An Integer Programming Approach to Item Bank Design

Wim J. van der Linden and Bernard P. Veldkamp, University of Twente

Lynda M. Reese, Law School Admission Council

An integer programming approach to item bank design is presented that can be used to calculate an optimal blueprint for an item bank, in order to support an existing testing program. The results are optimal in that they minimize the effort involved in producing the items as revealed by current item writing patterns. Also presented is an adaptation of the models, which can be used as a set of monitoring tools in item bank management. The approach is demonstrated empirically for an item bank that was designed for the Law School Admission Test. *Index terms: integer programming, item bank design, item response theory, linear programming, optimal test assembly, test assembly, test specifications.*

A variety of methods for automated assembly of test forms from an item bank have recently become available. These methods can be classified as belonging to one of the following classes (van der Linden, 1998a): (1) heuristics that select items sequentially, in order to match a target for the test information function (TIF) 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 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 (e.g., Armstrong & Jones, 1992; Armstrong, Jones, & Wang, 1994, 1995; Armstrong, Jones, & Wu, 1992); or (4) methods based on optimal design theory from statistics (e.g., Berger, 1994). Detailed descriptions and examples of these methods are given by van der Linden (1998b).

These four test-assembly methods result in tests that are optimal or close to optimal. However, even when the tests are optimal, the results are not necessarily satisfactory because the composition of the item bank imposes an important constraint on the quality of the tests. For example, an item bank could have enough items with the content attributes required by the test specifications, but not enough items with the required values for their statistical attributes. This could easily occur if an item bank is frequently used and certain categories of its items are depleted quickly. The result of a case like this is an optimal test with an information function that is too low on a relevant interval of the trait scale. Although the problem of item bank depletion is less likely to occur in larger item banks, they are not necessarily better. Often a considerable proportion of items in a bank might never be used. The presence of such items can be the result of attribute values either not needed by the test specifications or overrepresented in the bank. The costs of screening and pretesting items with unneeded or overrepresented attribute values are generally high, and typically involve a considerable loss of resources.

Applied Psychological Measurement, Vol. 24 No. 2, June 2000, 139–150

This paper presents an integer-programming method for item bank design. This method results in a *blueprint* for an item bank—a document that specifies the attributes of the items needed in a new bank or an extension of an existing bank. As will become clear, the blueprint is designed to allow for the assembly of a prespecified number of test forms from the bank, each with its own set of specifications. The blueprint is also optimal in the sense that the effort or "costs" involved in creating the item bank are minimized. A favorable consequence is that the number of unused items in the bank is also minimized.

Writing test items to satisfy a blueprint is difficult, not so much because of the content attributes of the items, but because of their statistical attributes. The values of statistical attributes for individual items are not easily predicted, but at the bank level they often show persistent patterns of correlation with content attributes. These patterns are used here to derive an empirical measure for item-writing effort that is minimized in the design model.

Item banks are not static; tests are assembled and then released, or obsolete items are removed. In most testing programs, new items are written and pretested on a continuous basis. Although the method presented here is for the design of a single item bank, it can serve as a tool for monitoring the item writing process on a continuous basis. The slight adaptation of the models needed for the latter use is introduced below.

The problem of item bank design has been addressed earlier by Boekkooi-Timminga (1991) and Stocking & Swanson (1998). Boekkooi-Timminga also used integer programming to calculate the number of items needed for future test forms. However, she used a sequential approach that maximized the TIF under the one-parameter logistic (Rasch) model. These results were then used to improve on the composition of an existing item bank. The model described here directly calculates a blueprint for the entire item bank. In addition, its objective is to minimize the costs of producing the bank by the current item writers, rather than maximizing the TIF. At the same time, the model guarantees that the targets for the TIF are met. The present model is not restricted to items calibrated under the Rasch model. Stocking & Swanson (1998) also did not use the Rasch model; they presented a method for assigning items from a master bank into a set of smaller banks for use in an adaptive testing program that are rotated during testing to minimize item security problems.

Analysis of the Design Problem

An important distinction between test specifications or constraints in mathematical programming models for test assembly is that between constraints on categorical item attributes, constraints on quantitative attributes, and constraints needed to represent inter-item dependencies (van der Linden, 1998a).

Categorical Constraints

A categorical item attribute (e.g., item content, cognitive level, format, author, answer key) partitions the item bank into a series of subsets. A test specification with respect to a constraint on a categorical attribute restricts the distribution of items in a test over these subsets. If the items are coded by multiple attributes, then their Cartesian product can be used to partition the bank. In this case, constraints on categorical attributes address the marginal distributions of the items, as well as the joint and conditional distributions.

An example for the case of two categorical attributes 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). The unconstrained distribution of the items is represented by n_{ij}, the number of items in cell (i, j); $n_{i.}$, the number of items in row i; $n_{.j}$, the number of items in

column j; and $n_{..}$, the total number of items. For the constrained distribution, 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: $r_{21} = 6$; $r_{1.} = 4$; and $r_{.1} = 8$. Note that this set of constraints not only fixes certain numbers directly, but also restricts the values possible for the other numbers in Table 1. For example, the first and last constraint together imply $r_{11} + r_{13} \leq 2$. This allows the same set of test specifications to be represented by different sets of constraints. Some of these sets might be smaller and, therefore, more efficient than others. The method presented here, however, is neutral with respect to such differences.

Table 1
Unconstrained Distribution of Items in the Bank and Constrained Distribution of Items in a Test Form [Two Categorical Attributes, Item Content (C1, C2, and C3) and Item Format (F1 and F2)]

	Unconstrained			Constrained		
	F1	F2	Total	F1	F2	Total
C1	n_{11}	n_{21}	$n_{1.}$	r_{11}	r_{21}	4
C2	n_{12}	n_{22}	$n_{2.}$	6	r_{22}	$r_{2.}$
C3	n_{13}	n_{23}	$n_{3.}$	r_{13}	r_{23}	$r_{3.}$
Total	$n_{.1}$	$n_{.2}$	$n_{..}$	8	$r_{.2}$	$r_{..}$

In a test assembly problem, values for r_{ij} are sought that allow the constraints on all distributions to be met and the combination of the values to optimize an objective function. In so doing, n_{ij} is fixed and serves as an upper bound to r_{ij}. The basic approach presented here is to reverse the role of these two quantities. n_{ij} is taken as the decision variable, and a function of the n_{ij} is optimized subject to all constraint sets involved by the specifications of the tests to be constructed from the bank.

Quantitative Constraints

Examples of quantitative item attributes include word counts, exposure rates, values for item response theory (IRT) information functions, expected response times, and classical test theory statistics. Unlike categorical constraints, quantitative constraints do not impose bounds directly on numbers of items, but instead impose them on a function of their joint attribute values, usually a sum or average.

In IRT-based test assembly, constraints on the TIF are important. They typically require the sum of values of the item information functions to meet certain bounds. Although each combination of item information functions defines a unique TIF, the reverse does not hold.

Unlike categorical attributes, constraints with quantitative attributes have no one-to-one correspondence with item distributions. Instead, they correspond with sets of distributions that are feasible with respect to the constraints. However, this should not be viewed as a disadvantage. It can be used to select a distribution to represent a quantitative constraint that is optimal with respect to an objective function. This is the approach used here. All constraints on quantitative attributes are translated into an optimal distribution of the items over tables defined by a selection of these values.

Choosing an objective function for the selection of an optimal item distribution can be used to solve the problem of writing items with prespecified values for their statistical attributes. That is,

it is possible to minimize an explicit objective function that measures the costs or effort involved in writing items with various possible combinations of attribute values.

There are several possible options for choosing a cost function. One is to assess the costs directly. This can be done, for example, by asking item writers to time their activities or to analyze the data in the log files (e.g., time elapsed, number of changes in items) produced by their word processors. A less time-consuming option is to use the distribution of statistical attributes for a recent item bank in order to define the costs involved in writing items with certain combinations of attribute values. This option assumes that items with combinations of attribute values with higher frequencies are easier or less "costly" to produce. The empirical example below elaborates on this option.

More realistic information about 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 be formalized for constraints on TIFs. However, the treatment also can easily be generalized for constraints on other quantitative attributes.

Example of a Cost Function

In this empirical example, the 3-parameter logistic model (3PLM) was used to calibrate the existing item bank (Lord, 1980, chap. 2):

$$p_i(\theta) \equiv P(U_i = 1|\theta) \equiv c_i + (1 - c_i)\{1 + \exp[-a_i(\theta - b_i)]\}^{-1}\,, \tag{1}$$

where

θ is the unknown trait level of an examinee,
$a_i \in [0, \infty]$ is the discrimination parameter for item i,
$b_i \in [-\infty, \infty]$ is the item difficulty parameter, and
$c_i \in [0, 1]$ is the pseudo-guessing parameter.

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 $d = 1, 2, \ldots, D$ grid points. The number of points on the grid, as well as their spacing, is arbitrary. Let Q be the table defined by the product of these grids for all quantitative attributes, with arbitrary cell q. C represents the full table defined by the categorical attributes, with an arbitrary cell $c \in C$. A cell in the joint table defined by C and Q is denoted as $(c, q) \in C \times Q$. Because the table is the product of the attributes, the number of cells can become very large (e.g., Table 2).

Table 2
Number of Item Attributes, Constraints, and Decision Variables for the Three Sections of the LSAT

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

*Not applicable.

Let x_{cq} denote the frequency of the items in cell (c, q) for a representative item bank. x_{cq} contains information on the effort 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. Such items are easy to produce. Empty cells point to combinations of attribute values that are difficult to produce.

A monotonically decreasing function of x_{cq}, denoted as $\varphi(x_{cq})$, is used as an empirical measure of the effort involved in writing items with the various possible combinations of attribute values. The generic term "cost function" is used for this function. In the model below, the costs for writing the new item bank are minimized using $\varphi(x_{cq})$.

A simple cost function is $\varphi(x_{cq}) = x_{cq}^{-1}$, which requires $x_{cq} > 0$. This function is based on the assumption that the smaller the supply of items with attribute values (c, q) in the existing bank, the more difficult it is to produce these items. Although other functions could be used, it will be obvious from Equation 3 that the choice of unit does not matter.

The cost function can be made more realistic by adding item writers as a categorical attribute to the table, if there are different patterns of correlation among the attributes in the bank for different item writers. When this option is used, a constraint on the number of items or stimuli to be written by each item writer must be added to the design models (see below). The blueprint of the new item bank then automatically shows which types of items must be written by whom.

If the existing item bank is relatively small, collapsing attributes in $C \times Q$ that show no substantial dependencies on any of the other attributes is recommended. This results in larger x_{cq} values, which result in more stable marginal cost estimates. In the objective function and constraints in the models below, these marginal cost estimates are substituted for the cells over which collapsing has occurred. The operation is thus not meant to reduce the size of the original design problem. The models still provide complete blueprints for the items in the bank.

The use of a cost function defined on item writing practices for a recent item bank is *not* conservative in the sense that old practices are automatically continued. The new item bank 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 those specifications. However, a potentially large set of item banks 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 on interdependent items deal with possible relations of exclusion and inclusion between the items in the bank. Items exclude each other when they overlap in content and, as a consequence, one item contains a clue to the correct answer to the other item (i.e., 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 items. In the present authors' experience, sets of enemies in existing pools are usually small (e.g., 2–3 items per set) in relation to the number of tests the bank has to serve. In test assembly based on 0-1 linear programming, it is possible to constrain the test so that there is no more than one item from each set of enemies. If enemies are present in an item bank, they can, thus, generally be distributed over different test forms. The position taken here is that sets of enemies in the bank are a problem of *test assembly*. They can therefore be ignored in the context of the item bank design problem.

An important type of inclusion relation exists between items that are organized around common stimuli. Two examples are a reading passage in a reading comprehension test and a description of an experiment in a biology test. "Item sets" is used here as a generic term for this part of a test.

Typically, the items in these sets are selected from larger sets available in the bank. 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 linear programming models for test assembly are available that deal with banks containing item sets (van der Linden, in press).

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

1. A blueprint for the collection of all *items* in the bank (both discrete and set-based items) is designed using the integer programming model in Equations 3–7 (below), 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.
2. A blueprint for a bank of *stimuli* for the item sets is designed using the same methods as for the bank 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.
3. 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 for the numbers of items available in the various cells of the $C \times Q$ table and the numbers required in the item sets. The objective function is of the same type as above; its specific form is explained below.

These steps are taken separately. They serve only to design an item bank with a specified structure. Of course, if the design is realized, then the actual stimuli and items in the sets are written simultaneously and in a coordinated fashion.

Models for Item Bank Design

Item Bank

The individual test forms the item bank should support are designated by $f = 1, 2, \ldots, F$. As an example of a quantitative attribute, the TIF of test form f is required to be equal to or greater than a set of target values, $T_f(\theta_k)$, $k = 1, 2, \ldots, K$. 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 $q \in Q$ is denoted $I_q(\theta)$. This quantity is calculated, for example, for the midpoints of the intervals of the item parameter values defining cell q of Q. The decision variables in the model are integer variables n_{fcq}. These variables represent the number of items in cell (c, q) required in the bank to support form f. That is, n_{fcq} indicate how many items with item parameter values represented by q and categorical attributes represented by c are required for form f. The complete bank is, thus, defined by

$$\sum_{f=1}^{F} n_{fcq} \, . \tag{2}$$

The model is

$$\text{minimize} \sum_{f} \sum_{c} \sum_{q} \varphi_{cq} n_{fcq} \, , \qquad \text{(minimim costs)} \tag{3}$$

subject to

$$\sum_{c} \sum_{q} I_q(\theta_k) n_{fcq} \geq T_f(\theta_k) \, , \quad f = 1, 2, \ldots, F, \; k = 1, 2, \ldots, K \, , \qquad \text{(test information)} \tag{4}$$

$$\sum_{c \in V_{fg}} \sum_{q} n_{fcq} \geq n_{fg} , \qquad f = 1, 2, \ldots, F, \; g = 1, 2, \ldots, G , \qquad \text{(categorical constraints)} \quad (5)$$

$$\sum_{c} \sum_{q} n_{fcq} \geq n_{f} , \qquad f = 1, 2, \ldots, F , \qquad \text{(length of forms)} \quad (6)$$

and

$$n_{fcq} = 0, 1, 2, \ldots , \qquad f = 1, 2, \ldots, F , c \in C , q \in Q . \qquad \text{(integer variables)} \quad (7)$$

The objective function in Equation 3 minimizes the sum of the item writing costs across all items in the F forms. For each form, the constraints in Equation 4 require the TIF to be larger than the target TIF at θ_k, $k = 1, 2, \ldots, K$. Because the objective function in Equation 3 implies a minimum number of items to be written, the TIF will approach the target TIF from above. The categorical constraints imposed on the forms are formulated in Equation 5. V_{fg} ($f = 1, 2, \ldots, F$ and $g = 1, 2, \ldots, G$) are the sets of cells in C on which the constraints must be imposed. For example, in the set of constraints in Table 1, the first constraint is imposed on the set of cells consisting only of cell (1,1), the second constraint on the set of cells in Row 1, the third on the set of cells in Column 1. Only lower bounds, n_{fg}, are set. The objective function in Equation 3 guarantees that the constraints are satisfied as an equality at optimality. The same happens to the constraints on the length of the forms in Equation 6. Other quantitative constraints can be added to the model following the same logic as in Equation 4.

Models for the simultaneous assembly of sets of tests from a given bank 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 are not of concern here. The current goal is only to determine how many items of certain types are needed in the item bank to assemble a specified set of test forms.

For smaller $C \times Q$ tables, the optimal values of the variables in the model can be calculated using an available implementation of the branch-and-bound algorithm for integer programming (e.g., Nemhauser & Wolsey, 1988; Adema, 1992). For larger tables, optimal values can 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 up. As already noted, it is always prudent to have a few spare items in the bank.

Stimulus Pool

Tables C' and Q' can be designated for the sets of categorical and quantitative attributes used to describe the stimuli in the test forms the item bank must support. Because psychometric attributes for stimuli are rare, Table Q' is expected to be much smaller than Q. Item sets often have aggregated statistical attributes, such as sums of proportion-correct values or average values for b_i. 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 3–7. The cost function $\varphi_{c'q'}$ is now defined for the distribution of stimuli in the previous item bank. Likewise, the bounds in Equations 4–5 are derived from the specifications for the item sets in the various test forms. Word counts are an example of a quantitative attribute for stimuli. 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

$$\sum_{c'}\sum_{q'} w_{q'} n_{fc'q'} \geq w_f \ , \quad f = 1, 2, \ldots, F \ . \qquad \text{(word counts)} \quad (8)$$

Again, because of minimization in Equation 3, the bounds on the constraints in Equation 8 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

$$\sum_{f} n_{fc'q'} \ . \qquad (9)$$

Assigning Items to Stimuli

To introduce the item assignment model, $s_f = 1, 2, \ldots, S_f$ denotes the item sets in form 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 above model for the pool of stimuli. 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 on (c', q') will remain implicit.

In addition, integer decision variables $z_{s_f cq}$ are defined to denote the number of items from cell (c, q) in the item table assigned to s_f. A cost function is defined on the Cartesian product of 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

$$\text{minimize} \sum_{f}\sum_{s_f}\sum_{c}\sum_{q} \varphi_{cqc'q'} z_{s_f cq} \ , \qquad \text{(minimizing costs)} \quad (10)$$

subject to

$$\sum_{c,q} z_{s_f cq} \geq n_{s_f} \ , \quad s_f = 1, 2, \ldots, S_f \ , \quad f = 1, 2, \ldots, F \ , \qquad \text{(the number of items needed)} \quad (11)$$

$$\sum_{f}\sum_{s_f} z_{s_f cq} \leq n_{cq} \ , \quad c \in C \ , \quad q \in Q \ , \qquad \text{(the number of items available)} \quad (12)$$

and

$$z_{s_f cq} = 0, 1, 2, \ldots, \quad s_f = 1, 2, \ldots, S_f \ ,$$
$$f = 1, 2, \ldots, F \ , \quad c \in C \ , \quad q \in Q \ . \qquad \text{(integer variables)} \quad (13)$$

The constraints in Equation 11 assign n_{s_f} items to s_f, whereas the constraints in Equation 12 ensure that no more items are assigned from cell (c, q) than are available in the blueprint for the item bank.

If constraints on aggregated quantitative item attributes must be imposed on some of the item sets, the model must be expanded. For example, if s_f must have an average proportion correct

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:

$$\sum_c \sum_q p_q z_{s_f cq} \geq n_{s_f} p_{s_f}^{(l)} , \quad s_f = 1, 2, \ldots, S_f , \quad f = 1, 2, \ldots, F , \quad \text{(lower bounds on } \bar{p}) \quad (14)$$

and

$$\sum_c \sum_q p_q z_{s_f cq} \leq n_{s_f} p_{s_f}^{(u)} , \quad s_f = 1, 2, \ldots, S_f , \quad f = 1, 2, \ldots, F . \quad \text{(upper bounds on } \bar{p}) \quad (15)$$

Models for Item Bank Management

As already noted, item banks are not static. In most testing programs, tests are assembled from the bank and new items are pretested on a continuous basis. Hence, two important tasks of item bank management are: (1) monitoring the developments in the item bank, and (2) instructing item writers on writing new items in order to complete the bank.

The models presented here can easily be adapted for use in item bank management. However, the decision variables in the models must be corrected for the number of items and stimuli available in the bank. This principle is illustrated for the model in Equations 3–7. Let v_{cq} be a constant representing the current number of items in cell (c, q) and let η_{cq} be a new decision variable denoting the number of items yet to be written for cell (c, q). The only adaptation necessary is substituting $v_{cq} + \eta_{cq}$ for the old decision variables in the model.

If the current items in the bank reveal new patterns of correlation between categorical and quantitative attributes, φ_{cq} can be updated. This is done by defining them on v_{cq} rather than using x_{cq} for the previous item bank, or perhaps on a weighted combination of both. Defining φ_{cq} on both is recommended, for example, if the item writers form a categorical attribute in the definition of $C \times Q$ and new item writers are contributing to the bank.

Example Application

The method described above was used to design a new item bank for the Law School Admission Test (LSAT). The LSAT bank must support three different sections in the test, labeled here as A, B, and C. Sections A and B 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 TIFs. The sets of constraints included such attributes as item (sub)types, item-set structures in the bank, types of stimuli, gender and minority orientation of the stimuli, answer key distributions, word counts, and TIFs. Also, a previous item bank was available to estimate a cost function. Because the original authors of the bank could not be identified, author was not used as an attribute. 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. The numbers of attributes for each section in the LSAT are given in Table 2.

The item bank was designed to support thirty forms. There were ten regular forms, ten forms with a target for the TIF shifted .6 to the left on the θ continuum, and ten with a target shifted .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 number of decision variables and constraints in the models for the item bank, stimulus pool, and assignment of the items to the stimuli are given in Table 2.

A previous bank 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 bank was small relative to the number of cells in the attribute tables, the tables were collapsed over some attributes before computing the cost function.

For example, because the values of the items for the guessing parameter in the 3PLM in Equation 1 (c_i) did not vary much, that parameter was not included as a dimension of the attributes table $C \times Q$. Instead, its average value was used 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 to obtain more stable estimates of the frequencies in the cost function. Collapsed cells in the attribute tables received a value for the cost function based on their marginal frequencies. However, the integer programming models were formulated over the full table, with marginal costs substituted for the collapsed cells.

Because the three sections in the LSAT had no overlap in items, three independent models needed 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 that the bank was assumed to support involved no constraints on interdependent items. Also, the objective functions in the models were sums of costs across these sets, and were minimal when the costs for each set were minimal. The models could, therefore, be solved independently for each set.

The best strategy to solve models of the size in the current application was through the simplex algorithm for the relaxed version of the models. The integer variables were replaced by (non-negative) real-valued variables, and the solutions were rounded upward. 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 was approximately one second of CPU time on a Pentium 133 MHz processor. No rounding appeared to be necessary; the algorithm found a direct integer solution for all variables. This occurred 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 does not generalize automatically to other applications of the integer programming models for item bank design presented here, but upward rounding of variables is a simple and effective strategy that always works.

Conclusions

Although it was not demonstrated here, a minor correction to the decision variables is necessary in order to use the models as tools for managing item banks in ongoing testing programs. The same correction can be used to cope with possible future changes in the right-hand side constants in the integer programming models due to modifications in the test specifications.

As already indicated, several options to specify cost functions in Equations 3 and 10 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 applications rely heavily on the cost functions adopted. The question of which cost function is best does not have a universal answer. It can be best answered when implementing the method for an actual testing program.

For the example above, a previous item bank was 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 when the test specifications at the time of the previous bank address a set of item and stimulus attributes that includes the set of attributes for the new item bank. If new attributes are introduced, this approach might face an unsolvable missing data problem. However, too many old attributes

does not constitute a practical problem because the old table can always be collapsed over attributes that are no longer used in the testing program.

One case not dealt with here is when the tests assembled from the bank are allowed to have item overlap. If overlap is allowed, the number of possible tests from a given bank increases. Determining how many different tests are possible under this condition involves a complicated combinatorial problem (Theunissen, 1996). If the overlap is small, an appropriate strategy might be to simply ignore it; the resulting item bank might be somewhat too large, but it will allow for all planned tests to be assembled. However, the problem of designing an item bank of minimal size in order to support tests with large overlap remains 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-stage 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* (Volume 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,* 101–112.

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: Wiley.

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.

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.

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

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 tests for the measurement of multiple traits. *Applied Psychological Measurement, 20,* 373–388.

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

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

van der Linden, W. J. (in press). Optimal assembly of tests with item sets. *Applied Psychological Measurement.*

van der Linden, W. J., & Adema, J. J. (1998). Simultaneous assembly of multiple test forms. *Journal of Educational Measurement, 35,* 185–198 (Erratum in Vol. 34, 90–91).

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.

Acknowledgments

This study received funding from the Law School Admission Council (LSAC). The opinions and conclusions contained in this paper are those of the authors and do not necessarily reflect the position or policy of LSAC.

Author's Address

Send requests for reprints or further information to W. J. van der Linden, Department of Educational Measurement and Data Analysis, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands. Email: vanderlinden@edte.utwente.nl.