when the two left-most bins (i.e., the left tail of the distribution) were ignored, the $\chi^2$ statistic had a value of 19.67 with 11 degrees of freedom, which was not significant ($p > .05$). Therefore, the fitted beta distribution adequately modeled the examinees with the most extreme values of the second principal component. Figure 2-8 shows the empirical distribution and the fitted beta distribution.

![Histograms of the second principal components for the initial group of likely braindump users and the fitted beta distribution.](image-url)
Using the two distributions, Bayes’ rule determines the probability that a value of the second principal component was drawn from the distribution for likely braindump users. These probability values can then be used to update the two distributions, and the process is repeated until it converges. In the equations that follow, examinees are indexed by \( j \) and iterations of the EM algorithm are indexed by \( k \). Distributions and statistics associated with examinees who did not use the braindump are denoted by the subscript 0, whereas distributions and statistics associated with examinees who likely did use the braindump are denoted by the subscript \( b \).

From Bayes’ rule, the probability of the value of the second principal component coming from the distribution of values for likely braindump users is

\[
p_{j,k} = \frac{f_b(x_j|\hat{\alpha}_{b,k}, \hat{\beta}_{b,k})\hat{\pi}_k}{f_b(x_j|\hat{\alpha}_{b,k}, \hat{\beta}_{b,k})\hat{\pi}_k + f_0(x_j|\hat{\mu}_{0,k}, \hat{\sigma}_{0,k}^2)(1 - \hat{\pi}_k)},
\]

(2-2)

where \( x_j \) is the value of the second principal component for examinee \( j \), and \( \hat{\pi}_k \) is the proportion of examinees who likely used the braindump. This is the maximization step of the EM algorithm. In Equation 2-2, the probability density function for the beta distribution on the interval \([-7, 42.5]\) is

\[
f_b(x_j|\hat{\alpha}_{b,k}, \hat{\beta}_{b,k}) = \frac{(x_j + 7)^{\hat{\alpha}_{b,k} - 1}(42.5 - x_j)^{\hat{\beta}_{b,k} - 1}}{B(\hat{\alpha}_{b,k}, \hat{\beta}_{b,k})(49.5)^{\hat{\alpha}_{b,k} + \hat{\beta}_{b,k} - 1}}.
\]

(2-3)

After computing the probabilities, \( \hat{\pi}_k \) and the distribution parameters are updated during the expectation step of the EM algorithm as follows:

\[
\hat{\pi}_{k+1} = \frac{1}{n} \sum_{j=1}^{n} p_{j,k}
\]

(2-4)
\[ \hat{\mu}_{0,k+1} = \frac{\sum_{j=1}^{n}(1 - p_{j,k})x_j}{\sum_{j=1}^{n}(1 - p_{j,k})} \]  
\[ \hat{\sigma}_{0,k+1}^2 = \frac{\sum_{j=1}^{n}(1 - p_{j,k})(x_j - \hat{\mu}_{0,k+1})^2}{\sum_{j=1}^{n}(1 - p_{j,k})} \]  
\[ \hat{\mu}_{b,k+1} = \frac{\sum_{j=1}^{n}p_{j,k}x_j}{\sum_{j=1}^{n}p_{j,k}} \]  
\[ \hat{\sigma}_{b,k+1}^2 = \frac{\sum_{j=1}^{n}p_{j,k}(x_j - \hat{\mu}_{b,k+1})^2}{\sum_{j=1}^{n}p_{j,k}}. \]  

For the beta distribution in Equation 2-3, the parameters \( \hat{\alpha}_{b,k} \) and \( \hat{\beta}_{b,k} \) were computed from \( \hat{\mu}_{b,k} \) and \( \hat{\sigma}_{b,k}^2 \) using the method of moments. These have equations

\[ \hat{\alpha}_{b,k} = \frac{\hat{\mu}_{b,k} + 7}{49.5} \left( \frac{\hat{\mu}_{b,k} + 7 \left( 1 - \frac{\hat{\mu}_{b,k} + 7}{49.5} \right) \hat{\sigma}_{b,k}^2}{\hat{\sigma}_{b,k}^2} - 1 \right) \]  
\[ \hat{\beta}_{b,k} = \left( 1 - \frac{\hat{\mu}_{b,k} + 7}{49.5} \right) \left( \frac{\hat{\mu}_{b,k} + 7 \left( 1 - \frac{\hat{\mu}_{b,k} + 7}{49.5} \right) \hat{\sigma}_{b,k}^2}{\hat{\sigma}_{b,k}^2} - 1 \right). \]  

Iterations of the EM algorithm were performed until both means had relative changes less than .001 between iterations. For the case study data, 18 iterations were required to meet these convergence criteria.

**Results**

This section presents and compares the results of analyzing the case study data using the three different methods.
**Angoff’s B Index Results**

The residuals obtained from the regression equation were standardized using the mean and standard deviation of the residuals for the baseline data. For the baseline data, the mean of the residuals was 0, and the standard deviation was 1.84. As discussed in the development of the outlier detection method, the testing program required a $5\sigma$ threshold for flagging an examinee as a likely braindump user. To enable comparison of the methods, that same threshold was applied to the standardized residuals from the modified Angoff’s B index.

There were 155 examinees with standardized residuals of 5 or greater. All 155 examinees took the exam after July 7, 2014, so no baseline examinees were flagged by the method. For this group, the number of incorrect responses that matched the flawed braindump key varied from 14 to 37. When the tests were scored using the flawed key, 140 of these 155 examinees had higher scores and the remaining 15 had lower scores or no change in score.

**Outlier Detection Results**

During the presentation of the outlier detection method, the $5\sigma$ flagging threshold for the second principal components for the case study data was determined to be 13.24. There were 157 test instances with second principal component values that exceeded this threshold. All 157 examinees took the exam after July 7, 2014, so no baseline examinees were flagged by the method. For this group, the number of incorrect responses that matched the flawed braindump key varied from 13 to 37. When the tests were scored
using the flawed key, 141 of these 157 examinees had higher scores and the remaining 16 had lower scores or no change in score.

**Bayesian Classifier Results**

The Bayesian classifier differs from the other two methods in that it computes the probability of membership in the group of likely braindump users, as opposed to the probability of an observed value under the hypothesis that the examinee did not use the braindump. In this situation, a 5σ threshold cannot be set, so a specific probability of membership in the group of likely braindump users will be used as the flagging threshold. Figure 2-9 shows this probability as a function of the second principal component.

**Figure 2-9.** Probability of membership in the group of likely braindump users as a function of the second principal component.

The distribution of second principal components for examinees who did not use the braindump (i.e., those who took the exam before it was stolen) was centered at 0, so values near 0 should have a low probability of membership in the group of likely
braindump users. Accordingly, the curve in Figure 2-9 does appear to be minimized near 0. However, the curve does have a spurious peak for values far to the left of 0, which is attributable to lack of fit by the beta distribution in the left tail. These values correspond to examinees who did not benefit when the flawed key was used to score the test. The two probability distribution functions used in the probability computation have low values far to the left of 0, which caused the spurious peak. Based on the results shown in Figure 2-9, and a desire to be conservative with the threshold, a probability of .9 was used as the flagging threshold for the Bayesian classifier.

There were 177 examinees for whom the probability of membership in the group of likely braindump users exceeded the .9 flagging threshold. One examinee in this group of 177 took the test before July 7, 2014, so one baseline examinee was flagged by the method. This examinee had 10 incorrect matches with the flawed key, a keyed score of 42, and a flawed score of 35. For the group of 177 flagged examinees, the number of incorrect responses that matched the flawed braindump key varied from 10 to 37. When the flawed key was used, 141 of the 177 examinees had higher scores and the remaining 36 had lower scores or no change in score.

**Method Comparison**

The results from Angoff’s B index were very similar to the results from the outlier detection method. Only four examinees were flagged differently by the two methods. One examinee was flagged by Angoff’s B index, but not by the outlier detection method. This examinee had a keyed score of 38, a flawed score of 37, and 16 incorrect matches with the flawed key. The standardized value of the second principal component for this
examinee was 4.81, so it was slightly below the flagging threshold for the outlier detection method. Three examinees were flagged by the outlier detection method, but not by Angoff’s B index. Keyed scores for these examinees ranged between 21 and 44, their flawed scores ranged between 33 and 40, and they had either 13 or 18 incorrect matches with the flawed key. The standardized values of the B index for these three examinees ranged between 3.99 and 4.88, so they were close to the flagging threshold.

The Bayesian classifier flagged the most examinees. Every examinee that was flagged by Angoff’s B index or the outlier detection method was also flagged by the Bayesian classifier. An additional 19 examinees were flagged by the Bayesian classifier, but not by Angoff’s B index or the outlier detection method. For these 19 examinees, the number of incorrect responses that matched the flawed braindump key varied from 10 to 15. When the tests were scored using the flawed key, all 19 of these examinees had lower scores or no change in score.

To further illustrate which examinees were flagged by the different methods, the (keyed score, flawed score) ordered pairs from Figure 2-2 are plotted again, but the markers were chosen based on how many methods flagged the examinee. Examinees flagged by two methods were flagged by the Bayesian classifier and either Angoff’s B index or the outlier detection method. Examinees flagged by one method were flagged by the Bayesian classifier. Figure 2-10 shows these results.
Every examinee with a flawed score of 41 or greater was flagged by all three methods. For the 154 examinees in this group, the differences between the flawed scores and the keyed scores ranged between -6 and 37. Eleven examinees in this group had negative score differences, and three had no change in score when the flawed key was used. The examinees flagged by one or two methods had flawed scores ranging between 32 and 40. For the examinees in these two groups, differences between the flawed scores and keyed scores ranged between -13 and 12.

**Discussion**

**Case Study Summary**

This case study examined the problem of determining which examinees used a braindump when the following criteria were met:
1. The answer key in the braindump contained errors (i.e., some responses listed as correct were actually incorrect) and

2. A group of examinees was known to have taken the exam before it was stolen. Another characteristic of the case study data was that item-level data, such as p-values and item parameters, were not available. Only the keyed score, the flawed score, and the number of incorrect matches with the flawed key were provided. Hence, the methods used in the case study relied on these data to flag examinees who used the braindump.

The following three methods were used to flag likely braindump users:

1. A modified version of Angoff’s B index,

2. An outlier detection method based on a principal components analysis, and

3. A Bayesian classifier based on the principal components analysis.

The results of the case study have demonstrated that the methods described in this paper have power to detect braindump users when the answer key in the braindump has flaws. It was previously shown that 154 examinees were flagged by all three methods. Because conservative flagging thresholds were used, it can be said with confidence that these 154 examinees used the flawed key from the braindump. With 599 examinees in total, this means that approximately 26% of the testing population used the braindump. However, this rate does not take into account the fact that 236 examinees took the exam before the exam was stolen. Therefore, the 154 examinees actually represent 42% of the population who took the exam after it was stolen.
Thoughts on the Methods

An important benefit of these methods is that they require only test scores, and not item parameters. There were 236 baseline examinees in this case study, which is a small sample size for item parameter estimation, especially for models that generate parameters for each response. However, this is a sufficient number of examinees for fitting a distribution or constructing a linear regression. Another benefit is that the methods are based on simple statistical techniques that are easy to explain to non-statisticians.

There are two limitations to these methods that also need to be discussed. First, the date that the test was stolen must be known so the baseline examinees can be appropriately determined. Without this knowledge, some braindump users may be included in the baseline examinees, which will contaminate the flagging threshold. Second, and more importantly, the methods only determine which examinees used a known flawed answer key. These methods cannot determine which examinees cheated in other ways. Therefore, these methods are not suitable for routine use in a test security program. However, as demonstrated by this case study, they are effective for detecting examinees who used a known flawed answer key, should the situation arise.

The modified Angoff’s B index and the outlier detection method had very similar results and were more conservative than the Bayesian classifier. Although the assumptions of linear regression were held by the case study data, it is unknown how well other data sets with a flawed answer key will conform to the assumptions. Similarly, the
baseline data in the case study were shown to follow a normal distribution. It is unknown whether other data sets with a flawed answer key will have this property.

The Bayesian classifier was the most liberal method, and there was a spurious peak in the probabilities it computed for second principal component values in the left tail, as shown in Figure 2-9. This spurious peak resulted from small values of both probability density functions in the left tail. When a beta distribution was used for the baseline data instead of the normal distribution, some probability values in the left tail exceeded .999. Therefore, the Bayesian classifier is sensitive to how the distributions for the baseline examinees and the likely braindump users were modeled.

**Future Research**

There is opportunity for future research based on this case study. First, simulation studies can be conducted to compare the type I error rates and detection rates of the three methods presented here. These simulation studies can also help answer the questions of how well regression assumptions are held, how well the data are modeled by specific distributions, and how to prevent the spurious peak in the left tail observed for the Bayesian classifier.

Another research opportunity lies in the fact that these methods only work when the flawed answer key is known. Additional techniques will need to be developed to address the situation where a flawed answer key is suspected to exist, but its structure is unknown. These methods also require knowing a date of theft for identifying the baseline examinees. It may be possible to develop techniques that analyze the keyed scores and flawed scores to determine an approximate date of theft.
Finally, it may be possible to use the methods presented in this case study to analyze examinees’ responses to Trojan Horse (TH) items. Currently, there are two ways of analyzing the TH items. The first is to build a probability model of answering a given number of TH items in accordance with the incorrect key. However, data contamination will bias these models if too many examinees have chosen the incorrect answers. The other way is to choose a threshold for flagging examinees based on how many TH items they answered according to the incorrect key. However, this lacks statistical rigor. Research on flawed answer keys may provide additional insight into the use of TH items.
CHAPTER III

USING THE ω STATISTIC TO ESTIMATE AN UNKNOWN FLAWED ANSWER KEY

Preknowledge of test content is a substantial threat to the validity of test scores because it can be difficult to determine whether an examinee’s score is the result of legitimate ability or preknowledge. A common way for examinees to gain preknowledge is to download stolen test content, known as a “braindump,” from the Internet. A presentation by Foster & Zervos (2006) illustrated the ubiquity of braindumps. For $199, they purchased a collection of braindumps from a braindump site that was closing its operations. Altogether, 58,000 high-quality items from 498 exams offered by 50 testing programs were obtained. They estimated the replacement costs for these items to be in the tens of millions of dollars.

In the past, high-quality braindumps could be produced because some test administration systems required a server download of the test items and their answer key for a computer-based test. Individuals who obtained this material would have exact copies of the items and their answers (International Test Commission, 2014). It is now becoming common for test items to be delivered over the Internet during a testing session. Under this administration model, the answer keys are no longer downloaded to the test site (Gibson & Mulkey, 2016).

Test thieves who are unable to acquire the live answer key must create one for distribution with the braindump. There is potential for error when the test thieves create their own answer key, and braindumps with “flawed answer keys” have been sold on the
Internet (Scott, Cooper, & Maynes, 2015). The work by Scott, Cooper, & Maynes also showed that scoring the tests using the actual answer key and the flawed answer key can provide sufficient information to determine which examinees used the flawed answer key from the braindump. For their detection method to work, the flawed answer key had to be known, and a group of examinees known to have not used the flawed answer key had to exist.

Maynes (2016) demonstrated that identical response patterns shared by a group of examinees can be analyzed probabilistically. When the probabilities are sufficiently strong, the identical response pattern reliably estimates the flawed answer key shared by the group of examinees. Once the flawed answer key has been estimated, Maynes showed that probabilities of using the flawed answer key were computable using Bayes’ theorem. Maynes’ approach is limited by the requirement that identical response patterns must be observed. This is a rather critical distinction, because the case studies presented by Maynes suggested that many examinees who used the flawed answer key did not recall it perfectly.

This research extends that of the aforementioned flawed answer key research by developing methods of estimating the unknown flawed answer key that was imputed by the test thieves. The answer-copying statistic $\omega$ (Wollack, 1997) was used for this research because examinees who used the flawed answer key were essentially copying from a response pattern that was external to the testing session. The $\omega$ statistic was specifically used because it is simple to implement, it has power to detect copiers, and it maintains its type I error rate. Answer-copying statistics are traditionally used for
evaluating a case of answer copying after a source and one or more copiers have been identified, not exhaustively mining the test result data. Consequently, this research extends current answer-copying analyses by using the power of computers to exhaustively search the test result data for an unknown source (or flawed key) that may have been used by a group of unknown copiers.

**Review of \( \omega \)**

In his derivation of \( \omega \), Wollack (1997) denotes the potential copier and the potential source as \( C \) and \( S \), respectively. An examinee’s ability is represented by the quantity \( \theta \). The value \( u_i \) is the response to item \( i \); hence \( u_{ic} \) is the response to item \( i \) chosen by the potential copier, and \( u_{is} \) is the response to item \( i \) chosen by the potential source. An examinee’s response pattern for the \( n \) items is represented by the vector \( U \), and the item parameters form the matrix \( \xi \). For convenience, define the indicator function \( I(u_{ic} = u_{is}) \) to equal 1 when \( u_{ic} = u_{is} \), and 0 otherwise.

With these definitions, the observed number of matches between \( C \) and \( S \) is given by:

\[
h_{CS} = \sum_{i=1}^{n} I(u_{ic} = u_{is}). \tag{3-1}
\]

Because the nominal response model was used, the probability of \( C \) selecting the same response as \( S \) can be computed for each item, and the sum of these probabilities,

\[
E(h_{CS}|\theta_C, U_S, \xi) = \sum_{i=1}^{n} p(u_{ic} = u_{is}|\theta_C, U_S, \xi), \tag{3-2}
\]
gives the expected value of \( h_{CS} \) conditioned on the potential copier’s ability, the source’s response pattern, and the item parameters. Because the item responses are locally independent and two responses can either match or not match, the number of matches is a sum of independent Bernoulli variables. Therefore, the variance of \( h_{CS} \) is given by

\[
\sigma^2_{h_{CS}} = \sum_{i=1}^{n} p(u_{iC} = u_{iS}|\theta_C, U_S, \xi)[1 - p(u_{iC} = u_{iS}|\theta_C, U_S, \xi)].
\] (3-3)

The value of \( \omega \) is the standardized number of matches, and it has the formula

\[
\omega = \frac{h_{CS} - E(h_{CS}|\theta_C, U_S, \xi)}{\sqrt{\sigma^2_{h_{CS}}}},
\] (3-4)

where \( h_{CS}, E(h_{CS}|\theta_C, U_S, \xi) \), and \( \sigma^2_{h_{CS}} \) are given by Equations 3-1, 3-2, and 3-3, respectively.

**Flawed Answer Key Estimation**

Examinees who used the same flawed answer key are likely to have similar response patterns. If a flawed answer key is being used, then at least one examinee should have a response pattern that most closely resembles it. By computing the \( \omega \) statistic for every possible source-copier pair, a statistical signal may develop around this particular response pattern, which then can be used as an estimator of the unknown flawed answer key. Four methods for using the results of the \( \omega \) statistic computations to estimate the unknown flawed answer key are presented and evaluated.
Four Methods for Using $\omega$ to Estimate the Flawed Answer Key

The following are descriptions of the four methods of estimating the flawed answer key that were studied in this research:

1. **Common_Source** – Use the response pattern from the potential source that had the greatest number of source-copier pairs with values of $\omega$ exceeding a predetermined threshold as the estimate of the flawed key;

2. **Max_Omega** – Use the response pattern from the potential source that had the source-copier pair with the largest value of the $\omega$ statistic as the estimate of the flawed key;

3. **Most_Flagged** – Use the response pattern from the test that appeared in the most source-copier pairs (as source or copier) with values of $\omega$ exceeding a predetermined threshold as the estimate of the flawed key; and

4. **Common_Max** – Use the response pattern from the potential source that was observed with the maximum value of $\omega$ for the most potential copiers (i.e., the greatest number of potential copiers attained their maximum values of $\omega$ with this potential source) as the estimate of the flawed key.

Real Data Sets for Evaluating the Methods

The four methods were applied to four sets of real test data. In the first data set, a flawed answer key was known to exist because it was included in a purchased braindump. In the other three data sets, a flawed answer key was suspected to exist due to the presence of a large group of tests with an identical response pattern. The methods were evaluated by comparing the estimate of the flawed key to the known flawed key (for
the first data set) or the response pattern used by the group of identical tests (for the last three data sets). The following are descriptions of the real data sets. Identifying information such as organization names and exam titles have been withheld in accordance with confidentiality covenants.

Real data set 1. A braindump for one form of an exam was purchased, and the answer key had incorrect answers to 22 of the 60 items, so the braindump key was only 63% accurate. This form of the exam was administered to 249 examinees. Preliminary analysis found that no examinees had response patterns that matched the flawed key exactly, but there was a group of 12 examinees with response patterns that matched the flawed answer key for 58 of the 60 items, with 20 incorrect matches. The response pattern shared by this group of 12 examinees matched the flawed key more than any other response pattern.

Real data set 2. An exam with 60 items was administered to 2,899 examinees, and a cluster of 30 tests with identical response patterns was detected. This response pattern had correct answers to 47 of the 60 items (78%), so it was plausible that the examinees had access to the same flawed answer key. This response pattern was considered to be the flawed key. Identical clusters of sizes 9, 10, and 17 were also detected.

Real data set 3. An exam with 60 items was administered to 720 examinees, and a cluster of 17 tests with identical response patterns was detected. This response pattern had correct answers to 55 of the 60 items (92%), so it was plausible that the examinees had access to the same flawed answer key. This response pattern was considered to be
the flawed key. Three clusters of seven tests with identical response patterns were also
detected.

**Real data set 4.** An exam with 100 items was administered to 387 examinees,
and the testing program noticed that 95 was a very common score. There were 37
examinees with a score of 95, and 32 of those 37 had the same response pattern.
Therefore, it was plausible that these 32 examinees had access to the same flawed answer
key. The response pattern for those 32 examinees was considered to be the flawed key.
The five incorrect answers in the flawed key were more commonly chosen by the entire
testing population than were the correct answers, so the braindump had widespread use.
This data set was one analyzed by Maynes (2016).

**Simulation Study for Evaluating the Methods**

Although real-life data are useful for evaluating the methods, they are insufficient
for determining how the methods will perform under varying conditions of flawed answer
key usage. For example, the real-life data are incapable of demonstrating how the
proportion of flaws in the answer key or the proportion of braindump users affects the
accuracy of the estimated flawed key. However, a simulation study can provide insight
into those effects. The simulation study consisted of the following five steps:

1. Construct an exam with a specified number of items and a specified proportion of
   items that have incorrect answers in the flawed answer key;

2. Administer the exam to 1,000 examinees, a specific proportion of which used the
   flawed answer key;
3. Compute ω for every possible source-copier pair and flag any source-copier pair with a value of ω that exceeded a predetermined threshold;

4. Estimate the flawed answer key using the four methods; and

5. Compare the estimates of the flawed answer key with the real flawed answer key and determine the accuracy of the estimated flawed keys.

Simulations were conducted for tests with 60, 80, and 100 items; flawed answer key proportions of .05, .10, and .15; and flawed key user proportions of .05, .10, and .20. In the fully-crossed experiment, there were 27 different sets of parameters. For each combination of parameters, 100 repetitions of the simulation were performed. The next subsections elaborate on the steps of the simulation study.

**Step 1: Construct the exam and the flawed key.** Simulated exams consisting of 60, 80, or 100 items, drawn from a pool of 100 items, were administered. Using the 2-parameter logistic model, each item was defined by a discrimination parameter, \( a \), and a difficulty parameter, \( b \). Discrimination parameters were uniformly sampled from the interval \([0.5, 2]\), and difficulty parameters were sampled from the \( N(0,1) \) distribution. The number of incorrect answers in the flawed answer key was controlled by establishing a proportion of flawed answers. In the simulations, flawed answer key proportions of .05, .10, and .15 were analyzed. Without loss of generality, the items with flawed answer keys were always the first items in the exam (e.g., if the flawed answer key had incorrect answers for ten items, then items 1-10 were chosen to be the items with the incorrect answers). Each item had four response options, denoted \( A, B, C, \) and \( D \), with \( A \)
considered to be the correct response. If the item was one with an incorrect answer in the flawed answer key, then $B$ was considered the incorrect response listed in the flawed key.

**Step 2: Administer the exam.** For the 100 iterations of the simulation, the exam was administered to 1,000 examinees, a certain proportion of which used the flawed answer key. For each examinee, an ability parameter, $\theta_j$, was drawn from the $N(0,1)$ distribution, and a set of responses was generated and scored. Generation of the response pattern depended on whether the examinee used the flawed answer key.

If examinee $j$ did not use the flawed answer key, then the response string was created by calculating the probability that the examinee would answer each item correctly. Using the 2-parameter logistic model, the probability of an examinee with ability $\theta_j$ correctly answering an item with discrimination $a_i$ and difficulty $b_i$ was:

$$p(\text{correct}) = \left[1 + e^{-a_i(\theta_j-b_i)}\right]^{-1}.$$  \hfill (3-5)

After computing the probability, a random value, $\gamma$, from the interval $[0, 1]$ was generated, and the following rules were used for determining the response:

1. If $p(\text{correct}) > \gamma$, then the response was $A$, meaning the examinee answered correctly;
2. If $p(\text{correct}) \leq \gamma$ and $2\gamma < p(\text{correct}) + 1$, then the response was $B$;
3. If the conditions of rules 1 and 2 were not met and $5\gamma < p(\text{correct}) + 4$, then the response was $C$; and
4. If the conditions of rules 1-3 were not met, then the response was $D$. 

The last three rules were in place so that for examinees who answered incorrectly, $B$ was chosen 50% of the time, $C$ was chosen 30% of the time, and $D$ was chosen 20% of the time.

Examinees who used the flawed key responded as if they were copying from the flawed key, and were given an answer-copying rate, $\gamma_c$, drawn from the random variable $U(0.75, 0.95)$. The answer-copying rate determined the braindump user’s recall accuracy of the flawed answer key. Items that were not recalled were answered according to the examinee’s ability. This behavior was simulated by generating a random number, $\gamma_1$, from the interval $[0, 1]$ and applying the following rules:

1. if $\gamma_c > \gamma_1$, then the examinee selected the response from the flawed key; and
2. if $\gamma_c \leq \gamma_1$, then the response was determined by generating a random number, $\gamma_2$, from the interval $[0, 1]$ and comparing it to the examinee’s probability of correctly answering the item using rules 1-4 described previously.

This is similar to how Wollack simulated answer copiers in his work on the $\omega$ statistic (1997).

**Step 3: Compute $\omega$ for all possible source-copier pairs.** Although the 2-parameter logistic model was used to generate item responses, the $\omega$ statistic was defined using the nominal response model (Bock, 1972; Wollack, 1997). Therefore, item parameters for this model and estimates of examinee ability based on the nominal response model had to be computed before $\omega$ could be calculated. For these simulations, in-house data forensics tools were used to calculate the item parameters and ability
estimates. Baker (1992, pp. 257-272) gives an explanation on how to compute the item parameters and ability estimates under the nominal response model.

The $\omega$ statistic was computed for every possible source-copier pair. With 1,000 examinees, this resulted in 999,000 (1,000 x 999) comparisons. Source-copier pairs with $\omega$ values exceeding the predetermined threshold were flagged. For the simulations, a flagging threshold of 2.326 was used, which corresponds to an upper tail probability of 0.01 because $\omega$ is asymptotically standard normal (Wollack, 1997).

**Step 4: Estimate the flawed answer key using the four methods.** The four methods for estimating the flawed answer key were then applied to the results of computing $\omega$ for all of the source-copier pairs. Because it was not guaranteed that the four methods would all identify the same response pattern as the latent source, there could be up to four different estimates of the flawed key for each iteration of the simulation.

**Step 5: Determine the accuracy of the estimates of the flawed key.** Finally, the estimates of the flawed key were compared with the actual flawed key, and three accuracy values were computed. The first, denoted “flawed accuracy,” is the proportion of items that had incorrect answers in the flawed key for which the estimated flawed key had the same incorrect responses. The second, denoted “non-flawed accuracy,” is the proportion of items that had correct answers in the flawed key for which the estimated flawed key had the same correct responses. The third, denoted “overall accuracy,” is the proportion of items for which the estimated flawed key had the responses as the actual flawed key.
To understand the need for analyzing flawed accuracy and non-flawed accuracy separately, suppose that the flawed answer key is 92% correct. Using the actual answer key as an estimate of the flawed answer key will result in an estimate that is 92% accurate, despite the fact that it has no incorrect responses. However, in this situation, the flawed accuracy is 0% and the non-flawed accuracy is 100%, which shows that this is not a good estimate of the flawed answer key. Separating the accuracy for items with incorrect responses in the flawed key and items with correct responses in the flawed key provides a clearer picture of how well the flawed key was estimated.

Other ways of evaluating a method’s performance were determining how often the method selected a response pattern that exactly matched the flawed key, when the data contained one or more response patterns that were exact matches, and determining how often the method selected the response pattern from a braindump user as the estimate of the flawed key.

**Results from the Real-Life Data Sets**

Table 3-1 presents the results of applying the four methods to the four real-life data sets. Based on the average flawed accuracy and non-flawed accuracy values, Common_Max was observed to be the most effective method for estimating the flawed answer keys, followed by Common_Source, then Most_Flagged, then Max_Omega.
Table 3-1

Accuracies of the Estimated Flawed Answer Keys for the Real-Life Data Sets

<table>
<thead>
<tr>
<th></th>
<th>Estimation Method</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Common_Source</td>
<td>Max_Omega</td>
<td>Most_Flagged</td>
<td>Common_Max</td>
</tr>
<tr>
<td>Data Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flawed</td>
<td></td>
<td>.864</td>
<td>.409</td>
<td>.818</td>
<td>.909</td>
</tr>
<tr>
<td>Non-Flawed</td>
<td></td>
<td>1.000</td>
<td>.816</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>.950</td>
<td>.667</td>
<td>.933</td>
<td>.967</td>
</tr>
<tr>
<td>Data Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flawed</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
<td>.846</td>
<td>1.000</td>
</tr>
<tr>
<td>Non-Flawed</td>
<td></td>
<td>1.000</td>
<td>.957</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>1.000</td>
<td>.967</td>
<td>.967</td>
<td>1.000</td>
</tr>
<tr>
<td>Data Set 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flawed</td>
<td></td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Non-Flawed</td>
<td></td>
<td>.855</td>
<td>.891</td>
<td>.855</td>
<td>1.000</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>.783</td>
<td>.817</td>
<td>.783</td>
<td>1.000</td>
</tr>
<tr>
<td>Data Set 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flawed</td>
<td></td>
<td>.200</td>
<td>.000</td>
<td>.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Non-Flawed</td>
<td></td>
<td>.832</td>
<td>.716</td>
<td>.968</td>
<td>1.000</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>.800</td>
<td>.680</td>
<td>.920</td>
<td>1.000</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flawed</td>
<td></td>
<td>.516</td>
<td>.352</td>
<td>.416</td>
<td>.977</td>
</tr>
<tr>
<td>Non-Flawed</td>
<td></td>
<td>.922</td>
<td>.845</td>
<td>.956</td>
<td>1.000</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>.883</td>
<td>.783</td>
<td>.901</td>
<td>.992</td>
</tr>
</tbody>
</table>

Common_Max had perfect estimates of the flawed answer keys for three data sets. It was impossible to perfectly estimate the flawed key for data set 1 because no examinee had a response pattern that perfectly matched the purchased flawed key. However, for this data set, Common_Max selected the response pattern from the group of 12 identical tests that most closely matched the purchased flawed key. Therefore, Common_Max selected the best response pattern possible for all four data sets. Common_Max clearly set itself apart from the other methods on data sets 3 and 4. For data set 3, Common_Max was the only method with a non-zero flawed accuracy value.
A likely reason for the high level of accuracy for Common_Max is that all four data sets had large clusters of tests with identical response patterns. Data sets 1-4 had clusters of sizes 12, 30, 17, and 32, respectively. A test instance in one of these clusters would likely be the source with the maximum value of $\omega$ for all of the other test instances in the cluster, which would elevate the number of potential copiers that achieved their maximum value of $\omega$ with this source.

Common_Source and Most_Flagged had similar average flawed accuracies and non-flawed accuracies, which makes sense because the two methods use similar data for selecting a response pattern as the estimate of the flawed answer key. Common_Source had a higher average flawed accuracy, and Most_Flagged had a higher average non-flawed accuracy. Both methods had their best performance for data set 2. For data set 2, Common_Source had a perfect estimate, and Most_Flagged had a flawed accuracy of .846 (11 out of 13) and perfect non-flawed accuracy.

These two methods struggled with data sets 3 and 4. Neither method had a flawed accuracy greater than zero for data set 3. For data set 4, Common_Source had a flawed accuracy of .200 (1 out of 5), and Most_Flagged had a flawed accuracy of zero. For data set 3, the methods appeared to have selected a response pattern from a group of examinees who were using a different flawed key. For data set 4, the braindump was so widely used that selecting the incorrect answers from the flawed key was not anomalous, so very few source-copier pairs had $\omega$ values exceeding the threshold. Also, data sets 3 and 4 had flawed answer keys that were 92% correct and 95% correct, respectively, so these methods may struggle with estimating high-quality flawed answer keys.
Max_Omega had the lowest average values for the flawed accuracy and the non-flawed accuracy. The method had its best performance for data set 2, with a perfect flawed accuracy and a non-flawed accuracy of .957 (45 out of 47). However, this method’s flawed accuracies for the other three data sets were .409 or zero, and the non-flawed accuracies varied between .716 and .891. It appeared that the method is prone to selecting response patterns from isolated instances of answer copying, such as when one examinee copies nearly all of another examinee’s responses. This intuitively makes sense because Max_Omega finds the source with the highest value of omega, which is likely to occur when one examinee copies nearly all of another examinee’s responses.

**Simulation Results**

Table 3-2 presents a summary of the flawed accuracy, non-flawed accuracy, and overall accuracy for the four methods. In the simulation study, there were 27 different combinations of test length, proportion of incorrect responses in the flawed key, and proportion of braindump users. Each combination of simulation parameters was simulated 100 times, so the methods were tested 2,700 times. The accuracy values listed in Table 3-2 are the averages of the 2,700 accuracy values.
Table 3-2

Average Accuracies of the Estimated Flawed Answer Keys for the Simulation Study

<table>
<thead>
<tr>
<th>Method</th>
<th>Flawed</th>
<th>Non-Flawed</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common_Source</td>
<td>.943</td>
<td>.922</td>
<td>.924</td>
</tr>
<tr>
<td>Max_Omega</td>
<td>.876</td>
<td>.896</td>
<td>.895</td>
</tr>
<tr>
<td>Most_Flagged</td>
<td>.928</td>
<td>.967</td>
<td>.963</td>
</tr>
<tr>
<td>Common_Max</td>
<td>.835</td>
<td>.897</td>
<td>.890</td>
</tr>
</tbody>
</table>

All four methods had overall accuracies of .890 or greater. Common_Source was the only method for which the flawed accuracy was greater than the non-flawed accuracy. As was observed in the results from the real-life data, Common_Source and Most_Flagged had similar performance, with Common_Source having the higher flawed accuracy, and Most_Flagged having the higher non-flawed accuracy. Max_Omega and Common_Max had similar performance, and were less accurate than Common_Source and Most_Flagged. It was peculiar that Common_Max had such low accuracy, considering how well it performed on the real-life data sets.

Simulation results for each method are presented individually. In Tables 3-3 through 3-6, the first two columns give the proportion of items with incorrect answers in the flawed key and the proportion of braindump users, respectively. The next three columns give the average flawed accuracy for the simulations for exams with 60 items, 80 items, and 100 items, respectively. The last three columns give the non-flawed accuracy in a similar manner. Overall accuracy is not shown.
Common_Source Simulation Results

Table 3-3 gives the simulation results for the Common_Source method. Flawed accuracy was almost always higher than non-flawed accuracy, except primarily for simulations with a braindump user proportion of .20. The flawed accuracy values ranged between .828 and 1.000, and the non-flawed accuracy values ranged between .872 and .979. There was one set of parameters for which the method had a perfect flawed accuracy. The method had its best performance for simulations with the highest rate of flaws in the key (.15) and lowest rate of braindump users (.05).

Table 3-3

<table>
<thead>
<tr>
<th>Flaws</th>
<th>Users</th>
<th>Flawed Accuracy</th>
<th>Non-Flawed Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>60 Items</td>
<td>80 Items</td>
</tr>
<tr>
<td>.05</td>
<td>.05</td>
<td>.923</td>
<td>.965</td>
</tr>
<tr>
<td>.05</td>
<td>.10</td>
<td>.910</td>
<td>.963</td>
</tr>
<tr>
<td>.05</td>
<td>.20</td>
<td>.940</td>
<td>.918</td>
</tr>
<tr>
<td>.10</td>
<td>.05</td>
<td>.997</td>
<td>.995</td>
</tr>
<tr>
<td>.10</td>
<td>.10</td>
<td>.942</td>
<td>.959</td>
</tr>
<tr>
<td>.10</td>
<td>.20</td>
<td>.903</td>
<td>.901</td>
</tr>
<tr>
<td>.15</td>
<td>.05</td>
<td>1.000</td>
<td>.999</td>
</tr>
<tr>
<td>.15</td>
<td>.10</td>
<td>.978</td>
<td>.978</td>
</tr>
<tr>
<td>.15</td>
<td>.20</td>
<td>.950</td>
<td>.893</td>
</tr>
</tbody>
</table>

Test length had the strongest effect on flawed accuracy for simulations with a low rate of flaws in the key (.05) and a high rate of braindump users (.20). On the other hand, test length hardly had any effect of the flawed accuracy for simulations with a high rate of flaws in the key (.15) and a low rate of braindump users (.05). There was no discernible pattern to how test length affected the non-flawed accuracy.
The flawed accuracy increased as the proportion of incorrect responses in the key increased, although this may have been caused by the flawed accuracy data becoming less granular. The non-flawed accuracy was not significantly affected by changing the proportion of incorrect responses in the flawed key, which makes sense because the majority of the items had correct responses in the flawed key.

Flawed accuracy tended to decrease for a given test length and flaw proportion as the proportion of braindump users increased. The greatest decreases were observed in simulations with 100 items. For these simulations, the flawed accuracy would decrease by .120 or more as a result of increasing the proportion of braindump users from .05 to .20. There was no strong pattern to changes in non-flawed accuracy as a result of changing the proportion of braindump users.

With 27 sets of parameters and 100 iterations for each set of parameters, test data were simulated 2,700 times. Common_Source selected a response pattern from an examinee who used the braindump as the estimate of the flawed answer key for 2,616 of the 2,700 iterations (96.9%). Common_Source was only able to select a response pattern that perfectly matched the flawed answer key, when such a response pattern was present, for simulations with a braindump user proportion of .05 and a flaw proportion of .10 or greater. For such simulations, the selection rate of exact matches ranged from 2.1% (60 items, .10 flaw proportion, and .05 user proportion) to 55.8% (100 items, .15 flaw proportion, and .05 user proportion).
Max_Omega Simulation Results

Table 3-4 gives the simulation results for the Max_Omega method. This method typically had a higher non-flawed accuracy than flawed accuracy. The flawed accuracy values ranged between .697 and .991, and the non-flawed accuracy values ranged between .814 and .960. The method had its best performance for simulations with the highest rate of flaws in the key (.15) and lowest rate of braindump users (.05).

Table 3-4

Simulation Results for Max_Omega

<table>
<thead>
<tr>
<th>Flaws</th>
<th>Users</th>
<th>60 Items</th>
<th>80 Items</th>
<th>100 Items</th>
<th>60 Items</th>
<th>80 Items</th>
<th>100 Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td>.05</td>
<td>.713</td>
<td>.888</td>
<td>.862</td>
<td>.814</td>
<td>.886</td>
<td>.865</td>
</tr>
<tr>
<td>.05</td>
<td>.10</td>
<td>.697</td>
<td>.855</td>
<td>.826</td>
<td>.824</td>
<td>.881</td>
<td>.878</td>
</tr>
<tr>
<td>.05</td>
<td>.20</td>
<td>.840</td>
<td>.900</td>
<td>.790</td>
<td>.885</td>
<td>.919</td>
<td>.901</td>
</tr>
<tr>
<td>.10</td>
<td>.05</td>
<td>.972</td>
<td>.981</td>
<td>.971</td>
<td>.919</td>
<td>.926</td>
<td>.928</td>
</tr>
<tr>
<td>.10</td>
<td>.10</td>
<td>.808</td>
<td>.884</td>
<td>.861</td>
<td>.851</td>
<td>.903</td>
<td>.882</td>
</tr>
<tr>
<td>.10</td>
<td>.20</td>
<td>.803</td>
<td>.843</td>
<td>.829</td>
<td>.870</td>
<td>.899</td>
<td>.905</td>
</tr>
<tr>
<td>.15</td>
<td>.05</td>
<td>.989</td>
<td>.991</td>
<td>.978</td>
<td>.946</td>
<td>.960</td>
<td>.945</td>
</tr>
<tr>
<td>.15</td>
<td>.10</td>
<td>.917</td>
<td>.948</td>
<td>.939</td>
<td>.899</td>
<td>.910</td>
<td>.900</td>
</tr>
<tr>
<td>.15</td>
<td>.20</td>
<td>.861</td>
<td>.836</td>
<td>.861</td>
<td>.886</td>
<td>.896</td>
<td>.911</td>
</tr>
</tbody>
</table>

Test length had the strongest effect on flawed accuracy for simulations with a low rate of flaws in the key (.05). For these simulations, changing the test length while holding the other parameters constant could change the flawed accuracy .110 or more. There was no discernible pattern to how test length affected the non-flawed accuracy.

The flawed accuracy increased as the proportion of incorrect responses in the key increased, although this may have been caused by the flawed accuracy data becoming
less granular. The non-flawed accuracy increased as well, but the increase was not as strong as that seen in the flawed accuracy.

Flawed accuracy tended to decrease for a given test length and flaw proportion as the proportion of braindump users increased. However, the opposite was true for simulations with 60 or 80 items and a flaw proportion of .05. There was no strong pattern to changes in non-flawed accuracy as a result of changing the proportion of braindump users.

Max_Omega selected a response pattern from an examinee who used the braindump as the estimate of the flawed answer key for 2,379 of the 2,700 iterations (88.1%). This was the lowest selection rate of braindump users for any of the four methods. Max_Omega was only able to select a response pattern that perfectly matched the flawed answer key, when such a response pattern was present, for simulations with a braindump user proportion of .05 and a flaw proportion of .10 or greater. For such simulations, the selection rate of exact matches ranged from 2.6% (80 items, .10 flaw proportion, and .05 user proportion) to 38.2% (80 items, .15 flaw proportion, and .05 user proportion).

Most_Flagged Simulation Results

Table 3-5 gives the simulation results for the Most_Flagged method. This method typically had a higher non-flawed accuracy than flawed accuracy. The flawed accuracy values ranged between .800 and 1.000, and the non-flawed accuracy values ranged between .909 and .997. There were three sets of parameters for which the method had a
perfect flawed accuracy. The method had its best performance for simulations with a high rate of flaws in the key (.10 or .15) and lowest rate of braindump users (.05).

Table 3-5

*Simulation Results for Most_Flagged*

<table>
<thead>
<tr>
<th>Flaws</th>
<th>Users</th>
<th>60 Items</th>
<th>80 Items</th>
<th>100 Items</th>
<th>60 Items</th>
<th>80 Items</th>
<th>100 Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td>.05</td>
<td>.873</td>
<td>.955</td>
<td>.968</td>
<td>.909</td>
<td>.960</td>
<td>.966</td>
</tr>
<tr>
<td>.05</td>
<td>.10</td>
<td>.913</td>
<td>.948</td>
<td>.880</td>
<td>.940</td>
<td>.964</td>
<td>.965</td>
</tr>
<tr>
<td>.05</td>
<td>.20</td>
<td>.883</td>
<td>.948</td>
<td>.828</td>
<td>.950</td>
<td>.966</td>
<td>.970</td>
</tr>
<tr>
<td>.10</td>
<td>.05</td>
<td>1.000</td>
<td>1.000</td>
<td>.999</td>
<td>.993</td>
<td>.995</td>
<td>.993</td>
</tr>
<tr>
<td>.10</td>
<td>.10</td>
<td>.952</td>
<td>.975</td>
<td>.955</td>
<td>.956</td>
<td>.965</td>
<td>.965</td>
</tr>
<tr>
<td>.10</td>
<td>.20</td>
<td>.807</td>
<td>.864</td>
<td>.800</td>
<td>.947</td>
<td>.971</td>
<td>.971</td>
</tr>
<tr>
<td>.15</td>
<td>.05</td>
<td>1.000</td>
<td>.999</td>
<td>.999</td>
<td>.997</td>
<td>.994</td>
<td>.991</td>
</tr>
<tr>
<td>.15</td>
<td>.10</td>
<td>.992</td>
<td>.987</td>
<td>.953</td>
<td>.957</td>
<td>.960</td>
<td>.966</td>
</tr>
<tr>
<td>.15</td>
<td>.20</td>
<td>.868</td>
<td>.902</td>
<td>.823</td>
<td>.953</td>
<td>.964</td>
<td>.969</td>
</tr>
</tbody>
</table>

Test length had the strongest effect on flawed accuracy for simulations with a low rate of flaws in the key (.05). For these simulations, changing the test length while holding the other parameters constant could change the flawed accuracy by as much as .120. There was no discernible pattern to how test length affected the non-flawed accuracy.

The flawed accuracy tended to increase as the proportion of incorrect responses in the key increased, but this was not true for simulations with a braindump user proportion of .20. The non-flawed accuracy did not appear to increase as the flaw proportion increased.

For a given test length and flaw proportion, the flawed accuracy for simulations with a braindump user proportion of .05 or .10 were generally close together in size.
Between the two braindump user proportions, the higher flawed accuracy was usually associated with the .05 user rate. However, simulations with a braindump user proportion of .20 had a flawed accuracy that was much lower than that observed for simulations with a .05 or .10 user rate. This difference was not observed for the non-flawed accuracy. Non-flawed accuracy for a given set of parameters was usually highest with a user rate of .05, but the non-flawed accuracy values for the different parameter sets never differed by more than .046 as a result of changing the user rate.

Most_Flagged selected a response pattern from an examinee who used the braindump as the estimate of the flawed answer key for 2,627 of the 2,700 iterations (97.3%). This was the highest selection rate of braindump users for any of the four methods. Most_Flagged had the widest range of parameter sets for which it was able to select an exact match, when one was present. For this method, the selection rate of exact matches ranged from 1.5% (100 items, .15 flaw proportion, and .10 user proportion) to 96.7% (60 items, .15 flaw proportion, and .05 user proportion). For simulations with a user proportion of .05 and a flaw proportion of .10 or .15, the selection rate of exact matches was 86.9% or greater.

**Common_Max Simulation Results**

Table 3-6 gives the simulation results for the Common_Max method. For all but two sets of simulation parameters, flawed accuracy was lower than non-flawed accuracy. The flawed accuracy values ranged between .613 and .915, and the non-flawed accuracy values ranged between .863 and .942. The method generally had its best performance for simulations with the lowest rate of braindump users (.05).
Table 3-6

Simulation Results for Common_Max

<table>
<thead>
<tr>
<th>Flaws</th>
<th>Users</th>
<th>60 Items</th>
<th>80 Items</th>
<th>100 Items</th>
<th>Flawed Accuracy</th>
<th>Non-Flawed Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td>.05</td>
<td>.847</td>
<td>.905</td>
<td>.882</td>
<td>.886</td>
<td>.911</td>
</tr>
<tr>
<td>.05</td>
<td>.10</td>
<td>.800</td>
<td>.890</td>
<td>.872</td>
<td>.867</td>
<td>.885</td>
</tr>
<tr>
<td>.05</td>
<td>.20</td>
<td>.863</td>
<td>.815</td>
<td>.880</td>
<td>.898</td>
<td>.873</td>
</tr>
<tr>
<td>.10</td>
<td>.05</td>
<td>.915</td>
<td>.890</td>
<td>.888</td>
<td>.891</td>
<td>.920</td>
</tr>
<tr>
<td>.10</td>
<td>.10</td>
<td>.780</td>
<td>.878</td>
<td>.860</td>
<td>.863</td>
<td>.903</td>
</tr>
<tr>
<td>.10</td>
<td>.20</td>
<td>.818</td>
<td>.828</td>
<td>.816</td>
<td>.892</td>
<td>.884</td>
</tr>
<tr>
<td>.15</td>
<td>.05</td>
<td>.897</td>
<td>.891</td>
<td>.844</td>
<td>.903</td>
<td>.922</td>
</tr>
<tr>
<td>.15</td>
<td>.10</td>
<td>.823</td>
<td>.848</td>
<td>.798</td>
<td>.875</td>
<td>.900</td>
</tr>
<tr>
<td>.15</td>
<td>.20</td>
<td>.639</td>
<td>.613</td>
<td>.766</td>
<td>.865</td>
<td>.865</td>
</tr>
</tbody>
</table>

Test length did not have a noticeable effect on the flawed accuracy or non-flawed accuracy. A difference of .153 was observed between test lengths of 80 and 100 items for simulations with a .15 flaw proportion and .20 braindump user proportion, but there was no clear pattern to changes in either type of accuracy due to changing the test length.

Increasing the flaw proportion most strongly affected the flawed accuracy when the braindump user proportion was .20. For simulations with a braindump user proportion of .20, increasing the flaw proportion from .05 to .15 decreased the flawed accuracy by .114 (for simulations with 100 items) to .224 (for simulations with 60 items). Non-flawed accuracy was not affected by changing the flaw proportion.

The effect that changing the braindump user proportion had on the flawed accuracy depended on the flaw proportion and item count. For simulations with a flaw proportion of .15 and 60 or 80 items, increasing the user proportion from .05 to .20 decreased the flawed accuracy by .258 and .278, respectively. All other combinations of test length and flaw proportion had changes in flawed accuracy of .135 or less as a result...
of changing the user proportion. Also, the non-flawed accuracy was not strongly affected by changes in the user proportion.

Common_Max selected a response pattern from an examinee who used the braindump as the estimate of the flawed answer key for 2,531 of the 2,700 iterations (93.7%). Common_Max was the only method that was able to select a response pattern that perfectly matched the flawed answer key, when such a response pattern was present, for simulations with a braindump user proportion of .20. It had the most success in doing so for simulations with 60 items, and the selection rate of exact matches ranged from 10.0% (60 items, .15 flaw proportion, and .20 user proportion) to 35.0% (60 items, .05 flaw proportion, and .20 user proportion).

**Discussion**

Four methods were developed for using the results from computing $\omega$ for every possible source-copier pair to estimate an unknown flawed answer key, and two of those have shown promise based on the results of the real-life examples and the simulation study. Common_Max had the best performance on the real-life data sets, but struggled in the simulation study. This was likely caused by the conditions of the simulation study being qualitatively different from those of the real-life data sets (i.e., the way flawed key usage was simulated may have been different from how it appears in practice).

Most_Flagged had average performance on the real-life data sets, but had the best performance in the simulation study.
Although Common_Max and Most_Flagged demonstrated strong performance, the intent of this research is not to recommend any particular method above the others. Rather, the intent is to determine whether it is possible to employ answer-copying statistics to accurately estimate unknown flawed answer keys. Overall, the results of this research do provide evidence that this is indeed possible. However, more research is needed to determine the best method for estimating the unknown flawed answer key.

Additional research in this area could focus on different ways to use the results of computing $\omega$ to estimate the unknown flawed answer key. The four methods presented and studied here are not the only ways these results can be used. Also, this research focused exclusively on using $\omega$ as the answer-copying statistic. Further research could use results from other answer-copying statistics and compare methods based on them. Also, the simulation study for this research only allowed for one flawed key. It would be interesting to see how the methods perform in situations where more than one flawed key is available. Finally, applying these methods to additional real-life data sets and evaluating them in other simulation studies would help to determine whether some are more suitable for practical application than others.

Lastly, it is important to discuss some limitations of this research. Because $\omega$ is based on IRT, high rates of braindump users will contaminate the item parameter estimates, and this contamination affects the efficacy of the methods. This was encountered in data set 4 and the simulations with a braindump user proportion of .20. Also, the nature of the four methods is such that they will always identify a response pattern as being the best approximation of the flawed key, even when no flawed key was
used by any member of the testing population. Therefore, these methods are not suited for routinely scanning the test result data to determine whether a flawed answer key has been used. Rather, they should be employed only after a testing program suspects that a flawed answer key has been used. Information from a test security tip line, or noticing that some items have lower p-values than expected, are some reasons for suspecting that a group of examinees may have used a flawed answer key.
CHAPTER IV
CASE STUDY: PARALLELIZING ANSWER-COPYING ANALYSIS

Analysis of item response data can be computationally intensive, and is only feasible using the power of computers (Baker, 1992, p. 86). In the field of test security, data forensics analysis of item data requires large amounts of computation. An example of this is looking for evidence of collusion by computing an answer-copying statistic for each potential source-copier pair in a testing population.

Answer-copying statistics have been used to test the hypothesis of copying when there is suspicion that one examinee copied from another. A proctor’s report of behavior that could indicate copying is one motivation for such an analysis. However, there are several reasons for computing an answer-copying statistic for every possible source-copier pair in a testing population. The first is that such a procedure is a standard part of monitoring test security through data forensics analysis. Maynes and Wollack (2017) demonstrated that clusters of extremely similar tests, such as would occur from answer copying, are strong evidence of test fraud. Another reason is for the purposes of research. When new answer-copying statistics are developed, their type I error rates and power must be analyzed. This requires using data with known answer copiers, computing the statistic for each possible pair, and determining whether the statistic correctly identified an examinee as an answer copier or normal test taker.

Computationally-intensive tasks such as answer-copying analysis are more feasible with parallel computing. Nowadays, computer hardware consisting of multiple processors with shared memory is being used to increase computer capability (Chapman,
Jost, & van der Pas, 2008, pp. 1-2). These processors can operate in parallel, and this type of architecture is found in both the central processing unit (CPU) and the graphics processing unit (GPU) of modern computers. Thus, the potential for parallel computing is widely available.

Two programming models that have been developed to enable parallel computing on shared-memory machines with multiple processors are OpenMP (Open Multi-Processing) and OpenACC (Open Accelerators). The purpose of this paper is to demonstrate the utility of OpenMP and OpenACC for parallelizing computer programs that analyze test data. To do this, the paper presents a case study in which OpenMP and OpenACC are used to parallelize an answer-copying analysis using the $\omega$ statistic (Wollack, 1997). It is hoped that individuals who work with test data will see how they could use OpenMP or OpenACC to parallelize their own code used for data analysis.

**Review of $\omega$**

In his derivation of $\omega$, Wollack (1997) denotes the potential copier and the potential source as $C$ and $S$, respectively. An examinee’s ability is represented by the quantity $\theta$. The value $u_i$ is the response to item $i$; hence $u_{ic}$ is the response chosen by the potential copier to item $i$, and $u_{is}$ is the response chosen by the potential source to item $i$. An examinee’s response pattern for the $n$ items is represented by the vector $U$, and the item parameters form the matrix $\xi$. For convenience, define the indicator function $I(u_{ic} = u_{is})$ to equal 1 when $u_{ic} = u_{is}$, and 0 otherwise.

With these definitions, the observed number of matches between $C$ and $S$ is given by:
Because the nominal response model (Bock, 1972) was used, the probability of \( C \) selecting the same response as \( S \) to item \( i \) is

\[
p(u_{ic} = u_{is} | \theta_c, u_{is}, \xi) = \frac{e^{\lambda_{uic} \theta_c + \zeta_{uic}}}{\sum_{k=1}^{m} e^{\lambda_{uk} \theta_c + \zeta_{uk}}},
\]

where \( k \) indexes the \( m \) response options for item \( i \), and \( \lambda_{uk} \) and \( \zeta_{uk} \) are the nominal response model parameters for response \( k \) to item \( i \). The subscript \( u_{is} \) is used to indicate the response chosen by the source. The sum of these probabilities,

\[
E(h_{CS} | \theta_c, U_s, \xi) = \sum_{i=1}^{n} p(u_{ic} = u_{is} | \theta_c, U_s, \xi),
\]

gives the expected value of \( h_{CS} \) conditioned on the potential copier’s ability, the source’s response pattern, and the item parameters (Wollack, 1997). Because the item responses are assumed to be statistically and locally independent, and two responses can either match or not match, the number of matches is a sum of independent Bernoulli variables.

Therefore, the variance of \( h_{CS} \) is given by

\[
\sigma^2_{h_{CS}} = \sum_{i=1}^{n} p(u_{ic} = u_{is} | \theta_c, U_s, \xi) [1 - p(u_{ic} = u_{is} | \theta_c, U_s, \xi)].
\]

The value of \( \omega \) is the standardized number of matches, and has the formula

\[
\omega = \frac{h_{CS} - E(h_{CS} | \theta_c, U_s, \xi)}{\sigma_{h_{CS}}},
\]

where \( h_{CS} \), \( E(h_{CS} | \theta_c, U_s, \xi) \), and \( \sigma^2_{h_{CS}} \) are given by Equations 4-1, 4-3, and 4-4, respectively.
For any pair of examinees, two values of $\omega$ can be computed because there are two ways of designating the source and copier. Therefore, computing $\omega$ for all possible source-copier pairs in a population of $N$ examinees will require $N(N - 1)$ computations. The distribution of $\omega$ is not population-dependent, and is known to asymptotically have a standard normal distribution (Wollack, 1997). Therefore, $\omega$ for one source-copier pair can be computed separately from that of another source-copier pair, even if the same tests are used in both pairs, which is ideal for parallelization.

**Case Study Programming Exercise**

A C program that computes $\omega$ for all possible source-copier pairs in a population of examinees was written and parallelized using OpenMP and OpenACC. Figure 4-1 shows pseudocode for the sequential version of this program.

```c
for (i=0; i<examinee_count; i++)
{
    for (j=0; j<i; j++)
    {
        Compute $\omega$ with source i and copier j;
    }
    for (j=i+1; j<examinee_count; j++)
    {
        Compute $\omega$ with source i and copier j;
    }
}
```

*Figure 4-1.* Sequential pseudocode for answer-copying analysis with $\omega$.

For the examples shown in this paper, response patterns to a 100-item test were generated. Examples where the code was parallelized over the CPU had a population of
3,000 examinees. Altogether, \( \omega \) was computed for 8,997,000 pairs, and the sequential program had a computation time of 611 seconds. Examples where the code was parallelized over the GPU had a population of 2,000 examinees. Altogether, \( \omega \) was computed for 3,998,000 pairs, and the sequential program had a computation time of 114 seconds.

This programming exercise was implemented on a Lenovo® ThinkPad™ laptop with an Intel® Core™ i7-6700HQ CPU containing eight cores and an NVIDIA® Quadro™ M1000M GPU. OpenMP code was compiled using the GNU Compiler Collection, version 5.3.0, which supports OpenMP 4.0, and OpenACC code was compiled using the PGI® Community Edition C Compiler, version 17.4, which supports OpenACC 2.0.

**Directive-Based Parallelization on the CPU**

The first attempt at parallelizing the code for the case study focused on parallelizing the code shown in Figure 4-1 across the cores of the CPU. Because OpenMP was originally designed for this task, this section describes how to use OpenMP to enable the parallelization.

**Overview of OpenMP**

OpenMP is an application programming interface (API) for C/C++ and Fortran that enables parallel processing on shared memory machines (Barney, 2013). OpenMP consists of compiler directives, environment variables, and runtime library routines. Most of the parallelism offered by OpenMP is enabled by these compiler directives. The
environment variables and runtime library routines allow the user to control the level and
amount of parallelism. Therefore, most of the discussion on OpenMP in this paper will
focus on the compiler directives. Environment variables and runtime library routines will
be discussed only as needed. The environment variables and compiler directives are
automatically understood by a compiler that has OpenMP compilation enabled.
However, to use the runtime library routines, a C/C++ program must include the header
file *omp.h*, and a Fortran program must include the module *omp_lib* (Chapman, Jost,

A compiler directive is a line of code that appears as a comment to a compiler,
unless the compiler has been designed with the capability to interpret and apply the
directive. There are two advantages to using compiler directives for enabling the
parallelization. The first is that this framework allows a programmer to parallelize
existing code simply by adding the appropriate directives rather than by rewriting the
code (Chapman, Jost, & van der Pas, 2007, p. 35). The second is that a program written
using OpenMP can still be compiled and run on a machine that is not configured to
accommodate OpenMP (Yliluoma, 2007). Compiler directives for C/C++ take the form
#pragma omp directive_name, and directives for Fortran take the form !$omp
directive_name. The programming for this research was done in C/C++, so the
remainder of the paper will present the directives in these languages. Syntax for
corresponding Fortran directives can be found in the complete OpenMP 4.5 specifications
The parallel Directive

OpenMP enables parallelism using the concept of multithreading (Chapman, Jost, & Van der Pas, 2008, pp. 23-24). A thread is an independent stream of instructions, and multithreading is the process of simultaneously creating several threads. OpenMP provides a framework for the threads to be executed concurrently on the multiple cores found in the CPU, which results in parallel processing. In its treatment of multithreading, OpenMP uses a fork-join model (Barney, 2013). At execution, one thread, called the master thread, is initialized. When the program reaches the code that is marked for parallelization, the master thread forks into a team of two or more threads. Upon completion of the parallelized code, the threads join back into one.

Creation of this team of threads is accomplished through the parallel directive. The combination of a parallel directive and the code it contains forms a parallel construct (Chapman, Jost, & van der Pas, 2007, p. 25). When the master thread encounters a parallel construct, the thread forks as described previously. All of the code contained within the parallel construct will be executed by each thread of the team. Although this may give redundant results, other directives, which will be discussed later, can prevent the redundancy.

This section concludes with a discussion on the number of threads created by the parallel directive. Typically, the number of threads equals the number of CPU cores (Yliluoma, 2007). OpenMP possesses an environment variable named OMP_NUM_THREADS, which can be set to a desired number of threads at the command line (Chapman, Jost, & van der Pas, 2007, p. 97). All parallel constructs in the program
will use this number of threads. For situations where different parallel constructs would require different numbers of threads, the function omp_set_num_threads() can be used. Also, the parallel directive has a clause named num_threads that declares how many threads are to be used for the parallel construct.

The **for** Directive

It has been shown that the parallel directive is used to get all threads to execute the same code. A more practical use of parallel processing is to get the threads to divide execution of the code among themselves. This is known as work-sharing, and a common application of it is to assign iterations of a loop to the threads. Loops in C/C++ can be parallelized through the **for** directive, and loops in Fortran can be parallelized through the **do** directive (Chapman, Jost, & van der Pas, 2007, p. 26). Using these directives is perhaps the simplest way to parallelize code with OpenMP.

The **for** directive can be combined with the parallel directive to form a compound directive. Figure 4-2 shows a version of the code in Figure 4-1 after it has been parallelized by adding the parallel and for directives to the outer loop. The private(j) clause indicates that each thread is to receive its own copy of the variable j, which controls the iterations of the inner loop (Chapman, Yost, & Van der Pas, 2007, p. 72). This prevents a thread from overwriting the looping variable used by another thread. Without this clause, iterations of the inner loop may be omitted or duplicated. By default, the looping variable used in a parallelized loop is made private, so the variable i used in the outer loop does not need to be included in the list of variables to be made private.
Enabling OpenMP and adding the directive `#pragma omp parallel for private(j)` to the outermost loop decreased the computation time to 106 seconds, so the code in Figure 4-2 was approximately 5.8 times faster than the code in Figure 4-1. This is a substantial increase from just one line of code, which demonstrates both the power and the elegance of OpenMP.

OpenMP allows for nested parallelism, so it is possible to parallelize the two inner loops as well. The environment variable `OMP_NESTED` enables or disables nested parallelism. Nested parallelism is enabled by assigning a non-zero integer value to this variable. This can be done at the command line, or through the function `omp_set_nested()` (Oracle, 2010, chapter 4). One requirement for utilizing nested parallelism is that multiple worksharing directives cannot share the same parallel directive (OpenMP Architecture Review Board, 2015, p. 56).

Figure 4-3 shows the code from the previous two examples with parallelism enabled for the outer and inner loops. Because the looping variable `j` is automatically
made private by the parallel for directive, the private(j) clause is no longer needed for the outermost loop. However, this clause can be retained without affecting the performance of the code. Explicit statements of which variables can be accessed by the threads can be beneficial in that they make it easier for others to understand the code, and they ensure that the compiler provides the correct variables to the threads during execution.

```c
omp_set_nested(1);
#pragma omp parallel for
for (i=0; i<examinee_count; i++)
{
    #pragma omp parallel for
    for (j=0; j<i; j++)
    {
        Compute ω with source i and copier j;
    }

    #pragma omp parallel for
    for (j=i+1; j<examinee_count; j++)
    {
        Compute ω with source i and copier j;
    }
}
```

*Figure 4-3.* Answer-copying analysis pseudocode with the outer and inner loops parallelized.

This version had a computation time of 96 seconds, which was approximately 6.4 times faster than the sequential version. At this point, only four lines of code have been added to the original program: three parallel for directives and a function call to enable nested parallelism. This demonstrates another benefit of OpenMP, in that a programmer can incrementally parallelize a program for maximum effect.
A different version of the code in Figure 4-3 was implemented in which the two inner for loops were combined into one. To prevent a response pattern from being compared to itself, the statement `if (i==j) continue;` was added to the inner loop. This version was slightly faster than the version shown in Figure 4-3, and had a computation time of 95 seconds.

**The sections Directive**

For the code in Figure 4-3, the two inner for loops have been parallelized, but the two inner loops themselves execute sequentially (i.e., the loop from 0 to i executes first, followed by the loop from i+1 to the end). Because the computations performed by the two loops are independent of each other, there is potential to further reduce the computation time by parallelizing the execution of the two inner loops. This can be accomplished by using the `sections` directive.

A sections construct consists of the `sections` directive and the code contained within it (OpenMP Architecture Review Board, 2015, p. 65). Each section of work within the sections construct is identified by the directive `#pragma omp section`. Figure 4-4 shows pseudocode for computing ω that utilizes the `sections` directive. The `private(j)` returns to ensure that the loop in each section does not interfere with the loop in another section. The `num_threads(2)` clause was included because the `sections` directive assigns one thread to each section. Hence, there was no reason to create more than two threads for this construct.
The code shown in Figure 4-4 had a computation time of 104 seconds, which was approximately 5.9 times faster than the sequential version. This was slightly faster than the computation time observed from adding the `parallel for` directive to the outermost loop, as shown in Figure 4-2. The reason why the `sections` directive did not produce a significant decrease in the computation time is that although the two inner loops were executed concurrently, the loops themselves were not parallelized.

It is possible to parallelize the loop in each section with the `#pragma omp parallel for` directive. This version of the code had a computation time of 96 seconds, so it was no faster than the version shown in Figure 4-3. It appears that the two inner loops executing sequentially was not a significant bottleneck to the execution, so there was not much to be gained by having them execute in parallel. This can be seen by the fact that

```c
omp_set_nested(1);
#pragma omp parallel for
for (i=0; i<examinee_count; i++)
{
    #pragma omp parallel sections private(j) \ 
    num_threads(2)
    {
        #pragma omp section
        for (j=0; j<i; j++)
        {
            Compute \( \omega \) with source \( i \) and copier \( j \);
        }
        #pragma omp section
        for (j=i+1; j<examinee_count; j++)
        {
            Compute \( \omega \) with source \( i \) and copier \( j \);
        }
    }
}
```

*Figure 4-4. Answer-copying analysis pseudocode using the `sections` directive.*
the version of the code with one inner loop did not have a significantly different computation time than the version with two inner loops.

**Parallelizing the Code That Computes $\omega$**

So far, each attempt to parallelize the code has focused on parallelizing the nested loops required to iterate over the set of possible source-copier pairs. None of the versions presented have attempted to parallelize the code that computes $\omega$ for a given source-copier pair. Computation of $\omega$ for a given source-copier pair involves looping over the items in the exam to calculate the number of matching responses, the expected number of matching responses, and the variance of the number of matching responses. These three values are then used to compute $\omega$ as shown in Equation 4-5. Pseudocode for this calculation is shown in Figure 4-5.

```
for (k=0; k<ITEMS; k++)
{
    if (copier_response == source_response)
    {
        obs_matches = obs_matches + 1;
    }
    compute probability of match (pmatch);
    exp_matches = exp_matches + pmatch;
    var = var + pmatch * (1 - pmatch);
}
omega = (obs_matches - exp_matches)/sqrt(var);
```

*Figure 4-5.* Pseudocode for computing $\omega$ for a given source-copier pair.

**Parallelizing with a for Directive**

Up to this point, the greatest decreases in computation time have been achieved by adding `#pragma omp parallel for` to the loops in the program. Therefore, it
makes sense to take that approach with the for loop in Figure 4-5. This version of the code with three nested for loops had a computation time of 3,095 seconds, so it was approximately 5.1 times slower than the sequential code. The reason for this dramatic increase in computation time is that parallelizing the innermost loop in this manner requires a parallel construct. For 3,000 examinees, the innermost loop is executed 8,997,000 times, so this parallel construct must be created 8,997,000 times. The computational overhead required for creating this parallel construct numerous times completely removed any benefit from parallelization.

The simd Directive

It was just demonstrated that creating a parallel construct in the innermost loop dramatically increased the computation time, so any effort to parallelize that loop must do so without creating a parallel construct. This is a situation where the simd directive can be of use. The term SIMD stands for “single instruction multiple data,” and it refers to the process of applying the same instruction to multiple data elements concurrently (Furht, 2008). When SIMD has been implemented, it is said that the instruction has been vectorized.

In OpenMP, the simd directive can be applied to a loop to inform the compiler that the loop may be vectorized (Tian & de Supinski, 2014). Additionally, the declare simd directive can be added to user-defined functions called within that loop so the compiler knows that they too can be vectorized. The key difference between the for directive and the simd directive is that the for directive parallelizes across cores while simd directive parallelizes within a core (Deslippe, He, Wasserman, & Yang, 2014).
Figure 4-6 shows pseudocode for computing $\omega$ that has been vectorized with the \texttt{simd} directive. The parallelized loops over the sources and copiers were maintained.

This version of the code with a vectorized function for computing $\omega$ had a computation time of 95 seconds, so it was just as fast as the version with the outer and inner loops over the examinees parallelized.

```c
#pragma omp simd reduction(+: obs_matches, \ exp_matches, var)
for (k=0; k<ITEMS; k++)
{
    if (copier_response == source_response)
    {
        obs_matches = obs_matches + 1;
    }
    compute probability of match (pmatch);
    exp_matches = exp_matches + pmatch;
    var = var + pmatch * (1 - pmatch);
}
omega = (obs_matches - exp_matches)/sqrt(var);
```

*Figure 4-6.* Vectorized pseudocode for computing $\omega$ for a given source-copier pair.

The clause \texttt{reduction(+: obs\_matches, exp\_matches, var)} is used to prevent a thread from modifying a variable at the same time that another thread is accessing it, which is known as a race condition (Chapman, Jost, & Van der Pas, 2007, p. 105). The syntax of the \texttt{reduction} clause shown in Figure 4-6 tells the compiler that the variables \texttt{obs\_matches, exp\_matches, and var} will be updated via addition by multiple threads. Also, it should be noted that the function for computing $\omega$ calls a function that computes the probability of a matching response. That function, in turn, calls another function that computes the natural logarithm of an exponential sum (i.e., the
logarithm of the denominator of the nominal response model). Although not shown in Figure 4-6, these two functions have the directive `#pragma omp declare simd` before their definitions so the compiler knows that they can be vectorized along with the other computations in the function for computing ω.

**Amdahl’s Law**

At this point, it is prudent to discuss how much speedup could potentially be gained from adding OpenMP directives to the sequential code. Amdahl (1967) demonstrated that speedup achieved through multiple processors has an upper bound determined by the proportion of the code that can be parallelized and the number of available processors. If $r_p$ is the fraction of the program than can be parallelized, $r_s$ is the fraction of the program that must remain sequential ($r_s + r_p = 1$), and $n$ is the number of available processors, then the maximum speedup, $S$, has the formula

$$S = \frac{1}{r_s + \frac{r_p}{n}}. \quad (4-6)$$

For situations where some code cannot be parallelized (i.e., $r_s \neq 0$), the potential speedup is bounded by $\frac{1}{r_s}$ as the number of processors tends to infinity. For situations where the entire program can be parallelized (i.e., $r_s = 0$ and $r_p = 1$), the potential speedup is $n$. Another factor that limits performance is accessing data in memory (Farber, 2017, p. 12). A program that relies on accessing data from memory is considered memory bound, so its speedup from parallelization is limited by the memory subsystem in addition to the number of processors.
The code in this case study is not completely parallelizable, as some time is needed to read in examinee and item data prior to computing $\omega$. When executing the case study code, the process of reading in the data was typically completed in less than one second. Therefore, it appears that the code is at least 99% parallelizable. The computer used for this programming exercise has eight cores, so code parallelized across the CPU can execute at most eight times faster than the sequential version. However, the code requires numerous memory accesses to get the item parameters and other values needed for the computation of $\omega$, so 8x speedup is likely unattainable. Indeed, the fastest version of the code obtained from OpenMP directives had a computation time of 95 seconds, which was a speedup of about 6.4. To decrease the computation time even more, additional processors are needed. This can be achieved by using the GPU (graphics processing unit), instead of the CPU, to execute the code.

**Directive-Based Parallelization on the GPU**

Support for offloading computation to an accelerator, such as a GPU, has been a feature of OpenMP since with the version 4.0 release. In OpenMP, offloading computation is enabled with the `target` directive. Before this directive can be used, the compiler must be built in such a way as to enable communication between the host device (the CPU) and the target device (the GPU or other desired accelerator). Beyer and Larkin (2016) give a thorough demonstration of using the Clang compiler and OpenMP to parallelize a program over the GPU.
A more straightforward way to access the GPU for its computational power is to use OpenACC, which was originally designed for offloading computation to accelerators, although it is also capable of parallelizing code over CPU cores in a manner similar to OpenMP. Compilers from The Portland Group® (PGI) are designed to enable offloading via OpenACC, so OpenACC and a PGI® compiler were used to offload the code to the GPU.

Because of memory constraints, the GPU could not accommodate the full group of 3,000 examinees in the case study problem. Therefore, a smaller group of 2,000 examinees was used to test the speedup offered by OpenACC. For this reason, and the fact that a different compiler was used, the results from parallelizing the code over the GPU using OpenACC will not be compared to the results from parallelizing the code over the CPU using OpenMP. Also, the intent of this paper is to demonstrate the speedups enabled by OpenMP and OpenACC, rather than to argue that one is superior to the other. On the PGI® compiler, the code for computing ω with 2,000 examinees had an execution time of approximately 114 seconds.

**Overview of OpenACC**

OpenACC is an API for C/C++ and Fortran that uses compiler directives to inform compilers how to parallelize the code and offload its execution to an accelerator device, such as a GPU (OpenACC, 2015, p. 3). Directives in OpenACC have a similar structure to those in OpenMP. For C/C++, the directives begin with `#pragma acc`, and for Fortran, the directives begin with `!$acc` (OpenACC, 2015, pp. 6-7). Also like OpenMP, OpenACC includes environment variables and runtime library routines that aid
in parallelization. However, to use the runtime library routines, a C/C++ program must include the header file `openacc.h`, and a Fortran program must include the module `openacc`.

In OpenMP, the parallelism is based on the concept of multi-threading. Teams of threads are created and they divide the computations among themselves. These threads can split into additional threads, as was seen in the examples with the nested parallelized loops. OpenACC also uses threads, but it has special terms to describe the level of parallelism (Farber, 2017, p.7). A worker is a group of threads, and is analogous to the concept of a warp in CUDA programming. A gang is a group of workers, and is analogous to the concept of a threadblock in CUDA programming. Last, a vector is used to synchronize workers for SIMD instructions.

The `kernels` directive

OpenACC has two directives for parallelizing code execution: the `parallel` directive, which is similar to the `parallel` directive in OpenMP, and will be discussed later, and the `kernels` directive, which is discussed here (Farber, 2017, p. 4). The `kernels` directive is a good starting point for programmers new to OpenACC because it shifts the task of identifying parallelism to the compiler (OpenACC, 2015, p. 15). Figure 4-7 shows pseudocode with the `kernels` directive.
The compiler analyzes the code enclosed by the curly braces and determines which loops can be safely parallelized and how to appropriately parallelize them. The compiler also determines which variables could potentially be affected by race conditions and performs the appropriate reduction on them. When using a PGI compiler, enabling the compiler option `-Minfo` produces output that summarizes the decisions the compiler made about parallelizing the code.

In addition to determining which loops can be parallelized, the compiler must also determine which data are needed by the program and move them onto the accelerator accordingly. One limitation of the `kernels` directive is that the compiler takes a conservative approach in moving data and sometimes moves data more times than is necessary (Larkin, 2017, p. 41). Unnecessary data movement between the CPU and the GPU can significantly slow down execution, so the code in Figure 4-7 contains two data movement clauses to prevent this. The item parameters and examinee data are moved to the GPU using the `copyin` clause, which means they are copied to the GPU, but are not copied back to the CPU after execution (OpenACC, 2015, p. 29). This is a sensible

```
#pragma acc kernels copyin(item_data, examinee_data) \ copyout(omega_values)
{
    for (i=0; i<examinee_count; i++)
    {
        for (j=0; j<examinee_count; j++)
        {
            if (i==j) continue;
            Compute ω with source i and copier j;
        }
    }
}
```

Figure 4-7. Pseudocode parallelized for the GPU using the `kernels` directive.
approach because these data are only used to compute $\omega$ and are not changed in the process. The array for storing the values of $\omega$ uses the `copyout` clause, which means memory is allocated to the array, but it is not initialized (OpenACC, 2015, p. 30). Upon completion of the kernel, the data are copied back to the CPU. Although it is not used in this example, a common clause for data movement is the `copy` clause, which copies data to the GPU at the start of the kernel and copies it back to the CPU upon completion (OpenACC, p. 29).

Before presenting the results of executing the version of the code shown in Figure 4-7, it is necessary to discuss how to handle functions called within a parallelized loop. When execution of code has been offloaded to an accelerator, the accelerator must have a copy of any functions that are called so they may be properly parallelized (OpenACC, 2015, p. 19). There are two ways of informing the compiler on how to parallelize function calls within parallelized loops. Version 2.0 of OpenACC introduced the `routine` directive, which can be placed before a function definition to tell the compiler that it can be called within a parallelized loop and what level of parallelism to use (e.g., gang or vector). The other method is to use the `-Minline` compiler option during compilation, which is exclusive to the PGI® compiler. This compiler option automatically inlines all functions, which is the equivalent of copying the function code directly into the code that is calling it. Automatic inlining was used for the code shown in Figure 4-7.

The code in Figure 4-7 had an execution time of six seconds, which was 19.0 times faster than the sequential version. This is a substantial speedup from adding one
OpenACC directive, which illustrates both the ease of use of OpenACC and the processing power of a GPU.

**The parallel Directive**

The other directive for parallelizing code execution is the **parallel** directive, and its usage is similar to that of the **parallel** directive found in OpenMP. Because parallelism is most frequently applied to loops, this directive is typically combined with a **loop** directive (OpenACC, 2015, p.16). The **loop** directive applies to both for loops in C and do loops in Fortran, as opposed to OpenMP, which had a separate directive for each type of loop.

Figure 4-8 shows a version of the code that uses the **parallel** directive instead of the **kernels** directive. Using the **parallel** directive instead of **kernels** required adding two lines of code, so its implementation was no more difficult than that of **kernels**. This version required the same data movement clauses as those used in the version shown in Figure 4-7. Also, compilation of the code necessitated the **-Minline** compiler option discussed previously to inline the function calls contained in the parallel construct. The code in Figure 4-8 had an execution time of six seconds, so there was no difference in performance between using **kernels** and using **parallel**.
Figure 4-8. Pseudocode parallelized for the GPU using the `parallel` directive.

The key difference between `kernels` and `parallel` is identification of parallelizable code. When using `kernels`, it is up to the compiler to decide whether a section of code can be parallelized (OpenACC, 2015, p. 17). As such, the compiler will only parallelize code that it is certain can be safely parallelized. For more complex programs, this can mean that some parallelism is missed. When using `parallel`, the programmer is asserting to the compiler that the code can be parallelized. The compiler will parallelize this code, regardless of whether it is correct to do so.

**Levels of Parallelism Clauses**

It was discussed previously that OpenACC has several levels of parallelism: gang, worker, and vector. These terms are also used as clauses in the `loop` directive to inform the compiler what level of parallelism to apply to the loop (OpenACC, 2015, 39). Additionally, the `seq` clause can be used to indicate that code should be executed sequentially. For nested loops, OpenACC specifies that the outermost loop must receive
the `gang` clause, and the innermost loop must receive the `vector` clause (OpenACC, 2015, 41). The `worker` clause may be used in any intermediate loop, and the `seq` clause can be used at any level in nested loops.

Using the PGI® compiler with the `-Minfo` compiler option produces output detailing how the compiler parallelized the code. This information includes what level of parallelism was applied to each loop. Figure 4-9 shows the output generated when compiling the code shown in Figure 4-8 with this option.

```
63, #pragma acc loop gang /* blockIdx.x */
66, #pragma acc loop vector(128) /* threadIdx.x */
156, #pragma acc loop seq
```

*Figure 4-9*. Sample output when compiling with the `-Minfo` compiler option.

This indicates that the loop on line 63 was assigned gang-level parallelism and the loop on line 66 was assigned vector-level parallelism with a vector length of 128, which is the maximum number of data elements that can be acted on simultaneously by a SIMT (single instruction, multiple threads) process on the GPU (OpenACC, 2015, p. 39). The loops on lines 63 and 66 correspond to the outer and inner loops over the examinees, respectively. The loop on line 156 is the loop over items used for computing $\omega$ for a given source-copier pair. Because no OpenACC directives were applied to this loop, it was executed sequentially. Based on the output shown in Figure 4-9, the compiler automatically assigned the proper level of parallelism to each of the two loops with OpenACC directives. Hence, adding them to the current version of the code would have
no effect on the computation time. However, they are not pointless, and can be used to improve the scheduling of computational resources.

The **collapse** Clause

Using OpenACC to accelerate a program requires minimizing data movement and maximizing the available parallelism. The **collapse** clause is a way to increase the amount of parallelism that can be exploited by the accelerator (OpenACC, 2015, p.42). This clause collapses the iteration spaces of several tightly-nested loops into one loop with a larger iteration space. For example, consider the case where a loop with $X$ iterations immediately leads into a loop with $Y$ iterations. Applying the **collapse** clause to these nested loops results in one loop with $XY$ iterations. The number of loops to collapse is given in parentheses after the clause. Speedup can be achieved because this larger iteration space offers more potential for parallelism. As such, the **collapse** clause is especially useful for combining the iteration space of multiple loops when one or more of the loops has a relatively small iteration space.

The requirement that the loops must be tightly-nested is strictly enforced. There can be no intervening code between the loop declarations. For the code in this case study, the loop over the items for computing $\omega$ was not close enough to the loops over the examinees, so the **collapse** clause could only be applied to the two loops over the examinees. Figure 4-10 shows this version of the code. This version of the code did not lead to a speedup beyond the 19.0x speedup obtained earlier, so there was already enough parallelism to put the GPU’s resources to their full use.
Discussion

The stated purpose of this paper was to demonstrate the utility of OpenMP and OpenACC for parallelizing computer programs that analyze test data. A case study in which an answer-copying analysis using the $\omega$ statistic was presented to show how a programmer could achieve impressive computational speedups by adding OpenMP or OpenACC compiler directives to existing code. The following is a summary of the primary benefits of using directive-based parallelization through OpenMP or OpenACC.

**Significant Computational Performance Increases**

The case study code computed $\omega$ for a population of 3,000 examinees, which amounted to computing $\omega$ 8,997,000 times. The original sequential code had a computation time of 611 seconds. Addition of OpenMP compiler directives to this code decreased that computation time to 95 seconds, so the augmented code was approximately 6.4 times faster than the original. The CPU had eight cores, so approximately 80% of the maximum potential speedup was achieved. Similarly, addition

```c
#pragma acc parallel loop collapse(2) copyout(omega_values) \ copyin(item_data, examinee_data)
{
    for (i=0; i<examinee_count; i++)
    {
        for (j=0; j<examinee_count; j++)
        {
            if (i==j) continue;
            Compute $\omega$ with source i and copier j;
        }
    }
}
```

*Figure 4-10. Pseudocode with the collapse clause.*
of OpenACC compiler directives decreased the computation time of a different version of the sequential code from 114 seconds to 6 seconds, so the augmented code was approximately 19.0 times faster than the original. Analysis of test data can involve some computationally-intensive tasks, and this work showed that OpenMP and OpenACC are capable of significantly decreasing the computation time required for these tasks.

**Ease of Implementation**

Another benefit of OpenMP and OpenACC is that they are relatively easy to implement. Their high-level nature, in which simple compiler directives enable parallelism, makes these APIs accessible to nearly anyone who understands either Fortran or C/C++. For the examples in which OpenMP was used to parallelize the code over the CPU, the greatest decrease in computation time came from adding four lines of code to the program. For the examples in which OpenACC was used to accelerate the code on the GPU, adding one directive was sufficient to decrease the computation time by a factor of 19.0. Additionally, it should be noted that the code did not have to be rewritten to accommodate these directives.

**Cost-Effective Performance Gains**

Finally, it should be mentioned that parallelizing code using OpenMP or OpenACC does not require a significant investment of time or money. As discussed in previously, the existing code did not have to be modified to accommodate the OpenMP or OpenACC directives. Eliminating the need to rewrite code is a strong incentive for using these directive-based approaches to parallelization. Also, there are open-source
compilers capable of implementing these APIs at no charge. The examples with OpenMP used GCC, which is free to anyone, and the examples with OpenACC used a free community edition of the PGI® compiler.
CHAPTER V

ADDITIONAL STUDIES

This chapter presents some results that are supplementary to those presented in chapters II and III, but are beyond the scope of the original papers.

**Type I Error and Power Analysis of the Outlier Detection Method**

The original version of the paper contained in chapter II focused exclusively on the outlier detection method. Therefore, a simulation study to determine the method’s type I error rate and power was included in the original paper. This section describes the design of the simulation study and presents its results.

**Simulation Study**

This subsection describes the steps of the simulation used to create test result data for validating the method described previously. Simulations were conducted for tests with 50, 75, and 100 items; flawed answer key rates of .05, .10, and .20; and baseline examinee counts of 100 and 300. Altogether, there were 18 different sets of parameters. For each combination of parameters, 30 repetitions of the simulation were performed.

*Construct the exam.* Simulated exams consisting of 50, 75, and 100 items were administered. For each item, a 2-parameter logistic model was built by randomly generating a discrimination parameter, $a$, and difficulty parameter, $b$. Discrimination parameters were drawn from the interval $[0.5, 2)$, and difficulty parameters were drawn from the standard normal distribution. The number of flaws in the braindump answer key was controlled by establishing a rate of flawed answers. In the simulations, flawed
answer key rates of .05, .10, and .20 were analyzed. For simplicity, the items with flawed answer keys were always the first items in the exam (e.g., if the flawed answer key had ten items with incorrect answers, then items 1-10 were chosen to be the items with the incorrect answers).

**Administer the exam to baseline examinees.** In the example with the real-life test security breach, tests taken before the theft of the items were used to establish a baseline for normal test taking. The simulations established these baselines by first administering the exam to a specific number of examinees who did not use the braindump. Baseline sizes of 100 and 300 were analyzed.

The random variable $\theta \sim N(0,1)$ was used to generate ability levels for each baseline examinee. Because each examinee needed two scores (one based on the true answer key and one based on the flawed answer key), the simulation created a response string for each examinee, which was then scored in two different ways. A response of $A$ was considered the correct response, $B$ was the most common distractor (or an incorrect answer in the flawed answer key), and $C$ was considered any other incorrect answer.

The simulation created an examinee’s response string by calculating the probability that the examinee would answer the item correctly. Using the 2-parameter logistic model, the probability of an examinee with ability $\theta_j$ correctly answering an item with discrimination $a_i$ and difficulty $b_i$ was

$$p(\text{correct}) = \left[1 + e^{-a_i(\theta_j - b_i)}\right]^{-1}. \quad (5-1)$$
After computing the probability, a random value, \( \gamma \), from the interval \([0, 1]\) was generated. Because the baseline examinees were not braindump users, the following rules were used for determining the response:

1. If \( p(\text{correct}) > \gamma \), then the response was A;
2. If rule 1 was false and \( 3.0\gamma < p(\text{correct}) + 2.0 \), then the response was B; and
3. If neither rule 1 nor rule 2 was true, then the response was C.

The last two rules were established so that B was chosen twice as often as C.

To score the test using the true answer key, the number of times A appeared in the response string was counted. To score the test using the braindump key, the number of times B appeared in the portion of the response string corresponding to items with incorrect answers in the flawed key was added to the number of times A appeared in the remainder of the response string.

**Perform the PCA and determine flagging thresholds.** A principal components analysis was then performed on the (keyed score, flawed score) ordered pairs. As seen in the case study presented in chapter II, the second principal component, which lies in the vertical direction, was the component of interest. In the case study, a normal distribution could be fit to the second principal components, so it was assumed that a normal distribution could be used for the simulated data.

The mean and standard deviation of the second principal components were calculated, and each flagging threshold was set to be a certain number of standard deviations above the mean. The number of standard deviations (z-score) was chosen so that a certain proportion of the standard normal distribution would be above the
threshold. For $\alpha = .01$, $z = 2.326$; for $\alpha = .005$, $z = 2.576$; for $\alpha = .001$, $z = 3.090$; for $\alpha = .0005$, $z = 3.291$; and for $\alpha = .0001$, $z = 3.719$. Given an $\alpha$-level, the corresponding threshold was

$$\text{Threshold} = \text{Mean} + z \times \text{SD}. \quad (5-2)$$

The simulations computed type I error rates and statistical power for all five thresholds.

**Simulate normal examinees and determine type I error rates.** After setting the thresholds, 20,000 normal examinees were simulated so that type I error rates could be computed. Ability levels for normal examinees were drawn from the random variable $\theta \sim N(0,1)$, as they were for the baseline examinees. Response strings for normal examinees were generated by computing the probability of the examinee answering the item correctly and comparing it to a random number, $\gamma$, from the interval $[0, 1]$. Rules for determining the response were the same as those for the baseline examinees.

The response strings were then scored using the two different keys, and the second principal components were computed. All of these examinees took the exam normally, so their second principal components should all be below the flagging thresholds. The proportion of examinees with second principal components greater than a flagging threshold was the type I error rate associated with that threshold.

**Simulate braindump users and determine statistical power.** For the final step, 20,000 braindump users were simulated so that the power of the method could be estimated. Ability levels for braindump users were obtained from the random variable $\theta \sim N(2.5,0.25)$. Response strings for braindump users were generated by computing the probability of the braindump user answering the item correctly and comparing it to a
random number, \( \gamma \), from the interval \([0, 1]\). Rules for determining the response depended on whether the item had a flawed answer key.

- For an item that had an incorrect answer in the flawed answer key, the following rules were used for determining the response:
  1. If \( p(\text{correct}) > \gamma \), then the response was \( B \);
  2. If rule 1 was false and \( 3.0 \gamma < p(\text{correct}) + 2.0 \), then the response was \( A \); and
  3. If neither rule 1 nor rule 2 was true, then the response was \( C \);

- For an item that did not have a flawed answer key in the braindump, the response was determined using the same set of rules as the baseline examinees.

The response strings were then scored using the two different keys, and the second principal components were computed. All of these examinees were braindump users, so their second principal components should all be above the flagging thresholds. The proportion of examinees with second principal components greater than a flagging threshold was the statistical power associated with that threshold.

**Type I Error Rates**

Tables 5-1 through 5-3 show the average type I error rates for the simulations with exams consisting of 50, 75, and 100 items, respectively.
Table 5-1

**Type I Error Rates for Simulations with a 50-Item Exam**

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>( \alpha ) Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.01</td>
</tr>
<tr>
<td>.05</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.00430</td>
</tr>
<tr>
<td>300</td>
<td>.00317</td>
</tr>
<tr>
<td>.10</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td><strong>.01016</strong></td>
</tr>
<tr>
<td>300</td>
<td>.00890</td>
</tr>
<tr>
<td>.20</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td><strong>.01254</strong></td>
</tr>
<tr>
<td>300</td>
<td><strong>.01209</strong></td>
</tr>
</tbody>
</table>

*Note.* Type I error rates greater than the nominal rates are in boldface.

Table 5-2

**Type I Error Rates for Simulations with a 75-Item Exam**

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>( \alpha ) Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.01</td>
</tr>
<tr>
<td>.05</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.00555</td>
</tr>
<tr>
<td>300</td>
<td>.00628</td>
</tr>
<tr>
<td>.10</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td><strong>.01136</strong></td>
</tr>
<tr>
<td>300</td>
<td><strong>.01015</strong></td>
</tr>
<tr>
<td>.20</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td><strong>.01477</strong></td>
</tr>
<tr>
<td>300</td>
<td><strong>.01387</strong></td>
</tr>
</tbody>
</table>

*Note.* Type I error rates greater than the nominal rates are in boldface.
Table 5-3

*Type I Error Rates for Simulations with a 100-Item Exam*

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>( \alpha ) Level</th>
<th>.01</th>
<th>.005</th>
<th>.001</th>
<th>.0005</th>
<th>.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td>100</td>
<td>.00930</td>
<td>.00425</td>
<td>.00065</td>
<td>.00029</td>
<td>.00003</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>.00924</td>
<td>.00426</td>
<td>.00061</td>
<td>.00022</td>
<td>.00003</td>
</tr>
<tr>
<td>.10</td>
<td>100</td>
<td>.01202</td>
<td>.00628</td>
<td>.00142</td>
<td>.00073</td>
<td>.00019</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>.01186</td>
<td>.00607</td>
<td>.00119</td>
<td>.00054</td>
<td>.00009</td>
</tr>
<tr>
<td>.20</td>
<td>100</td>
<td>.01375</td>
<td>.00757</td>
<td>.00202</td>
<td>.00108</td>
<td>.00030</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>.01319</td>
<td>.00701</td>
<td>.00174</td>
<td>.00096</td>
<td>.00021</td>
</tr>
</tbody>
</table>

*Note.* Type I error rates greater than the nominal rates are in boldface.

Several observations about the type I error rates can be made from the results shown in Tables 5-1 through 5-3. First, the size of the baseline did not have a strong effect on whether the nominal type I error rate was held. Although the method typically had lower type I error rates for simulations with a baseline of 100 examinees than for simulations with a baseline of 300 examinees, the rates were usually similar in magnitude.

Second, the flaw proportion had the greatest effect on whether the method was able to maintain the nominal type I error rate for each set of simulation parameters. When the flaw proportion was .05, the nominal type I error rate was held for all sets of simulation parameters. When the flaw proportion increased to .10, the nominal type I error rate was held for approximately half of the sets of simulation parameters. At the greatest flaw proportion, .20, the nominal type I error rate was not held for any set of simulation parameters.
The primary reason for this increase in type I errors as the flaw proportion increased was that greater flaw proportions allowed for greater variation between an examinee’s scores when the two different keys were used. At a flaw proportion of .05, there were three differences between the keys for a 50-item exam, four differences for a 75-item exam, and five differences for a 100-item exam. Conversely, at a flaw proportion of .20, there were 10 differences between the keys for a 50-item exam, 15 differences for a 75-item exam, and 20 differences for a 100-item exam. These larger numbers of differences between the keys allowed for more low-performing examinees, who did not use the braindump, to be flagged.

Finally, the test length had a small effect on whether the nominal type I error rate was held. This small effect was most pronounced for the simulations in which the flaw proportion was .10. For the simulations with 50-item exams and a flaw proportion of .10, the nominal type I error rate was not held for $\alpha = .01$ and a baseline of 100. For the simulations with 75-item exams and a flaw proportion of .10, the nominal type I error rate was not held for any $\alpha$-level with a baseline of 100, and it was not held for $\alpha = .01$ and a baseline of 300. For the simulations with 100-item exams and a flaw proportion of .10, the nominal type I error rate was held only for $\alpha = .0001$ and a baseline of 300.

Detection Rates

Tables 5-4 through 5-6 show the average detection rates for the simulations with exams consisting of 50, 75, and 100 items, respectively.
Table 5-4

Detection Rates for Simulations with a 50-Item Exam

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>( \alpha ) Level</th>
<th>.01</th>
<th>.005</th>
<th>.001</th>
<th>.0005</th>
<th>.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.684</td>
<td>.584</td>
<td>.424</td>
<td>.352</td>
<td>.245</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>.691</td>
<td>.577</td>
<td>.396</td>
<td>.317</td>
<td>.190</td>
<td></td>
</tr>
<tr>
<td>.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.999</td>
<td>.997</td>
<td>.991</td>
<td>.985</td>
<td>.967</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>.999</td>
<td>.998</td>
<td>.994</td>
<td>.990</td>
<td>.976</td>
<td></td>
</tr>
<tr>
<td>.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-5

Detection Rates for Simulations with a 75-Item Exam

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>( \alpha ) Level</th>
<th>.01</th>
<th>.005</th>
<th>.001</th>
<th>.0005</th>
<th>.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.551</td>
<td>.461</td>
<td>.263</td>
<td>.203</td>
<td>.108</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>.603</td>
<td>.484</td>
<td>.245</td>
<td>.193</td>
<td>.098</td>
<td></td>
</tr>
<tr>
<td>.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.000</td>
<td>.999</td>
<td>.997</td>
<td>.995</td>
<td>.986</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1.000</td>
<td>1.000</td>
<td>.999</td>
<td>.997</td>
<td>.992</td>
<td></td>
</tr>
<tr>
<td>.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>
Table 5-6

Detection Rates for Simulations with a 100-Item Exam

<table>
<thead>
<tr>
<th>Flaw Proportion and Baseline</th>
<th>.01</th>
<th>.005</th>
<th>.001</th>
<th>.0005</th>
<th>.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.624</td>
<td>.516</td>
<td>.276</td>
<td>.204</td>
<td>.094</td>
</tr>
<tr>
<td>300</td>
<td>.645</td>
<td>.549</td>
<td>.264</td>
<td>.196</td>
<td>.095</td>
</tr>
<tr>
<td>.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.999</td>
<td>.998</td>
<td>.992</td>
<td>.985</td>
<td>.957</td>
</tr>
<tr>
<td>300</td>
<td>1.000</td>
<td>.999</td>
<td>.995</td>
<td>.991</td>
<td>.972</td>
</tr>
<tr>
<td>.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>300</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

*Note.* Power values that rounded to 1.000 but were not truly 1.000 are in boldface.

Several observations about the detection rates can be made from the results shown in Tables 5-4 through 5-6. First, the baseline size did not have an effect on the detection rates. Detection rates for simulations with a baseline of 100 were very similar to the detection rates for simulations with a baseline of 300. Neither baseline size had detection rates that were consistently lower or higher than the detection rates for the other baseline size.

As with the type I error rates, the flaw proportion had the strongest effect on the detection rates. In Tables 5-4 through 5-6, the method had detection rates ranging between .09 and .25 at $\alpha = .0001$ for simulations with flaw proportions of .05. The detection rates were greater than .95 at $\alpha = .0001$ for all simulations with a flaw proportion of .10. When the flaw proportion was increased to .20, the detection rates exceeded 0.999 at $\alpha = .0001$ for all simulations.
As noted earlier, the method had low type I error rates for simulations with flaw proportions of .05 because the number of answer key flaws was small, which limited the variation between the two different scores. The detection power of the method was impaired in these simulations for the same reason. Detection rates were nearly perfect for simulations with a flaw proportion of .20. The method failed to detect two braindump users at the $\alpha = .0005$ level and two braindump users at the $\alpha = .0001$ level for the simulation with 100 items, a flaw proportion of .20, and a baseline of 300 examinees.

Test length did not have an effect on the detection rate. When the flaw proportion was .05, the method had higher detection rates for simulations with 50 items than for simulations with 75 or 100 items. However, this difference was not maintained when the flaw proportion increased to .10 or .20.

**Discussion**

The simulation study showed that the outlier detection method tends to be liberal in flagging examinees. Inspection of the values in Tables 5-1 through 5-3 shows that the type I error rate in the simulations exceeds the nominal rate by a factor of 3.67 or less. This section contains a discussion as to why that occurs. Because the simulations with a flaw proportion of .20 were the worst at maintaining the type I error rate, the discussion will focus on this flaw proportion. Figure 5-1 shows a distribution of the second principal components for the 20,000 non-braindump users in a simulation with 50 items, a flaw proportion of .20, and a baseline of 100 examinees, and Figure 5-2 zooms in on the upper tail of the distribution of Figure 5-1.
Figure 5-1. Simulated data and a fitted normal distribution.

Figure 5-2. Upper tails of the distributions in Figure 5-1.

It can be seen from the Figure 5-1 that the normal distribution used in the simulation seems to fit the simulated data. The upper tails of the distributions shown in Figure 5-2 have values of the second principal component that are typically at the flagging levels used in the simulations. Figure 5-2 shows that the upper tail of the
simulated data is heavier than the upper tail of the normal distribution in this region. Therefore, it comes as no surprise that the outlier detection method was liberal when flagging examinees. This means that a testing program that wishes to use this method should employ a conservative threshold, such as the $5\sigma$ threshold used in the case study.

In terms of statistical power, the simulation study demonstrated that the outlier detection method can detect examinees who used a flawed answer key. At a flaw proportion of .10, the detection rates exceeded 90%. Although the detection rates would not likely be this high if a $5\sigma$ threshold was used, these results are promising.

**Effect of Sample Size on Estimating an Unknown Flawed Answer Key**

In chapter III, it was observed that the methods of estimating an unknown flawed answer key had different results for real-life data and simulated data. This was especially true for the Common_Max method, which had the best performance for the real-life data, but had poor performance for the simulated data. One possible explanation is that the simulated data used a standard sample size of 1,000 examinees, and the real-life data had different sample sizes. This section investigates whether sample size was responsible for the different results.

**Simulation with 500 Examinees**

In chapter III, data sets 1, 3, and 4 had 249, 720, and 387 examinees, respectively. Rather than run several simulations with sample sizes similar to these values, it was decided to have one simulation with 500 examinees. Because the results of chapter III showed that test length did not have a strong effect on accuracy, only tests with 60 or 100
items were simulated. Data sets 1 and 3 had tests with 60 items, and data set 4 had a test with 100 items. No changes were made to the values of the flaw proportion and braindump user proportion used in the simulations, and 100 iterations of each set of simulation parameters were performed, as before. Table 5-7 presents a summary of the flawed accuracy, non-flawed accuracy, and overall accuracy for the four methods, and is similar to Table 3-2.

Table 5-7

*Average Accuracies of the Estimated Flawed Answer Keys for the Simulation with 500 Examinees*

<table>
<thead>
<tr>
<th>Method</th>
<th>Flawed</th>
<th>Non-Flawed</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common_Source</td>
<td>.916</td>
<td>.919</td>
<td>.918</td>
</tr>
<tr>
<td>Max_Omega</td>
<td>.824</td>
<td>.873</td>
<td>.870</td>
</tr>
<tr>
<td>Most_Flagged</td>
<td>.900</td>
<td>.956</td>
<td>.952</td>
</tr>
<tr>
<td>Common_Max</td>
<td>.812</td>
<td>.907</td>
<td>.896</td>
</tr>
</tbody>
</table>

Comparison of this table with Table 3-2 shows that Common_Source, Max_Omega, and Most_Flagged had lower accuracy values for the simulations with 500 examinees. Common_Max had a lower flawed accuracy for the simulations with 500 examinees, but the non-flawed accuracy and overall accuracy increased by .010 and 0.006, respectively.

**Simulation with 3,000 Examinees**

In chapter III, data set 2 had 2,899 examinees and a test with 60 items. Thus, a simulation with 3,000 examinees and a test with 60 items was performed. Because computing $\omega$ for every possible source-copier pair in a population of 3,000 examinees
required considerably more time than it does in a population of 1,000 examinees, the number of iterations in the simulation was reduced to 20. No changes were made to the flaw proportions or braindump user proportions.

In contrast to the simulations with 500 examinees, there was a notable change in performance when 3,000 examinees were used. Common_Source had an increase in flawed accuracy and a slight decrease in non-flawed accuracy. Max_Omega did not have a change in performance, and it continued to be the least effective of the methods. However, Most_Flagged and Common_Max had improvements in both flawed accuracy and non-flawed accuracy. These improvements are described further.

**Most_Flagged Results.** Table 5-8 shows the average flawed accuracy and non-flawed accuracy for the Most_Flagged method in the simulation with 3,000 examinees and a test with 60 items. Also included is a count of the number of times the estimated flawed key exactly matched the simulated flawed key.

Table 5-8

*Results for the Most_Flagged Method in the Simulation with 3,000 Examinees*

<table>
<thead>
<tr>
<th>Flaws</th>
<th>Users</th>
<th>Flawed</th>
<th>Non-Flawed</th>
<th>Exact Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05</td>
<td>.05</td>
<td>1.000</td>
<td>.956</td>
<td>0</td>
</tr>
<tr>
<td>.05</td>
<td>.10</td>
<td>.900</td>
<td>.964</td>
<td>0</td>
</tr>
<tr>
<td>.05</td>
<td>.20</td>
<td>.900</td>
<td>.968</td>
<td>0</td>
</tr>
<tr>
<td>.10</td>
<td>.05</td>
<td>1.000</td>
<td>1.000</td>
<td>20</td>
</tr>
<tr>
<td>.10</td>
<td>.10</td>
<td>.992</td>
<td>.974</td>
<td>0</td>
</tr>
<tr>
<td>.10</td>
<td>.20</td>
<td>.875</td>
<td>.973</td>
<td>0</td>
</tr>
<tr>
<td>.15</td>
<td>.05</td>
<td>1.000</td>
<td>1.000</td>
<td>20</td>
</tr>
<tr>
<td>.15</td>
<td>.10</td>
<td>1.000</td>
<td>.954</td>
<td>1</td>
</tr>
<tr>
<td>.15</td>
<td>.20</td>
<td>.939</td>
<td>.974</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 5-8 shows that the Most_Flagged method perfectly estimated the unknown flawed key for all 20 iterations when the proportion of braindump users was .05 and the proportion of flaws in the flawed key was .10 or .15. When this method was applied to the real data sets in chapter III, it had the best performance on the first two, which seemed to have fewer braindump users than the latter two. There were two other sets of simulation parameters for which the method had a perfect flawed accuracy, so the increased sample size led to better performance in the simulation for this method.

**Common_Max Results.** Table 5-9 shows the average flawed accuracy and non-flawed accuracy for the Most_Flagged method in the simulation with 3,000 examinees and a test with 60 items. The columns in Table 5-9 have the same meaning as those in Table 5-8.

Table 5-9

<table>
<thead>
<tr>
<th>Flaws Users</th>
<th>Flawed</th>
<th>Non-Flawed</th>
<th>Exact Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>.05 .05</td>
<td>.883</td>
<td>.866</td>
<td>0</td>
</tr>
<tr>
<td>.05 .10</td>
<td>.900</td>
<td>.894</td>
<td>8</td>
</tr>
<tr>
<td>.05 .20</td>
<td>1.000</td>
<td>1.000</td>
<td>20</td>
</tr>
<tr>
<td>.10 .05</td>
<td>.917</td>
<td>.862</td>
<td>0</td>
</tr>
<tr>
<td>.10 .10</td>
<td>.900</td>
<td>.915</td>
<td>8</td>
</tr>
<tr>
<td>.10 .20</td>
<td>1.000</td>
<td>1.000</td>
<td>20</td>
</tr>
<tr>
<td>.15 .05</td>
<td>.889</td>
<td>.889</td>
<td>0</td>
</tr>
<tr>
<td>.15 .10</td>
<td>.839</td>
<td>.878</td>
<td>3</td>
</tr>
<tr>
<td>.15 .20</td>
<td>1.000</td>
<td>1.000</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5-9 shows that the Common_Max method perfectly estimated the unknown flawed key for all 20 iterations when the proportion of braindump users was .20. In
Chapter III, this was the only method that performed well on data sets 3 and 4, which seemed to have the most braindump users. Overall, the performance of this method for the simulation with 3,000 examinees and a test with 60 items was more closely aligned with its performance on the real-life data sets in chapter III.

Discussion. Increasing the simulation sample size to 3,000 led to increased performance, particularly for the Common_Max method. In this simulation, Common_Max had performance that was more closely aligned with its performance on the real-life data sets. The reason for this may be that the number of exact matches with the flawed answer key increased when 3,000 examinees were used.

In chapter III, it was discussed that the Common_Max method excelled when there were clusters of examinees with identical response patterns. Examinees whose response patterns exactly match the flawed answer key would obviously be identical. Further support for this hypothesis is that during the original simulations, in which Common_Max had poor performance, the average number of exact matches per set of simulation parameters was 3.9. For the simulation with 3,000 examinees, the average number of exact matches per set of simulation parameters increased to 19.9.
CHAPTER VI

SUMMARY AND CONCLUSIONS

As stated in the introduction, determining which examinees had preknowledge of test content can be a difficult task. The three papers presented in this dissertation describe a partial solution to this problem in that they address the feasibility of determining which examinees used a braindump when the braindump has a flawed answer key.

The first paper used a case study to demonstrate that it is possible to determine which examinees used a flawed answer key. Knowing the date of the theft and the scores from using both the real key and the flawed key were all that was required for the analysis. The three techniques used for identifying likely braindump users in the case study gave very similar answers, which supports the conclusion that it is possible to identify examinees who used a flawed answer key based on their scores using the real and flawed keys. Also, the simulation study presented in chapter V for one of techniques showed that the method does have power to detect braindump users. However, because it did not maintain the nominal type I error rate, a conservative threshold should be used if it is put into practice.

There were two limitations to the methods used in the case study. First, the date of the theft had to be known so a group of known non-braindump users could be used as a baseline. Future research could examine ways of looking at item responses to determine an approximate date of the theft. The second limitation is that the flawed answer key had to be known so the tests could be scored both ways.
This second limitation was addressed by the second paper, which explored using an answer-copying statistic to estimate an unknown flawed answer key. Four methods of using the results of the answer-copying analysis to estimate the unknown flawed answer key were developed. Applying these methods to both real and simulated data showed that they had potential to accurately estimate the hypothesized and simulated flawed answer keys. Two methods in particular, known as Most_Flagged and Common_Max, showed the greatest potential for being put into practice. However, the purpose of the paper was to demonstrate that it is possible to estimate an unknown flawed answer key using an answer-copying statistic, and not to propose the best way of doing so.

It was observed that some methods had different performance on the real data versus the simulated data. One possible explanation for this difference in performance is that there was a disconnect between how braindump usage appears in real life and how it was simulated. Another theory is that the sample sizes in the simulations needed to match those of the real-life data examples. This theory was tested using sample sizes of 500 and 3,000. Although worse performance was observed for the methods with a sample size of 500, increasing the sample size to 3,000 significantly improved the performance for two of the methods and produced results more in line with those from the real-life data sets. An increased number of response patterns that exactly matched the flawed answer key may be the reason that the methods typically had better performance in the simulation with 3,000 examinees.

There is potential for more research on estimating an unknown flawed answer key. Answer-copying statistics other than $\omega$ could be studied. Also, there may be better
ways of using the results of the answer-copying analysis to estimate the flawed key than
the methods that were described and studied in chapter III. Finally, it would be
interesting to see how using an estimate of the flawed answer key would affect the results
of using the techniques from the first paper to determine which examinees used the
braindump.

One limitation of using an answer-copying statistic to estimate an unknown
flawed answer key is that computing the statistic for every source-copier pair is
computationally intensive, and thus is not feasible for large populations of test takers.
This is addressed by the third paper, which used OpenMP and OpenACC to parallelize
the answer-copying analysis over the computer’s hardware. By adding a few lines of
code to an existing computer program, it was possible to accelerate the answer-copying
analysis by a factor of 6.4 using OpenMP and 19.0 using OpenACC. This can make
estimating an unknown flawed answer key much more feasible.
REFERENCES


CURRICULUM VITAE

Marcus Scott
(December 2017)

EDUCATION:

BS in Mathematics, Brigham Young University, Provo, Utah. (4/07) GPA: 3.59.
   Emphasis in applied mathematics.

MS in Industrial Mathematics, Utah State University, Logan, Utah. (9/10) GPA: 3.84.
   Awarded research assistantship funded by USTAR.

Ph.D. in Mathematical Sciences, Utah State University, Logan, Utah (expected 5/18).

EXPERIENCE

Data Forensics Scientist, Caveon Test Security (1/13 – present). Analyzed data and
   wrote reports for clients. Also participated in standard setting work and research
   projects involving item cloning and flawed answer key analysis.

Graduate Teaching Assistant, Utah State University (8/07 – 12/12). Taught one
   undergraduate mathematics course each semester.

CONFERENCE PRESENTATIONS

2017 Conference on Test Security, Madison, Wisconsin. Using the ω Statistic to
   Estimate an Unknown Flawed Answer Key. Poster session.

2015 Conference on Test Security, Lawrence, Kansas. Analysis of Flawed Answer Keys
   to Detect Braindump Usage. Paper session.

2014 Conference on Test Security, Iowa City, Iowa. Using Answer Key Imputation for
   Making Bayesian Inference About Cheating on Tests. Paper session.