A Model and Heuristic For Solving Very Large Item Selection Problems

Len Swanson and Martha L. Stocking

Educational Testing Service

A model for solving very large item selection problems is presented. The model builds on previous work in binary programming applied to test construction. Expert test construction practices are applied to situations in which all specifications for item selection cannot necessarily be met. A heuristic for selecting items that satisfy the constraints in the model also is presented. The heuristic is particularly useful for situations in which the size of the test construction problem exceeds the limits of current implementations of linear programming algorithms. A variety of test construction problems involving real test specifications and item data from actual test assemblies were investigated using the model and the heuristic. *Index terms: expert systems, heuristic algorithms, item response theory, linear programming, mathematical programming, test assembly, test construction, test design.*

There has been considerable interest recently in methods of automated item selection in test construction (e.g., Ackerman, 1989; Adema, 1990; Boekkooi-Timminga, 1989; Theunissen, 1985; Van der Linden, 1987; Yen, 1983). These methods, frequently implemented on microcomputers, provide several potential benefits to test specialists, including reducing labor costs and increasing the efficiency of test construction. They can also increase the consistency of successive forms of a test through the development and codification of specifications that are refined and perfected by test experts.

Much of the recent interest in automated item selection methods has been stimulated by the development of large-scale computerized item banks (e.g., Boekkooi-Timminga, 1989; Hsu & Sadock, 1985). These item banks often contain very detailed statistical and content-related information about hundreds or, in many cases, thousands of test items. Computerized item banks make it possible for the test specialist to rapidly assemble representative test forms, review the psychometric characteristics and content attributes of those forms, refine the test specifications as needed, and repeat the process until a satisfactory test is assembled.

Research in this area also has been stimulated by developments in item response theory (IRT; Lord, 1952, 1980). Under IRT, the test information function (TIF) is the sum of the independent item information functions for the items in a test (Lord, 1980). The additivity of item information makes it possible to construct tests to target test information functions (TTIFs), as suggested by Birnbaum (1968) and Lord (1980).

Test construction problems have been formulated as binary programming problems, which offers several advantages. The problems are binary in the sense that an item is or is not included in the test. This is not to be confused with binary response models, in which items are scored correct or incorrect. The methods described in this paper are independent of the way in which the items are scored. Viewing test construction as a binary programming problem allows statistical and nonstatistical test specifications to be expressed in mathematical terms, as constraints on test optimization. The

APPLIED PSYCHOLOGICAL MEASUREMENT
Vol. 17, No. 2, June 1993, pp. 151-166

0146-6216/93/020151-16$2.05

binary programming formulation also allows test construction problems to be solved with classical linear programming algorithms.

Several possible formulations of test construction as a binary programming problem are possible and many have been researched and reported in the literature (e.g., Boekkooi-Timminga, 1989; Theunissen, 1985; Van der Linden, 1987). These formulations and algorithms for solving them (i.e., for finding a set of items that satisfy the model constraints) have been used successfully, although generally not with very large item pools and/or very large numbers of test specifications. The model presented below is particularly well suited to very large test construction problems. It builds on previous formulations and can be solved using standard binary programming techniques.

Test Construction Constraints and Binary Programming

Test Construction Constraints

Test specifications are rules for including items in a test. These rules invariably include constraints on both psychometric and content-related item properties, and may also include consideration of other properties not directly related to content or statistical behavior. Psychometric constraints can be IRT-based—for example, conformance to a TTIF (Lord, 1980). Alternatively, they can be based on conventional statistics, such as conformance to frequency distributions of item difficulty and/or discrimination.

Nonpsychometric constraints can be of many types. Content restrictions set boundaries on the number of items with a particular content classification that may be included in the test. For example, a mathematics test might include a specified number of geometry, algebra, and trigonometry items. Content specifications typically involve a hierarchy of content-related classifications and are often several levels deep. The number of constraints imposed by these content classification levels can be quite large, and constraints are often not mutually exclusive (i.e., an item may contribute to the satisfaction of several constraints simultaneously).

Item format or item type restrictions constrain the number of items having each of the possible formats in the test. For example, a verbal test might include specifications for the number of sentence completion, word matching, and sentence correction items to be administered. Similarly, specifications along cognitive dimensions might constrain the number of items that deal with computational skills, the number that require abstract reasoning, and so forth. As with content specifications, these restrictions might involve several levels of classification.

Other item attributes may be included in test specifications. For example, limitations may be placed on the number of items written by a particular item writer. Constraints may be imposed on the number of items that reference particular groups (e.g., the number of references to minorities, males, or females). The positions of the correct alternatives in a multiple-choice test also may be required to conform to a prespecified distribution. Or it may be necessary to avoid selecting items that are related to each other in undesirable ways, such as when one item suggests the correct answer to another (referred to as item overlap).

Many other examples of test specifications based on item attributes are possible. Van der Linden & Boekkooi-Timminga (1989) provided an extensive discussion of possible test constraints. Table 1 shows several examples of nonpsychometric test constraints. The column labeled "lower bound" shows the minimum number of items with the specified property that must be included in the test; the column labeled "upper bound" shows the maximum number of items that can be included.

All these test specifications can be expressed mathematically as linear constraints, in the tradition of linear programming. For example, a specification such as "select at least two but no more than

Table 1
Typical Nonpsychometric Test Specifications

Item Property	Lower Bound	Upper Bound
Geometry Content	10	15
Algebra Content	12	15
Trigonometry Content	10	12
Abstract Reasoning Skills	15	20
Computation Skills	15	20
Female Reference	0	5
Minority Reference	0	5

five geometry items'' takes the form

$$2 \leq Y \leq 5 , \tag{1}$$

where Y is the number of selected items having the property ''geometry.'' Conformance to a specified frequency distribution of item difficulties takes the form of upper and lower bounds on the number of selected items falling into each specified item difficulty range.

Similarly, conformance to a TTIF takes the form of upper and lower bounds on the sum of the individual item information functions at selected trait levels. This is based on the premise that it is adequate to consider the TIF at discrete trait levels. This is a reasonable assumption given that TIFs are typically relatively smooth and that trait levels can be selected to be arbitrarily close to each other (Van der Linden, 1987).

Binary Programming Models

A typical formulation of the binary programming model has the following mathematical form. Let $i = 1, \ldots, N$ index the items in the pool, and let x_i denote the decision variable that determines whether item i is included in ($x_i = 1$) or excluded from ($x_i = 0$) the test. Let $j = 1, \ldots, J$ index the item properties associated with the nonpsychometric constraints; let L_j and U_j be the lower and upper bounds (which may be equal), respectively, on the number of items in the test having each property; and let a_{ij} be 1 if item i has property j and 0 if it does not. Then, the model optimizing the objective function z (discussed below) for a test of fixed length n is specified as:

Minimize (or maximize) z subject to:

$$\sum_{i=1}^{N} x_i = n , \tag{2}$$

$$\sum_{i=1}^{N} a_{ij} x_i \geq L_j, \quad j = 1, \ldots, J , \tag{3}$$

$$\sum_{i=1}^{N} a_{ij} x_i \leq U_j, \quad j = 1, \ldots, J , \tag{4}$$

and

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, N . \tag{5}$$

Equation 2 fixes the test length, and Equations 3 and 4 express the nonpsychometric constraints as lower and upper bounds on the number of items in the test with the specified properties.

The objective function, z, can take on several possible forms (Van der Linden & Boekkooi-Timminga, 1989, Table 3). It typically maximizes conformance to the psychometric constraints. Examples

include maximizing absolute test information, minimizing the sum of the positive deviations from the target test information, or minimizing the largest positive deviation from the target. Models that minimize the maximum deviation from an absolute or relative target are referred to as "minimax" models (Van der Linden, 1987; Van der Linden & Boekkooi-Timminga, 1989). z can also take the form of minimizing test length (Theunissen, 1985), or minimizing other characteristics of the test, such as administration time or frequency of item administration. Finally, z can be a dummy variable that is simply used to cast the problem into a linear programming framework. Boekkooi-Timminga (1989) provided a thorough discussion of possible objective functions.

A Model for Solving Large Test Construction Problems

Feasibility of Models

If the binary programming model described above is feasible (i.e, has an integer solution), it can be solved using standard mixed integer linear programming (MILP) algorithms (e.g., Nemhauser & Wolsey, 1988). Considerable attention also has been devoted to methods of speeding up the MILP procedure (e.g., Adema, 1988; Boekkooi-Timminga, 1989).

Binary programming models, together with various procedures and heuristics for solving them, have been successful in solving many test construction problems. However, it is not always the case that the model has a feasible solution. This may occur because one or more of the constraints in Equation 3 or 4 is difficult or impossible to satisfy, or simply because the item pool does not have sufficient items to satisfy all constraints simultaneously. In general, the binary programming model is more likely to be infeasible when the number of constraints is large because of the complexity of the interaction of constraints.

Studies reported in the literature generally have dealt with relatively small problems: Item pool sizes have been 1,000 or less, and the number of constraints typically has been less than 50 (e.g., Adema, 1988; Kester, 1988; Theunissen, 1985; Van der Linden & Boekkooi-Timminga, 1989). In practice, pool sizes range from 300 to 5,000, and the number of constraints ranges from 50 to 300. Moreover, many, if not most, of these constraints are not mutually exclusive, so that it is not possible to use them to partition the pool into mutually independent subsets. Problems of this size, with this degree of constraint interaction, greatly increase the likelihood that the model specified above will not have a feasible solution.

Heuristic procedures for solving the model often resolve the feasibility problem. For example, Adema (1988) derived a (relaxed) linear solution by removing Equation 5 to permit decision variables to take on values between 0 and 1. This allows the problem to be solved using linear programming algorithms. Various rules then can be used to set decision variables to 0 or 1 (e.g., Adema, 1988; Van der Linden & Boekkooi-Timminga, 1989). Heuristics such as these are designed to reduce computer time, but in many cases they also ensure a feasible (if not optimal) solution to the binary model if there is a feasible solution to the (relaxed) linear model.

Constraints Versus Test Specifications

A more direct way of resolving the feasibility problem is to explicitly recognize the possibility of an infeasible solution and revise the problem statement to produce the best possible test under those conditions. Test specialists generally are concerned less with optimizing some function of the items selected (e.g., maximizing test information or minimizing test length), or even with meeting all the constraints, than they are with coming "as close as possible" to all constraints simultaneously. That is, test specialists recognize the possibility that all constraints and objectives cannot always be met,

but want to make certain that the extent to which they are not met is in some sense minimized. In fact, optimal solutions (i.e., solutions in which the objective function is minimized or maximized) are often unnecessary because test specialists will revise the final test based on intangible characteristics of items. This changes the statement of the test construction problem from one of constrained optimization to one of accommodating the inability to meet all specifications simultaneously, which will lead to a very different model from that given above.

These considerations suggest that constraints should be thought of more as "desired properties" than constraints in the mathematical sense of binary programming. The possibility of failing to meet some of these desired properties should be recognized, but the aggregate failures should be minimized. Moreover, some control over which specifications can be violated should be exercised. Some test specifications are very important and it may be better to sacrifice others for them; for example, if a reading test is to contain exactly one science passage, a test that included two passages or no passages would not be tolerated. Similarly, if one constraint is twice as important as another, or is twice as difficult to meet given the characteristics of the item pool, then its influence in determining the appropriateness of an individual item should be double that of the other. This suggests that the constraints, or desired properties, should be weighted so that test specialists can quantitatively prioritize them.

This leads to a reformulation of the goal: *minimize the weighted sum of positive deviations from the constraints.* The basic principle of the binary programming model is retained, but test specifications are now moved from the constraints—Equations 3 and 4—to the objective function. This is accomplished through a procedure commonly used in mathematical programming to make an infeasible model feasible (e.g., Brooke, Kendrick, & Meeraus, 1988, p. 159). The equations are rewritten so that they contain explicit "slack" variables that accommodate differences between the desired and obtained values. Boekkooi-Timminga (1989, p. 59) used a similar device for dealing with situations in which it may not be possible to meet all test specifications simultaneously. In that application, the technique was used not to select items, but rather to determine the optimal number of items to include in pool partitions.

With the introduction of slack variables, Equation 3 is replaced by

$$\sum_{i=1}^{N} a_{ij} x_i + d_{L_j} - e_{L_j} = L_j, \quad j = 1, \ldots, J, \tag{6}$$

where d_{L_j} and e_{L_j} are non-negative variables. The d_{L_j} are the positive (or 0) deviations from the lower bounds—that is, the differences between the lower bounds and the sums whenever the lower bounds are not met. Similarly, the e_{L_j} represent the differences between the sums and the lower bounds whenever the lower bounds are exceeded, and might be interpreted as the unneeded "surplus" quantity. Note that for a given j, one or both of these variables must take on the value 0 (i.e., the sum cannot both exceed and fail to meet the lower bound). The examples below will clarify this.

Similarly, Equation 4 is replaced by

$$\sum_{i=1}^{N} a_{ij} x_i - d_{U_j} + e_{U_j} = U_j, \quad j = 1, \ldots, J, \tag{7}$$

where d_{U_j} represents the non-negative difference between the sums and the upper bounds whenever the upper bounds are exceeded, and e_{U_j} represents the non-negative difference between the upper bounds and the sums whenever the upper bounds are not exceeded.

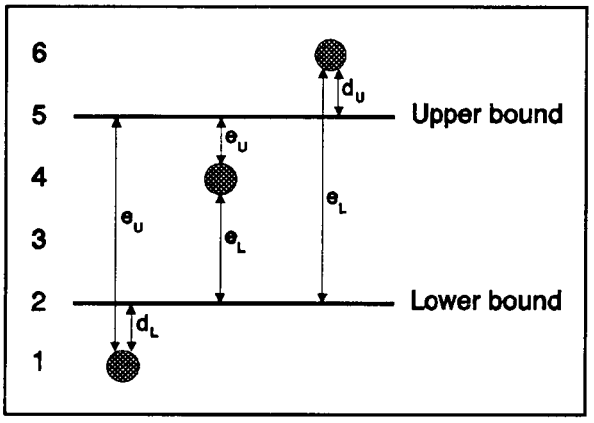
For example, consider the specification "at least 2 but no more than 5 geometry items," and consider the three situations in which 1, 4, and 6 geometry items are selected. The respective values of d and e for each of these situations are shown in Table 2. Note that the number of geometry items

Table 2
Examples of Property Deviations

Number of Geometry Items Chosen	d_L	e_L	d_U	e_U
1	1	0	0	4
4	0	2	0	1
6	0	4	1	0

selected deviates from the lower bound when 1 geometry item is selected and from the upper bound when 6 are selected. Figure 1 shows this graphically.

Figure 1
Examples of Property Deviations



Given this formulation of the model constraints, the objective function now becomes: Minimize

$$\sum_{j=1}^{J} w_j d_{L_j} + \sum_{j=1}^{J} w_j d_{U_j} , \tag{8}$$

where w_j is the weight assigned to constraint j.

For convenience, "constraints" will continue to be referred to in the sense in which test specialists think of them, recognizing that they are not constraints in the mathematical sense of binary programming. Equations 6-8 reflect the mathematical formalism by which the test specialist's "constraints" are transferred to the objective function. That is, in the original binary programming model the test specifications are also model constraints (Equations 3 and 4), which may be very difficult to satisfy simultaneously. These specifications now have been moved to the objective function and have been replaced with new model constraints (Equations 6 and 7) that are trivially satisfied by appropriate choices of the ds and es.

Target Test Information Functions

For IRT-based tests, conformance to a TTIF also can be considered as a set of test specifications. Let $r = 1, \ldots, R$ be points on the trait metric, θ, at which the test specialist wishes to constrain the TIF (e.g., points between -3.0 and +3.0). Let $I_i(\theta_r)$ be the item information for item i at θ_r. Let $I_L(\theta_r)$ and $I_U(\theta_r)$ be the lower and upper bounds, respectively, on test information at θ_r. Then con-

formance to the target test information is expressed by

$$\sum_{i=1}^{N} I_i(\theta_r)x_i + d_{L_r} - e_{L_r} = I_L(\theta_r), \quad r = 1, \ldots, R \tag{9}$$

and

$$\sum_{i=1}^{N} I_i(\theta_r)x_i - d_{U_r} + e_{U_r} = I_U(\theta_r), \quad r = 1, \ldots, R\,. \tag{10}$$

These equations are simply special forms of Equations 6 and 7, respectively. For test information constraints, the a_{ij} become item information at points on the trait metric, instead of 0 or 1, and the L_j and U_j become lower and upper bounds on information, rather than bounds on the number of items having a specified property. Thus, the nonpsychometric test specifications are formulated as bounds on the number of items having specified properties, and the TTIF constraint is expressed as bounds on information at a desired set of points on the trait metric.

This procedure also places a band on test information rather than simply maximizing it. There are two reasons for this. First, in many cases tests are required to be approximately parallel, so that one edition of the test does not measure differently than another. To accomplish this, both the minimum and maximum information provided by the test should be controlled. Second, and more importantly, if no limit is placed on information, the item selection process would tend to select the more informative items and exhaust the pool of those items. The psychometric quality of tests produced from a pool would thereby decrease over time (Boekkooi-Timminga, 1989). Establishing both upper and lower bounds on information addresses these problems.

Item Sets

An additional consideration in formulating a model appropriate to large item selection problems is dealing with item sets. An item set is a group of items related to each other through a common stimulus. Examples include a reading passage, a chart, or a graph, in which a set of questions refer to the common stimulus. Test specialists often have test specifications that apply to the set itself (or any subset of the set); for example, "select at most one set on medieval literature." In addition, they often select only a subset of the set to be included in a particular test (if the set is included at all). The items in that subset must, along with all other items included in the test, conform to the global psychometric and nonpsychometric test specifications.

If the test specialist has the option of selecting a subset, as opposed to the full set, then the number of possibilities to be considered is large. In particular, if a set has p items, then the number of possible nonempty subsets to be considered is $2^p - 1$ (in practice, test specialists avoid sets with one item, so the number is more typically $2^p - p - 1$). For example, for a 10-item set the test specialist would have to consider over 1,000 possibilities.

Item sets can be represented in the model by redefining the item pool to include all possible subsets of each set, as well as the discrete items (items not associated with any set). Thus, N becomes the number of discrete items plus subsets of sets, x_i indicates whether the ith item/subset in the pool is included in the test, and a_{ij} indicates the presence of property j (for nonpsychometric test specifications) or the item/subset information (for IRT-based target test specifications) for the ith item/subset in the pool. Note that for the target test specifications, if i represents a subset, then a_{ij} is the sum of the item information over the items in that subset.

Given this redefinition of the pool, the test length is no longer the sum of the x_is because some of the is represent subsets with more than one item. For convenience, define another variable, g, indexed on the pool, and set g_i to the number of items in the subset if i represents a subset, or to 1

if i represents a discrete item.

A final complication of item sets is that it would be illogical to select more than one subset from a given set. To control for this, an additional constraint (in the strict binary programming sense) is defined that limits the selection to at most one subset of any set. Let $s = 1, \ldots, S$ index the item sets in the original pool, and let b_{is} be 1 if item/subset i is a subset of set s and 0 otherwise. Then, the additional constraint is specified as

$$\sum_{i=1}^{N} b_{is} x_i \leq 1, \quad s = 1, \ldots, S . \tag{11}$$

Weighted Deviations Model

In summary, the complete specification for the new model is as follows:
Minimize

$$\sum_{j=1}^{J} w_j d_{L_j} + \sum_{j=1}^{J} w_j d_{U_j} , \tag{12}$$

subject to

$$\sum_{i=1}^{N} g_i x_i = n , \tag{13}$$

$$\sum_{i=1}^{N} a_{ij} x_i + d_{L_j} - e_{L_j} = L_j, \quad j = 1, \ldots, J , \tag{14}$$

$$\sum_{i=1}^{N} a_{ij} x_i - d_{U_j} + e_{U_j} = U_j, \quad j = 1, \ldots, J , \tag{15}$$

$$\sum_{i=1}^{N} b_{is} x_i \leq 1, \quad s = 1, \ldots, S , \tag{16}$$

$$d_{L_j}, d_{U_j}, e_{L_j}, e_{U_j} \geq 0, \quad j = 1, \ldots, J , \tag{17}$$

and

$$x_i \in \{0,1\}, \quad i = 1, \ldots, N . \tag{18}$$

This model is referred to as the *weighted deviations model* (WDM). It was investigated by constructing tests using several actual item pools and test specifications. Many of the cases involved relatively large item pools, and all of the cases involved large numbers of constraints (in the test specialist's sense). All data were taken from actual test assemblies for active testing programs.

A Heuristic for Solving the Model

The WDM is a binary programming problem and therefore, as noted earlier, can be solved using standard MILP algorithms. These algorithms, in general, produce the best possible solution to the problem by obtaining the optimal value for the objective function and by meeting all the constraints (assuming a feasible solution exists). However, very large test construction problems are difficult to solve using MILP algorithms. They often require considerable amounts of computer time and, more importantly, sometimes exceed the size limitations of MILP implementations. The size of a linear programming model is often measured in terms of the number of *variables* (i.e., decision variables) and the number of *equations* (constraints and objective function) in the model. Problems have been encountered involving as many as 140,000 variables and 400 equations (see below); such problems

exceed the limits of linear programming implementations for microcomputers.

Heuristic solutions to binary programming problems, although often suboptimal, overcome these difficulties. Any number of heuristics might be considered; however, the objective function for the WDM suggests selecting items in such a way that the weighted sum of the positive deviations is as small as possible. As a sequential process, the heuristic seeks to monotonically increase progress toward the optimum value of the objective function.

Phases of the Heuristic

The heuristic algorithm used to implement the WDM consists of two phases. In the first phase, items are successively selected so that the expected weighted sum of positive deviations is minimized at each iteration. Details on the selection process and what is meant by "expected" are provided below; for now, note that for each item in the pool, the weighted sum of positive deviations that would be expected if this item were added to the test is computed. Then, the item with the smallest expected weighted sum of positive deviations is selected and added to the test.

Once the desired test length has been reached a replacement phase is entered. In this phase, previously selected items are successively replaced until no further improvement in the weighted sum of positive deviations can be made.

This essentially follows the pattern of a "goal seeking" or "greedy" heuristic (e.g., Nemhauser & Wolsey, 1988, chap. II.5) commonly found in linear programming applications. The heuristic is similar to a method used by Luecht & Hirsch (1992) in which items are selected using moving averages of the distance to a TTIF. It is also similar to an approach used by Ackerman (1989) for the construction of multiple parallel tests. Others who have followed this kind of approach include Webb (1969) and Kester (1988). Adema (1990) referred to such algorithms as combined *construction* and *improvement* heuristics. In addition to making the solution more tractable, they offer the opportunity to incorporate considerable flexibility into the construction process. Examples of this are given below.

Local Optimality and Item Set Constraints

Local optimality. Sequential selection procedures are often locally optimal but globally suboptimal. For example, early in the process the deviations from the lower bounds will generally be positive, although there are generally no deviations from the upper bounds. If items were selected strictly on the basis of current deviations from the bounds, items having properties that are relatively abundant in the pool may inadvertently be selected early because they contribute to satisfaction of the lower bounds, even though their presence may cause difficulties later in meeting the upper bound constraints. To correct for this, an estimate of how the deviations might be affected by future selections must be included in the computation of the deviations.

The simplest way to do this is to assume that future selections will be random, and will therefore contribute to the satisfaction of bounds in proportion to the distribution of item properties in the pool. The assumption that the remaining items are randomly selected implies that the expected value of the sum of the constraint properties is equal to the number of items remaining to be selected times the average of the constraint values taken over the remaining items in the pool. This may be approximated mathematically as follows. If the kth item in an n-item test is selected and then the appropriateness of the tth item in the pool is considered, compute for $j = 1, \ldots, J$

$$\left(\sum_{i=1}^{N} a_{ij} x_i\right) + (n - k)v_j + a_{tj} , \tag{19}$$

where the first term reflects the properties of the items selected so far, and v_j is the average occur-

rence of property j (or the average item information at the specified trait level if j represents a test information constraint) across the pool. This quantity is then compared to both the upper and lower bounds to compute the expected deviations.

The effect of this adjustment is that scarce items, that is, items having properties with a relatively small average occurrence in the pool, will tend to be selected early. This will happen because the v_j for sparse properties is small compared to that for abundant properties, resulting in a smaller contribution to Equation 19. This will cause items having a sparse property ($a_{ij} = 1$) to improve the weighted deviations more than do items lacking this property but abundant in another property, which in turn will result in early selection of the items with the sparse property.

The use of average property occurrence (or item information) to project future test properties is one of many ways to deal with the problem of local versus global optimization. More refined methods might include estimates of the probabilities that items not yet selected will in fact be selected, or might take into account more precise computations of average property occurrence given the items already selected. However, it is by no means clear that more elaborate methods would improve the selection process, and this heuristic is simple and fast to compute.

Item set constraints. As each item is considered, the heuristic determines whether selecting the item would "bring in" its parent stimulus; that is, does the item have a stimulus and, if so, has the stimulus already been included in the test? This results in a count of the number of sets included in the test that have the properties associated with each of the set constraints. The deviations between these counts and the constraint bounds are then included in the computation of the expected weighted sum of positive deviations.

If more than one set corresponding to a given set constraint will be included in the test, the algorithm also attempts to balance (i.e., make approximately equal) the number of items selected from each of the sets (a requirement of test specialists). It does this by adding to the sum of the weighted positive deviations the difference between the average number of items across the selected sets and the minimum or maximum number of items across the selected sets. Similarly, because one-item sets are considered undesirable, a penalty (set by the test specialist but defaulted to the equivalent of a deviation of 1) is added to the sum of the weighted deviations if a set is included in the test with only one of its items selected. One advantage of a heuristic is the flexibility it allows in incorporating such features.

Details of the Heuristic

The selection phase consists of the following four steps:

1. For every item t not already included in the test, compute the expected weighted sum of deviations that would be obtained if item t were added to the test. To do this, first compute d_{L_j} and d_{U_j} for each constraint j by comparing the quantity given by Equation 19 to the lower and upper bounds, respectively, for constraint j. This computation will depend on whether constraint j is an item constraint, a set constraint, or a test information (target) constraint. The computation for a set constraint must consider the status of an item's parent stimulus in the test, as well as matters such as set balancing and single-item sets. The computation for an item constraint considers the (binary) presence or absence of item properties; the computation for test information constraints considers item information rather than binary item properties. The details of each of these three computations follow.
   a. If j is an item constraint, compute q_j [the number of items already included in the test having property j (i.e., the summation in Equation 19)], plus 1 if item t has property j (a_{tj} in Equation 19), plus the quantity $(n - k)v_j$, where v_j is the average occurrence of property j across

the pool, and this is the kth item in the test. If $q_j < L_j$, set $d_{L_j} = L_j - q_j$ (i.e., the deviation from the lower bound), otherwise $d_{L_j} = 0$. Similarly, if $q_j > U_j$ then set $d_{U_j} = q_j - U_j$ (i.e., the deviation from the upper bound), otherwise $d_{U_j} = 0$. This estimates the number of items in the test that will have property j if this and all previously selected items are included in the test and all remaining $(n - k)$ items are selected randomly (in the sense described earlier), and then compares that estimate to the upper and lower bounds.

b. If j is a set constraint, compute q_j (the number of sets already included in the test having property j), plus 1 if item t belongs to a set having property j and that set was not previously included in the test, plus the quantity $(n - k)v_j$, where v_j is the average occurrence of property j across all sets in the pool. As before, set $d_{L_j} = L_j - q_j$ if $q_j < L_j$, otherwise $d_{L_j} = 0$; and $d_{U_j} = q_j - U_j$ if $q_j > U_j$, otherwise $d_{U_j} = 0$. To incorporate set balancing, add to the deviations the larger of the absolute difference between the average number of items across the selected sets and the minimum or maximum number of items across the selected sets. To avoid one-item sets, add to the deviations a penalty (default 1) if a set would be included in the test with only one of its items selected. Here, the number of sets in the test is being effectively estimated that will have property j, if this and all previously selected items are included in the test and all remaining $(n - k)$ items are selected randomly. Then that estimate is being compared to the upper and lower bounds, and the results are being adjusted to accommodate set balancing and the penalty for one-item sets.

c. If j is a test information (target) constraint, compute q_j (the sum of item information over all items already included in the test), plus the item information for item t, plus the quantity $(n - k)v_j$, where v_j is the average item information at the specified trait level. As before, set $d_{L_j} = L_j - q_j$ if $q_j < L_j$, otherwise $d_{L_j} = 0$; and $d_{U_j} = q_j - U_j$ if $q_j > U_j$, otherwise $d_{U_j} = 0$. Here the test information is being effectively estimated if this and all previously selected items are included in the test and all remaining $(n - k)$ items are selected randomly. That estimate then is compared to the upper and lower bounds.

2. Sum the weighted d_{L_j} and d_{U_j} across all constraints j, as specified in Equation 12, to form the weighted sum of positive deviations that would be expected if item t were added to the test.
3. Select the item t with the smallest expected weighted sum of positive deviations, and add it to the test.
4. Repeat Steps 1 through 3 until n items have been selected.

The replacement phase consists of the following three steps:

5. Select the $(n + 1)$ item according to Steps 1 through 3, except eliminate the expected values $[(n - k)v_j]$ from the computations (these values are meaningless for $k > n$). Provisionally add the item to the test.
6. Find the item already included in the test whose removal would most reduce the weighted sum of positive deviations. The details for performing this step are much the same as Steps 1 through 3, except that here only items already included in the test are being considered and the computations assume that the item will be removed from, rather than added to, the test. Item set constraints are handled by determining whether the removal of an item would remove its parent stimulus (because the set would otherwise be empty), and the impact on set constraints is then computed as appropriate.
7. If the removal and replacement process would reduce the weighted sum of positive deviations, then add the replacement item (from Step 5) to the test, remove the item located in Step 6, and repeat Steps 5 and 6. Otherwise stop.

By definition, the replacement phase will monotonically improve the weighted sum of positive deviations: If it is not possible to find a pair of items whose replacement in the test would result in a smaller

weighted sum of positive deviations, then the process stops.

Various methods of pairing the replacement items, that is, the item removed and the item added, were considered. The method suggested in Steps 5 through 7 is simple and efficient. An elaboration of this procedure would be to consider simultaneously the effect on the weighted deviations of all possible pairs of items currently selected against items available to be added. Other elaborations are of course possible, although analysis of the examples described below showed that inclusion of the compensating expectation usually made the replacement phase unnecessary.

Note that the details of the computations in this heuristic are tied to the particular features of the three types of constraints involved (item, set, and test information constraints). These are the types typically encountered in test construction problems; other types of constraints could be accommodated within the overall logic of the heuristic by extending the details of the computations to the particulars of those types. Alternatively, a single computational method could have been used for all types of constraints by appropriate recasting of the data, although this would have been less efficient.

Results

The WDM and the heuristic for solving it were evaluated by applying them to several test construction problems. The data for these problems were obtained from actual test assemblies for operational tests. In each case, the data consisted of real test specifications and actual item data.

Many, if not most, test assemblies resulted in a value of 0 for the objective function (Equation 12). However, a number of assemblies resulted in an objective function greater than 0. It is these cases which are of interest because the model expressed in Equations 2–5 would have been infeasible.

The Heuristic Versus MILP

To implement a MILP solver, the model was first formulated using the general algebraic modeling system (GAMS; Brooke et al., 1988). This system provides a means of expressing linear programming models and applying several types of solvers to them. The model was then solved using the zero/one optimization method. This method first solves the linear programming relaxation, then applies a pivot and complement heuristic, and finally uses a branch-and-bound procedure (see Brooke et al., 1988, Appendix E, for details). The weights applied to each test specification are typically 1, so that deviations usually reflect the number of items by which the test deviated from the specifications.

Table 3 shows a comparison of the heuristic and the MILP for eight tests ("cases"). In this table, n is the intended test length (number of items), and J is the number of test specifications ("constraints" in the test specialist's terminology). Note that both lower and upper bounds must be applied to each of these J specifications, resulting in $2J$ actual constraints. N is the number of discrete items plus subsets of sets in the pool. Var and Equ are the number of variables and the number of equations, respectively, in the binary programming problem, and thus characterize the size of the problem from a mathematical programming point of view. "Weighted deviations" are the values of the objective function obtained by the heuristic and by the MILP solver. The models were all run on a 386-based 20 MHz personal computer with 4 MB internal memory (RAM).

Cases 1 and 2 represent assemblies of sections of larger tests. The item pools for Cases 1 and 2 were relatively small. No sets were involved in the first pool, but 38 sets were included in the second pool. Both assemblies involved a relatively small number of test specifications, with five of the specifications representing points on the trait metric at which test information is constrained. The specifications were generally consistent with the proportion of items in the pool; that is, few of the specifications called for much larger or much smaller numbers of items to be selected than would be expected given the representation of the characteristic in the pool. Thus, it would be expected that these assemblies

Table 3
Weighted Deviations and CPU Time In Seconds for Heuristic and MILP Solutions

Test, Case, and n	J	N	Var	Equ	Weighted Deviations		CPU Time	
					Heuristic	MILP	Heuristic	MILP
Sentence Completion								
Case 1, n = 27	39	452	617	84	1	1	15	435
Logical Reasoning								
Case 2, n = 25	38	298	459	120	3	3	17	2,056
Verbal								
Case 3, n = 70	64	343	748	204	3	2	61	163
Mathematics								
Case 4, n = 25	103	506	919	217	9	3	34	1,175
Sentence Completion								
Case 5, n = 40	60	1,025	1,314	146	4	1	85	1,958
Verbal								
Case 6, n = 76	54	1,064	1,333	136	3	3	144	434
Mathematics								
Case 7, n = 65	155	741	1,370	326	4	3	165	969
Case 8, n = 60	126	903	1,424	262	11	7	155	13,522
Case 9, n = 60	129	6,647	7,164	446	1		996	
Arithmetic Reasoning								
Case 10, n = 24	52	11,521	11,730	152	6		32	
Reading								
Case 11, n = 28	61	25,505	25,750	159	2		52	
Verbal								
Case 12, n = 40	34	46,114	46,255	154	12		95	
Case 13, n = 70	98	143,381	143,786	270	3		4,411	
Reading and Sentence Completion								
Case 14, n = 40	72	146,202	146,491	397	2		4,791	

would readily satisfy most of the test specifications.

In Case 1, the heuristic and the MILP procedure failed to meet one of the test specifications, and each missed the same specification by one item. This was a particularly difficult specification to satisfy because few items in the pool had the relevant characteristics. In Case 2, the heuristic and the MILP procedure again failed to meet one of the test specifications, and each missed the same specification, but in this case each selected three fewer items than desired.

Case 3 represents a full-length verbal test with a relatively small item pool with no sets. The assembly was constrained by a moderate number of specifications, again including bounds on the TIF at five trait levels. The specifications are consistent with the characteristics of the pool, with only a few that are difficult to meet. The heuristic and the MILP solver both failed to meet one specification, with the heuristic failing by three items and the MILP solver failing by two.

Case 4 was a short version of a corresponding full-length test. It had a relatively small pool with a small number of sets, and a fairly large number of test specifications. This test did not use IRT statistics; most non-IRT based tests express statistical specifications as upper and lower bounds on the number of items that may be selected from each interval in a frequency distribution of item difficulty and/or discrimination. The specifications were largely consistent with the pool characteristics, although several were difficult to meet. The heuristic failed to meet eight specifications by one item and one specification by three items (weights less than 1 resulted in nine deviations). The MILP solver missed three specifications by one item, with two of those specifications also missed by the heuristic. In this case it was

clear that the problem in meeting specifications was in simultaneously trying to satisfy all the specifications, rather than any one or two of the specifications being extraordinarily difficult to meet.

Case 5 shows the results of assembling a section of a longer test from a large pool from which the sets were removed. A moderate number of test specifications was involved, with no test information requirements, and a few of the test specifications were difficult to meet. The heuristic missed four specifications by one item each, and the MILP solver missed a different specification by one item.

Case 6 was assembled from a large pool with no sets. The number of test specifications was moderate, with no test information constraints. In this case more than half of the test specifications were difficult to satisfy. Both the heuristic and the MILP solver failed to meet one of the particularly difficult specifications, in both cases by three items.

Case 7 was a full-length test assembled from a moderate size pool with a small number of sets. A very large number of test specifications were involved, including constraints on test information at 11 points on the trait metric. The specifications were generally consistent with the pool, although some had too few items. The heuristic missed three nonstatistical specifications by one item and failed to meet the target test information at three points. The MILP solver failed on the same three nonstatistical specifications, but met the test information targets.

In Case 8, a full-length test was assembled from a large item pool with no sets. The number of test specifications was large, no IRT targets were included, and the test specifications were generally difficult to satisfy. The heuristic failed to meet five specifications by one item and one specification by two items. The MILP solver missed three specifications by one item, all of which were also missed by the heuristic (weights greater than 1 resulted in deviations of 11 and 7, respectively).

In most cases, the tests selected by the heuristic and the MILP solver were quite different. The number of items in common between the two solvers varied from 0 to approximately 60%. The most typical result was one-quarter to one-third of the items in common.

The Heuristic With Large Numbers of Items

Table 3 also shows six cases in which only the heuristic was attempted. These cases all involved large numbers of sets, which resulted in pools (N; items and subsets) that were extremely large. The resulting MILP model required numbers of variables that exceeded the implementation limitations of the microcomputer being used.

The heuristic performed well on these problems, obtaining reasonably small weighted deviations. CPU times were in some cases substantially larger than Cases 1 through 8 because of the additional complexity of handling sets. This was particularly true in cases for which the sets tended to contain many items, as in Cases 13 and 14.

Discussion

A variety of real test construction problems were investigated using the WDM with both the heuristic and a MILP solver. As described above, these problems varied along several dimensions: item pool size and characteristics, presence or absence of item sets, number of test specifications and the difficulty of meeting them, and presence or absence of test information targets. The resulting MILP formulations varied in size (as indicated by the number of variables and equations)—several formulations were very large. Also, some of the cases involved specifications that were very difficult to meet individually, whereas others had combinations of specifications that were difficult to meet simultaneously.

The WDM was successful in solving these problems, none of which could have been solved if the test specifications were formulated as model constraints (as in Equations 2–5). The solutions, although not optimal in terms of satisfying all test specifications, were judged by test specialists to be good

solutions. The model, therefore, was generally useful for solving very large or very difficult test construction problems.

The heuristic also worked well in practice. In most cases, it produced a result reasonably close to that obtained by the MILP solver, and in some cases produced the identical result. The primary advantages of the heuristic are (1) that it can solve very large problems that cannot be run with microcomputer implementations of a MILP solver and (2) that CPU times are usually substantially shorter. It should be noted, however, that CPU time can depend on many factors, such as the speed of the machine and the particular way an algorithm is coded. Table 3 shows that CPU times did not vary directly in any reliable way with the size of the problem, particularly for the MILP solver. Moreover, item data must in practice be retrieved from a database, and this retrieval time will often be quite large. The relative time differences between the heuristic and the MILP solver appear to be significant, and the user needs to balance the time advantages of the heuristic against the improved solution often obtained by the MILP solver.

An added advantage of the heuristic is that it provides increased flexibility for incorporating additional considerations in the test construction process. For example, it lends itself well to the control of item overlap. As a sequential process, items that overlap with a selected item can be excluded easily from further consideration. Similarly, the simultaneous construction of multiple parallel tests can be incorporated easily into the model and heuristic by minimizing the sum of weighted positive deviations across all tests simultaneously, or by minimizing the largest of the deviations.

The WDM and the heuristic are now being used operationally to assemble a variety of tests. Comparisons between the results obtained with this model and the results of manual test assemblies (Stocking, Swanson, & Pearlman, 1991, 1993) have confirmed its effectiveness in practice.

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 for Measurement in Education, San Francisco.

Adema, J. J. (1988). *A note on solving large-scale zero-one programming problems* (Research Rep. 88-4). Enschede, The Netherlands: Department of Education, University of Twente.

Adema, J. J. (1990). *Models and algorithms for the construction of achievement tests.* The Haag, The Netherlands: CIP-gegevens Koninklijke Bibliotheek.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In F. M. Lord & M. R. Novick, *Statistical theories of mental test scores* (pp. 395–479). Reading MA: Addison-Wesley.

Boekkooi-Timminga, E. (1989). *Models for computerized test construction.* The Haag, The Netherlands: Academisch Boeken Centrum.

Brooke, A., Kendrick, D., & Meeraus, A. (1988). *GAMS: A user's guide* [Computer program and manual]. Redwood City CA: The Scientific Press.

Hsu, T., & Sadock, S. (1985). *Computer-assisted test construction: A state of the art* (TME Report 88). Princeton NJ: ERIC Clearinghouse on Tests, Measurement, and Evaluation, Educational Testing Service.

Kester, J. G. (1988). *Various mathematical programming approaches toward item selection* (Report No. 3 of the Project "Optimal Item Selection"). Arnhem, The Netherlands: CITO.

Lord, F. M. (1952). A theory of test scores. *Psychometric Monograph,* No. 7.

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

Luecht, R. M., & Hirsch, T. M. (1992). Item selection using an average growth approximation of target information functions. *Applied Psychological Measurement, 16,* 41–51.

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

Stocking, M. L., Swanson, L., & Pearlman, M. (1991). *Automatic item selection (AIS) methods in the ETS testing environment* (Research Memorandum 91-5). Princeton NJ: Educational Testing Service.

Stocking, M. L., Swanson, L., & Pearlman, M. (1993). The application of an automated item selection method to real data. *Applied Psychological Measurement, 17,* 167–176.

Theunissen, T. J. J. M. (1985). Binary programming

and test design. *Psychometrika, 50,* 411–420.

Van der Linden, W. J. (1987). Automated test construction using minimax programming. In W. J. Van der Linden (Ed.), *IRT-based test construction* (pp. 1–16). Enschede, The Netherlands: Department of Education, University of Twente.

Van der Linden, W. J., & Boekkooi-Timminga, E. (1989). A maximin model for test design with practical constraints. *Psychometrika, 54,* 237–248.

Webb, J. (1969). *A system for the computer assisted assembly of tests* (Test Development Systems Rep. 69.3). Princeton NJ: Educational Testing Service.

Yen, W. M. (1983). Use of the three-parameter model in the development of a standardized achievement test. In R. K. Hambleton (Ed.), *Applications of item response theory* (pp. 123–141). Vancouver BC: Educational Research Institute of British Columbia.

Author's Address

Send requests for reprints or further information to Len Swanson, Educational Testing Service, Princeton NJ 08541, U.S.A. Internet: lswanson@rosedale.org.