Automated parallel test form assembly based on discrete algorithms

Kazuma Fuchimoto

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Engineering – Dr.Eng
Graduate School of Informatics and Engineering
The University of Electro-Communications

March 2025

Supervisory Committee

- Prof. Yoshio Okamoto
- Prof. Yusaku Yamamoto
- Assoc. Prof. Yasuhiko Takenaga
- Assoc. Prof. Masaki Uto
- Prof. Maomi Ueno

論文の和文概要

論文題目	Automated parallel test form assembly based on discrete algorithms
氏　　名	渕本　壱真

近年，Computer Based Testing (CBT)の普及に伴い，自動平行テスト構成が注目されている．平行テストでは，異なる問題項目で構成されるにもかかわらず，受検者得点を同一精度で測定できる．そのために，平行テストは，受検者得点の予測誤差が等質となるように，問題項目のデータベースから組合せ最適化を用いて可能な限り多く自動構成する必要がある．これにより，複数回受検や任意の時間・場所での実施が可能となる．最先端手法では，空間計算量が大きく，約10万の平行テスト構成が限界であった．しかし，実際の試験（例えば，American College Testing）では100万人以上が受験する場合があり，同一テストが重複して実施されてきた．この場合，全ての受検者が異なるテストを受けるためには100万以上の平行テストを構成しなければいけない．平行テストの構成数を大幅に改善するために，本研究では最大クリークと整数計画法を用いた二段階並列探索手法を提案する．第一段階では，時間計算量は小さいが空間計算量の大きい最大クリーク手法により，メモリの限界まで平行テスト構成する．第二段階では，時間計算量は大きいが空間計算量の小さい整数計画法を用いた並列探索手法に切り替える．これにより，提案手法は1週間で約30万の平行テスト構成が可能となった．しかし，整数計画法の時間計算量が大きく，平行テスト数の改善には限界がある．そこで，本研究では，場合分け二分木の圧縮表現であるZero-suppressed Binary Decision Diagram (ZDD)を用いた手法を提案する．この手法では，各節点を問題項目，各節点をその問題項目をテストに含むか否かとした場合分け二分木をZDDにより圧縮する．具体的には，この場合分け二分木を幅優先で展開し，その節点までのテストの問題項目数や受検者得点の予測誤差が等価な節点を共有する．しかし，予測誤差が等価な節点は限りなく少ないため，従来のZDDでは節点が共有できずメモリオーバーを引き起こす．この問題を解決するために，本研究では，①幅優先探索の過程で同じ深さのある二節点における予測誤差の差が閾値以下の場合に節点を共有し，共有節点の予測誤差を二節点の平均値を取りZDDを構築する．ただし，閾値はメモリが許す限り経路（テスト）数が最大となるように決定する．②構築されたZDDから予測誤差の制約を満たす経路を探索して厳密な予測誤差を（共有節点の近似を用いず）再計算し，制約を満たす経路を列挙する．その結果，提案手法は，978問題項目を持つ実際のデータベースから，最大約150万の平行テスト構成を1日で達成できた．

Abstract

In recent years, along with the spread of Computer Based Testing (CBT), automated parallel test form assembly has emerged. Parallel test forms allow for the measurement accuracy of examinees' test scores on the same scale, even when the test form consists of different question items. An exceedingly important task in automated parallel test form assembly is to assemble as many parallel test forms as possible to administer CBT multiple times or at any time. However, the state-of-the-art maximum clique algorithm is limited by high space complexity. It is therefore capable of assembling a maximum of only 100,000 parallel test forms, although some actual examinations require over 1,000,000 parallel test forms annually.

To increase the number of parallel test forms, this study proposes a two-stage method using the maximum clique algorithm and integer programming. In the first stage, the proposed method assembles parallel test forms by extracting a maximum clique that is found within the computation time. However, because of the high space complexity of the maximum clique algorithm, the first stage is constrained in increasing the number of parallel test forms. Consequently, in the second stage, the proposed method switches to integer programming, which has high time complexity but low space complexity. This approach can assemble approximately 200,000 parallel test forms in one week under specific test constraints from an actual item pool with 978 items, surpassing conventional methods.

However, the two-stage method remains incapable of assembling a sufficient number of parallel test forms for large-scale CBT because of the high time complexity of integer programming. Therefore, this study proposes a method using Zero-suppressed Binary Decision Diagram (ZDD), a compressed representation of a binary decision tree. In this method, each vertex represents a question item. Each vertex has two outgoing edges, indicating whether the corresponding question item is included in parallel test form, or not. Specifically, the ZDD

method is expanded via a breadth-first search. Vertices with the identical number of question items in parallel test form and the identical measurement accuracies of examinees' test scores are shared. However, extremely few vertices have identical measurement accuracy to examinees' test scores, leading to insufficient sharing vertices. Consequently, the conventional breadth-first search causes computer memory overflow because vertices cannot be shared.

To resolve this issue, this study proposes a two-stage algorithm. (1) During the breadth-first search, vertices are shared when the difference in the measurement accuracy of examinees' test scores between two vertices at the same depth is less than a threshold. The shared vertex's measurement accuracy of examinees' test scores is averaged from the two vertices. The threshold is determined to increase the number of paths (parallel test forms) to as many as possible up to a computer memory limit. (2) Paths that satisfy the measurement accuracy constraint are searched and enumerated from the constructed ZDD. The exact measurement accuracy of examinee's test scores for each of the enumerated paths is recalculated to enumerate paths that satisfy the measurement accuracy constraint exactly. As a result, the ZDD method assembles parallel test forms that exactly satisfy all test constraints.

Empirical experiments have demonstrated that the ZDD based method can assemble up to approximately 1,500,000 parallel test forms in one day under specific test constraints from an actual item pool with 978 items, thereby exceeding the approximately 100,000 parallel test forms assembled using earlier reported methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer based testing, web-based systems, and computerized adaptive testing, collectively known as e-testing, enable examinees to participate as subjects of educational assessments at any time and from any location (e.g., [1, 2, 3]). In light of these beneficial features, many educational assessments have transitioned from traditional paper-based formats to e-testing. For example, the Scholastic Aptitude Test (SAT) [4] in the United States will adopt an e-testing format in 2024.

The e-testing guidelines adhere to international standards (ISO/IEC 23988:2007) [5], which mandate the use of parallel test forms [6]. According to Samejima [6], parallel test forms are expected to guarantee equivalent measurement accuracy for estimating examinees' test scores based on Item Response Theory (IRT) while using different sets of question items (e.g., [7]). For purposes of this paper, "question item" is simply designated as "item" hereinafter. As a result, parallel test forms guarantee that the test score remain consistent, even when examinees with equal ability take different versions of the test.

Figure 1.1 depicts the measurement errors of 9,600 examinees' test scores obtained when using parallel test forms in an actual examination. In the figure, the horizontal axis shows the test scores of examinees who took different parallel test forms. The vertical axis represents the measurement errors of those test scores. In e-testing, it

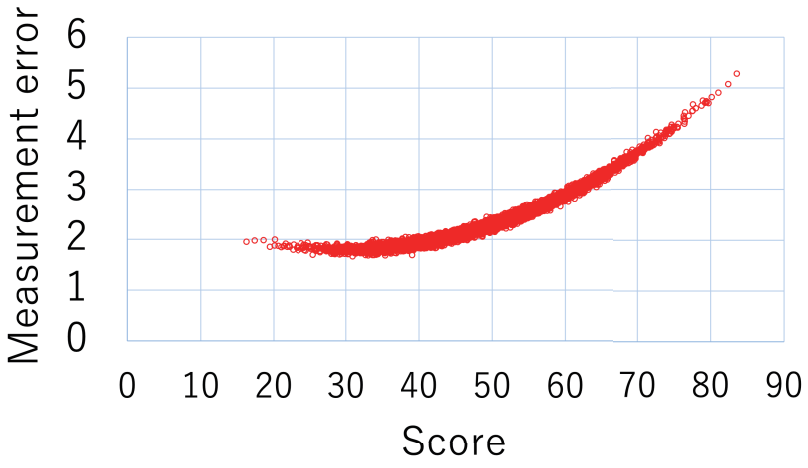


Figure 1.1: Measurement error of examinees' test scores obtained using parallel test forms for an actual examination.

is desirable to decrease the maximum value of the measurement error of the test score at the passing line. In fact, as shown in Figure 1.1, the measurement error becomes a small value of less than 2.0 at around the test score 40 which is the passing line for this examination. Additionally, when the vertical width of each plot for each test score becomes small, the parallel test forms have equivalent measurement accuracy. Therefore, parallel test forms are assembled to decrease the variance of measurement errors of examinees who have the same test score. Consequently, parallel test forms enable equivalent assessments for tests administered at different times and locations.

The most important task for e-testing is to increase the number of parallel test forms to as many as possible. The number of parallel test forms is expected to outnumber the examinees. For example, more than 200,000 examinees take the National Examination of Information Processing Engineers, the so-called "IT Passport", annually in Japan. Also, more than 1,300,000 examinees take the American College Testing (ACT) annually in the United States. Consequently, test

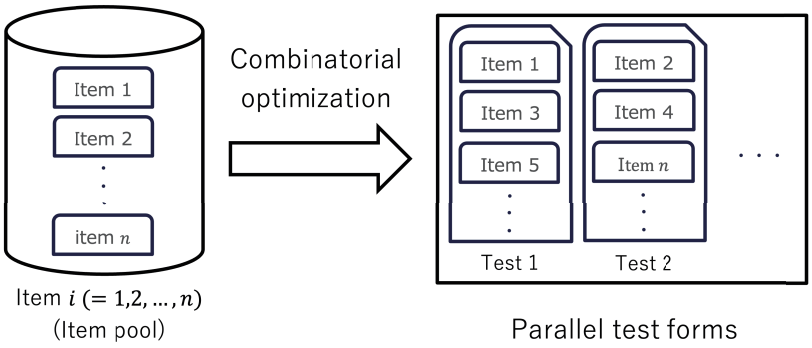


Figure 1.2: Outline of the automated parallel test form assembly.

organizations have tried to adopt e-testing by assigning different parallel test forms to each group of examinees who take the examination at different times and locations.

Many automated parallel test form assembly methods (e.g., [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]) have been proposed using IRT. These methods have formulated automated parallel test form assembly as a combinatorial optimization problem from an item pool, which is a database in which the items are stored, as shown in Figure 1.2. Specifically, from the item pool, the automated parallel test form assembly selects combinations of items that satisfy specific test constraints, such as constraint of the measurement accuracy of examinees' test scores, constraint of test length (number of items in each parallel test form), and constraint of the maximum number of overlapping items (the so-called overlapping item constraint), where overlapping items mean common items between any pair of parallel test forms. Here, when overlapping items are not allowed, the automated parallel test form assembly cannot increase the number of parallel test forms because each item is used only once. Allowing some overlap items enables repeated use of items, thereby increasing the total number of parallel test forms. However, when the maximum number of overlap items is too large, the parallel test forms might become nearly identical items. Accordingly, the

maximum number of overlapping items is usually set as 20%–30% of the test length (e.g., [29, 30, 32, 33]).

Among these methods, the Big Shadow Test (BST) method [34] using a Mixed-Integer Programming (MIP) problem is widely recognized. The MIP problem has binary variables indicating whether each item is included in the assembled parallel test form, or not, and a single real variable in the objective function. This real variable simultaneously minimizes two differences: the difference between the examinees' test scores of the currently assembled parallel test form and the target examinees' test scores, and the difference between the examinees' test scores of the remaining items in an item pool and a proportional target examinees' test scores based on the number of remaining items. This proportional target examinees' test scores are determined as the ratio of remaining items to the test length. Then, BST sequentially assembles parallel test forms to minimize this difference by solving the MIP problem. Accordingly, BST mitigates the rapid decrease of items with high measurement accuracy of examinees' test scores from the item pool. Although BST can readily apply test constraints to the automated parallel test form assembly using MIP, it has two important limitations. First, the measurement accuracy of examinees' test scores decreases as the number of parallel test forms increases. Second, BST does not ensure maximization of the number of parallel test forms.

To minimize differences of the measurement accuracies among parallel test forms, several earlier studies have addressed the problem by formulating it as a large-scale integer programming (IP) problem (e.g., [35, 36, 22, 37, 38]). The IP problem has binary variables denoting whether each item is included in the assembled parallel test form, or not. These approaches directly minimize the differences in measurement accuracy of examinees' test scores among parallel test forms. Moreover, these methods are designed to assemble parallel test

forms that maintain the same measurement accuracy of examinees' test scores, even as the number of forms increases.

To mitigate the high computational costs associated with automated parallel test form assembly, Sun et al. [25] proposed a heuristic approach using a Genetic Algorithm (GA). The GA assembles multiple parallel test forms simultaneously, aiming to minimize differences in the measurement accuracy of examinees' test scores among parallel test forms while fitting with determined target values set by the test organization. Based on this approach, Songmuang and Ueno [28] enhance performance by application of a Bees Algorithm (BA) to the automated parallel test form assembly. The GA and the BA methods can realize the equivalent measurement accuracy of examinees' test scores among parallel test forms. Nevertheless, these methods do not guarantee maximizing the number of parallel test forms.

To guarantee the maximum number of parallel test forms, Belov and Armstrong [39] proposed automated parallel test form assembly using a Maximum Set-Packing (MSP) algorithm. The MSP method partitions the item pool into the maximum number of parallel test forms. Each parallel test form satisfies the given constraints. The MSP method guarantees the maximum number of parallel test forms. Nevertheless, the MSP method has a daunting limitation: it cannot include overlapping items, meaning that each item can only appear in one parallel test form. This crucially important limitation is that it severely decreases the number of assembled parallel test forms because it prevents the reuse of items among these form.

To address this limitation, Ishii et al. [29] proposed a method to allow overlapping items among parallel test forms using a Maximum Clique Algorithm (MCA), which extracts a graph structure in which any two vertices are connected with an edge. The MCA method is recognized for assembling the largest number of parallel test forms while maintaining the highest measurement accuracy of examinees'

test scores. The MCA method generates a graph in which the vertices represent parallel test forms that satisfy the test constraints without overlapping constraint. The edges represent the sufficiency of overlapping item constraint between two vertices (parallel test forms). Then, the MCA method extracts the maximum clique from this graph to assemble parallel test forms. This approach guarantees the maximum possible number of assembled parallel test forms. However, the MCA method has high time complexity and high space complexity [40]. Here, the time and space complexities are defined respectively as the number of basic operations and the total amount of memory required by an algorithm to complete its execution. Specifically, this paper employs the $\mathcal{O}$-notation as time and space complexities. Complexity of a non-negative function $f(m)$ with the input size $m \in \mathbb{N}$ is defined as

$$\mathcal{O}(g(m)) = \{f(m) : \exists c \in \mathbb{R}^+, \exists m_0 \in \mathbb{N},\ \forall m \geq m_0, \\ 0 \leq f(m) \leq c \cdot g(m)\}, \qquad (1.1)$$

where $g(m)$ represents a non-negative function. According to the $\mathcal{O}$-notation, the MCA method has a high time complexity of $\mathcal{O}(2^{|V|})$ and a high space complexity of $\mathcal{O}(|V|^2)$, where V represents a set of vertices in the graph. Furthermore, the number of vertices $|V|$ can be at most the total number of combinations of selecting L (test length) items from n items in item pool. Namely, $|V| = \binom{n}{L}$. As a result, extracting the maximum clique $C \subseteq V$ becomes computationally infeasible for large-scale item pools because the graph size becomes too large to be stored in computer memory.

To mitigate the time and space complexities difficulty, Ishii et al. proposed an approximation based on a random search method, which is known as the Random Maximum Clique Algorithm (RndMCA) [30]. After RndMCA constructs graphs by sequentially and randomly assembling as many vertices (parallel test forms) as possible, it then extracts the maximum clique from these graphs. Specifically, RndMCA samples random subgraphs from the overall graph and ex-

tracts the maximum clique within each subgraph. The maximum clique extraction from a randomly sampled subgraph in RndMCA has a high time complexity of $\mathcal{O}(\binom{n}{L} + 2^{SS})$ and a high space complexity of $\mathcal{O}(\binom{n}{L} + SS^2)$, where $SS(|C| \leq SS \ll |V|)$ represents the sampling size of each random sampling vertices from the overall graph. Although RndMCA has the high time complexity, it is known [30] to assemble more parallel test forms than conventional methods do, even when the computation is interrupted within a short time limit. However, because of the high space complexity and its attendant computer memory limitations, RndMCA cannot extract the maximum clique with more than 100,000 vertices. As a result, the number of parallel test forms is restricted to a mere hundred thousand. This number of parallel test forms is insufficient for practical use in examinations with more than 1,000,000 examinees annually, such as the ACT. Therefore, this study proposes three discrete algorithms aimed at assembling more than 1,000,000 parallel test forms.

First, to mitigate the high space complexity of RndMCA, this study proposes a new method: Random Integer Programming for Maximum Clique Algorithm (RIPMCA). RIPMCA dynamically searches for a vertex in a maximum clique one by one using IP, which has binary variables indicating whether each item is included in a parallel test form, or not. Specifically, RIPMCA searches for a vertex connected to all vertices in the current clique, which is a set of vertices already found by IP. Then, RIPMCA sequentially expands the maximum clique by repeating the search. Also, RIPMCA has a lower space complexity of $\mathcal{O}(|C|)$ compared to the space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$ of RndMCA, because it does not store the graph structure of the maximum clique on the computer memory. The benefit of reducing space complexity increases the number of parallel test forms within limited memory. Nevertheless, the improvement of RIPMCA is limited because of the high time complexity $\mathcal{O}(|C| \cdot m_c \cdot 2^n)$. Here, m_c represents the number of constraints in IP formulation. In this formulation,

m_c increases as $|C|$ becomes larger.

Second, to relax the computational costs for RIPMCA, this study proposes a new two-stage parallel method: a Hybrid Maximum Clique Algorithm with Parallel Integer Programming (HMCAPIP). In the first stage, HMCAPIP assembles as many parallel test forms as possible up to a computer memory limit using RndMCA within a computation time limit. However, because of the high space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$, RndMCA is constrained in increasing the number of parallel test forms. Consequently, in the second stage, HMCAPIP searches for a vertex that is connected to all vertices of the current clique from the remaining vertices using the IP with a low space complexity of $\mathcal{O}(|C|)$ but with a high time of complexity $\mathcal{O}(|C|\cdot m_c \cdot 2^n)$. To address this time complexity, we parallelize the second stage. Specifically, the key idea is efficient parallelization of the search for vertices connected to all vertices of the current clique using the IP up to a determined number. Afterward, the second stage identifies the maximum clique from the searched vertices, which is then combined with the current clique. As a result, HMCAPIP can reduce the computation time necessary for assembling parallel test forms. Findings from numerical experiments demonstrate that HMCAPIP can assemble 1.5–2.7 times more parallel test forms than earlier methods can. Particularly under specific test constraints from an actual item pool with 978 items, HMCAPIP can assemble 200,000 parallel test forms.

Despite its advantages, HMCAPIP still has high time complexity associated with the IP. For instance, HMCAPIP takes a week or more to assemble 200,000 parallel test forms. Additionally, the efficiency of HMCAPIP's parallel search is highly dependent on the performances of the computer's multi-core processors. Therefore, HMCAPIP remains unable to assemble a sufficient number of parallel test forms for large-scale e-testing because of the high time complexity of IP.

Third, to increase the number of parallel test forms, we propose

a new method of Automated Parallel Test Form Assembly (ATA) using a Zero-suppressed Binary Decision Diagram (ZDD) [41], which is designated as ATA-ZDD. The ZDD provides an efficient graphical structure for combination. It is obtained by application of two reduction rules [41] to a Binary Decision Tree (BDT). The ZDD provides the important benefits of reducing both the computation time and memory usage. Because of these benefits, the ZDD has been applied in various fields, including Stackelberg models in combinatorial congestion games [42], architectural floor planning [43], grid power loss minimization [44], and region partitioning for disaster evacuation [45].

In ATA-ZDD, each vertex in the BDT represents an item from the item pool. Each vertex has two outgoing edges, indicating whether the corresponding item is included in a parallel test form, or not. The BDT can enumerate all parallel test forms that satisfy the test constraints. However, the BDT has high space complexity of $\mathcal{O}(2^n)$. To mitigate this space complexity, the proposed method uses a breadth-first search [46] to compress the BDT into the ZDD. This breadth-first search is recognized as an efficient approach for reducing both computer memory usage and computation time. Specifically, in ATA-ZDD, vertices with identical test lengths and identical measurement accuracies of examinees' test scores are shared. However, identical the measurement accuracy of examinees' test scores between two vertices is rare. Consequently, the conventional breadth-first search causes computer memory overflow because nodes cannot be shared.

To resolve this issue, this study proposes a two-stage algorithm. In the first stage, vertices are shared when the difference in the measurement accuracy of examinees' test scores between two vertices at the same depth is less than a threshold value. Then, the shared vertex's measurement accuracy of examinees' test scores is averaged from the two vertices. The threshold value is determined to increase the num-

ber of paths (parallel test forms) to as large as possible within a computer memory limit. However, the first stage does not guarantee that the enumerated paths satisfy the measurement accuracy constraints. To address this difficulty, in the second stage, paths that satisfy the measurement accuracy constraints are sought and enumerated using random sampling [41] from the constructed ZDD. The exact measurement accuracy of examinee's test score for each path without using the value of the approximated shared vertex is recalculated to enumerate paths that satisfy the measurement accuracy constraints exactly. Therefore, ATA-ZDD enumerates parallel test forms that satisfy all test constraints exactly in the second stage.

Numerical experiments show that ATA-ZDD can assemble more parallel test forms than the other automated test assembly from simulated and actual item pools. It is noteworthy that ATA-ZDD can assemble 1,500,000 parallel test forms in 24 hr under specific test constraints from an actual item pool with 978 items, whereas HMCAPIP is able to assemble only 100,000 parallel test forms.

Chapter 2

Related Work

2.1 Item Response Theory

In many automated parallel test form assemblies, an examinee's test score is estimated using Item Response Theory (IRT) [7, 47]. For IRT, u_{ij} represents the response of examinee $j(= 1, 2, \ldots, J)$ to item $i(= 1, 2, \ldots, n)$ as

$$u_{ij} = \begin{cases} 1, & \text{examinee } j \text{ answers item i correctly,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

The widely adopted IRT model, the Three-Parameter Logistic Model (3PLM), calculates the probability that an examinee j with test score $\theta_j \in (-\infty, \infty)$ correctly answers item i as

$$p(u_{ij} = 1|\theta_j) = c_i + \frac{1 - c_i}{1 + \exp(-Da_i(\theta_j - b_i))}, \tag{2.2}$$

where $a_i \in [0, \infty)$, $b_i \in (\infty, \infty)$, and $c_i \in [0, 1]$ respectively represent the item discrimination parameter, the item difficulty parameter, and the item guessing parameter. Additionally, constant D (typically set to 1.701) scales the model to approximate the standard normal distribution. When the guessing parameter c_i is zero, the model simplifies to the two-parameter logistic model (2PLM).

The asymptotic variance of the test score estimate based on IRT has

been shown to converge to the inverse of Fisher information [7, 47]. Consequently, Fisher information is employed as an index to represent the measurement accuracy of the examinee's test score estimation. For 3PLM, the Fisher information function $I_i(\theta)$ for item i, given an examinee's test score θ, is defined as

$$I_i(\theta) = \frac{[p'(u_i = 1|\theta)]^2}{p(u_i = 1|\theta)[1 - p(u_i = 1|\theta)]}, \tag{2.3}$$

where

$$p'(u_i = 1|\theta) = \frac{\partial}{\partial\theta} p(u_i = 1|\theta). \tag{2.4}$$

Using this information, we define the test information function $TI(\theta)$ of a test form as

$$TI(\theta) = \sum_{i=1}^{n} I_i(\theta) x_i, \tag{2.5}$$

where

$$x_i = \begin{cases} 1, & \text{if the item } i \text{ is selected for the test form, and} \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

The asymptotic standard error $SE(\hat{\theta})$ of a estimate $\hat{\theta}$ is the reciprocal of the square root of the test information function at a given test score level $\hat{\theta}$, as

$$SE(\hat{\theta}) = \frac{1}{\sqrt{TI(\hat{\theta})}}. \tag{2.7}$$

Consequently, the test information function is the sum of Fisher information function for each item in the test form. It reflects the precision with which the test form can estimate an examinee's test score at different test score points. Therefore, in parallel test forms,

maintaining a uniform test information value across all examinees' test score levels ensures that examinees can take the test at any time and from any location, without compromising the measurement accuracy of estimation. However, because the test information function is continuous, it is difficult to compute these values uniformly across all examinees' test score levels. Earlier automated parallel test form assembly methods (e.g., [48, 35, 49, 36, 34, 28, 25]) have addressed this difficulty by discretizing the values of the test information function. For example, these methods evaluate the test information function at specific points θ_k $(k = 1, 2, \ldots, K)$ on the test score scale θ. Following traditional approaches, this study applies a similar treatment to the test information function.

2.2 Automated Parallel Test Form Assembly

2.2.1 Big Shadow Test

Van der Linden proposed the Big Shadow Test (BST) method [34] using Mixed-Integer Programming (MIP), which is the most widely used automated parallel test form assembly.

Step by step, BST assembles parallel test forms using MIP to reduce the gap in test information between the assembled parallel test form and the remaining items in the pool. Specifically, BST solves the MIP problem outlined below. The objective is to minimize

$$y.$$

Subject to

$$\sum_{k=1}^{K} | \sum_{i=1}^{n} I_i(\theta_k) x_i - TI_{\text{target}}(\theta_k) | \leq L y, \tag{2.8}$$

$$\sum_{k=1}^{K} | \sum_{i=1}^{n} I_i(\theta_k) z_i - TI_{\text{ST}}(\theta_k) | \leq L_{\text{ST}} y, \tag{2.9}$$

$$\sum_{i=1}^{n} x_i = L, \tag{2.10}$$

$$\sum_{i=1}^{n} z_i = L_{\text{ST}}, \tag{2.11}$$

$$x_i + z_i \leq 1, \tag{2.12}$$

where

$$\begin{aligned} x_i &= \begin{cases} 1, & \text{if item } i \text{ is selected for the currently assembled} \\ & \text{parallel test form,} \\ 0, & \text{otherwise,} \end{cases} \\ y &\geq 0, \\ z_i &= \begin{cases} 1, & \text{if item } i \text{ is remaining items in item pool,} \\ 0, & \text{otherwise,} \end{cases} \\ TI_{\text{ST}}(\theta_k) &= \frac{L_{\text{ST}}}{L} TI_{\text{target}}(\theta_k). \end{aligned} \tag{2.13}$$

Here, L and L_{ST} respectively represent the test length and the number of remaining items in the item pool. The objective variable y represents the minimum difference between the information function of the assembled parallel test form and the target value $TI_{\text{target}}(\theta_k)$ of the test information function at the test score level θ_k. Similarly, $TI_{\text{ST}}(\theta_k)$ denotes the target value of the test information function for the remaining items in the item pool, which is the target test information multiplied by the ratio of the remaining items L_{ST} to the total test length L.

First, the constraint equation (2.8) minimizes the difference of test information between the currently assembled parallel test form and the target value $TI_{\mathrm{target}}(\theta_k)$. Then the constraint equation (2.9) minimizes the difference of test information between the set of the remaining items and its target value $TI_{\mathrm{ST}}(\theta_k)$. The MIP problem simultaneously minimizes the differences in test information function values as formulated in constraint (2.8) and constraint (2.9).

The constraint equation (2.9) ensures that items with high Fisher information are not depleted too quickly. Without the constraint equation (2.9), BST might leave only items with low Fisher information in the item pool. Consequently, the constraint equation (2.9) relaxes the rapid decrease of test information which occurs as the number of assembled parallel test forms increases.

By solving the MIP problem, BST sequentially assembles parallel test forms. In fact, BST can easily apply test constraints to the MIP. However, because of the greedy algorithm property, this BST's sequential algorithm reduces the measurement accuracy of examinees' test scores as the number of parallel test forms increases.

2.2.2 Maximum Clique Algorithm

Automated parallel test form assembly using a maximum clique algorithm (MCA) by Ishii et al. [29] (designated as ExMCA) assembled the largest number of parallel test forms with the highest measurement accuracy of examinees' test scores at that time of its publication. The MCA (e.g., [50, 51]) extracts a maximum clique in a graph. In the maximum clique, a graph $G = \{V, E\}$ consists of a finite set of vertices V and edges E. The MCA extracts the largest clique $C \subseteq V$ in

the graph G, formally defined as

$$\begin{aligned} &\textbf{maximize} && |C|, \\ &\textbf{subject to} && \\ &&& \forall v', \forall v'' \in C, \{v', v''\} \in E. \end{aligned} \tag{2.14}$$

For ExMCA, let $V = \{T_1, T_2, \ldots, T_H\}$ be a set of vertices, where each $T_h (h = 1, 2, \ldots, H)$ is a subset of items from item pool $I = \{1, 2, \ldots, n\}$. Each T_h represents a parallel test form that satisfies all test constraints except for the overlapping item constraint. Let E be a finite set of edges, where each edge exists between two vertices T_h and $T_{h'} (h \neq h')$ when the corresponding parallel test forms satisfy the overlapping item constraint. In accord with earlier studies [29, 30, 32], in this study, all test constraints are defined as presented below.

The test length constraint is given as

$$\sum_{i=1}^{n} x_i = L, \tag{2.15}$$

where L represents the test length.

The test information constraint is given as

$$I_{\mathrm{LB}}(\theta_k) \leq TI(\theta_k) \leq I_{\mathrm{UB}}(\theta_k), \tag{2.16}$$

where $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively denote the lower and upper bounds of the test information function at test score level θ_k.

The overlapping item constraint is defined as

$$\forall T_h, \forall T_{h'} \in V, |T_h \cap T_{h'}| \leq OC, \tag{2.17}$$

where OC is defined as the maximum number of common items between any pair of parallel test forms.

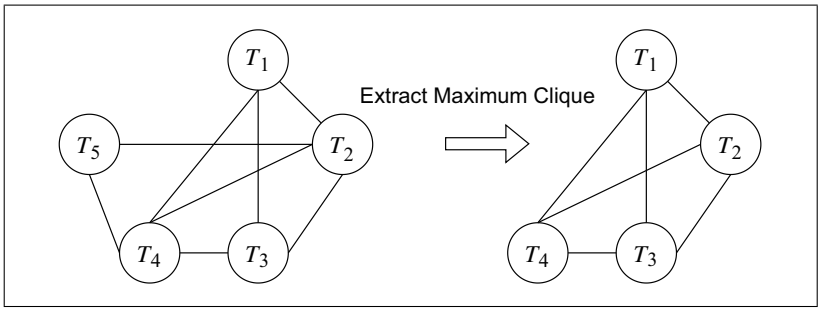


Figure 2.1: Example of ExMCA.

As a result, the maximum clique represents parallel test forms that satisfy all test constraints.

Figure 2.1 presents an example of ExMCA. In the figure, a graph has five vertices (test forms) and eight edges. In the figure, $T_1, T_2, T_3,$ and T_4 are the vertices included in the maximum clique in the graph.

The ExMCA proceeds with the following three procedures.

Procedure 1: (Generating parallel test forms)
Using a branch-and-bound approach (e.g., [52]), Procedure 1 assembles all parallel test forms that satisfy test constraints, except for the overlapping item constraint.

Procedure 2: (Constructing a graph)
Procedure 2 constructs the graph by counting overlapping items between pairs of parallel test forms, which are represented as vertices. Only pairs that satisfy the overlapping item constraint are connected by edges.

Procedure 3: (Extracting the maximum clique)
Procedure 3 extracts a maximum clique from the graph, representing the largest number of parallel test forms that satisfy all test constraints.

Although the ExMCA guarantees the maximum number of parallel

test forms, the computational cost of extracting the maximum clique is considerable. Specifically, the time and space complexities of in the ExMCA are, respectively, $\mathcal{O}(2^{|V|})$ and $\mathcal{O}(|V|^2)$. Furthermore, the number of vertices $|V|$ can be at most the total number of combinations of selecting L (test length) items from n items in an item pool. Namely, $|V| = \binom{n}{L}$. As a result, ExMCA becomes computationally infeasible for large-scale item pools because the graph size exceeds practical computer memory limitations.

To address the high complexities of ExMCA, Ishii et al. [30] proposed an approximation algorithm, the Random Maximum Clique Algorithm (RndMCA), which requires the following inputs.

- Constant value parameter $SS(|C| \leq SS \ll |V|)$ represents the sampling size of each random sampling vertices from V.
- Constant time parameter T_{limit} stands for the computation time limit to search for vertices (parallel test forms) in Procedure 1 or to extract a maximum clique in Procedure 3. Specifically, when Procedure 1 does not find SS parallel test forms within the computation time limit T_{limit}, it returns parallel test forms found up to that time. Similarly, when Procedure 3 does not extract the maximum clique within T_{limit}, it returns the largest clique that time.
- Constant time parameter LT denotes the algorithm's total computation time limit.
- Finite set I represents a set of items in an item pool.
- Constant value parameter L stands for the test length.
- Constant value parameter n represents the number of items in the item pool.
- Constant value parameter K denotes the number of discretized points for the test information function.

- Constant value parameters $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively denote the lower and upper bounds of the test information function at test score level θ_k.
- Constant value parameter OC signifies the maximum number of common items between any pair of parallel test forms.

Algorithm 1 provides a description of RndMCA. The output of Algorithm 1 is the set $C_{\max}$, which represents the maximum number of parallel test forms found within the computation time LT.

Algorithm 1: RndCMA

---

procedure RndMCA

Input: $SS, T_{\mathrm{limit}}, LT, I, L, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC$

Output: $C_{\max}$

$C \leftarrow \emptyset, C_{max} \leftarrow \emptyset$

$st \leftarrow now()$

▷ $now()$ retrieves the current timestamp to track elapsed computation time.

while $(now() - st) < LT$ **do**

▷ Procedure 1

$V \leftarrow$ RandomSearch($SS, T_{\mathrm{limit}}, I, L, n, K,$ $I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k)$)

▷ RandomSearch searches for at most SS vertices that satisfy test constraints except for overlapping item constraint from I, L, n $K, I_{\mathrm{LB}}(\theta_k)$, and $I_{\mathrm{UB}}(\theta_k)$ within computation time limit T_{limit}.

▷ Procedure 2

$E \leftarrow$ ConnectVertices(V, OC)

▷ ConnectVertices constructs edges between pairs of vertices that satisfy the overlapping item constraint.

$G \leftarrow (V, E)$

▷ Procedure 3

$C \leftarrow$ MCA(G, T_{limit})

▷ MCA extracts the maximum clique from the graph G within computation time limit T_{limit}.
    if $|C_{max}| < |C|$ **then**
        $C_{max} \leftarrow C$
    end if
end while
Output $C_{\max}$
▷ $C_{\max}$ represents the found parallel test forms within computation time limit LT.
end procedure

RndMCA repeatedly extracts the maximum number of parallel test forms from a subgraph of the overall graph. The time and space complexities of a single iteration through Procedures 1 to 3 in RndMCA can be analyzed based on the complexity of its three procedures.

In the worst-case, Procedure 1 requires exploring all combination of selecting L items from the n items in the item pool. Consequently, Procedure 1 has a high time complexity of $\mathcal{O}(\binom{n}{L})$, which depends on the input sizes L and n. Additionally, to avoid duplicate combinations, Procedure 1 stores explored parallel test forms on the computer memory. As a result, the space complexity of Procedure 1 is $\mathcal{O}(\binom{n}{L})$, which depends on the input sizes L and n.

Procedure 2 constructs a graph by verifying the number of overlapping items between each pair of SS parallel test forms. Then, Procedure 2 has a time complexity of $\mathcal{O}(SS^2)$ depending on the input size SS. Additionally, Procedure 2 requires a space complexity of $\mathcal{O}(SS^2)$ to store the adjacency structure of the graph depending on the input size SS.

Procedure 3 has a high time complexity of $\mathcal{O}(2^{SS})$ and a high space

complexity of $\mathcal{O}(SS^2)$ to extract a maximum clique in the graph depending on the input size SS.

Therefore, the single iteration in RndMCA has a high time complexity of $\mathcal{O}(\binom{n}{L}+2^{SS})$ and a high space complexity of $\mathcal{O}(\binom{n}{L}+SS^2)$ through Procedures 1 to 3. RndMCA repeats Procedures 1 to 3 with the high time and space complexity within the time limit LT. However, it is known [30] to assemble more parallel test forms than conventional methods do, even when the computation is interrupted under short time limits LT and T_{limit}. However, when the number of parallel test forms becomes too large, the high space complexity of RndMCA makes it impossible to assemble more than 100,000 parallel test forms because of a computer memory limit.

Chapter 3

Automated Parallel Test Form Assembly using Integer Programming for Maximum Clique

3.1 Random Integer Programming Maximum Clique Algorithm for Automated Parallel Test Form Assembly

RndMCA was recognized for assembling the largest number of parallel test forms at the time of its publication [30], but it remains restricted to, at most, 100,000 parallel test forms because of its high space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$. To mitigate RndMCA's space complexity, this study proposes a new automated parallel test form assembly: Random Integer Programming Maximum Clique Algorithm (RIPMCA). The key idea of RIPMCA is iterative search for a vertex that is connected to all vertices in the current clique using IP, thereby expanding the clique step by step. An IP problem formulation for assembling parallel test forms is the following. The objective is to maximize

$$\sum_{i=1}^{n} \lambda_i x_i, \tag{3.1}$$

where the binary variable x_i is defined as follows: it equals 1 if item i is selected for the parallel test form, and 0 otherwise. Additionally,

λ_i are random variables that are distributed uniformly in the range [0,1]; these values are resampled after each problem is solved. The IP with the sampled weight λ_i randomly assembles parallel test forms from feasible solutions. Consequently, the weight λ_i ensures a more uniform item exposure of parallel test forms, defined as how many times each item is used. Subject to

$$\sum_{i=1}^{n} x_i = L, \tag{3.2}$$

$$\forall k \in \{1, 2, \ldots, K\}, I_{\mathrm{LB}}(\theta_k) \leq \sum_{i=1}^{n} I_i(\theta_k) x_i \leq I_{\mathrm{UB}}(\theta_k), \tag{3.3}$$

$$\forall r \in \{1, 2, \ldots, |C|\}, \sum_{i=1}^{n} X_{i,r} x_i \leq OC, \tag{3.4}$$

where $X_{i,r}$ equals 1 if the item i is included in the parallel test form r of the current clique C, and 0 otherwise. The IP problem randomly searches a vertex (parallel test form) that is connected to all vertices (parallel test forms) in the current clique C. Equation (3.2) restricts test length L in the parallel test form. The term of $\sum_{i=1}^{n} I_i(\theta_k) x_i$ in equation (3.3) restricts the value of test information function of assembling parallel test form at each test score level θ_k. Specifically, equation (3.3) restricts the value of the test information function by setting the lower bound $I_{\mathrm{LB}}(\theta_k)$ and upper bound $I_{\mathrm{UB}}(\theta_k)$. The term $\sum_{i=1}^{n} X_{i,r} x_i$ restricts the number of overlapping items between the assembling parallel test form and parallel test form r in the current clique C. Equation (3.4) restricts the number of overlapping items between any two parallel test forms in the current clique C. Consequently, the constraint equations (3.2)–(3.4)guarantee an assembling vertex (parallel test form), which is connected to all vertices (parallel test forms) in the current clique C.

By solving the IP problem, RIPMCA searches for a vertex connected to all vertices in the current clique. Then, RIPMCA sequen-

tially expands the maximum clique by repeating the search. For this sequential maximum clique expansion, RIPMCA requires the following inputs.

- Constant time parameter LT represents the algorithm's total computation time limit.
- Finite set I denotes a set of items in an item pool.
- Constant value parameter L stands for the test length.
- Constant value parameter n represents the number of items in the item pool.
- Constant value parameter K represents the number of discretized points for the test information function.
- Constant value parameters $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively signify the lower and upper bounds of the test information function at test score level θ_k.
- Constant value parameter OC stands for the maximum number of common items between any pair of parallel test forms.

Algorithm 2 provides a description of RIPMCA. The output of Algorithm 2 is the set $C_{\max}$, which represents the maximum number of parallel test forms found within the computation time limit LT.

Algorithm 2: RIPMCA

---

procedure RIPMCA
Input: $LT, I, L, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC$
Output: $C_{\max}$
$C \leftarrow \emptyset, C_{max} \leftarrow \emptyset$
$st \leftarrow now()$
▷ $now()$ retrieves the current timestamp to track the elapsed computation time.
while $(now() - st) < LT$ **do**
$T \leftarrow \mathrm{IP}(C, I, L, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC)$
▷ IP searches a vertex that is connected with all vertices in the current clique C.
if $T \neq \emptyset$ **then**
$C \leftarrow C \cup T$
if $|C_{max}| < |C|$ **then**
$C_{max} \leftarrow C$
end if
else
$C \leftarrow \emptyset$
end if
end while
Output $C_{\max}$
▷ $C_{\max}$ represents the found parallel test forms within the computation time limit LT.
end procedure

In RIPMCA, the time complexity of solving the IP to search for a vertex is $\mathcal{O}(m_c \cdot 2^n)$. The reason is that the IP problem is solved using the branch-and-bound method, which explores the solution space by branching on decision variables and bounding sub problems to prune

infeasible regions. In the worst case, this process requires evaluating up to 2^n possible solutions while ensuring that they satisfy the m_c constraints. Here, m_c increases as $|C|$ in equation (3.4) becomes larger. Therefore, the time complexity of IP computation in RIPMCAP is $\mathcal{O}(|C| \cdot 2^n)$. Furthermore, RIPMCA expands the maximum clique sequentially by repeating this search. When the search continues until no new vertex is found to expand the clique, RIPMCA has a high time complexity of $\mathcal{O}(|C|^2 \cdot 2^n)$, which depends on the input sizes C and n, because the IP problem is solved $|C|$ times repeatedly. On the other hand, RIPMCA does not store the graph structure for the maximum clique search. Instead, RIPMCA only stores $|C|$ parallel test forms found on the computer memory. Consequently, RIPMCA has a low space complexity of $\mathcal{O}(|C|)$, which depends on the input size $|C|$. As a result, RIPMCA might reduce memory usage compared to RndMCA, which has the high space complexity of $\mathcal{O}(\binom{n}{L} + SS^2)$.

3.2 Hybrid Maximum Clique Algorithm using Integer Programming for Automated Parallel Test Form Assembly

Actually, RIPMCA's space complexity $\mathcal{O}(|C|)$ is lower than RndMCA's space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$. This benefit is expected to increase the number of parallel test forms within a computer memory limit. However, the improvement of RIPMCA remains limited because of its high time complexity $\mathcal{O}(|C|^2 \cdot 2^n)$. To address this difficulty, this study proposes a new two-stage parallel method for automated parallel test form assembly: Hybrid Maximum Clique Algorithm with Parallel Integer Programming (HMCAPIP).

The first stage is to extract a maximum clique found within a computation time that is as large as possible up to a computer memory limit using RndMCA, which has high space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$. Although RndMCA assembles parallel test forms within the computation time, because of the computer memory limit, it becomes impractical to search for a clique with more than a hundred thousand vertices. Therefore, once the computer memory limit is reached, HMCAPIP transitions to the second stage using RIPMCA, where it searches repeatedly for vertices connected to all vertices in the current clique from the remaining vertices using IP, which has low space complexity $\mathcal{O}(|C|)$ but high time complexity $\mathcal{O}(|C|^2 \cdot 2^n)$. Nevertheless, the number of additionally assembled parallel test forms using IP is limited because of its high time complexity. To address this limitation, HMCAPIP parallelizes the second stage. However, the second stage is difficult to parallelize because it requires adding overlapping item constraint expressions to IP when a new vertex is found. Therefore, HMCAPIP searches vertices through the following procedures, as presented in Figure 3.1.

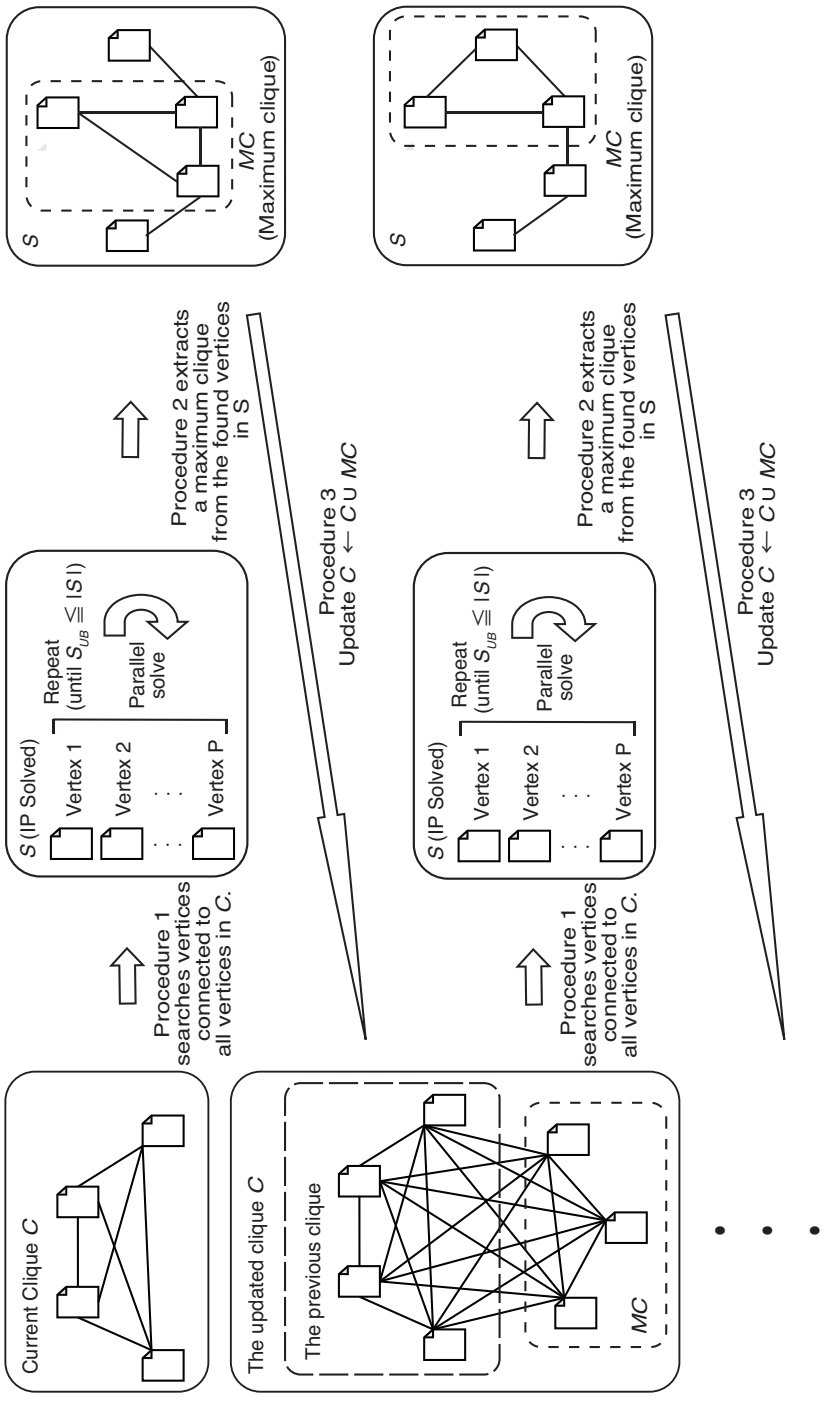


Figure 3.1: Outline of the second stage in HMCAPIP.

Procedure 1: Using multi-core processors, Procedure 1 searches in parallel for P vertices that are connected to all vertices of the current clique using IP, where P is the number of parallelisms for searching a vertex using the IP. Here, the optimal solution in IP varies with each search because of the resampling of λ_i. Then, the found vertices are added to a set $S \subseteq V$, which represents the set of vertices connected to all vertices in the current clique. Additionally, the previously found solutions in the set S remain feasible under the new objective function because the constraints of the IP problem remain unchanged. Consequently, the next IP search can start with the maximum value of the new objective function, taken from the previously found solutions, as the initial lower bound. Procedure 1 continues until $S_{\mathrm{UB}} \leq |S|$, where S_{UB} is the limit of the number of vertices imposed by IP.

Procedure 2: Procedure 2 extracts the maximum clique MC from the set S, which is obtained in Procedure 1.

Procedure 3: The found maximum clique MC in Procedure 2 is combined with the current clique C, thereby expanding the clique size. This combined clique becomes the new current clique.

Procedure 4: When Procedure 1 fails to find a solution for the IP, the search might fall into a local solution of the maximum clique. In such a case, to avoid the local solution, Procedure 4 randomly removes some vertices from the current clique C.

HMCAPIP requires the following inputs.

- Constant value parameter $SS(|C| \leq SS \ll |V|)$ represents the sampling size of each random sampling vertices from V in RndMCA.

- Constant time parameter T_{limit} denotes the computation time limit to extract the maximum clique.
- Constant time parameter LT' represents RndMCA's total computation time limit.
- Constant time parameter LT stands for the algorithm's total computation time limit.
- Tuning parameter $S_{\mathrm{UB}}(S_{\mathrm{UB}} \leq SS)$ represents the upper bound of the number of vertices using IP in the second stage.
- Constant value parameter D denotes the number of vertices removed from the current clique when the IP fails to find a solution. Although the value of D can be adjusted, preliminary experiments suggest it leads to slight differences in the number of parallel test forms. Accordingly, D is set to 100 for this study.
- Tuning parameter P represents the number of parallelisms for searching a vertex using IP.
- Finite set I represents a set of items in an item pool.
- Constant value parameter L stands for the test length.
- Constant value parameter n denotes the number of items in the item pool.
- Constant value parameter K represents the number of discretized points for the test information function.
- Constant value parameters $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively denote the lower and upper bounds of the test information function at test score level θ_k.
- Constant value parameter OC signifies the maximum number of common items between any pair of parallel test forms.

Algorithm 3 provides a description of HMCAPIP. The output of Algorithm 3 is the set $C_{\max}$, which represents the maximum number of parallel test forms found within the computation time limit LT.

Algorithm 3: HMCAPIP

1: **procedure** HMCAPIP
2: **Input:** $SS, T_{\text{limit}}, LT', LT, S_{\text{UB}}, D, P, I, L, n,$
3: $K, I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k), OC$
4: **Output:** $C_{\max}$
5: $st \leftarrow now()$
▷ $now()$ retrieves the current timestamp to track the elapsed computation time.
6: $C \leftarrow$ RndMCA($SS, T_{\text{limit}}, LT', I, L, n, K, I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k)$
7: $, OC$)
▷ First stage
8: $C_{max} \leftarrow$ SecondStage($C, LT, S_{\text{UB}}, D, P, I, L, n,$
9: $K, I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k), OC, st$)
10: **Return** $C_{\max}$
11: **end procedure**

12: **function** SecondStage
13: **Input:** $C, LT, S_{\text{UB}}, D, P, I, L, n, K, I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k), OC, st$
14: **Output:** $C_{\max}$
15: $C_{max} \leftarrow C$
16: **while** $(now() - st) < LT$ **do**
17: $S \leftarrow \emptyset$
18: **repeat**
19: $Sol \leftarrow \emptyset$
20: **parallel for** $p \leftarrow 1 \ldots P$ **do**
▷ Search P vertices in parallel.
21: $Sol \leftarrow Sol \cup$ SearchVertex($C, S, I, L, n, K,$
22: $I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k), OC$)
▷ SearchVertex searches a vertex that is connected to all vertices in C using IP.
23: **end parallel for**
24: **if** $Sol \neq \emptyset$ **then**

▷ IP has solutions

$S \leftarrow S \cup Sol$

else

$C \leftarrow$ Remove(C, D)

▷ Remove(C, D) removes D vertices from C

$break$

end if

until $S_{\mathrm{UB}} \leq |S|$

if $S \neq \emptyset$ **then**

$E \leftarrow$ ConnectVertices(S, OC)

▷ ConnectVertices constructs edges between pairs of vertices that satisfy the overlapping item constraint.

$G \leftarrow (S, E)$

▷ (S, E) constructs a graph that corresponds to the set S with the overlapping item constraint.

$MC \leftarrow \mathrm{MCA}(G, T_{\mathrm{limit}})$

▷ MCA(G, T_{limit}) extracts the maximum clique from the graph G within calculation time T_{limit}.

$C \leftarrow C \cup MC$

if $|C_{max}| < |C|$ **then**

$C_{max} \leftarrow C$

end if

end if

end while

return $C_{\max}$

end function

function SearchVertex

Input: $C, S, I, L, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC$

Output: A vertex is connected to all vertices in C using IP.

$LB \leftarrow \max_{T_h \in S} \left(\sum_{i \in T_h} \lambda_i x_i \right)$

▷ Objective function is equation (3.1)

return IP$(C, S, I, L, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC, LB)$

▷ $\mathrm{IP}()$ solves IP with initial lower bound LB
48: **end function**

3.3 Tuning Parameter Optimization and Lower Bound Evaluation for HMCAPIP

This section presents determination of the optimal tuning parameters for HMCAPIP using a simulated item pool to maximize the number of parallel test forms. In addition, this section presents evaluation showing that HMCAPIP can increase the number of assembled parallel test forms using the initial lower bound in Procedure 1.

3.3.1 Optimization of Parameter S_{UB}

This experiment ascertains the optimal value to maximize the number of parallel test forms by modifying the value of the tuning parameter S_{UB} using a simulated item pool with 1000 items, close to the number of items in the real-world dataset used by Ishii et al. [30]. Items in the simulated item pools have discrimination parameters and difficulty parameters based on IRT according to van der Linden [34]. Specifically, the item discrimination parameters were generated independently for each item as $\log_2 a \sim N(0, 1)$. The item difficulty parameters were also generated independently for each item as $b \sim N(0, 1)$. Furthermore, the discrimination and difficulty parameters were generated independently of each other. The values of guessing parameters c_i of all items in the simulated item pools are 0. Additionally, this study sets all test constraints as presented below.

1. The test length L is 25 items.

2. Test information constraints are given by the lower and upper bounds of the test information function, as shown in Table 3.1.

3. The maximum number of common items OC is increased from 0 to 10 in increments of 1, corresponding to 0%–40% of the test length L, with increments of 4%.

Table 3.1: Test information constraints for HMCAPIP experimentation

$I_{\mathrm{LB}}(\theta_k)/I_{\mathrm{UB}}(\theta_k)$				
$\theta_1 = -2.0$	$\theta_2 = -1.0$	$\theta_3 = 0.0$	$\theta_4 = 1.0$	$\theta_5 = 2.0$
2.0/2.4	3.2/3.6	3.2/3.6	3.2/3.6	2.0/2.4

For HMCAPIP, we set SS to 100,000, T_{limit} to 3 hr, LT' to 3 hr, P to 1, D to 100, and LT to 24 hr. The values of $SS, T_{\mathrm{limit}}, LT'$, and LT are determined as described for an experiment by Ishii et al. [30].

The first experiment focuses on determining the value of S_{UB}. To isolate the effect of S_{UB}, the solutions found in IP are not used as the initial lower bound for subsequent searches. Additionally, the number of parallelisms P is set to 1. This experiment compares the number of parallel test forms for $S_{\mathrm{UB}} = 10, 100, \mathrm{and}\ 1000$.

This experiment was conducted on a computer equipped with an Core i9-9900X 3.50 GHz CPU (Intel Corp.), 128 GB of RAM, and running a 64-bit Linux (Ubuntu) operating system. The same RAM capacity was used by Ishii et al. [30]. HMCAPIP employed CPLEX 12.9 [54] to solve IP.

Table 3.2: Performance of HMCAPIP achieved by modifying the tuning parameter S_{UB} values

Item Pool Size	OC	Proposal								
		$S_{\mathrm{UB}} = 10$			$S_{\mathrm{UB}} = 100$			$S_{\mathrm{UB}} = 1000$		
		No. tests	Avg. ST (MCA) [s]	Avg. ST (IP) [s]	No.tests	Avg. ST (MCA) [s]	Avg. ST (IP) [s]	No. tests	Avg. ST (MCA) [s]	Avg. ST (IP) [s]
1000	0	**34**	**0.0001**	55.24	**34**	0.8632	47.52	**34**	457.4781	**37.23**
	1	243	**0.0001**	24.81	**261**	0.2375	20.24	253	180.0563	**10.07**
	2	1595	**0.0001**	33.29	**1639**	0.1338	25.63	1349	177.5153	**9.14**
	3	**7522**	**0.0001**	12.41	7355	0.0010	11.05	6936	180.0590	**6.36**
	4	33150	**0.0001**	3.11	**33160**	0.0008	3.04	31012	0.0583	**2.96**
	5	63706	**0.0001**	4.01	**65458**	0.0006	**3.58**	63719	0.0535	4.00
	6	105367	**0.0001**	**4.46**	**105391**	0.0007	**4.46**	105239	0.0463	4.51
	7	108286	**0.0001**	7.97	**108765**	0.0007	**7.56**	108279	0.0443	7.99
	8	108759	**0.0001**	8.08	**109245**	0.0006	**7.66**	108761	0.0455	8.10
	9	108698	**0.0001**	8.12	**109202**	0.0007	**7.66**	108746	0.0475	8.10
	10	108699	**0.0001**	8.10	**109203**	0.0005	**7.68**	108744	0.0472	8.12

Table 3.2 presents the relative performance results of HMCAPIP based on different S_{UB} values. Table 3.2 includes the following details: "No. tests" represents the number of parallel test forms; "Avg. ST (MCA) [s]" denotes the average calculation time required to search for the maximum clique from S in Procedure 2 of HMCAPIP's second stage; and "Avg. ST (IP) [s]" stands for the average of search calculation time using IP for searching a vertex in Procedure 1 of HMCAPIP's second stage.

When $OC \leq 3$, HMCAPIP provides a tradeoff between "Avg. ST (MCA)" and "Avg. ST (IP)". Particularly as the S_{UB} value increases, "Avg. ST (MCA)" increases whereas "Avg. ST (IP)" decreases. In contrast, when $OC > 3$, the effect of this tradeoff is limited because the difference between "Avg. ST (MCA)" and "Avg. ST (IP)" is small at different S_{UB} values. As a result, in most cases, when $S_{\mathrm{UB}} = 100$, HMCAPIP assembles the greatest number of parallel test forms. Consequently, for this study, the value of S_{UB} was determined as 100.

3.3.2 Performance of HMCAPIP with Lower Bound

In Procedure 1 of HMCAPIP, the next IP search can start with the maximum value calculated from equation (3.1) as the initial lower bound, using the previously found solutions. For this experiment, we refer to this method with the lower bound as "HMCAPIP with the lower bound" and to that without as "HMCAPIP without the lower bound". Results obtained from this experiment demonstrate that "HMCAPIP with the lower bound" increases the number of parallel test forms compared to "HMCAPIP without the lower bound" using the same simulated item pool, computer, and test constraints as those described in subsection 3.3.1. For "HMCAPIP with the lower bound" and "HMCAPIP without the lower bound", we set SS to 100,000, T_{limit} to 3 hr, LT' to 3 hr, S_{UB} to 100, P to 1, D to 100, and LT to 24 hr.

Table 3.3 presents comparison of performance results obtained using "HMCAPIP with the lower bound" and "HMCAPIP without the lower bound". The following are presented in Table 3.3. In the table, "No. tests" represents the number of parallel test forms. "Avg. SN" and "Avg. CT [s]" respectively denote the number of search vertices and the average of calculation time using IP for searching a vertex in Procedure 1 of HMCAPIP's second stage. "Avg. SN" and "Avg. CT [s]" are measured to evaluate whether HMCAPIP with the lower bound reduces the search space and calculation time in IP.

In fact, results show that "HMCAPIP with the lower bound" assembles more parallel test forms than "HMCAPIP without the lower bound" does. Furthermore, both "Avg. SN" and "Avg. CT [s]" of "HMCAPIP with the lower bound" are lower than those of "HMCAPIP without the lower bound" because "HMCAPIP with the lower bound" prunes the search space using the branch-and-bound process of the IP, to reduce the computation time. These results confirm the effectiveness of "HMCAPIP with the lower bound" to increase the number of assembled test forms. "HMCAPIP with the lower bound" is simply stated as "HMCAPIP" hereinafter.

Table 3.3: Performance of "HMCAPIP with the lower bound" and "HMCAPIP without the lower bound"

Item Pool Size	OC	Method					
		HMCAPIP without lower bound			HMCAPIP with lower bound		
		No. tests	Avg. SN	Avg. ST [s]	No. tests	Avg. SN	Avg. ST [s]
1000	0	**34**	254232.3	47.52	**34**	**234232.3**	**45.82**
	1	261	144763.9	22.72	**293**	**134352.1**	**20.24**
	2	1639	146458.8	26.05	**1670**	**135961.9**	**25.63**
	3	7355	18974.5	11.05	**7383**	**17793.0**	**11.04**
	4	33160	1307.7	3.04	**36325**	**965.4**	**2.66**
	5	65458	233.2	3.58	**74654**	**219.8**	**2.33**
	6	105391	154.7	4.46	**107527**	**149.6**	**3.84**
	7	108765	133.0	7.56	**114296**	**128.1**	**4.52**
	8	109245	128.4	7.66	**114835**	**126.8**	**4.55**
	9	109202	129.5	7.66	**114796**	**126.1**	**4.54**
	10	109203	129.7	7.68	**114781**	**126.6**	**4.57**

3.3.3 Optimization of the number of parallelisms

To mitigate the high time complexity of IP, HMCAPIP repeats the parallel search of the vertices that are connected with all vertices of the current clique using IP. The improvement of the parallel search is influenced by the value of the tuning parameter P. This experiment was conducted using the same simulated item pool, computer, and test constraints as in subsection 3.3.1 to evaluate the HMCAPIP performance achieved by modifying the tuning parameter P.

For HMCAPIP, we set SS to 100,000, T_{limit} to 3 hr, LT' to 3 hr, S_{UB} to 100, D to 100, and LT to 24 hr. The experiment was conducted to compare the number of parallel test forms by $P = 1, 2, 5, \text{and } 10$.

Table 3.4 presents the number of parallel test forms as the value of tuning parameter P is modified. The following are shown in Table 3.4. "No. tests" refers to the number of parallel test forms, and "Avg. ST [s]" represents the average search time for searching a vertex using IP in Procedure 1 of HMCAPIP's second stage. In fact, "Avg. ST [s]" was measured to demonstrate that increasing the tuning parameter P reduces the average search time for finding a single vertex.

When OC is small, the numbers of parallel test forms remain almost identical across all values of P because the maximum number of assembled parallel test forms is limited by the tight OC. However, as OC increases, $P = 10$ tends to assemble the greatest number of parallel test forms. Results suggest that the efficiency of parallelization improves as the number of parallel test forms increases. Based on results obtained from the experiment, this study sets P to 10.

Table 3.4: Performance of parallel search of HMCAPIP

Item Pool Size	OC	Method							
		$P = 1$		$P = 2$		$P = 5$		$P = 10$	
		No. tests	Avg. ST [s]	No. tests	Avg. ST [s]	No. tests	Avg. ST [s]	No. tests	Avg. ST [s]
1000	0	**34**	45.82	**34**	24.32	**34**	10.31	**34**	**6.24**
	1	**293**	20.24	290	14.36	236	5.44	272	**2.97**
	2	**1670**	25.63	1584	18.29	1605	9.63	1657	**5.33**
	3	7383	11.04	7972	8.74	**9462**	6.35	9000	**4.09**
	4	36325	2.66	36057	1.99	37333	1.44	**39970**	**1.28**
	5	74654	2.33	76997	2.02	94127	1.08	**106881**	**0.66**
	6	107527	3.84	106967	3.78	119624	1.82	**134050**	**0.98**
	7	114296	4.52	114668	4.18	126496	2.00	**139172**	**1.07**
	8	114835	4.55	115203	4.20	126759	2.01	**139757**	**1.09**
	9	114796	4.54	115090	4.21	126703	2.02	**140059**	**1.07**
	10	114781	4.57	115092	4.20	126702	2.04	**140067**	**1.04**

3.4 Comparison of HMCAPIP to Earlier Methods

This study then compares HMCAPIP's performance with the respective performances of previously reported methods, using both simulated and actual item pools.

3.4.1 Comparing HMCAPIP to earlier methods

This experiment demonstrates that HMCAPIP can assemble more parallel test forms than the other methods can. For this purpose, this experiment compares the number of parallel test forms assembled using HMCAPIP, RIPMCA, and earlier methods (BST [34] in section 2.2.1 and RndMCA [30] in subsection 2.2.2) using the test constraints and computer as described in subsection 3.3.1 for both simulated and actual item pools.

These simulation item pools have 1000 and 2000 items. The items in the simulated item pools include discrimination and difficulty parameters based on IRT. Actually, the item discrimination parameters were generated independently for each item as $\log_2 a \sim N(0, 1^2)$. The item difficulty parameters were also generated independently for each item as $b \sim N(0, 1^2)$. Furthermore, the discrimination and difficulty parameters were generated independently of each other. The values of guessing parameters c_i of all items from simulated item pool are 0. As reported by van der Linden [34], simulations frequently assume these distributions to approximate real-world item pools.

Table 3.5 presents specific information about the actual item pool. This actual item pool is used for the synthetic personality inventory (SPI) examination: a widely used aptitude test in Japan [55]. The table shows that real-world item pools sometimes fail to satisfy the parameter distribution assumptions commonly used in simulations. This

Table 3.5: Specific information about the actual item pool

Item pool size	Item discrimination parameter a			Item difficulty parameter b		
	Range	Mean	SD	Range	Mean	SD
978	$0.12 - 3.08$	0.46	0.19	$-4.00 - 4.55$	-0.22	1.57

study demonstrates that HMCAPIP can assemble more parallel test forms than the earlier methods can, even in the actual item pool.

This experiment used 24 hr as a computation time limit for all methods. For RndMCA, this experiment determined the values $SS = 100,000$, $T_{\mathrm{limit}} = 3$ hr, and $LT = 24$ hr according to the explanation provided by Ishii et al. [30]. We determine the target value $TI_{\mathrm{target}}(\theta_k)$ of information function for BST as

$$TI_{\mathrm{target}}(\theta_k) = \frac{I_{\mathrm{LB}}(\theta_k) + I_{\mathrm{UB}}(\theta_k)}{2}.$$

In addition, BST [34], RIPMCA, and HMCAPIP employed CPLEX 12.9 [54] to solve the IP.

Table 3.6: Numbers of assembled parallel test forms for all methods using each item pool

Item Pool Size	OC	BST [56]	RndMCA [30]	RIPMCA	HMCAPIP
1000	0	25	17	**34**	**34**
	1	25	61	**318**	272
	2	25	282	**1892**	1431
	3	25	1585	7557	**9000**
	4	25	9793	20653	**39970**
	5	25	46162	55024	**106881**
	6	25	90127	96527	**134050**
	7	25	99396	106834	**139172**
	8	25	99979	107942	**139757**
	9	25	99998	107735	**140059**
	10	25	100000	107672	**140067**
2000	0	61	32	**70**	**70**
	1	61	186	**1531**	988
	2	61	1463	6963	**7569**
	3	61	12456	25364	**51401**
	4	61	62424	72520	**108165**
	5	61	96859	103354	**129257**
	6	61	99891	106362	**131791**
	7	61	99993	107434	**132273**
	8	61	100000	107774	**132090**
	9	61	100000	107998	**133550**
	10	61	100000	107783	**140700**
978 (actual)	0	31	18	**35**	**35**
	1	31	63	**348**	286
	2	31	297	**1844**	1334
	3	31	1717	6960	**7050**
	4	31	10252	14866	**31724**
	5	31	45746	52126	**73693**
	6	31	88947	93704	**108935**
	7	31	99993	104339	**118165**
	8	31	100000	105823	**119797**
	9	31	100000	105805	**119758**
	10	31	100000	105956	**124200**

Table 3.6 provides the number of assembled parallel test forms for all methods using each item pool by varying the value of OC. When $OC \geq 3$, results show that HMCAPIP assembles more parallel test forms than the other methods do. Especially, the difference of the number of parallel test forms between RIPMCA and HMCAPIP remains large even when the number of parallel test forms becomes larger than 100,000. Results reflect that HMCAPIP reduces the computation time of RIPMCA by the two-stage algorithm and parallel search of IPs.

By contrast, when $OC \leq 2$, HMCAPIP assembles an almost identical number of assembled parallel test forms as RIPMCA does. The numbers converge to the maximum number of assembled parallel test forms because, as a result of the tight overlapping item constraint, the maximum number of parallel test forms is not large.

In fact, because RIPMCA has lower space complexity than RndMCA has, RIPMCA assembles more parallel test forms than the earlier methods do. However, the difference in the number of parallel test forms between RIPMCA and RndMCA becomes smaller as their numbers increase. The results suggest that the performance of RIPMCA is constrained by its high time complexity.

Regarding the methods described above, except when $OC = 0$, RndMCA assembles more parallel test forms than BST does. The reason is that BST cannot increase the number of parallel test forms because it disallows overlapping items.

3.4.2 Comparison of HMCAPIP to earlier methods with extended computation time

Both RIPMCA and HMCAPIP have lower space complexity than RndMCA has, which allows for more assembled parallel test forms

within memory constraints. However, RIPMCA's improvement is constrained by the high time complexity of IP. By contrast, HMCAPIP relaxes the computation time across multi-core processors. As a result, the difference in the number of parallel test forms between RIPMCA and HMCAPIP might increase as the computation time increases. To explore this possibility, this experiment compares the number of parallel test forms using RndMCA, RIPMCA, and HMCAPIP by extending the computation time limit to 168 hr (seven days). The comparison is conducted using a simulation item pool with 2000 items, which assembles the greatest number of parallel test forms in the simulated pools, as well as an actual item pool including 978 items.

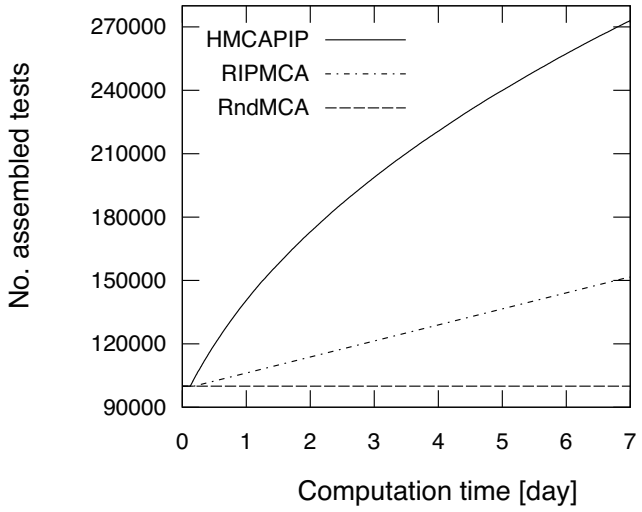
Table 3.7: Numbers of assembled parallel test forms in 168 hr

item pool size	OC	RndMCA	RIPMCA	HMCAPIP
2000	0	32	**70**	**70**
	5	96859	149403	**254418**
	10	100000	151592	**274900**
978 (actual)	0	18	**35**	**35**
	5	45746	103763	**139048**
	10	100000	140185	**214350**

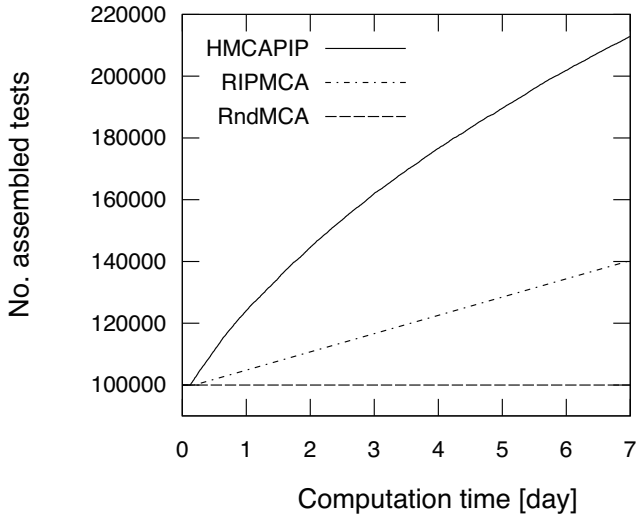
Table 3.7 presents the number of assembled parallel test forms for each method, given the time limitation of 168 hr. The results demonstrate that RIPMCA and HMCAPIP can assemble more parallel test forms than RndMCA can because RIPMCA and HMCAPIP have lower space complexity than RndMCA has. Because of the space complexity limitation, the number of parallel test forms by RndMCA does not increase as the computation time increases. In contrast, HMCAPIP assembles 1.5–2.7 times more parallel test forms than RndCMA does, except for the case of $OC = 0$. When $OC = 0$, the numbers of parallel test forms assembled by the proposed methods

are equal because they converge to a maximal number of parallel test forms. When OC becomes large, the differences of the number of parallel test forms between RIPMCA and HMCAPIP become large because HMCAPIP divides the high time complexity of IP.

Figure 3.2 depicts the number of parallel test forms of HMCAPIP and RIPMCA with $OC = 10$, for (a) simulated item pool size 2000, and for (b) actual item pool size 978, which assembled the largest number of parallel test forms in Table 3.7. The figure is presented as a line plot, where the horizontal axis represents the computation time. The vertical axis shows the number of parallel test forms for each method. As the figure shows, the difference in the number of assembled parallel test forms between HMCAPIP and RIPMCA becomes large as the computation time increases. Results demonstrate that HMCAPIP assembles more parallel test forms, given a longer computation time.



(a) Simulated item pool



(b) Actual item pool

Figure 3.2: Line plot of number of parallel test forms for each method in 168 hr.

Chapter 4

Automated Parallel Test Form Assembly using Zero-suppressed Binary Decision Diagram

As described in section 3.2, HMCAPIP improved the performance of the current automated parallel test form assembly based on the maximum clique method. However, its improvement is inadequate because of the high time complexity of IP in the second stage. Furthermore, the HMCAPIP's parallel search performance depends heavily on the performance of the computer's multi-core processors.

To address this issue, this study proposes a new Automated Parallel Test Form Assembly method using Zero-suppressed Binary Decision Diagram (ATA-ZDD). A Zero-suppressed Binary Decision Diagram (ZDD) is a compact and efficient representation of Binary Decision Tree (BDT) that reduces both memory usage and computation time by eliminating redundancy through shared identical subtrees. This compact graph structure works efficiently for enumerating large families of sets such as parallel test forms. For ATA-ZDD, each vertex in the ZDD represents an item. Each edge shows whether the item is included in the parallel test forms or not. This ZDD is expanded through a breadth-first search, sharing vertices that have identical test lengths and identical test information values. However, extremely few vertices have identical test information, leading to insufficient sharing of vertices. Consequently, ATA-ZDD often causes the difficulty of computer

memory overflow. This study examines a two-stage algorithm to address this difficulty. (1) The first stage constructs a ZDD with approximated test information values of shared vertices. Specifically, during the breadth-first search, vertices are shared when the difference in test information values between two vertices at the same depth is smaller than a determined threshold parameter value. Then, the test information values of the shared vertex are averaged from the two vertices. (2) The second stage enumerates paths which satisfy the test constraints from the approximated ZDD. Specifically, paths that satisfy all test information constraint are sampled randomly and enumerated from the constructed ZDD. The exact test information values for each path are then recalculated to ensure that all constraints are satisfied.

4.1 Zero-suppressed Binary Decision Diagram (ZDD)

A ZDD is derived from a Binary Decision Tree (BDT) by the application of two reduction rules that eliminate redundancy. These reduction rules provide the ZDD with the advantages of reducing computation time and memory usage. As a result, the ZDD achieves compactness and efficiency by representing subsets using binary variables, as explained below.

Given a finite set $I = \{x_1, x_2, \ldots, x_n\}$ with ordered binary variables, a family of sets $\mathcal{F} \subseteq 2^I$ exists, where each subset $R \subseteq I$ is a set of binary variables x_i from the finite set I. Each binary variable x_i represents whether $x_i \in R$ or $x_i \notin R$ for each subset R, defined as

$$x_i = \begin{cases} 1 & \text{if } x_i \in R, \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, each subset $R \in \mathcal{F}$ can be represented as a unique combination of binary values for each binary variable x_i in the finite set I.

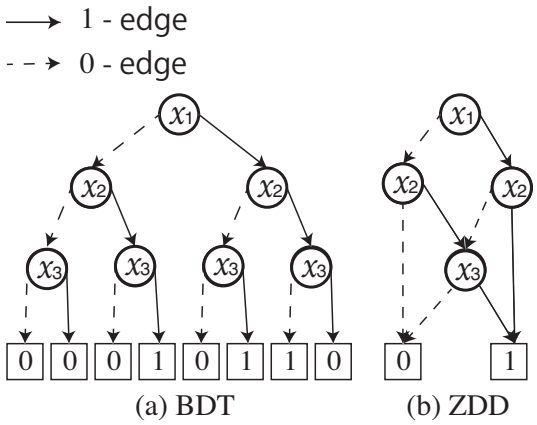


Figure 4.1: BDT and ZDD.

A ZDD is a directed acyclic graph (DAG) that represents the family of sets $\mathcal{F}$ compactly and has two types of terminal vertices: a 1-terminal vertex representing valid subsets in $\mathcal{F}$; and a 0-terminal vertex representing subsets not included in $\mathcal{F}$. Consequently, each path from the root to the 1-terminal vertex corresponds to a unique subset $R \in \mathcal{F}$.

Figures 4.1(a) and 4.1(b) respectively represent examples of a BDT and a ZDD, where the finite set is given as $I = \{x_1, x_2, x_3\}$. In addition, these graph structures have two terminal vertices, which are shown as rectangles in Figure 4.1: 1-terminal and 0-terminal. A path from the root vertex to the 1-terminal vertex in these graph structures corresponds to a unique subset $R \in \mathcal{F}$. Every non-terminal vertex is presented as a circle in Figure 4.1. Each non-terminal vertex is labeled using a binary variable x_i. Moreover, each vertex has two outgoing edges: a 1-edge and a 0-edge. The 1-edge and the 0-edge respectively signify the parent vertex is an element of each subset R, and not. For example, in Figure 4.1, the subset $\{x_1, x_3\}$ is represented in both the BDT and the ZDD by following the 1-edge at x_1 (indicating $x_1 \in R$), the 0-edge at x_2 (indicating $x_2 \notin R$), and the

1-edge at x_3 (indicating $x_3 \in R$) before reaching the 1-terminal vertex. This traversal ensures that x_1 and x_3 are included in the subset R and that x_2 is excluded. Accordingly, in Figure 4.1, both the BDT and the ZDD correspond to the same family of sets $\mathcal{F}$, which consists of $\{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}\}$. This comparison demonstrates that the ZDD can represent the same family of sets with fewer vertices than the BDT can.

The ZDD is obtained by applying the two reduction rules by Minato [41] to the BDT. Specifically, the two reduction rules are defined as the following.

Reduction rule 1 When two non-terminal vertices represent the identical binary variable x_i and their 1-edge and 0-edge point to vertices that represent identical subtrees, these two vertices are shared into a single vertex. Reduction rule 1 eliminates duplicate vertices representing the same subtrees, thereby reducing redundancy in the graph structure.

Reduction rule 2 Vertices with a 1-edge pointing to the 0-terminal vertex are removed because these vertices are not elements of any valid subset R in the family of sets $\mathcal{F}$. Reduction rule 2 simplifies the graph structure by eliminating redundant vertices that cannot engender the 1-terminal vertex.

By applying these two reduction rules, a canonical ZDD representing the family of sets $\mathcal{F}$ is obtained. The canonical ZDD provides a unique and compact representation of the family of sets $\mathcal{F}$, ensuring that redundant vertices and subtrees are fully eliminated.

4.2 Automated Parallel Test Form Assembly using ZDD

To increase the number of parallel test forms, this study proposes the new method of Automated Parallel Test Form Assembly using ZDD (ATA-ZDD). For ATA-ZDD, we define a finite set $I = \{x_1, x_2, \ldots, x_n\}$ with ordered binary variables, where n represents the number of items in the item pool. Each binary variable x_i is defined as presented below.

$$x_i = \begin{cases} 1, & \text{if the item } i \text{ is selected for the parallel test form, and} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

Additionally, we define the family of sets $\mathcal{F} \subseteq 2^I$ as the set of parallel test forms, where each subset $R \in \mathcal{F}$ is defined as a set of binary variables that satisfy the following constraints.

$$\sum_{i=1}^{n} x_i = L, \tag{4.2}$$

$$\forall k \in \{1, 2, \ldots, K\}, I_{\mathrm{LB}}(\theta_k) \leq \sum_{i=1}^{n} I_i(\theta_k) x_i \leq I_{\mathrm{UB}}(\theta_k). \tag{4.3}$$

Equation (4.2) and equation (4.3) respectively represent the test length constraint and the test information constraint. Consequently, each subset R corresponds to a parallel test form that satisfies all test constraints except for the overlapping item constraint.

To efficiently enumerate all such parallel test forms, the first stage in ATA-ZDD compresses a BDT into a ZDD using frontier-based search [58, 46], which is one of the most commonly used methods for ZDD compression. Frontier-based search directly constructs a ZDD using top-down and breadth-first approaches without increasing the computer memory usage and computation time compared to the BDT. Specifically, ATA-ZDD merges two vertices into a single vertex when

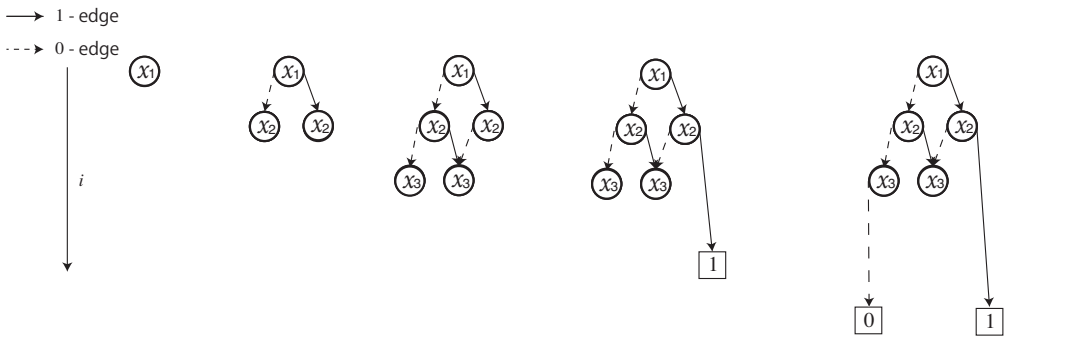


Figure 4.2: Outline of ATA-ZDD.

they have identical test lengths and identical test information values at all test score levels during the top-down and breadth-first approaches.

The time complexity of frontier-based search depends on the number of vertices $N_{\mathrm{V}}(N_{\mathrm{V}} \leq 2^n)$ in the constructed ZDD because each vertex is processed only once during construction. Similarly, the space complexity is also $\mathcal{O}(N_{\mathrm{V}})$, as the method retains only unique vertices on the computer memory.

For frontier-based search, we designate a test length variable as *tl* and a test information array as *tis*($tis = [ti_1, ti_2, \ldots, ti_K]$), where K represents the number of discretized points for the test information function. The test length variable value and each element value in the test information array respectively correspond to $\sum_{i=1}^{n} x_i$ in equation (4.2) and $\sum_{i=1}^{n} I_i(\theta_k)x_i$ in equation (4.3). Frontier-based search calculates the values of these variables for each vertex. Specifically, the main algorithm of ATA-ZDD consists of five procedures, as presented in Figure 4.2.

1. Procedure 1 creates a root vertex. Then, Procedure 1 sets zero to test length variable tl value and zero to each element $tis[k]$ value in the test information array.

2. Procedure 2 creates a 0-child vertex with a 0-edge and a 1-child vertex with a 1-edge. Then, Procedure 2 adds one to the test length variable tl value for 1-child vertices. Subsequently, Procedure 2 adds Fisher information $I_i(\theta_k)$ of depth i to every element $tis[k]$ value in the test information array for 1-child vertices.

3. Procedure 3 merges two vertices into a single vertex when they have the identical test length variable tl value and identical each element $tis[k]$ value in the test information array. Here, the first stage in the proposed method merges two nodes into a single node when the difference in each element $tis[\gamma]$ value in the test information array is less than the threshold value $I_{\rm th}$. Then, each element $tis[\gamma]$ value in the test information array for the merged node is calculated as the average of the corresponding values from the two original nodes. However, since the test information values are approximated by averages, accurate calculations cannot be performed. To address this, the exact test information values are recalculated in the second stage.

4. Procedure 4 connects a 1-edge to the 1-terminal vertex when the test length variable tl value and each element $tis[k]$ value in the test information array satisfy the following constraints, which correspond to equation (4.2) and equation (4.3):

   Condition 1. $L = tl$,

   Condition 2. $\forall k \in \{1, 2, \ldots, K\}, I_{\rm LB}(\theta_k) \leq tis[k] \leq I_{\rm UB}(\theta_k)$.

5. In Procedure 5, a 1-edge and a 0-edge are connected to the 0-terminal vertex when one of the following constraints is satisfied because the test constraints are not satisfied:

   Condition 1. $L < tl$,

   Condition 2. $\exists k \in \{1, 2, \ldots, K\}$ s.t. $I_{\rm UB}(\theta_k) < tis[k]$,

Condition 3. $\exists k \in \{1, 2, \ldots, K\}$ s.t. $L = tl$ and $tis[k] < I_{\mathrm{LB}}(\theta_k)$.

6. In Procedure 6, the ATA-ZDD executes Procedures 2–5 sequentially for all items in the finite set I, resulting in a ZDD representing the family of parallel test forms $\mathcal{F}$. Then, the two reduction rules are applied to the constructed ZDD to remove redundant vertices and identical subtrees because frontier-based search does not guarantee a canonical graph structure [58]. Consequently, a canonical ZDD representing the family of parallel test forms $\mathcal{F}$ is obtained by application of the two reduction rules.

The first stage in ATA-ZDD requires the following inputs.

- Tuning parameter I_{th} represents the threshold value in Procedure 3.
- Finite set I represents a set of items in an item pool.
- Constant value parameter L denotes the test length.
- Constant value parameter n stands for the number of items in the item pool.
- Constant value parameter K represents the number of discretized points for the test information function.
- Constant value parameters $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively denote the lower and upper bounds of the test information function at test score level θ_k.

Algorithm 4 provides a description of the first stage in ATA-ZDD according to an earlier report of the relevant literature [46]. The output of Algorithm 4 is the set $\mathcal{F}$, which represents parallel test forms that satisfy all test constraints without overlapping item constraint.

Algorithm 4: ATA-ZDD (first stage)

1: **procedure** ATA-ZDD (FIRST STAGE)
2: **Input:** $I_{\mathrm{th}}, I, n, L, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k)$
3: **Output:** $\mathcal{F}$
4: Create a new vertex v_{root}
▷ root vertex
5: $v_{\mathrm{root}}.state.tl \leftarrow 0$
6: $v_{\mathrm{root}}.state.tis \leftarrow Array[K]$
▷ Declare an array of size K
7: **for** $k \leftarrow 1$ **to** K **do**
8: $v_{\mathrm{root}}.state.tis[k] \leftarrow 0$
9: **end for**
10: $V_1 \leftarrow \{v_{\mathrm{root}}\}$
▷ V_i is a set of vertices of depth i
11: **for** $i \leftarrow 2$ **to** n **do**
12: $V_i \leftarrow \emptyset$
13: **end for**
14: $V_{n+1} \leftarrow \{$0-terminal vertex, 1-terminal vertex$\}$
15: **for** $i \leftarrow 1$ **to** n **do**
16: **for each** $v \in V_i$ **do**
17: **for each** $x_i \in \{0, 1\}$ **do**
▷ 0-edge, 1-edge
18: $\{i', state'\} \leftarrow$ Child$(i, L, v.state, x_i)$
▷ i' is the depth of the child vertex. $state'$ is tl and tis of the child vertex.
19: $v' \leftarrow$create a new vertex
▷ child vertex
20: **if** $\{i', state'\}$ is $\{n+1, 0\}$ **then**
21: $v' \leftarrow$ 0-terminal vertex
22: **else if** $\{i', state'\}$ is $\{n+1, 1\}$ **then**
23: $v' \leftarrow$ 1-terminal vertex
24: **else**

```
                    v'.state ← state'
                    share_vertex ← False
                    for each w ∈ V_{i+1} do
                        if v'.state.tl = w.state.tl then
                            for k ← 1 to K do
                                if I_th ≤ |v'.state.tis[k] − w.state.tis[k]| then
                                    next w
                                end if
                            end for
                            UpdateState(v', w)
                            v' ← w
  ▷ share vertex
                            share_vertex ← True
                            break
                        end if
                    end for each
                    if share_vertex is False then
                        V_{i+1} ← V_{i+1} ∪ v'
                    end if
                end if
                v.child[x_i] ← v'
            end for each
        end for each
    end for
    F ← ReductionRule(v_root)
  ▷ ReductionRule applies the two reduction rules to the constructed ZDD.
    Output F
end procedure
procedure CHILD(i, L, state, x_i)
    if x_i = 1 then
        state'.tl ← state.tl + 1
```

```
54:         for k ← 1 to K do
55:             state′.tis[k] ← state.tis[k] + I_i(θ_k)
    ▷ I_i(θ_k) in eq(2.3)
56:         end for
57:     end if
58:     if state′.tl = L then
59:         for k ← 1 to K do
60:             if not I_LB(θ_k) < state′.tis[k] < I_UB(θ_k) then
61:                 return {n + 1, 0}
    ▷ 0-terminal vertex
62:             end if
63:         end for
64:         Return {n + 1, 1}
    ▷ 1-terminal vertex
65:     end if
66:     if state′.tl + n − i < L then
67:         for k ← 1 to K do
68:             if I_UB(θ_k) < state′.tis[k] then
69:                 Return {n + 1, 0}
    ▷ 0-terminal vertex
70:             end if
71:         end for
72:     end if
73:     Return {i + 1, state′}
74: end procedure
75: procedure UPDATESTATE(v′, w)
76:     for k ← 1 to K do
77:         w.state.tis[k] ← (v′.state.tis[k] + w.state.tis[k])/2
78:     end for
79: end procedure
```

In Algorithm 4, each element value in the test information array of a shared vertex is approximated by the average of the test information values of two vertices. Consequently, paths that include the shared vertex might not exactly satisfy test information constraint in equation (4.3). Additionally, the first stage in ATA-ZDD is unable to control overlapping items. To overcome these limitations, the second stage in ATA-ZDD enumerates paths that exactly satisfy the test information and overlapping item constraints. Specifically, the second stage in ATA-ZDD sequentially recalculates the exact test information value for each path in the constructed canonical ZDD without the value of the approximated shared vertex using random sampling [41] from the constructed canonical ZDD. As a result, ATA-ZDD enumerates parallel test forms that exactly satisfy all test constraints.

For the second stage in ATA-ZDD, we define parallel test forms as the families of sets $\mathcal{P} \subseteq \mathcal{F}$ that satisfy all test constraints. The second stage of ATA-ZDD searches parallel test forms using the following procedures.

1. Procedure 1 sets $\mathcal{P}$ to the empty set.
2. Procedure 2 searches a subset R from the constructed canonical ZDD $\mathcal{F}$ using random sampling [41].
3. Procedure 3 proceeds to Procedure 4 when the binary variables of the sampled subset R satisfy the test information constraint in equation (4.3); otherwise, it returns to Procedure 2.
4. In Procedure 4, the sampled subset R is added to the family of sets $\mathcal{P}$ ($\mathcal{P} \leftarrow \mathcal{P} \cup \{R\}$) when the binary variables of the sampled subset R satisfy the following overlapping item constraint; otherwise, it returns to Procedure 2.

$$\forall P \in \mathcal{P}, \sum_{i \in I} x_i^R x_i^P \leq OC, \tag{4.4}$$

where x_i^R denotes a binary variable x_i in the subset R and x_i^P represents a binary variable x_i in the subset P.

5. ATA-ZDD repeats Procedures 2–4 until a determined computation time is reached.

The second stage in ATA-ZDD requires the following inputs.

- Constant value time LT stands for the algorithm's total computation time limit.
- Finite set I represents a set of items in an item pool.
- Constant value parameter n stands for the number of items in the item pool.
- Constant value parameter K represents the number of discretized points for the test information function.
- Constant value parameters $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ respectively denote the lower and upper bounds of the test information function at test score level θ_k.
- Constant value parameter OC is the maximum number of common items between any pair of parallel test forms.

Algorithm 5 provides a description of the second stage in ATA-ZDD. The output of Algorithm 5 is the family of sets $\mathcal{P}$, which represents parallel test forms that satisfy all test constraints.

Algorithm 5: ATA-ZDD (second stage)

1: **procedure** ATA-ZDD (SECOND STAGE)
2: **Input:** $I_{\mathrm{th}}, I, n, K, I_{\mathrm{LB}}(\theta_k), I_{\mathrm{UB}}(\theta_k), OC$
3: **Output:** $\mathcal{P}$

4: $st \leftarrow now()$

▷ $now()$ retrieves the current timestamp to track the elapsed computation time.

5: $\mathcal{F} \leftarrow$ATA-ZDD (first stage) $(I_{\text{th}}, I, n, K, I_{\text{LB}}(\theta_k), I_{\text{UB}}(\theta_k))$

6: $\mathcal{P} \leftarrow \emptyset$

7: **while** $(now() - st) < LT$ **do**

8: $R \leftarrow RandomSampling(\mathcal{F})$

▷ random sampling subset R from the constructed canonical ZDD $\mathcal{F}$

9: **for** $k \leftarrow 1$ **to** K **do**

10: $TI(\theta_k) \leftarrow TestInfo(R, n, \theta_k)$

▷ TestInfo calculates the test information $\sum_{i=1}^{n} I_i(\theta_k)x_i, x_i \in R$ at test score level θ_k from the binary variables in subset R.

11: **if** $TI(\theta_k) < I_{\text{LB}}(\theta_k)$ *or* $I_{\text{UB}}(\theta_k) < TI(\theta_k)$ **then**

12: Next while

13: **end if**

14: **end for**

15: **for each** $P \in \mathcal{P}$ **do**

16: **if** $\sum_{i \in I} x_i^R x_i^P > OC$ **then**

17: Next while

18: **end if**

19: **end for**

20: $\mathcal{P} \leftarrow \mathcal{P} \cup \{R\}$

21: **end while**

22: **Output** $\mathcal{P}$

23: **end procedure**

Consequently, the two-stage algorithm ATA-ZDD enumerates parallel test forms that exactly satisfy all test constraints. To evaluate the efficiency of the second stage of ATA-ZDD, the time and space complexities of a single iteration through Procedure 2 to Procedure 4 are

detailed below.

In Procedure 2, the time and space complexities of random sampling from a ZDD are $\mathcal{O}(n)$, which depends on the input size n because the process involves traversing from the root vertex to the 1-terminal vertex. In Procedure 3, the total test information $\sum_{i=1}^{n} I_i(\theta_k)x_i$ for the items included in the sampled subset R is calculated at K discretized points, as defined in equation (4.3). As a result, The time and space complexities of Procedure 3 respectively are $\mathcal{O}(n \cdot K)$ and $\mathcal{O}(K)$, which depends on the input sizes n and K. In Procedure 4, all subsets $S \subseteq \mathcal{P}$ are verified to satisfy the overlapping item constraint with the sampled subset R. Consequently, the time and space complexities of Procedure 4 is $\mathcal{O}(|\mathcal{P}|)$.

Finally, the time and space complexities of the second stage of ATA-ZDD respectively are $\mathcal{O}(\kappa \cdot (n \cdot K + |\mathcal{P}|))$ and $\mathcal{O}(|\mathcal{P}|)$ when $\kappa(|\mathcal{P}| \leq \kappa)$ iterations are performed within the computation time limit LT.

4.3 ATA-ZDD Experiments

4.3.1 Threshold Parameter Effectiveness

This experiment is aimed ascertaining the value of threshold parameter I_{th} to obtain the optimal value, which maximizes the number of parallel test forms. Specifically, this experiment was conducted to compare the performances of ATA-ZDD by modifying the values of the threshold parameter I_{th} using both simulated and actual item pools. The threshold parameter I_{th} value of ATA-ZDD is decreased from 0.45 to 0.01 in decrements of 0.01 to maximize the number of parallel test forms for each item pool.

Items in the simulated item pools have discrimination parameters

Table 4.1: Test information constraints for ATA-ZDD experimentation

$I_{\mathrm{LB}}(\theta_k)/I_{\mathrm{UB}}(\theta_k)$				
$\theta_1 = -2.0$	$\theta_2 = -1.0$	$\theta_3 = 0.0$	$\theta_4 = 1.0$	$\theta_5 = 2.0$
8.0/9.6	12.8/14.4	12.8/14.4	12.8/14.4	8.0/9.6

and difficulty parameters based on the IRT. This study generated the item discrimination parameters as $\log_2 a \sim N(0, 1^2)$ and the item difficulty parameters as $b \sim N(0, 1^2)$. Table 3.5 provides the specifics of the actual item pool. Additionally, this study sets the three test constraints as presented below.

1. Test length L is 100 items.
2. Test information constraints are given by the lower and upper bounds of the test information function, as shown in Table 4.1.
3. Overlapping items are not controlled for this experiment.

Here, Ishii et al. [30] used 25 items as the test length in their experiment. However, large-scale standardized examinations such as IT Passport, ACT, and SAT typically have total test lengths exceeding 100 items. Accordingly, in the experiments described hereinafter, we set the test length as 100 items to reflect the constraints imposed by large-scale examinations. Additionally, the upper and lower bounds of the test information constraints were also set to four times the values used for an experiment conducted by Ishii et al. [30], consistent with adjustment of the test length. This experiment was conducted using a computer equipped with a Ryzen 9 5950X 3.40 GHz CPU (AMD Inc.), 128 GB of RAM, and running a 64-bit Linux (Ubuntu) operating system.

Table 4.2 presents performance indicators of ATA-ZDD by modifying the value of I_{th}. In the table, "No. tests" stands for the number

of parallel test forms. Also, "RC" represents the compression rate, defined as

$$\mathrm{RC} = \frac{N_{\mathrm{ZDD}}}{2^{n+1} - 1},$$

where N_{ZDD} is the number of vertices in the canonical ZDD obtained after applying the two reduction rules to the constructed ZDD in Procedure 6 of the first stage. Also, n represents the maximum depth of a full binary decision tree. Additionally, "Time" represents the calculation time (minute) for the ZDD construction. Here, for threshold values below 0.27, all conditions caused computer memory overflow. Accordingly, they are listed in the table as "below 0.27".

Table 4.2: Performance of ATA-ZDD by modifying the threshold parameter I_{th} value

	Item pool size = 1000			Item pool size = 2000			Item pool size = 978		
I_{th}	No. tests	RC	Time	No. tests	RC	Time	No. tests	RC	Time
below 0.27	Memory Over			Memory Over			Memory Over		
0.28	Memory Over			Memory Over			4.65×10^{127}	3.77×10^{-288}	1202
0.29	Memory Over			Memory Over			4.21×10^{126}	2.37×10^{-288}	220
0.30	Memory Over			Memory Over			3.01×10^{128}	1.57×10^{-288}	193
0.31	Memory Over			Memory Over			1.34×10^{130}	1.18×10^{-288}	142
0.32	Memory Over			Memory Over			9.11×10^{123}	8.04×10^{-289}	99
0.33	Memory Over			Memory Over			1.56×10^{123}	8.22×10^{-289}	78
0.34	1.67×10^{136}	1.20×10^{-293}	1122	Memory Over			5.29×10^{120}	5.50×10^{-289}	61
0.35	1.43×10^{136}	8.33×10^{-294}	654	Memory Over			7.82×10^{124}	3.73×10^{-289}	40
0.36	1.38×10^{136}	6.84×10^{-294}	338	Memory Over			3.11×10^{117}	2.05×10^{-289}	39
0.37	8.12×10^{135}	6.00×10^{-294}	258	Memory Over			5.33×10^{120}	3.24×10^{-289}	24
0.38	3.20×10^{136}	4.41×10^{-294}	132	2.11×10^{167}	2.16×10^{-594}	1361	1.17×10^{129}	3.08×10^{-289}	11
0.39	1.03×10^{136}	3.25×10^{-294}	78	6.34×10^{166}	1.85×10^{-594}	959	1.07×10^{128}	2.65×10^{-289}	5
0.40	9.10×10^{135}	2.51×10^{-294}	49	3.66×10^{167}	1.44×10^{-594}	550	9.87×10^{123}	1.57×10^{-289}	5
0.41	1.51×10^{136}	1.73×10^{-294}	25	2.56×10^{167}	1.21×10^{-594}	330	2.73×10^{119}	1.19×10^{-289}	5
0.42	1.93×10^{135}	1.37×10^{-294}	17	2.11×10^{167}	8.95×10^{-595}	260	1.75×10^{121}	9.59×10^{-290}	3
0.43	1.51×10^{136}	1.08×10^{-294}	12	1.05×10^{167}	7.99×10^{-595}	210	1.31×10^{120}	8.53×10^{-290}	3
0.44	9.54×10^{135}	8.54×10^{-295}	8	4.18×10^{167}	6.13×10^{-595}	133	8.62×10^{116}	8.21×10^{-290}	3
0.45	3.48×10^{135}	7.73×10^{-295}	7	1.13×10^{167}	4.77×10^{-595}	85	2.19×10^{119}	4.15×10^{-290}	2

Table 4.3: Determined values of threshold parameter I_{th}

Item pool size	I_{th}
1000	0.38
2000	0.44
978 (actual)	0.31

The table shows that varying the threshold parameter I_{th} causes tradeoffs involving "No. tests" with both "RC" and "Time" in the ZDD. Particularly, smaller values of the threshold parameter I_{th} tend to increase the number of parallel test forms, but ATA-ZDD might cause computer memory overflow. By contrast, larger I_{th} values reduce the calculation time and the number of vertices by increasing the number of shared vertices. However, this also results in fewer parallel test forms. Accordingly, as presented in Table 4.3, the threshold parameter I_{th} was determined to maximize the number of parallel test forms for each item pool. In practice, I_{th} depends on various factors such as the item pool, the test information constraint, and the test length. A realistic approach is to start with a large value and then to decrease it gradually. This approach is useful to ascertain I_{th} to maximize the number of parallel test forms within the limits imposed by available computation time and memory resources.

To analyze the ZDD construction process, Table 4.4 presents the number of pruned vertices and the effects of the reduction rules. In the table, "No. pruned vertices" represents the number of vertices connected to the 0-terminal vertex during Procedure 5. "Reduction rules" denotes the compression rate of the constructed canonical ZDD, defined as

$$\frac{N_{\mathrm{ZDD}}}{N_{\mathrm{pre}}},$$

where N_{pre} represents the number of vertices in the constructed ZDD

before applying the two reduction rules in Procedure 6 of the first stage.

The table shows that a smaller value of threshold parameter I_{th} increases the number of pruned vertices because of the fewer shared vertices and additional branches. However, the compression ratio remains consistent irrespective of the threshold parameter I_{th} value.

Table 4.4: Effects of modifying the threshold parameter I_{th} on pruned vertices and reduction rules in the ZDD

	Item pool size = 1000		Item pool size = 2000		Item pool size = 978	
I_{th}	No. pruned vertices	Reduction rule	No. pruned vertices	Reduction rule	No. pruned vertices	Reduction rule
below 0.27	Memory Over		Memory Over		Memory Over	
0.28	Memory Over		Memory Over		5892801	0.08
0.29	Memory Over		Memory Over		5389210	0.01
0.30	Memory Over		Memory Over		4868817	0.06
0.31	Memory Over		Memory Over		4100665	0.06
0.32	Memory Over		Memory Over		3413287	0.04
0.33	Memory Over		Memory Over		3034305	0.05
0.34	9239892	0.79	Memory Over		2692426	0.04
0.35	8234729	0.78	Memory Over		2070770	0.03
0.36	7537993	0.79	Memory Over		1939610	0.02
0.37	6636419	0.78	Memory Over		1865923	0.04
0.38	4852511	0.80	17023890	0.64	1179711	0.06
0.39	3634472	0.79	15892039	0.63	719357	0.00
0.40	2794332	0.79	14591090	0.63	857468	0.04
0.41	1921650	0.79	14023900	0.63	827739	0.03
0.42	1523517	0.78	13716345	0.63	612405	0.03
0.43	1196380	0.78	12961675	0.63	547942	0.00
0.44	956115	0.78	10889604	0.64	547434	0.03
0.45	901037	0.77	8359099	0.64	435407	0.02

4.3.2 Comparison of ATA-ZDD to earlier methods

This experiment demonstrates the benefits of ATA-ZDD by comparing the number of parallel test forms with those assembled using earlier methods, specifically BST (BST in subsection 2.2.1 and HMCAPIP in section 3.2) with the item pools described in the preceding subsection 4.3.1. The test length and the test information constraints are set similarly, as shown in the preceding subsection 4.3.1. Additional test constraints are set as presented below.

1. The maximum number of common items OC is increased from 4 to 40 in increments of 4, corresponding to 4%–40% of the test length L, with increments of 4%.
2. The time limitation for all methods is 24 hr.

The parameter values for HMCAPIP were set based on the explanation provided in section 3.2. For this study, we applied CPLEX 12.9 [54] to the IP for HMCAPIP and BST.

Table 4.5 presents the numbers of assembled parallel test forms produced using ATA-ZDD and using the earlier methods by modifying the item pool sizes and overlapping item constraints.

Table 4.5: Numbers of assembled parallel test forms in 24 hr

Item Pool Size	OC	BST	HMCAPIP	ATA-ZDD
1000	4	5	**19**	4
	8	5	**24**	7
	12	5	**43**	28
	16	5	200	**220**
	20	5	2198	**3654**
	24	5	4240	**57996**
	28	5	5658	**121255**
	32	5	6189	**124400**
	36	5	6235	**124449**
	40	5	6514	**125192**
2000	4	6	**28**	10
	8	6	**103**	77
	12	6	1802	**1879**
	16	6	10163	**34422**
	20	6	11122	**52271**
	24	6	11798	**53311**
	28	6	12121	**54126**
	32	6	12333	**54392**
	36	6	12365	**54446**
	40	6	12412	**54928**
978 (actual)	4	4	**20**	3
	8	4	**26**	6
	12	4	**59**	21
	16	4	403	**473**
	20	4	7038	**8783**
	24	4	97423	**147086**
	28	4	99852	**1545602**
	32	4	100021	**1548327**
	36	4	100222	**1548498**
	40	4	100314	**1548902**

Table 4.6: Random sampling iterations conducted by ATA-ZDD for each item pool

Item pool size	Random sampling iterations	No. test (first stage)
1000	1,640,142,109	3.20×10^{136}
2000	632,367,331	4.18×10^{167}

As presented in the table, ATA-ZDD can assemble more parallel test forms than the earlier methods can when the number of parallel test forms exceeds 200. It is noteworthy that, under specific test constraints, when $OC \geq 28$, ATA-ZDD can assemble more than 1,500,000 parallel test forms within 24 hr (precisely 23 hr) from the actual item pool with 978 items, whereas HMCAPIP is limited to assembling only 100,000 parallel test forms. Accordingly, ATA-ZDD can assemble the number of parallel test forms which are sufficient for practical use in examinations with more than 1,000,000 examinees annually, such as the ACT. In fact, ATA-ZDD enables the assessment to be administered to these examinees from any location, at any time, and on the same scale.

By contrast, when $OC \geq 24$, ATA-ZDD with the simulated item pool size of 1,000 can assemble more parallel test forms than ATA-ZDD can with the pool size of 2,000. To elucidate the reason this result was obtained, we counted the random sampling iterations conducted by ATA-ZDD within the time limit, as presented in Table 4.6, for each simulated item pool when $OC = 24$. In the table, "No. test (first stage)" represents the number of parallel test forms assembled in the first stage of ATA-ZDD.

In the table, ATA-ZDD with pool size of 1000 performs more random sampling iterations than with the pool size of 2000 within the limited computation time. By contrast, ATA-ZDD with pool size of 2000

assembles more parallel test forms than with the pool size of 1000 in the first stage. Accordingly, when the computation time is extended, ATA-ZDD with pool size of 2000 might assemble more parallel test forms than with pool size of 1000 in the second stage. To validate this hypothesis, future studies must examine whether extension of the computation time allows ATA-ZDD with a pool size of 2000 to assemble more parallel test forms than with a pool size of 1000.

By contrast, in Table 4.5, when OC is small, the difference in the number of parallel test forms between ATA-ZDD and HMCAPIP is also small. The results demonstrate that the number of parallel test forms converges to the maximum possible number when a tight overlapping item constraint restricts the exact maximum number of parallel test forms. With this overlapping item constraint, ATA-ZDD cannot assemble more parallel test forms than HMCAPIP can because ATA-ZDD cannot exactly guarantee that all paths satisfy the test constraints to approximate by the average when vertices are shared. Under this constraint, ATA-ZDD cannot assemble more parallel test forms than HMCAPIP can because ATA-ZDD averages test information values when sharing vertices, which makes it unable to guarantee that all paths satisfy the test constraint. To evaluate the accuracy of this approximation, we calculate $P_{\text{valid,info}}$, which represents the rate of valid paths that satisfy the test information constraint as

$$P_{\text{valid,info}} = \frac{N_{\text{valid,info}}}{N_{\text{sampled}}}.$$

Here, $N_{\text{valid,info}}$ represents the number of valid paths which satisfy the test information constraint. N_{sampled} stands for the total number of random sampling iterations.

Table 4.7 presents the rate of valid paths $P_{\text{valid,info}}$ that satisfy the test information constraint. For the actual item pool, 51% of the paths found from ATA-ZDD through random sampling satisfy the test information constraint. In fact, because of this high rate of valid paths

Table 4.7: Rates of valid paths that satisfy the test information constraint

Item pool size	$P_{\text{valid,info}}$
1000	0.07
2000	0.07
978 (actual)	0.51

$P_{\text{valid,info}}$, ATA-ZDD enables assembly of over 1,500,000 parallel test forms within 24 hr from the actual item pool. In contrast, only 7% of the paths found from ATA-ZDD through random sampling in the simulated item pools satisfy the test information constraint. Despite this markedly lower rate of valid paths $P_{\text{valid,info}}$, ATA-ZDD can assemble more parallel test forms than HMCAPIP can.

Furthermore, the second stage of ATA-ZDD assembles parallel test forms by selecting paths that satisfy the overlapping item constraint from those which satisfy the test information constraints. To assess the rate of such paths, we calculate $P_{\text{valid,oc}}$, which represents the rate of valid paths that satisfy overlapping item constraint as

$$P_{\text{valid,oc}} = \frac{N_{\text{valid,oc}}}{N_{\text{valid,info}}}.$$

Here, $N_{\text{valid,oc}}$ represents the number of valid paths that satisfy the test information constraint and the overlapping item constraint.

Table 4.8 presents the rates of valid paths $P_{\text{valid,oc}}$ that satisfy the overlapping item constraint. As shown in the table, as OC increases for each item pool, the rate of paths $P_{\text{valid,oc}}$ satisfying the overlapping item constraint increases because more overlap between test items is allowed. Consequently, as shown in the preceding Table 4.5, a larger OC allows for assembling more parallel test forms. However, in Table 4.8, when $OC \geq 28$, $P_{\text{valid,oc}}$ shows nearly equal values. Similarly, as

shown in Table 4.5, the number of parallel test forms becomes almost identical.

These experimentally obtained results demonstrate the effectiveness of ATA-ZDD. By sharing vertices when the difference in test information values is below a threshold parameter value and by using averaged values to maximize the number of parallel test forms within a computer memory limit, ATA-ZDD constructs a ZDD efficiently. Then, ATA-ZDD randomly samples paths that satisfy all test constraints by recalculating their exact test information and the overlapping item constraint. As a result, ATA-ZDD is capable of assembling more parallel test forms than the earlier methods do.

Table 4.8: Rates of valid paths satisfying the overlapping item constraint.

Item Pool Size	OC	$P_{\text{valid,oc}}$
1000	4	3.46×10^{-8}
	8	6.06×10^{-8}
	12	2.42×10^{-7}
	16	1.90×10^{-6}
	20	3.16×10^{-5}
	24	5.02×10^{-4}
	28	1.05×10^{-3}
	32	1.08×10^{-3}
	36	1.08×10^{-3}
	40	1.08×10^{-3}
2000	4	2.26×10^{-7}
	8	1.74×10^{-6}
	12	4.24×10^{-5}
	16	7.78×10^{-4}
	20	1.18×10^{-3}
	24	1.20×10^{-3}
	28	1.22×10^{-3}
	32	1.23×10^{-3}
	36	1.23×10^{-3}
	40	1.24×10^{-3}
978 (actual)	4	4.61×10^{-9}
	8	9.23×10^{-9}
	12	3.23×10^{-8}
	16	7.27×10^{-7}
	20	1.35×10^{-5}
	24	2.26×10^{-4}
	28	2.38×10^{-3}
	32	2.38×10^{-3}
	36	2.38×10^{-3}
	40	2.38×10^{-3}

Chapter 5

Conclusions

For e-testing, assembling parallel test forms of a quantity exceeding the number of examinees is extremely important. The state-of-the-art method, Random Maximum Clique Algorithm (RndMCA), is limited by its high space complexity $\mathcal{O}(\binom{n}{L}+SS^2)$, which restricts the maximum number of parallel test forms to 100,000. That number of parallel test forms is insufficient for large-scale examinations requiring over 1,000,000 forms. For instance, over 200,000 examinees take the IT Passport examination in Japan annually. More than 1,300,000 examinees annually take the ACT in the United States. To achieve this objective, this study proposed three automated parallel test form assembly methods based on discrete algorithms capable of assembling more than 1,000,000 parallel test forms.

First, to mitigate the high space complexity of RndMCA, this study proposed a new method: Random Integer Programming for Maximum Clique Algorithm (RIPMCA). The main idea of RIPMCA is that it dynamically sought a vertex connected to all vertices in a maximum clique using IP. RIPMCA has lower space complexity to $\mathcal{O}(|C|)$ than the space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$ of RndMCA. Accordingly, RIPMCA allowed more parallel test forms to be assembled within memory resource constraints. However, the high time complexity $\mathcal{O}(|C|^2 \cdot 2^n)$ of RIPMCA constrained improvement in the number of assembled parallel test forms.

Second, to address the high time complexity of RIPMCA, this study proposed a new method: Hybrid Maximum Clique Algorithm using Parallel Integer Programming (HMCAPIP). In HMCAPIP, the first stage used the RndMCA with constant time but high space complexity $\mathcal{O}(\binom{n}{L} + SS^2)$, which assembled parallel test forms until a computer memory limit were reached. The second stage switched to RIPMCA with low space complexity $\mathcal{O}(|C|)$ but high time complexity $\mathcal{O}(|C|^2 \cdot 2^n)$ of IP to assemble parallel test forms further. Furthermore, this study parallelized the second stage of HMCAPIP to reduce the computation time. Specifically, in the second stage of HMCAPIP, IPs are used in parallel to search for vertices that are connected to all vertices in the current clique. Afterward, the second stage identifies the maximum clique from the searched vertices, which is then combined with the current clique. HMCAPIP assembled 1.5–2.7 times more parallel test forms than earlier methods did. Particularly under specific test constraints from the actual item pool with 978 items, HMCAPIP assembled 200,000 parallel test forms, exceeding the annual number of examinees for the IT Passport examination. Nevertheless, because of the high space complexity of IP, HMCAPIP is inadequate for assembly of more than 1,000,000 parallel test forms.

Third, to increase more parallel test forms, this study proposed a new method: Automated parallel Test Form Assembly using Zero-suppressed binary Decision Diagram (ATA-ZDD). Each vertex represented an item. Each edge determined whether to include the item in the parallel test form. Specifically, ATA-ZDD was expanded via breadth-first search and two vertices that have identical numbers of test lengths and the identical test information values. However, extremely few vertices have identical test information. Accordingly, ATA-ZDD caused computer memory overflow. To mitigate this difficulty, this study examined a two-stage algorithm. (1) During the breadth-first search, vertices are shared when the difference in the test information values between the two vertices at the same depth is less than a thresh-

old. The shared vertex's test information values are then averaged from the two vertices. The threshold parameter values are determined to maximize the number of paths (parallel test forms) to as many as possible within a computer memory limit. (2) Paths satisfying the test information constraint are sought from the constructed ZDD. Then the exact accuracy is recalculated. ATA-ZDD assembles up to 1.5 million parallel test forms in one day, far surpassing earlier methods.

These results were obtained under specific test constraints using one actual item pool. By contrast, the simulated item pools were generated based on idealized parameter distribution assumptions commonly used in earlier studies, which were expected to allow ATA-ZDD to assemble more parallel test forms. Contrary to this expectation, as demonstrated by the experiment of subsection 4.3.2, ATA-ZDD using the actual item pool assembled more parallel test forms than methods using the simulated item pools did.

To analyze this unexpected result, Figure 5.1 presents the averages and standard deviations of test information values for 100 randomly sampled items (the test length constraint specified in the experiment of subsection 4.3.2) from each item pool 10,000 times. Let $I = \{1, 2, \ldots, n\}$ represent the set of items in each item pool, and let $R_t \subseteq I$ denote the set of 100 items randomly sampled at the t-th iteration ($t = 1, 2, \ldots, 10,000$). For a given test score θ_k, where $k = -2.00, -1.75, \ldots, 2.00$, the average test information value is calculated as:

$$\mu(\theta_k) = \frac{1}{10,000} \sum_{t=1}^{10,000} \sum_{i \in R_t} I_i(\theta_k),$$

where $I_i(\theta_k)$ is defined in equation (2.3). Similarly, the standard devi-

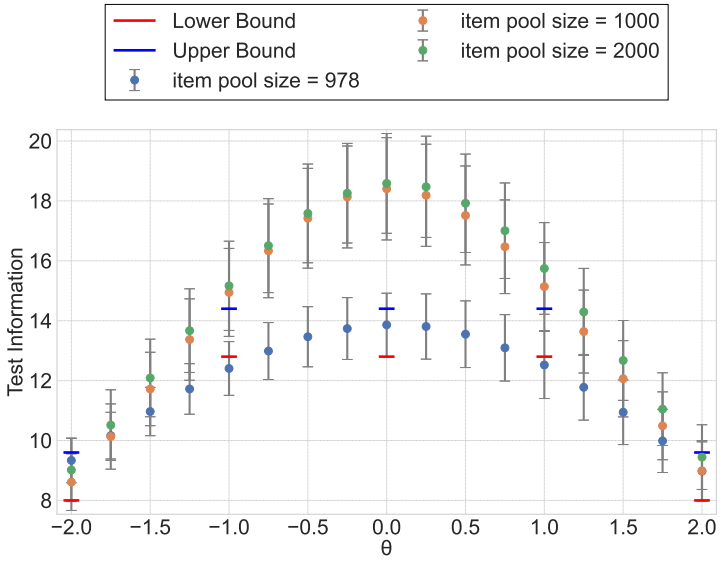


Figure 5.1: Averages and standard deviations of test information values for 100 randomly sampled items for each item pool.

ation for each test score θ_k is given by:

$$\sigma(\theta_k) = \sqrt{\frac{1}{10,000} \sum_{t=1}^{10,000} \left(\sum_{i \in R_t} I_i(\theta_k) - \mu(\theta_k) \right)^2}.$$

In this figure, the horizontal axis represents the test scores θ, and the vertical axis represents the test information values. The points in each plot indicate the average test information values $\mu(\theta_k)$, with error bars representing their standard deviations $\sigma(\theta_k)$, for each item pool. The figure also includes the upper and lower bounds of the test information constraints used in the experiment of subsection 4.3.2. Figure 5.1 shows that the item pool size of 978 represents the actual item pool. The item pool sizes of 1000 and 2000 represent simulated item pools.

The figure shows that the upper and lower bounds of the test information constraints are close to the average and standard deviation of test information values for 100 items sampled randomly from the ac-

tual item pool. Conversely, for the simulated item pools, the averages and standard deviations of test information values for 100 randomly sampled items around $\theta = 0$ deviate from the upper and lower bounds of the test constraints. From this deviation, one can infer that the test information constraints might reduce the number of assembled parallel test forms for the simulated item pools.

Accordingly, future work should be aimed develop systematic methods for determining the upper and lower bounds of test information constraints. For example, the test information constraints $I_{\mathrm{LB}}(\theta_k)$ and $I_{\mathrm{UB}}(\theta_k)$ could be defined as follows:

$$I_{\mathrm{LB}}(\theta_k) = \mu(\theta_k), \quad I_{\mathrm{UB}}(\theta_k) = \mu(\theta_k) + \sigma(\theta_k).$$

Using this test information constraints, when $OC = 28$, ATA-ZDD successfully assembled approximately 950,000 parallel test forms from simulated item pools with 1000 items, although HMCAPIP assembled only 130,000 parallel test forms. This suggests that adjusting the test information constraints based on the characteristics of the item pool could further increase the number of parallel test forms.

By contrast, ATA-ZDD does not guarantee exactly all enumerations with the satisfaction of test constraints: each element of the test information array is approximated by the mean when vertices are shared. Consequently, the proposed method still has room for improvement. To resolve this difficulty, we expect an exact solution algorithm using ZDD with depth-first search such as [59] in future work.

Additionally, this study only specifically examines automated parallel test form assembly considering only constraints related to test length, test information, and overlapping items. However, in real-world scenarios, other constraints are necessary for examinations. For example, the proposed method does not impose adversarial item or categorical item constraints in parallel test forms. Adversarial items

are those with related content that must not appear simultaneously in the same parallel test form, although categorical constraints limit the number of items selected from each category, such as vocabulary or reading comprehension in language examinations. Particularly, ATA-ZDDs might cause computer memory overflow ZDD construction in the first stage because the number of shared vertices decreases as the number of constraints increases. Therefore, HMCAPIP might be more efficient than ATA-ZDD when test organizations require many test constraints.

Furthermore, the proposed methods do not control how often each item is used in the assembled test forms. This lack of control of item exposure frequency leads to a biased distribution of item usage: a difficulty known as item exposure bias [60]. The item exposure bias adversely affects the reliability of both the items and the measurement accuracy [60]. To address this issue, future work could incorporate a probabilistic approach to item exposure control, such as the probabilistic eligibility model suggested by van der Linden [61] for computerized adaptive testing (CAT). Similar challenges related to item exposure bias are also observed in Computerized Adaptive Testing (CAT). To mitigate this issue in CAT, a two-stage stratified adaptive testing approach [62, 63, 64] has been proposed, which leverages automated test assembly techniques to construct parallel test forms. Inspired by this approach, the proposed methods could potentially be adapted to reduce item exposure bias in two-stage stratified adaptive testing. Exploring this possibility remains an avenue for future research.

Acknowledgments

This dissertation comprehensively describes the research contents of my doctoral studies at the University of Electro Communications, Tokyo, Japan. I am grateful to numerous people who have helped me to accomplish my work. First, I would like to express my sincere gratitude to my supervisor, Professor Maomi Ueno, for his valuable comments, suggestions, and encouragement throughout the research period. Secondly, I would like to acknowledge Professor Yoshio Okamoto, Professor Yusaku Yamamoto, Associate Professor Yasuhiko Takenaga, and Associate Professor Masaki Uto. Their comments and suggestions related to my research presentations were very helpful in improving my research and completing this thesis. Thirdly, I would like to thank Professor Shin-ichi Minato for his valuable advice and generous support. Additionally, I would like to thank the students of the Ueno Laboratory for their continuous support and encouragement during my studies. This research was supported by JSPS KAKENHI Grant Numbers JP19H05663 and JP24KJ1124. Finally, I would like to express my gratitude to my parents for their tremendous encouragement.

Related journal papers

1. Kazuma Fuchimoto, Takatoshi Ishii, and Maomi Ueno. Hybrid maximum clique algorithm using parallel integer programming for uniform test assembly. *IEEE Transactions on Learning Technologies*, 15(2):252–264, 2022. (Chapter 3)

2. Kazuma Fuchimoto, Shin-ichi Minato, and Maomi Ueno. Automated parallel test forms assembly using zero-suppressed binary decision diagrams. *IEEE Access*, 11:112804–112813, 2023. (Chapter 4)

Bibliography

[1] Maomi Ueno. Ai based e-testing as a common yardstick for measuring human abilities. In *2021 18th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5. IEEE, 2021.

[2] Maomi Ueno, Kazuma Fuchimoto, and Emiko Tsutsumi. E-testing from artificial intelligence approach. *Behaviormetrika*, 48(2):409–424, 2021.

[3] Bopelo Boitshwarelo, Alison Kay Reedy, and Trevor Billany. Envisioning the use of online tests in assessing twenty-first century learning: a literature review. *Research and Practice in Technology Enhanced Learning*, 12:1–16, 2017.

[4] College Board and National Merit Scholarship Corporation. Scholastic aptitude test. https://satsuite.collegeboard.org/, 2024. Accessed: 2024-12-01, Last updated: 2024.

[5] ISO/IEC. Iso/iec 29992:2018 assessment of outcomes of learning services — guidance. https://www.iso.org/standard/68490.html, 2018. Accessed: 2024-12-01, Last updated: 2024.

[6] Fumiko Samejima. Weakly parallel tests in latent trait theory with some criticisms of classical test theory. *Psychometrika*, 42(2):193–198, 1977.

[7] F.M. Lord and M.R. Novick. *Statistical theories of mental test scores*. Addison-Wesley Pub. Co., 1968.

[8] Frederic M. Lord. *Applications of Item Response Theory To Practical Testing Problems*. Routledge, 1980.

[9] T. J. J. M. Theunissen. Binary programming and test design. *Psychometrika*, 50(4):411–420, December 1985.

[10] T. J. J. M. Theunissen. Some applications of optimization algorithms in test design and adaptive testing. *Applied Psychological Measurement*, 10(4):381–389, 1986.

[11] Wim J. van der Linden and Ellen Boekkooi-Timminga. *A zero-one programming approach to Gulliksen's matched random subtest method*. Department of Education of the University of Twente, 1986.

[12] Ellen Boekkooi-Timminga. Simultaneous test construction by zero-one programming. *Methodika*, 1:101–112, 1987.

[13] Frank B. Baker, Alan S. Cohen, and B. Ross Barmish. Item characteristics of tests constructed by linear programming. *Applied Psychological Measurement*, 12(2):189–199, 1988.

[14] Jos J. Adema and Wim J van der Linden. Algorithms for computerized test construction using classical item parameters. *Journal of Educational Statistics*, 14:279–290, 1989.

[15] Terry A. Ackerman. An alternative methodology for creating parallel test forms using the irt information function. *Paper presented at the Annual Meeting of the National Council on Measurement in Education*, pages 1–25, 1989.

[16] Jos J. Adema. Models and algorithms for the construction of achievement tests. *Ph.D., University of Twente*, 1990.

[17] Jos J. Adema, Ellen Boekkooi-Timminga, and Wim J van der Linden. Achievement test construction using 0–1 linear programming. *European Journal of Operational Research*, 55(1):103–111, 1991.

[18] Jos J. Adema. Methods and models for the construction of weakly parallel tests. *Applied Psychological Measurement*, 16(1):53–63, 1992.

[19] Len Swanson and Martha L. Stocking. A model and heuristic for solving very large item selection problems. *Applied Psychological Measurement*, 17(2):151–166, 1993.

[20] H.L. Jeng and S.G. Shih. A comparison of pair-wise and group selections of items using simulated annealing in automated construction of parallel tests. *Psychological Testing*, 44(2):195–210, 1997.

[21] Richard M. Luecht. Computer-assisted test assembly using optimization heuristics. *Applied Psychological Measurement*, 22(3):224–236, 1998.

[22] Wim J. van der Linden and Jos J. Adema. Simultaneous assembly of multiple test forms. *Journal of Educational Measurement*, 35(3):185–198, 1998.

[23] Richard B. Fletcher. *A review of linear programming and its application to the assessment tools for teaching and learning (as TTLE) projects*. University of Auckland, Auckland, New Zealand, 2000.

[24] Gwo-Jen Hwang, Peng-Yeng Yin, and Shu-Heng Yeh. A tabu search approach to generating test sheets for multiple assessment criteria. *IEEE Transactions on Education*, 49(1):88–97, 2006.

[25] Koun-Tem Sun, Yu-Jen Chen, Shu-Yen Tsai, and Chien-Fen Cheng. Creating irt-based parallel test forms using the genetic algorithm method. *Applied Measurement in Education*, 2(21):141–161, 2008.

[26] Angela J. Verschoor. Genetic algorithms for automated test assembly. *Ph.D. dissertation, University of Twente*, 2007.

[27] Kejing He, Li Zheng, Shoubin Dong, Liqun Tang, Jianfeng Wu, and Chunmiao Zheng. Pgo: A parallel computing platform for global optimization based on genetic algorithm. *Computers and Geosciences*, 33(3):357–366, 2007.

[28] Pokpong Songmuang and Maomi Ueno. Bees algorithm for construction of multiple test forms in e-testing. *IEEE Transactions on Learning Technologies*, 4:209–221, 2011.

[29] Takatoshi Ishii, Pokpong Songmuang, and Maomi Ueno. Maximum clique algorithm for uniform test forms. *The 16th International Conference on Artificial Intelligence in Education*, pages 451–462, 2013.

[30] Takatoshi Ishii, Pokpong Songmuang, and Maomi Ueno. Maximum clique algorithm and its approximation for uniform test form assembly. *IEEE Transactions on Learning Technologies*, 7(1):83–95, 2014.

[31] M. Luan Nguyen, S. Cheung Hui, and A. C.M. Fong. Large-scale multiobjective static test generation for web-based testing with integer programming. *IEEE Transactions on Learning Technologies*, 6(1):46–59, 2013.

[32] Takatoshi Ishii and Maomi Ueno. Clique algorithm to minimize item exposure for uniform test forms assembly. In *International Conference on Artificial Intelligence in Education*, pages 638–641. Springer, 2015.

[33] Takatoshi Ishii and Maomi Ueno. Algorithm for uniform test assembly using a maximum clique problem and integer programming. In *Artificial Intelligence in Education*, pages 102–112, 2017.

[34] Wim J. Van der Linden. *Linear Models for Optimal Test Design*. Springer, 2005.

[35] Ellen Boekkooi-Timminga. The construction of parallel tests from irt-based item banks. *Journal of Educational Statistics*, 15(2):129–145, 1990.

[36] Ronald D. Armstrong, Douglas H. Jones, and Charles S. Kunce. Irt test assembly using network-flow programming. *Applied Psychological Measurement*, 22(3):237–247, 1998.

[37] Ting-Yi Chang and You-Fu Shiu. Simultaneously construct irt-based parallel tests based on an adapted clonalg algorithm. *Applied Intelligence*, 36(4):979–994, 2012.

[38] Jordi Pereira and Mariona Vila. Variable neighborhood search heuristics for a test assembly design problem. *Expert Systems with Applications*, 42(10):4805–4817, 2015.

[39] Dmitry I. Belov and Ronald D. Armstrong. A constraint programming approach to extract the maximum number of non-overlapping test forms. *Computational Optimization and Applications*, 33:319–332, 2006.

[40] Qingfu Zhang, Jianyong Sun, and Edward Tsang. An evolutionary algorithm with guided mutation for the maximum clique problem. *Evolutionary Computation, IEEE Transactions on*, 9(2):192–200, april 2005.

[41] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277, 1993.

[42] Shinsaku Sakaue and Kengo Nakamura. Differentiable equilibrium computation with decision diagrams for stackelberg models of combinatorial congestion games. *Advances in Neural Information Processing Systems*, 34:9416–9428, 2021.

[43] Atsushi Takizawa, Yushi Miyata, and Naoki Katoh. Enumeration of floor plans based on a zero-suppressed binary deci-

sion diagram. *International Journal of Architectural Computing*, 13(1):25–44, 2015.

[44] Takeru Inoue, Keiji Takano, Takayuki Watanabe, Jun Kawahara, Ryo Yoshinaka, Akihiro Kishimoto, Koji Tsuda, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, 2014.

[45] A. Takizawa, Y. Takechi, A. Ohta, N. Katoh, T. Inoue, T. Horiyama, J. Kawahara, and S.-I. Minato. *Enumeration of region partitioning for evacuation planning based on ZDD*, pages 65–72. 2014.

[46] Hiroaki Iwashita and Shin-ichi Minato. *Efficient top-down ZDD construction techniques using recursive specifications*, 2013.

[47] F.B. Baker and S.H. Kim. *Item Response Theory: Parameter Estimation Techniques, Second Edition*. Taylor & Francis, 2004.

[48] Wim J. Van der Linden and Ellen Boekkooi-Timminga. A maximin model for irt-based test design with practical constraints. *Psychometrika*, 54(2):237–247, 1989.

[49] Ronald D. Armstrong, Douglas H. Jones, and Zhaobo Wang. Automated parallel test construction using classical test theory. *Journal of Educational Statistics*, 19(1):73–90, 1994.

[50] Etsuji Tomita, Kohei Yoshida, Takuro Hatta, Atsuki Nagao, Hiro Ito, and Mitsuo Wakatsuki. A much faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Frontiers in Algorithmics*, pages 215–226, 2016.

[51] Chu-Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017.

[52] Jos J. Adema. Implementations of the branch-and-bound method for test construction problems. *Project Psychometric Aspects of Item Banking, Department of Education*, 1989.

[53] Kazuma Fuchimoto, Takatoshi Ishii, and Maomi Ueno. Hybrid maximum clique algorithm using parallel integer programming for uniform test assembly. *IEEE Transactions on Learning Technologies*, 15(2):252–264, 2022.

[54] IBM. Ilog cplex optimization studio cplex 12.9. `https://www.ibm.com/products/ilog-cplex-optimization-studio`, 2019. Accessed: 2024-12-01, Last updated: 2019.

[55] Recruit. Synthetic personality inventory (spi). http://www.spi.recruit.co.jp/, 2024. Accessed: 2024-12-01, Last updated: 2024.

[56] Dmitry I. Belov. Uniform test assembly. *Psychometrika*, 73(1):21–38, 2008.

[57] Kazuma Fuchimoto, Shin-ichi Minato, and Maomi Ueno. Automated parallel test forms assembly using zero-suppressed binary decision diagrams. *IEEE Access*, 11:112804–112813, 2023.

[58] Donald Ervin Knuth. The art of computer programming: Bitwise tricks & techniques. *Binary Decision Diagrams*, 4, 2009.

[59] Shin ichi Minato, Mutsunori Banbara, Takashi Horiyama, Jun Kawahara, Ichigaku Takigawa, and Yutaro Yamaguchi. Interval-memoized backtracking on zdds for fast enumeration of all lower cost solutions, 2022.

[60] Howard Wainer. Rescuing computerized testing by breaking zipf's law. *Journal of Educational and Behavioral Statistics*, 25:203–224, 2000.

[61] Wim J Van der Linden and Seung W Choi. Improving item-exposure control in adaptive testing. *Journal of Educational Measurement*, 57(3):405–422, 2020.

[62] Maomi Ueno and Yoshimitsu Miyazawa. Uniform adaptive testing using maximum clique algorithm. In *International Conference on Artificial Intelligence in Education*, pages 482–493. Springer, 2019.

[63] Maomi Ueno and Yoshimitsu Miyazawa. Two-stage uniform adaptive testing to balance measurement accuracy and item exposure. In *Artificial Intelligence in Education: 23rd International Conference*, pages 626–632, 2022.

[64] Wakaba Kishida, Kazuma Fuchimoto, Yoshimitsu Miyazawa, and Maomi Ueno. Item difficulty constrained uniform adaptive testing. In *International Conference on Artificial Intelligence in Education*, pages 568–573, 2023.