

■ **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**



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.

Copyright© 2000 by Law School Admission Council, Inc.

All rights reserved. This report may not be reproduced or transmitted, in whole or in part, by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission of the publisher. For information, write: Communications, Law School Admission Council, Box 40, 661 Penn Street, Newtown, PA 18940-0040

LSAT® and the Law Services logo are registered marks of the Law School Admission Council, Inc.

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	1
Abstract	1
Introduction	1
Analysis of Design Problems	2
<i>Categorical Constraints</i>	2
<i>Quantitative Constraints</i>	3
<i>Constraints on Interdependent Items</i>	4
Models for Item Pool Design	5
<i>Pool of Items</i>	5
<i>Pool of Stimuli</i>	6
<i>Assigning Items to Stimuli</i>	7
Models for Item Pool Management	7
Empirical Example.	8
Concluding Remarks	9
References	10

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:

$$\begin{aligned} r_{12} &= 6; \\ r_{.1} &= 4; \\ r_{1.} &= 8. \end{aligned}$$

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

	F1	F2		F1	F2	
C1	n_{11}	n_{21}	$n_{.1}$	C1	r_{11}	r_{21} 4
C2	n_{12}	n_{22}	$n_{.2}$	C2	6	r_{22} $r_{.2}$
C3	n_{13}	n_{23}	$n_{.3}$	C3	r_{13}	r_{23} $r_{.3}$
	$n_{1.}$	$n_{2.}$	$n_{..}$		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 Constraints

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 $d = 1, \dots, D$ 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 $\phi(x_{cq})$, 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 $\phi(x_{cq}) = x_{cq}^{-1}$, 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 $\phi(x_{cq})$, 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 $C \times Q$ 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 $f = 1, \dots, F$ 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 $T_f(\theta_k)$, $k = 1, \dots, K$, 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:

$$\text{minimize } \sum_f \sum_c \sum_q \phi_{cq} n_{fcq} \quad (\text{minimizing costs}) \quad (2)$$

subject to

$$\sum_c \sum_q I_q(\theta_k) n_{fcq} \geq T_f(\theta_k), \quad f = 1, \dots, F, \quad k = 1, \dots, K, \quad (\text{test information}) \quad (3)$$

