DOCUMENT RESUME

ED 467 819 TM 034 363

AUTHOR van der Linden, Wim J.; Veldkamp, Bernard P.; Reese, Lynda M.
TITLE An Integer-Programming Approach to Item Pool Design. Law School Admission Council Computerized Testing Report. LSAC Research Report Series.
INSTITUTION Law School Admission Council, Princeton, NJ.
REPORT NO LSAC-R-98-14
PUB DATE 2000-09-00
NOTE 14p.
PUB TYPE Reports - Research (143)
EDRS PRICE EDRS Price MF01/PC01 Plus Postage.
DESCRIPTORS *Item Banks; Test Construction; *Test Items; Testing Programs
IDENTIFIERS *Integer Programming

ABSTRACT

Presented is an integer-programming approach to item pool design that can be used to calculate an optimal blueprint for an item pool to support an existing testing program. The results are optimal in the sense that they minimize the efforts involved in actually producing the items as revealed by current item writing patterns. Also presented is an adaptation of the models for use as a set of monitoring tools in item pool management. The approach is demonstrated empirically for an item pool designed for the Law School Admission Test. (Contains 2 tables and 30 references.) (Author/SLD)

LSAC RESEARCH REPORT SERIES

TM

An Integer-programming Approach to Item Pool Design

Wim J. van der Linden

Bernard P. Veldkamp

University of Twente

Lynda M. Reese

Law School Admission Council
Computerized Testing Report 98-14
September 2000

LAW Services

A Publication of the Law School Admission Council

The Law School Admission Council is a nonprofit corporation that provides services to the legal education community. Its members are 197 law schools in the United States and Canada.

This study is published and distributed by the Law School Admission Council (LSAC). The opinions and conclusions contained in these reports are those of the authors and do not necessarily reflect the position or policy of the Law School Admission Council.

Table of Contents

Executive Summary

Recently, a body of research on the problem of optimal test assembly has been developing. Researchers have taken various approaches to the problem of automatically assembling a number of test forms from a pool of test items (questions). The result is a prespecified number of test forms that optimally meet a predefined set of test specifications. These test specifications may include a balance of such things as test content, answer key distribution, word count limitations, and statistical specifications.

While many of the approaches to optimal test assembly have been very successful, the results of these methods are limited by the composition of the item pool to which they are applied. While a method of optimal test assembly may result in test forms that are optimal for the item pool, these forms may be unacceptable in important ways if the item pool is lacking. For example, the item pool may contain items of the required content characteristics, but the items may be too difficult or too easy. The end result could be a content-balanced test form that is inappropriate in terms of difficulty level. Also, an item pool may appear to be large, but may contain many items that will never be selected for inclusion on a test form. In such a case, the size of the pool is deceiving from the point of view of test assembly.

The method described in this paper presents an integer-programming approach to item pool design. The result of this approach is a document, referred to here as a "blueprint," specifying what attributes the items in a new item pool or an update to an existing pool should have. The blueprint is specified to allow for the assembly of a prespecified number of test forms. The solution is optimal in that the efforts needed to achieve this item pool are minimized. The number of unused items in the pool is minimized as well.

This study addressed the definition of an item pool to support the assembly of a prespecified number of paper-and-pencil test forms. Paper-and-pencil test assembly represents a simpler problem than the selection of items for inclusion in a computerized adaptive test. Future research on this problem will address adaptive testing. This approach is very promising for the monitoring of a computerized-testing item pool.

Abstract

An integer-programming approach to item pool design is presented that can be used to calculate an optimal blueprint for an item pool to support an existing testing program. The results are optimal in the sense that they minimize the efforts involved in actually producing the items as revealed by current item writing patterns. Also, an adaptation of the models for use as a set of monitoring tools in item pool management is presented. The approach is demonstrated empirically for an item pool designed for the Law School Admission Test (LSAT).

Introduction

Recently, a variety of methods for automated assembly of test forms from an item pool have become available. Each of these methods can be classified as belonging to one of the following classes (van der Linden, 1998): (1) heuristics that select items sequentially to match a target for the test information function or to fit a weighted combination of the test specifications (e.g., Ackerman, 1989; Luecht, 1998; Sanders & Verschoor, 1998; Swanson & Stocking, 1993); (2) methods that model the test assembly problem as a 0-1 linear programming (LP) problem and then use a search algorithm or heuristic to find a simultaneous solution to the problem (e.g., Adema, 1990, 1992; Adema, Boekkooi-Timminga, & van der Linden, 1991; Boekkooi-Timminga, 1987, 1990; Theunissen, 1985; Timminga & Adema, 1995; van der Linden, 1994, 1996; van der Linden & Boekkooi-Timminga, 1989); (3) methods based on network-flow programming with Lagrange relaxation and/or embedding of the network model in a heuristic (Armstrong & Jones, 1992; Armstrong, Jones, & Wang, 1994, 1995; Armstrong, Jones, & Wu, 1992); and (4) methods based on optimal design theory from statistics (e.g., Berger, 1994). Detailed descriptions and examples of these methods are given in a recent special issue of *Applied Psychological Measurement* on optimal test assembly (van der Linden, 1998).

These test assembly methods result in tests that are optimal or close to optimality. However, even when optimal, the results may not be satisfactory because an important constraint on the quality of the tests is imposed by the composition of the item pool. For example, an item pool can have enough items with the content attributes required by the test specifications but their statistical attributes may be off target in the sense that the items are too difficult or too easy. This case can easily occur if an item pool is frequently used and certain categories of items in the pool are depleted quickly. The result is then an optimal test with an information function too low on a relevant interval on the ability scale. Though the problem of item pool depletion is less likely to happen for larger item pools, it should not be inferred that larger item pools are necessarily optimal. On the contrary, a well-known phenomenon in item pool management is that a considerable proportion of the items in the pool may never be used. The presence of such "wallflowers" can be the result of attribute values not needed by the test specifications or overrepresented in the pool. Since the

costs of screening and pretesting items are generally high, items in either category typically involve a considerable loss of resources.

This paper presents an integer-programming method for item pool design. The method results in a *blueprint* for an item pool, that is, a document specifying what attributes the items in a new item pool or an update of an existing pool should have. As will become clear below, the blueprint is designed to allow for the assembly of a prespecified number of test forms from the pool, each with its own set of specifications. At the same time, it is optimal in the sense that the efforts or "costs" involved in realizing the item pool are minimized. A favorable consequence of this objective is that the number of unused items is also minimized. (In practice, it may be prudent to have a few spare items though; see the discussion later in this paper.)

The actual task of writing test items to a blueprint is difficult. This difficulty arises mainly as a result of statistical attributes, such as p-values, item-test correlations, and item response theory (IRT) parameters, rather than content attributes. It is a common experience that the values of statistical attributes of *individual* items are only loosely predictable. At the same time, however, at the level of a pool of items, statistical attributes often show persistent patterns of correlation with content attributes. In this paper, these patterns are used to derive an empirical measure for item writing efforts that is minimized in the design model.

The point of view taken in this paper is that item pools are not static entities. Tests are assembled from the pool and subsequently released, or items may be removed from the pool because they become obsolete. In most testing programs, new items are therefore written and pretested on a continuous basis. Though presented as a method for designing a single pool, it is believed that in practice the models in this method will serve as tools for monitoring the item writing process on a more continuous basis. The slight adaptation needed to use the models for item pool management is presented later in this paper. As will become clear below, if the models are used in this mode, possible differences between the statistical attributes in the blueprint and their actual values from the pretest of the items in the existing pool are automatically detected and lead to an optimal update of the blueprints for the items that still have to be written.

The problem of item pool design has been addressed earlier in Boekkooi-Timminga (1991) and Stocking and Swanson (1998). The former paper also uses integer-programming to calculate the number of items needed for future test forms but follows a sequential approach maximizing the information function of each subsequent test under the Rasch or one-parameter logistic model. The results are then used to improve on the composition of an existing item pool. The model proposed in this paper directly calculates a blueprint for the entire item pool (though it is sometimes efficient to do the calculations sequentially). In addition, its objective is to minimize the costs of actually producing the pool by the current item writers rather than maximizing the test information functions. At the same time, the model guarantees that the targets for the test information functions are met. Finally, this model is not restricted to items calibrated under the Rasch model. The paper by Stocking and Swanson does not deal with the problem of designing an item pool as such. Rather, it presents a method for assigning items from a master pool to a set of smaller pools accessed randomly in an adaptive testing program to minimize item security problems.

The remainder of the paper is organized as follows: First, the problem of item pool design is analyzed, making an important distinction between test specifications based on categorical and quantitative item attributes. Then the design method is presented, and it is shown how its models minimize the expected costs involved in item writing. In addition, it is explained how the method can be used if the item pool has to support a testing program with sets of items related to common stimuli. The next section of the paper explains how the proposed models can be adapted for use as monitoring tools in item pool management. Finally, an empirical application of the models to the problem of designing an item pool for the Law School Admission Test (LSAT) is presented.

Analysis of Design Problem

An important distinction between test specifications or constraints in mathematical programming models for test assembly is the one between constraints on categorical item attributes, on quantitative attributes, and constraints needed to represent inter-item dependencies (van der Linden, 1998). This distinction also plays a critical role in the item pool design model presented in this paper.

Categorical Constraints

The defining characteristic of a categorical item attribute, such as item content, cognitive level, format, author, or answer key, is that it partitions the item pool into a series of subsets. A test specification with respect to a constraint on a categorical attribute constrains the distribution of the items in the test over these subsets. If the items are coded by multiple attributes, their Cartesian product introduces a partition of the pool. In this case, constraints on categorical attributes do not only address the marginal distributions of items on attributes but also their joint and conditional distributions.

A natural way to represent categorical attributes is by a table. An example for the case of two categorical constraints with a few constraints on their distributions is given in Table 1. One attribute is item content, C (with levels C1, C2, and C3); the other is item format, F (with levels F1 and F2). In the first panel, the full distribution of the items in the pool is represented by the numbers n_{ij}, $n_{i.}$, $n_{.j}$, and $n_{..}$ that are the numbers of items in cell (i,j) row i, column j, and the total table, respectively. Likewise, in the second panel the numbers of items in the test are denoted as r_{ij}, $r_{i.}$, $r_{.j}$, and $r_{..}$. The following set of constraints is imposed on the test:

$r_{12} = 6;$
$r_{.1} = 4;$
$r_{1.} = 8.$

TABLE 1
Distribution of items in the pool and constrained distribution of items in a test form (case of two categorical attributes)

	F1	F2	
C1	n_{11}	n_{21}	$n_{.1}$
C2	n_{12}	n_{22}	$n_{.2}$
C3	n_{13}	n_{23}	$n_{.3}$
	$n_{1.}$	$n_{2.}$	$n_{..}$

	F1	F2	
C1	r_{11}	r_{21}	4
C2	6	r_{22}	$r_{.2}$
C3	r_{13}	r_{23}	$r_{.3}$
	8	$r_{2.}$	$r_{..}$

Note that this set not only fixes certain numbers directly but also restricts the values possible for the other numbers in the table. For example, the first and last constraint together imply the constraint $r_{11} + r_{13} \leq 2$. This fact sometimes allows us to represent the same set of test specifications by different sets of constraints. Some of these sets may be smaller and therefore more efficient than others. The method in this paper, however, is neutral with respect to such differences.

In a test assembly problem, values for the numbers r_{ij} are sought such that the constraints on all distributions are met and the combination of the values optimizes an objective function. In so doing, the numbers n_{ij} of items in the pool are fixed and serve as upper bounds to the numbers r_{ij}. The basic approach to the item pool design problem in this paper is *to reverse the role of these two quantities*. The number n_{ij} are now taken as the decision variables and a function of them is optimized subject to all constraint sets involved by the specifications of the tests the pool has to support.

Quantitative Contraints

Examples of quantitative item attributes are word counts, exposure rates, values for IRT information functions, expected response times, and classical parameters as *p*-values and item-test correlations. Unlike categorical constraints, quantitative constraints do not impose bounds directly on numbers of items but on a function of their joint attribute values, mostly a sum or an average.

In IRT-based test assembly, important constraints are those on the information function of the test. These constraints typically require the sum of values of the item information functions to meet certain bounds. It is important to note that though each combination of item information functions defines a unique test information function, the reverse does not hold. Unlike categorical attributes, constraints with quantitative attributes have no one-to-one correspondence with item distributions but sets of distributions that are feasible with respect to the constraints. However, this property should not be viewed as a disadvantage. It can be exploited to choose a distribution to represent a quantitative constraint that is *optimal* with respect to an objective function. This paper follows this approach and translates all constraints on quantitative attributes into an optimal distribution of the items over tables defined by a selection of their values.

The option of choosing an objective function for the selection of an optimal item distribution is used to solve a problem alluded to earlier—the difficulty involved in writing items with prespecified values for their statistical attributes. More concretely, it is proposed to minimize an explicit objective function that measures the costs or efforts involved in writing items with the various possible combinations of attribute values by the item writers.

One possible option is to assess these costs directly, for example, by asking item writers to time their activities or by analyzing log files produced by their word processors. Another, possibly less time-consuming option is to use the distribution of the statistical attributes for a recent item pool to define the costs involved in writing items with certain combinations of attribute values. In so doing, the assumption can be made that items with combinations of attribute values with higher frequencies are easier or less "costly" to produce. In view of the empirical example later in this paper, the latter option will be elaborated. However, this choice does not imply that this type of objective function is necessarily best.

More realistic information on item writing costs is obtained if the joint distribution of all quantitative and categorical attributes is used. If persistent differences between item writers exist, further improvement is possible by choosing item writers as one of the (categorical) attributes defining the joint distribution. This choice will now be formalized for constraints on test information functions. However, the treatment can easily be generalized to constraints on other quantitative attributes.

In the empirical example, the 3-parameter logistic model was used to calibrate the existing item bank:

$$P_i(\theta) \equiv \text{Prob}\{U_i = 1|\theta\} \equiv c_i + (1 - c_i)\{1 + \exp[-a_i(\theta - b_i)]\}^{-1}, \quad (1)$$

where θ is the unknown ability of the examinee and $a_i \in [0,\infty]$, $b_i \in [-\infty,\infty]$, and $c_i \in [0,1]$, are the discrimination, difficulty, and guessing parameter for item i, respectively (Lord, 1980, chap. 2). First, the scales of the parameters a_i, b_i, and c_i are replaced by a grid of discrete values, (a_{id}, b_{id}, c_{id}), with $d = 1, \ldots, D$ grid points. The number of points on the grid as well as their spacing is free. Let Q be the table defined by the product of these grids for all quantitative attributes, with arbitrary cell q. The symbol C is used to represent the full table defined by the categorical attributes, with an arbitrary cell denoted by $c \in C$. A cell in the joint table defined by C and Q is denoted as $(c,q) \in C \times Q$. Since the table is the product of the attributes, the number of cells can become very large. For a realistic example, see Table 2.

Let x_{cq} denote the frequency of the items in cell (c,q) for a representative pool. These frequencies contain information on the efforts involved in writing items for the various cells in the table. Cells with relatively large frequencies represent combinations of categorical and quantitative attribute values that tend to go together often; apparently, such items are easy to produce. On the other hand, empty cells seem to point at combinations of attribute values that are difficult to produce. A monotonically decreasing function of x_{cq}, denoted as $\varphi(x_{cq})$, will be used as an empirical measure of the efforts involved in writing items with the various possible combinations of attribute values. The generic term "cost function" will be used for this function. In the model below, the costs for writing the new item pool will be minimized using this function.

A simple cost function is $\varphi(x_{cq}) = x_{cq}^{-1}$, which requires $x_{cq} > 0$. The cost function is thus based on the assumption that the smaller the supply of items with attribute values (c,q) in the existing pool, the more difficult it might have been to produce such items. Other choices of function are possible. However, as will be obvious from the definition of the objective function in Equation 2, the choice of unit does not matter. If different patterns of correlation exist for different item writers, the cost function can be made more realistic by adding item writers as a categorical attribute to the table. If this option is used, a constraint has to be added to the design models below on the number of items or stimuli to be written by each item writer. The blueprint of the new item pool then automatically shows which types of items have to be written by which author.

Also, if the existing item pool is relatively small, before calculating $\varphi(x_{cq})$, it is recommended to collapse over attributes in the $C \times Q$ table that show no substantial dependencies on any of the other attributes. This operation will result in larger frequencies and hence in more stable marginal cost estimates. In the objective function and constraints in the models below these marginal estimates are substituted within the cells that have been collapsed. The operation is thus not to reduce the size of the original design problem. The models still provide complete blueprints for the items in the pool.

It should be noted that the use of a cost function defined on item writing practices for a recent item pool is *not* conservative in the sense that old practices are automatically continued. The new item pool can be planned freely to support any new sets of test specifications, and the integer-programming model guarantees that test forms can be assembled to these specifications. The point is, however, that a potentially large set of item pools can be expected to be feasible for the integer-programming model. The cost function is used only to select a solution from this set that minimizes the costs of item writing.

Constraints on Interdependent Items

The constraints in this category deal with possible relations of exclusion and inclusion between the items in the pool. Two items exclude each other if they overlap in content and, as a consequence, one item contains a clue to the key of the other item ("enemies"). Generally, the larger the number of attributes used in the test specifications, the more specific the blueprints and the less likely it is to have overlap between written items. To the experience of the authors, sets of enemies in existing pools are usually small (2-3 items per set), in particular, in relation to the number of tests the pool has to serve. In 0-1 LP-based test assembly, it is possible to constrain the test to have no more than one item from each set of enemies. If enemies are present in an item pool, they can thus generally be distributed over different test forms. The position taken in this paper is therefore that sets of enemies in the pool are a problem of *test assembly*. In the item pool design problem in this paper, they will be ignored.

An important type of inclusion relation exists between items that are organized around common stimuli, for example, a reading passage in a reading comprehension test or a description of an experiment in a biology test. We will use "item sets" as a generic term for this part of a test. Typically, the items in these sets are selected from larger sets available in the pool. Selection of item sets often involves constraints on categorical (e.g., content) and quantitative (e.g., word counts) attributes for the stimuli. Several versions of 0-1 LP models for test assembly are available to deal with pools with item sets (van der Linden, submitted).

The problem of designing a pool with item sets is solved by the following three-stage procedure:

> First, a blueprint for a pool of *items* is designed using the integer-programming model in Equations 2-6 ignoring the item set structure. The model constrains the distributions of the items over their categorical and quantitative attributes. The objective function minimizes a cost function for writing the items.
>
> Next, a blueprint for a pool of *stimuli* for the item sets is designed using the same methodology as for the pool of items. The model now constrains the distribution of the stimuli over their categorical and quantitative attributes, and the objective function minimizes a cost function for writing the stimuli.
>
> Finally, items are assigned to the stimuli to form *item sets*. The assignment is done using a separate integer-programming model formulated for this task. The constraints in the model control the assignment both for the number of items available in the various cells of the C x Q table and the number required in the item sets. The objective function is of the same type as above; its specific form will be explained later in this paper.

The fact that these steps are taken separately should not come as a surprise. They only serve to *design* an item pool with a set structure. Of course, if the design is realized, the actual stimuli and items in the sets are to be written simultaneously and in a coordinated fashion.

Models for Item Pool Design

In this section the various models introduced above will be explained. First, the model for designing the pool of items is presented. Then the case of item sets is addressed and the models for designing a pool of stimuli and assigning items to stimuli are explained.

Pool of Items

The following notation is needed to present the model. Index $f = 1, ..., F$ will be used to represent the individual test forms the item pool should support. Still, the symbols C and Q will be used to represent the tables defined by the categorical and quantitative attributes, respectively. As an example of a quantitative attribute the information function of test form f will be required to approach a set of target values $T_f(\theta_k)$, $k = 1, ..., K$, from above. The model in Equation 1 implies a three-dimensional table Q, with one dimension for each item parameter. The information on θ in a response to an item in cell $q \in Q$ will be denoted as $I_q(\theta)$; this quantity is calculated, for example, for the midpoints of the intervals of the item parameter values defining cell q of the table. The decision variables in the model are integer variables n_{fcq}. These variables represent the number of items in cell (c,q) needed in the pool to support form f, that is, these variables indicate how many items with item parameter values represented by q and categorical attributes represented by c would be needed for form f. The complete pool is thus defined by the numbers

$$\sum_{f=1}^{F} n_{fcq}.$$

The cost function is still denoted as φ_{cq}.

The model is as follows:

$$\text{minimize} \sum_{f} \sum_{c} \sum_{q} \varphi_{cq} n_{fcq} \qquad \text{(minimizing costs) (2)}$$

subject to

$$\sum_{c} \sum_{q} I_q(\theta_k) n_{fcq} \geq T_f(\theta_k), \; f = 1, ..., F, \; k = 1, ..., K, \qquad \text{(test information) (3)}$$

$$\sum_{c \in V_{fg}} \sum_{q} n_{fcq} \geq n_{fg}, f = 1, ..., F, g = 1, ..., G, \quad \text{(categorical constraints) (4)}$$

$$\sum_{c} \sum_{q} n_{fcq} \geq n_{f}, f = 1, ..., F, \quad \text{(length of forms) (5)}$$

$$n_{fcq} = 0,1, ..., f = 1, ..., F, c \in C, q \in Q. \quad \text{(integer variables) (6)}$$

The objective function in Equation 2 minimizes the sum of the item writing costs across all items in the F forms. For each form the constraints in Equation 3 require the information function to be larger than the target values at θ_k, $k = 1, ..., K$. Because the objective function in Equation 2 implies a minimum number of items to be written, the test information functions approach these target values from above. The categorical constraints imposed on the forms are formulated in Equation 4. The sets V_{fg}, $f = 1, ..., F$ and $g = 1, ..., G$, are the sets of cells in C on which the constraints have to be imposed. For example, in the test form in Table 1 the first constraint imposed on the set of cells consists only of cell (1,1), the second on the set of cells in Row 1, and the third on the set of cells in Column 1. Lower bounds n_{fg} are set only; the objective function in Equation 2 guarantees that the constraints are satisfied as an equality at optimality. The same happens to the constraints on the lengths of the forms in Equation 5. Other quantitative constraints can be added to the model following the same logic as in Equation 3.

Models for the simultaneous assembly of sets of tests from a given pool need large numbers of constraints to prevent the same item from being assigned to more than one test form (Boekkooi-Timminga, 1987; van der Linden & Adema, 1998). However, such constraints need not bother us here. The current goal is only to determine how many items of certain types are needed in the item pool to assemble this set of test forms.

For smaller $C \times Q$ tables, the optimal values of the variables in the model can be calculated using one of the available implementations of the branch-and-bound algorithm for integer-programming. For larger tables optimal values can always be obtained by relaxing the model and using the simplex algorithm. The simplex algorithm is capable of handling problems with thousands of variables in a small amount of time (Wagner, 1978). Fractional values in the solution can then be rounded upwardly; as already noted, it is always prudent to have a few spare items in the pool.

Pool of Stimuli

Tables C' and Q' are now defined for the sets of categorical and quantitative attributes used to describe the stimuli in the test forms the item pool has to support. Since psychometric attributes for stimuli are rare, table Q' is expected to be much smaller than Q. Item sets do often have *aggregated* statistical attributes, such as sums of p-values or average values for b_is. However, these aggregates belong to the set of items associated with a stimulus—not to the stimulus itself. Constraints on such aggregated attributes are dealt with in the item assignment model below.

The model is analogous to that in Equations 2-6. The cost function $\varphi_{c'q'}$ is now defined for the distribution of stimuli in the previous item pool. Likewise, the bounds in Equations 3-4 are derived from the specifications for the item sets in the various test forms. As an example of a quantitative attribute for stimuli, word counts are used. Let $w_{p'}$ be the number of words for a stimulus in cell q and w_f the target for the number of words for a stimulus in form f. The constraints needed are

$$\sum_{c'} \sum_{q'} w_{q'} n_{fc'q'} \geq w_f, f = 1, ..., F. \quad \text{(word counts) (7)}$$

Again, because of minimization in Equation 2, the bounds in the constraints in Equation 7 are approximated from above and serve as targets for the number of words per stimulus.

The output from the model is an optimal array of frequencies $n_{fc'q'}$ for form f. The blueprint for the complete pool of stimuli is determined by the numbers

$$\sum_{f} n_{fc'q'}.$$

Assigning Items to Stimuli

To introduce the item assignment model, index $s_f = 1, ..., S_f$ is defined to denote the item sets in form $f = 1, ..., F$. Each of these sets is associated with one stimulus, that is, one of the cells (c', q'). The total number of stimuli associated with the cells satisfies the optimal numbers $n_{fc'q'}$ from the model in the previous section. The association is arbitrary and assumed to be made prior to the item assignment. For each set the

attribute values of its stimulus are thus assumed to be known; however, for notational convenience, the dependence of s_f (c',q') will remain implicit in this paper.

In addition, integer decision variables z_{s_fcq} are defined to denote the number of items from cell (c,q) in the item table assigned to set s_f. Finally, a cost function $\varphi_{cqc'q'}$ is defined on the Cartesian product of the tables $C \times Q$ and $C' \times Q'$. This function reflects the costs of writing an item with attributes (c,q) for a stimulus with attributes (c',q').

The item assignment model is as follows:

$$\text{minimize} \sum_f \sum_{s_t} \sum_c \sum_q \varphi_{cqc'q'} z_{s_fcq} \qquad \text{(minimizing costs) (8)}$$

subject to

$$\sum_{c,q} z_{s_fcq} \geq n_{s_t}, s_f = 1, ..., S_f, f = 1, ..., F, \qquad \text{(number of items needed) (9)}$$

$$\sum_f \sum_{s_t} z_{s_fcq} \geq n_{cq}, c \in C, q \in Q, \qquad \text{(number of items available) (10)}$$

$$z_{s_fcq} = 0, 1, ..., s_f = 1, ..., S_f, f = 1, ..., F, c \in C, q \in Q \qquad \text{(integer variables) (11)}$$

The constraints in Equation 9 assign n_{s_t} items to item set s_f whereas the constraints in Equation 10 ensure that no more items are assigned from cell (c,g) than available in the blueprint for the item pool.

If constraints on aggregated quantitative item attributes have to be imposed on some of the item sets, the model has to be expanded. For example, if item set s_f has to have an average p-value between lower and upper bounds $p_{s_f}^{(l)}$ and $p_{s_f}^{(u)}$, respectively, the following two constraints should be added to the model:

$$\sum_c \sum_q p_q z_{s_fcq} \geq n_{s_f} p_{s_t}^{(l)}, s_f = 1, ..., S_f, f = 1, ..., F, \qquad \text{(lower bounds on } \bar{p}) \text{ (12)}$$

$$\sum_c \sum_q p_q z_{s_fcq} \leq n_{s_f} p_{s_f}^{(u)}, s_f = 1, ..., S_f, f = 1, ..., F. \qquad \text{(upper bound on } \bar{p}) \text{ (13)}$$

Models for Item Pool Management

As already indicated, item pools are not static entities. In most testing programs, tests are assembled from the pool and new items are pretested on a continuous basis. Hence, two important tasks of item pool management are: (1) monitoring the developments in the item pool; and (2) instructing item writers to write new items to complete the pool.

The models in this paper can easily be adapted for use in item pool management. The only thing needed is to correct the decision variables in the models for the numbers of items and stimuli currently available in the pool. The principle is illustrated for the model in Equations 2-6. Let v_{cq} be a constant representing the current number of items in cell (c,q) in the pool and η_{cq} a new decision variable denoting the number of items to be written yet for this cell. The only adaptation necessary is *substituting* $v_{cq} + \eta_{cq}$ *for the old decision variables in the model.*

If the current items in the pool reveal new patterns of correlation between categorical and quantitative attributes, the cost functions φ_{cq} can be updated by defining them on v_{cq} rather than the frequencies x_{cq} for the previous item pool, or perhaps on a weighted combination of both. This practice is recommended, for example, if the item writers form a categorical attribute in the definition of table $Q \times C$ and new item writers have been hired.

Empirical Example

The method in this paper was used to design a new item pool for the LSAT. The purpose of this study was to illustrate the use of the method for an actual item pool design problem.

The LSAT pool has to support three different sections in the test, labeled here as A, B, and C. The first two sections have items organized as sets with a common stimulus; the last section consists of discrete items. The three sections in the test were assembled to meet experimental sets of test specifications and targets for their information functions. The sets of constraints included such attributes as item (sub)types, item-set structures in the pool, types of stimuli, gender and minority orientation of the stimuli, answer key

distributions, word counts, and test information functions. Also, a previous item pool was available to estimate a cost funtion. As the original authors of the pool could not be identified, item author was not used as an attribute in the current example. The values of the a_i and b_i parameters in the model in Equation 1 were grouped into 8 and 10 intervals, respectively, using the midpoints of the intervals to calculate the information function values. For security reasons, the exact item attributes and stimulus in this study are not revealed. The numbers of attributes for each section in the LSAT are given in Table 2.

TABLE 2

Number of item attributes, constraints, and decision variables for the three sections of the LSAT

Model	Number of Attributes	Number of Constraints	Number of Decision Variables
Section A			
Item Pool	5	70	1,920
Stimulus Pool	4	1	8
Assignment	*	4	24
Section B			
Item Pool	6	97	6,144
Stimulus Pool	4	6	31
Assignment	*	10	12
Section C			
Item Pool	5	65	11,520

Note. Cells with "*" are not applicable

The item pool was designed to support 10 regular forms of the LSAT, 10 forms with a target for the test information function shifted 0.6 to the left on the θ continuum, and 10 with a target shifted 0.6 to the right. The (integer) decision variables in these constraints represented the frequencies needed for the cells in the full attribute tables, $C \times Q$. The numbers of decision variables and constraints in the models for the item pool, stimulus pool, and assignment of the items to the stimuli are given in the fourth column of Table 2.

A previous pool of 5,316 items was available to define cost functions for the models. The functions were defined as $\varphi(x_{cq}) \equiv x_{cq}^{-1}$, with an arbitrary large value substituted for empty cells in the table. Because the number of items in the pools was small relative to the numbers of cells in the attribute tables, the tables were collapsed over some attributes before computing the cost function. For example, since the values of the items for the guessing parameter in the 3-PL model in Equation 1, c_i, did not vary much, this parameter was left out as a dimension of the attributes table $C \times Q$, using its average value to calculate the information function values in the models. Neighboring values of other attributes with approximately the same conditional distribution were grouped. The purpose of all grouping and collapsing was to reduce the number of cells in the table and get more stable estimates of the frequencies in the cost function. Cells that were collapsed in the attribute tables received a value for the cost function based on their marginal frequencies. However, the integer-programming models in this example were still formulated over the full table, with marginal costs substituted for the cells that were collapsed.

As the three sections in the LSAT have no overlap in items, three independent models had to be solved. The numbers of constraints and decision variables in the integer-programming models are given in Table 2. The three sets of test specifications the pool was assumed to support involved no constraints on interdependent items between them. Also, the objective functions in the models are sums of costs across these sets which are minimal if the costs for each set are minimal. The models could therefore be solved independently for each set.

The best strategy to solve models of the size in the current application is through the simplex algorithm for the relaxed version of the models; that is, with the integer variables replaced by (nonnegative) real-valued variables, rounding the solutions upwardly. The simplex algorithm as implemented in the Consolve module in the test assembly software package ConTEST was used (Timminga, van der Linden, &

Schweizer, 1996). The solution times for all models were approximately 1 second of CPU time on a PC. No rounding appeared to be necessary; the algorithm found a direct integer solution for all variables. This result happened because the matrix of coefficients for the models appeared to have a unimodular structure (for this property, see Nemhauser & Wolsey, 1988, chap. II.3). This feature does not generalize automatically to other applications of the integer-programming models for item pool design in this paper. However, rounding the variables upwardly is a simple but effective strategy that always works.

Concluding Remarks

The example showed how the integer-programming models in this paper can be used to calculate blueprints for new item pools to support a testing program. Although not illustrated in the example, only a minor correction to the decision variables is necessary to use the models as tools for managing item pools in an ongoing testing program. The same correction can be used to cope with possible future changes in the right-hand side bounds in the test specifications in the models.

As already indicated, several options to specify cost functions in Equation 2 and 8 are available, including options that directly estimate the costs or the amount of time involved in writing items with certain attributes. The results from the models in real-life applications rely heavily on the cost functions adopted. The question regarding what cost function is best does not have a universal answer. It can be answered best when implementing the method for an actual testing program.

For the empirical example above, a previous item pool could be used to define a cost function on the distribution of the items over the tables defined by the item and stimulus attributes. This option is available only if the test specifications at the time of this previous pool address a set of item and stimulus attributes that include the set of attributes for the new item pool. If new attributes are introduced, this approach may face an unsolvable missing data problem. However, too many old attributes do not constitute a practical problem because the old table can always be collapsed over attributes that are no longer used in the testing program.

A case not dealt with in this paper is the one in which the tests assembled from the pool are allowed to have item overlap. If overlap is allowed, the number of possible tests from a given pool goes up. Determining how many different tests are possible under this condition involves a complicated combinatorial problem (Theunissen, 1996). If the overlap is small, a wise strategy might be to just ignore this possibility, the result being an item pool somewhat too large but allowing for all planned tests to be assembled. However, the problem of how to design an item pool of minimal size to support tests for which large overlap is allowed remains still to be solved.

References

Ackerman, T. (1989, March). *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, San Francisco.

Adema, J. J. (1990). The construction of customized two-staged tests. *Journal of Educational Measurement, 27*, 241-253.

Adema, J. J. (1992). Implementations of the branch-and-bound method for test construction. *Methodika, 6*, 99-117.

Adema, J. J., Boekkooi-Timminga, E., & van der Linden, W. J. (1991). Achievement test construction using 0-1 linear programming. *European Journal of Operations Research, 55*, 103-111.

Armstrong, R. D. & Jones, D. H. (1992). Polynomial algorithms for item matching. *Applied Psychological Measurement, 16*, 365-373.

Armstrong, R. D., Jones, D. H., & Wang, Z. (1994). Automated parallel test construction using classical test theory. *Journal of Educational Statistics, 19*, 73-90.

Armstrong, R. D., Jones, D. H., & Wang, Z. (1995). Network procedures to optimize test information curves with side constraints. In K.D. Lawrence (Ed.), *Applications of management science: Network optimization applications* (Vol. 8; pp. 189-212). Greenwich, CT: JAI Press.

Armstrong, R. D., Jones, D. H., & Wu, I.-L. (1992). An automated test development of parallel tests. *Psychometrika, 57*, 271-288.

Berger, M. P. F. (1994). A general approach to algorithmic design of fixed-form tests, adaptive tests, and testlets. *Applied Psychological Measurement, 18,* 141-153.

Boekkooi-Timminga, E. (1987). Simultaneous test construction by zero-one programming. *Methodika, 1,* 1101-112.

Boekkooi-Timminga, E. (1990). The construction of parallel tests from IRT-based item banks. *Journal of Educational Statistics, 15,* 129-145.

Boekkooi-Timminga, E. (1991, June). *A method for designing Rasch model based item banks.* Paper presented at the annual meeting of the Psychometric Society, Princeton, NJ.

Lord, F. M. (1980). *Applications of item response theory to practical testing problems.* Hillsdale, NJ: Erlbaum.

Luecht, R. D. (1998). Computer-assisted test assembly using optimization heuristics. *Applied Psychological Measurement, 22,* 224-236.

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization.* New York: John Wiley & Sons.

Sanders, P. F., Verschoor, A. J. (1998). Parallel test construction using classical item parameters. *Applied Psychological Measurement, 22,* 212-223.

Stocking, M. L., & Swanson, L. (1998). Optimal design of item pools for computerized adaptive tests. *Applied Psychological Measurement, 22,* 271-279.

Swanson, L. & Stocking, M. L. (1993). A model and heuristic for solving very large item selection problems. *Applied Psychological Measurement, 17,* 151-166.

Timminga, E., van der Linden, W. J., & Schweizer, D. A. (1996). *ConTEST [Computer program and manual].* Groningen, The Netherlands: iec ProGAMMA.

Theunissen, T. J. J. M. (1985). Binary programming and test design. *Psychometrika, 50,* 411-420.

Theunissen, T. J. J. M. (1996). *Combinatorial issues in test construction.* Unpublished doctoral dissertation, University of Amsterdam, The Netherlands.

Timminga, E. & Adema, J. J. (1995). Test construction from item banks. In G. H. Fischer & I. W. Molenaar (Eds.), *The Rasch model: Foundations, recent developments, and applications* (pp. 111-127). New York: Springer-Verlag.

van der Linden, W. J. (1994). Optimum design in item response theory: Applications to test assembly and item calibration. In G.H. Fischer & D. Laming (Eds.), *Contributions to mathematical psychology, psychometrics, and methodology* (pp. 308-318). New York: Springer-Verlag.

van der Linden, W. J. (1996). Assembling test for the measurement of multiple traits. *Applied Psychological Measurement, 20,* 373-388.

van der Linden, W. J. (Ed.). (1998). Optimal test assembly [Special issue]. *Applied Psychological Measurement, 22*(3).

van der Linden, W. J. (1998). Optimal assembly of psychological and educational tests. *Applied Psychological Measurement, 22,* 195-211.

van der Linden, W. J. (submitted). *Optimal assembly of tests with item sets.*

van der Linden, W. J., & Adema, J. J. (1998). Simultaneous assembly of multiple test forms. *Journal of Educational Measurement, 35*(3), 185-198.

van der Linden, W. J., & Boekkooi-Timminga, E. (1989). A maximin model for test design with practical constraints. *Psychometrika, 54,* 237-247.

Wagner, H.M. (1978). *Principles of operations research with applications to managerial decisions.* London: Prentice-Hall.

U.S. Department of Education
Office of Educational Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

NOTICE

Reproduction Basis

X	This document is covered by a signed "Reproduction Release (Blanket)" form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.
	This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").