RR-91-9

RESEARCH REPORT

AUTOMATED ITEM SELECTION USING ITEM RESPONSE THEORY

Martha L. Stocking
Len Swanson
Mari Pearlman

Educational Testing Service
Princeton, New Jersey
February 1991

AUTOMATED ITEM SELECTION USING ITEM RESPONSE THEORY*

Martha L. Stocking
Len Swanson
Mari Pearlman

Educational Testing Service
Princeton, New Jersey 08541

February 1991

*This work was supported by Educational Testing Service through the Program Research Planning Council. Part of this paper was presented at the annual meeting of the National Council on Measurement in Education, Boston, 1990.

AUTOMATED ITEM SELECTION USING ITEM RESPONSE THEORY

Abstract

This paper presents a new heuristic approach to interactive test assembly that is called the successive item replacement algorithm. This approach builds on the work of van der Linden (1987a) and van der Linden and Boekkooi-Timminga (1989) in which methods of mathematical optimization are combined with Item Response Theory (IRT) to construct tests from larger collections or pools of items. This new approach is contrasted with two more formal models as well as other heuristic approaches that appear in the literature. An experiment using quasi-realistic data is performed that serves to illustrate the differences in approaches for typical practical test construction applications.

Key Words: item response theory, test construction, mathematical programming, heuristic algorithms.

AUTOMATED ITEM SELECTION USING ITEM RESPONSE THEORY

Introduction

Test construction -- that is, selecting items from a set of available items to form a final test -- is a complex process. Until recently this process had been virtually unassisted by modern psychometrics. Psychometric developments over the past twenty years have suggested that model-based measurement models such as Item Response Theory (IRT) could be helpful in the process. However, the practical use of IRT and the test construction algorithms suggested by Lord (1980) and Birnbaum (1968) require modern computers, available only recently. Further, this merging of psychometrics and computing technology suggests the potential for the integration of mathematical optimization algorithms into the test construction process, as exemplified by the work of van der Linden (1987a) and Boekkooi-Timminga (1989). In these works, some aspect of the items to be selected is optimized subject to constraints on other item properties. These constraints may formally incorporate characteristics of items, such as content or type, that are important in test construction.

This paper presents a new heuristic approach to interactive test assembly that we call the successive item replacement algorithm. This approach will be contrasted to two elegant mathematical models for interactive test assembly developed by van der Linden (1987a) and van der Linden and Boekkooi-Timminga (1989) and to other heuristic approaches that have appeared in the literature. An experiment using quasi-realistic data is described that serves to illustrate the advantages of the new algorithm for typical practical test construction applications.

Theoretical Framework

A key foundation of the theoretical framework developed by van der Linden and Boekkooi-Timminga (1989) is the idea that it is adequate to consider the test information function (Lord, 1980, equation 5-6) at discrete ability levels. This is a reasonable assumption given that test information functions are typically relatively smooth and that ability levels can be chosen to be arbitrarily close to each other.

In the mathematical expression of models in this framework, decision variables x_i, $i = 1,\ldots,N$, are defined for each item in an N-item pool. These decision variables take on the values of $x_i = 0$ if the item is excluded from the test being assembled and $x_i = 1$ if the item is included in the test. The basic concept of all models in this framework is to optimize some objective function of interest subject to linear constraints on the decision variables.

Model 1: Relative Information

This model, due to van der Linden and Boekkooi-Timminga (1989), is used to build a test with an information function having a predetermined shape but of unknown height. It is useful in a context in which the location of optimal measurement is known, but the precision of that optimal measurement is unknown. In the language of linear and integer programming algorithms, it is a maximin model, that is, it seeks to maximize some minimum test information at selected locations on the ability continuum.

Let $i = 1,\ldots,N$ index the items in the item pool,

θ_k, $k = 1,\ldots,K$ be the K values of θ at which the test information function, or item information functions, are evaluated,

r_k, $k = 1,\ldots,K$ be the relative desired height of the target test information function.

Then the relative information optimization model is

$$\text{Maximize } y \tag{1}$$

subject to the constraints

$$\sum_{i=1}^{N} I_i(\theta_k)x_i - r_k\, y \geq 0,\ k = 1,\ldots,K\ , \tag{2}$$

$$\sum_{i=1}^{N} x_i = n\ , \tag{3}$$

$$x_i \in (0,1)\ ,\ \text{and} \tag{4}$$

$$y \geq 0\ , \tag{5}$$

where n is the desired number of items in the test.

The variable y in equation 1 is a mechanism used in both maximin and minimax problems. Its role in the current formulation can be more easily seen if we rewrite equation 2 as

$$\sum_{i=1}^{N} (I_i(\theta_k)\, x_i)/r_k - y \geq 0\ ,\ k = 1,\ldots,K\ .$$

In this form, y can be viewed as the minimum of the weighted sums of decision variables. The $x_i = 1$ are selected so that this minimum is maximized, hence the name "maximin."

Additional linear constraints on the optimization problem can be incorporated easily into this mathematical framework. One kind of constraint

may be called a categorical or set constraint. Items may be considered as having or not having the feature which defines a particular category or set. This type of constraint is expressed as

$$a \leq \sum_{i \epsilon V_j} x_i \leq b \ . \tag{6}$$

The (a,b) are the (lower, upper) bounds on the constraint, that is, the bounds on the number of items from a category, and V_j is the set of items in the pool with feature or characteristic j.

A second type of constraint is noncategorical. This is used for items having some feature of interest that is most easily expressed as a number rather than as category or set membership. For noncategorical constraints of interest, the formulation is

$$a \leq \sum_{i=1}^{N} l_i x_i \leq b \ , \tag{7}$$

where l_i is the property of interest associated with item i, and (a,b) are the (lower,upper) bounds on the total amount of this property desired in the test.

Optimization problems such as these, where the function to be optimized as well as all constraints are linear, are typically solved by linear or integer programming methods (see, for example, Fletcher, 1987). These methods seek the global (vs. local) optimum of the function in the space defined by the decision variables and the constraints. Integer programming methods require that the decision variables take on only integer values; linear programming methods allow decision variables to take on any (usually positive)

real values. This is the direction currently being followed by most researchers.

Model 2: Absolute Information

In many test construction environments, there is typically no mystery about the desired properties of test forms under construction; in many testing programs, new editions of a test are constructed to be as parallel as possible to previous editions. In fact, test information functions can be computed and averaged for all older editions of a test, thus giving an absolute target test information function for new test editions. A second model developed by van der Linden (1987b) seems more relevant to this environment. In this model we have

$$\text{Minimize } y \tag{1'}$$

subject to the constraints

$$\sum_{i=1}^{N} I_i(\theta_k)\, x_i - I(\theta_k) \leq y,\ k = 1,\ldots,K\ , \tag{2'}$$

and

$$\sum_{i=1}^{N} I_i(\theta_k)\, x_i \geq I(\theta_k),\ k = 1,\ldots,K\ , \tag{2''}$$

where $I(\theta_k)$ are the k values of the target information function. In this formulation, y is the maximum positive deviation from the target. Equation 2'' guarantees that the target is met at all values of θ_k , but adds K more constraints to the problem. The other constraints contained in equations 3, 4 and 5, plus any additional constraints on test content, are added to this model in the same manner as before.

Heuristic Approaches

While the search for a global optimum seems mathematically elegant, it appears reasonable to question whether the search for a global optimum is necessary. Test development specialists concentrate on meeting various constraints placed on their selection of items to include in test editions. In their work they do not search for the optimum set of items, but rather seek to find any set that satisfies these constraints. It seems reasonable to propose that any algorithmic test construction process perform in a similar way.

This conclusion has clearly been reached by Adema (1988) in his concentration on heuristics to improve integer programming algorithms. In his proposed algorithms he is satisfied with locally optimum solutions that are within a known percentage of the first noninteger solution. A problem with this approach is that it may result in no solution at all. Adema (1989) also investigates other improvements in standard branch and bound algorithms used to solve problems in integer programming.

Boekkooi-Timminga (1988, 1989) takes a slightly different approach in which the available item pool is subdivided into clusters with identical statistical properties as well as identical content classifications. This approach works well when the number of item classifications is small, but becomes unrealistic when the number is large, as well as when models more complex than the 1-parameter logistic (1PL) are required to adequately describe the item response functions.

Ackerman (1989) takes a very different approach -- one that is particularly appropriate for the construction of multiple parallel tests. He considers two factors: the desired number of items per content area, and the

difference between an absolute target information function and the instantaneous test information for each test as it is being constructed. To select an item he determines the test and the θ at which deviation from the target is greatest, and then chooses the most informative item at that θ for a content area not yet filled for that test. This process is repeated until all content requirements are met. Several refinements to the basic algorithm are used to further minimize any information differences among the multiple tests.

Ackerman's results are promising. While the algorithm considers only information and the number of items per content area, it clearly could be extended to control for other kinds of constraints.

Webb (1969) adopted an approach that is similar, but within the confines of classical item statistics. His algorithm considers two statistical properties, difficulty (as measured by the item delta) and discrimination (as measured by the item r-biserial), and a set of content classifications. For each content category he computes, for each item, a weighted linear combination of content and statistical terms that reflect the degree to which the item satisfies the test requirements *and the difficulty of fulfilling those requirements*. This novel addition forces the algorithm to focus on those test requirements which are most difficult to meet, just as a human test developer would. After selecting the most satisfactory item he modifies the test requirements to reflect their partial fulfillment by selection of that item. The process is repeated until the desired number of items for the content category have been selected, and then repeats for each of the other content categories.

When implemented almost twenty years ago, the Webb algorithm worked well in practice, though its implementation suffered from the lack of an appropriate technology for making it interactive. Its particular appeal is that it seeks multiple goals (i.e., both content and statistical properties) simultaneously, and at the same time gives the test developer some control, through the weightings, over the relative importance of particular factors.

Successive Item Replacement Algorithm

The approach we have developed is a blend of the simplicity and flexibility of the Ackerman and Webb heuristics with the elegant mathematical approach developed by van der Linden and his associates. It is motivated by our failure to achieve satisfactory solutions with the van der Linden approach, while only partially solving the problem of excessive computer time through its various extensions (more on these topics below). These results led us to rethink our purposes.

In some sense, we are less concerned with maximizing test information, or even meeting all of the constraints of interest, than we are with coming "as close as possible" to all constraints simultaneously. Phrased another way, we would rather miss on two or three constraints, but come very close, than meet all but one constraint but miss that one by a large margin.

This is generally, but not universally, true. Some constraints are very important and we would rather sacrifice others for them. For example, if we want a reading test to contain exactly one science passage we would probably not be tolerant of an assembly that yielded two, or none.

These considerations suggest that we need to think of "constraints", including conformance to upper and/or lower target information functions, as more "desired properties" than true constraints. We want to recognize the

possibility of failing to meet these desired properties, but we also want to minimize our aggregate failures. And, in some sense, if we miss on a constraint then the extent to which we miss it matters, in a roughly linear way (e.g., missing by two items is twice as bad as missing by one).

Moreover, we want to control the importance of individual constraints by weighting them. If one constraint is twice as important as another then its weight in determining the appropriateness of an individual item should be double that of the other.

These considerations led us to reformulate our goal in very simple terms: *minimize the weighted sum of deviations from the constraints*. We retain the van der Linden model for expressing constraints, except that we now consider conformance to the target information function as a set of constraints like any other, rather than as an objective function to be minimized or maximized. Thus, linear constraints are formulated as bounds on the number of items having specified properties. The target information function constraint is expressed as lower and upper bounds on information at the desired set of θ's. The complete model is formulated as follows:

Minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{m+2K} w_j d_{ij} \tag{8}$$

subject to

$$\sum_{i=1}^{N} I_i(\theta_k) x_i \geq I_L(\theta_k), \quad k = 1, \ldots, K, \tag{9}$$

$$\sum_{i=1}^{N} I_i(\theta_k) x_i \leq I_U(\theta_k), \quad k = 1, \ldots, K, \tag{10}$$

$$\sum_{i=1}^{N} a_{ij}x_i \geq L_j,\ j = 1,\ldots,m\ , \tag{11}$$

$$\sum_{i=1}^{N} a_{ij}x_i \leq U_j,\ j = 1,\ldots,m\ , \tag{12}$$

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

and

$$x_i \in (0,1),\ i = 1,\ldots,N\ , \tag{14}$$

where m is the number of linear constraints; $m + 2K$ is the total number of constraints to be considered; the w_j are the weights to be applied to each constraint; the $I_L(\theta_k)$ and $I_U(\theta_k)$ are the set of lower and upper target information values, respectively; the L_j and U_j are the lower and upper bounds on the linear constraints, respectively; and the a_{ij} are 1 if item i has property j, else 0. The d_{ij} are defined as follows:

$$d_{ij} = I_L(\theta_k) - \sum_{i=1}^{N} I_i(\theta_k)x_i \geq 0 \text{ (lower targets)},$$

or

$$d_{ij} = \sum_{i=1}^{N} I_i(\theta_k)x_i - I_U(\theta_k) \geq 0 \text{ (upper targets)},$$

or

$$d_{ij} = L_j - \sum_{i=1}^{N} a_{ij}x_i \geq 0 \text{ (lower bounds on linear constraints)},$$

or

$$d_{ij} = \sum_{i=1}^{N} a_{ij}x_i - U_j \geq 0 \text{ (upper bounds on linear constraints)}.$$

Note that each of these terms are taken as zero as soon as the relevant constraint is met.

The algorithm that implements this model consists of two phases. In the first phase we successively select items so as to minimize the weighted sum of deviations (d_{ij}'s). That is, for each item in the pool we compute the weighted sum of deviations from the bounds that would apply if this item were added to the test. We then choose the item with the smallest weighted sum of deviations and add it to the test.

Once n items have been selected we enter a replacement phase. After the $(n + 1)^{st}$ item is added, we compute for each item now in the test the weighted sum of deviations from the bounds that would apply if this item were removed from the test. We then choose the item with the smallest weighted sum of deviations -- that is, the one whose removal will most improve the test -- and remove it. A new $(n + 1)^{st}$ item is then selected and all items are again examined for possible removal. This process is continued until no further improvement is possible.

A Realistic Experiment

An experiment was designed and carried out to compare the behavior of both the relative and absolute information models and the successive item replacement algorithm. To obtain the most informative comparison possible, the experiment mimics current test construction practices at Educational Testing Service (ETS) and uses quasi-realistic data.

Current Test Assembly Practices at ETS

Test construction practices at ETS are the result of over forty years of experience in this field. Typically, unique test specifications exist for a particular test that include the consideration of both content and statistical

properties of items. Test construction specialists use these unique specifications to produce different editions of a test that are as parallel to each other as possible in all of the dimensions that are considered important for the test.

In addition to unique test specifications there exist more general specifications incorporating good test construction practices that apply to a large class of tests. These more general specifications are considered by test development specialists to be at least as important as the unique test specifications. Their applications, however, may be more idiosyncratic, thus making parallelism between different test editions even more difficult to achieve. A side benefit of the more automatic test assembly algorithms considered in this paper is the codification and implementation of the same set of unique and general constraints for all parallel test editions.

Table 1 displays an example of the complete (that is, both the unique and the general) test specifications for a particular 25-item ETS test. Each of the 25 items in this test consists of a single sentence in which a portion representing a particular construction (writing problem) in English has been underlined. The examinee is asked to consider the underlined portion and to determine if a better construction exists among the options presented.

Insert Table 1 about here

In Table 1, all but the fourth specification represent categorical constraints. The first specification stipulates that items should come from a variety of item writers. It is thought that a test containing items from only one or two item writers would appear unpleasantly uniform, since item writers

tend to develop a distinctive style. The second specification is that the location of the correct answer must appear in each possible position with about the same frequency. This specification prevents examinees when they do not know the correct answer from always marking an answer choice in a particular position and hence being unfairly advantaged over other examinees who also do not know the correct answer.

The underlined portion of the item may appear at the beginning, middle, or end of the sentence, or may include the entire sentence. Test development specialists feel that a balance of these positions across the items in the test prevents the test from appearing too uniform; thus, we have the third specification. The noncategorical specification four requires that the sentences be roughly the same length. Specification five stipulates that the majority of sentences should be simple sentences, with complex, compound, and compound/complex sentences represented with roughly equal frequency. Specification six balances the subject matter of the sentences, again for the sake of a reasonable balance.

Specification seven in Table 1 considers the various types of writing problems presented in both the sentence and each of the answer choices. No particular type of writing problem is allowed to dominate. Note that for this specification, the classification of items is not mutually exclusive. That is, the sentence may represent an agreement problem, while the answer choices may represent problems in grammatical construction, rhetorical construction, diction, idiom, etc. In contrast, each of the specifications one through six represents a mutually exclusive classifications of items. For example, for specification five, a sentence is either simple, complex, compound, or compound/complex.

Each category of a specification in Table 1 contributes a constraint to the optimization problem. Thus each item writer, each correct answer position, each underlined portion position, etc., is a separate constraint. If we assume that there are 10 different item writers and all items have five choices, Table 1 specifies 42 constraints on the test assembly process. If we add to that list the constraint on the total number of items in the test (equation 3) and decide to evaluate the relative height of the test information at four different abilities ($K = 4$ in equation 2), then the optimization problem for van der Linden's relative information model (Model 1), for example, has a total of 47 constraints.

The Data

A pool of 480 writing items from which the above test is typically built was obtained. Each item had already been calibrated using the 3-parameter logistic (3PL) item response model and the computer program LOGIST (Wingersky, 1983). These items and their estimated parameters are typical of the items that exist in pools for this particular testing program. The average test information function for five previous editions of this test is shown as the bottom curve in Figure 3.

Item characteristics displayed in Table 1, such as item writer, position of correct answer, sentence length, etc., were assigned to the 480 items at random. It would have been better, of course, to use real item properties. However, easy mechanisms did not exist at the time to obtain information about these properties. To the extent that such item characteristics are correlated with each other and with the estimated item parameters, this set of test data is unrealistic.

Results for the Relative Information Model (Model 1)

It was decided to try to build a 25-item test from the 480-item pool that had the same relative height of information as previous test editions at four different abilities, θ = -1, 0, .5, and +1. The results of these attempts are shown in Figure 1 and Table 2. In Figure 1, the horizontal axis is ability; the vertical axis is test information. The dashed lines in Figure 1 are various information functions that exhibit the target relative values of information, that is, the desired shape of the information function, at the four abilities of interest. These are the criteria against which the results should be compared. Any number of these criterion information functions could be drawn, but only three are shown for clarity. The solid lines in Figure 1, each with different plotting symbols at the four ability values, are the test information functions that resulted from five attempts to apply the standard methods of linear and integer programming. None of these five attempts produced an information function with the same shape as the target relative information.

Insert Figure 1 and Table 2 about here

Integer programming, that is, methods of solving optimization problems in which the decision variables are constrained to be integer, are methods of searching the functional space for all possible integer solutions and then choosing as the correct one that which optimizes the function in question. These methods typically implement some form of a "branch and bound" algorithm in conjunction with more straightforward linear programming algorithms in which the decision variables can take on any integer or noninteger values to

solve the linear subproblems (see Fletcher, 1987). In this experiment the revised simplex method for solving the linear subproblems was used as implemented in the IMSL (1989) subroutine library and the branch and bound algorithm was from Fletcher (1987, Chapter 13).

Integer programming problems with a large number of decision variables, as we have here, and a large number of constraints, as we also have here, are typically very difficult to solve in realistic amounts of computer time. Thus five different heuristic approaches to a solution were tried, some of which were recommended by van der Linden and Boekkooi-Timminga (1989). The five methods were as follows:

Method 1: Crude linear rounding

In this method, only the solution to the initial "relaxed" problem is obtained. The decision variables are allowed to take on noninteger values between zero and one, and the results are rounded to zero or one. This method is not guaranteed to find an integer solution that is optimum or even one that satisfies all of the constraints.

Method 2: Improved linear rounding

This method is similar to Method 1, but the method of rounding is different. Here, the decision variables are sorted in descending order and the first n of them are rounded to one, where n is the desired number of items. Like the first method, this method does not guarantee an optimum solution satisfying all constraints.

Method 3: Optimal rounding

This method obtains the relaxed linear solution and then reduces the solution space of the problem by fixing at zero or one any decision variables that had attained these values. The next stage then seeks the

optimum integer solution in this reduced space. Like the first two methods, this method does not guarantee an optimum solution to the original problem satisfying all constraints.

Method 4: First zero-one solution

In the process of finding the global solution, branch and bound algorithms consider and discard many local solutions. The notation "first 0-1 solution" indicates that this procedure was stopped after the first integer solution. This method is guaranteed to produce a solution that satisfies all constraints (if one exists), but it is probably not an optimum solution.

Method 5: Second zero-one solution

The procedure for finding the global optimum was stopped after the second integer solution was found. As in stopping after the first integer solution, constraint satisfaction is guaranteed, but the function is unlikely to be optimum.

Information about the five attempts is shown in Table 2. The simplest attempt, crude linear rounding (Method 1), is the fastest in terms of computer time[1], but resulted in a test that was longer than 25 items, and which violated one of the constraints. Improved linear rounding (Method 2) violated three constraints and was slightly more demanding in terms of CPU time than Method 1. Optimal rounding (Method 3) satisfied all constraints but failed to produce a very close approximation to the desired relative shape.

The first zero-one solution (Method 4) produced results similar to that of optimal rounding, but took about 30 times as long in terms of computer time. The second zero-one solution (Method 5) produced a test information

[1]All times shown here and in Tables 2 and 3 are based on an IBM 3090-300S running at 57 MIPS.

function that most closely matched the desired shape. However, this method took far too long (20 CPU minutes) to be practical. Note that none of the methods tried produced the globally optimal solution. Based on the times required to find only the first two integer solutions, it seems certain that the conventional methodology for solving integer programming problems is unlikely to work in these circumstances.

Results for the Absolute Information Model (Model 2)

The same data as previously described were used for trying out van der Linden's absolute information model (Model 2). Only the results for crude linear rounding (Method 1), optimal rounding (Method 3), and first zero-one solution (Method 4) are reported. Improved linear rounding (Method 2) produced results identical with Method 1. The second zero-one solution (Method 5) was attempted but ran out of computer time after 20 CPU minutes without finding the second zero-one solution. The results are shown in Figure 2 and Table 3. In Figure 2, the target test information function at the four abilities of interest is shown by a dashed line. The results of the three methods are shown by solid lines. The simplest method, crude linear rounding, produced a test information function slightly above the target, but violated 7 constraints. Optimal rounding violated 6 constraints and failed to produce a test information function that met the target. The first zero-one solution exceeded the target at all points and met all of the constraints, but took slightly over 60 times as long as crude linear rounding to produce a solution.

Insert Figure 2 and Table 3 about here

It is clear that the attempts to find strictly globally optimal solutions to either the relative or absolute information models with these data are unsuccessful. It is also clear that continuing along this line of inquiry does not hold much promise. The best locally optimal solution found in the case of the relative information model (second zero-one solution) took over 20 CPU minutes; the best locally optimal solution found in the case of the absolute information model took over 2 CPU minutes.

Results for the Successive Item Replacement Algorithm

The successive item replacement algorithm was implemented and tried out with the same data with one addition: Because we wished to include an upper target information function as well as a lower one, four additional constraints were added bringing the total to 51 constraints. The upper target information constraints correspond to a value of 1.2 times the lower target at each θ_k.

Insert Figure 3 about here

The results were quite promising. Figure 3 shows the resulting test information (solid line) plotted against the upper and lower target curves (dashed lines) at the four abilities of interest. The targets were comfortably met at all θ's. All except one of the linear constraints were met, and that constraint failed by one item. Close examination of the data shows that this constraint in combination with related ones is exceedingly

difficult to meet. CPU time was approximately 0.8 seconds.[2] This compares favorably with the data shown in Tables 2 and 3, especially since the number of constraints on test information was doubled (in fact, adjusting for that difference would reduce the time to approximately two-thirds of a second).

Discussion, Extensions and Future Research

The implementation of the successive item replacement algorithm allowed us to overcome the limitations of implementations of the van der Linden models and proceed with experiments on the use of interactive item selection methods in a realistic setting. The consideration of constraints as more "desired properties" rather than true constraints, and the minimization of our aggregate failure to achieve the desired properties, provide needed flexibility in practical problems. The facility to treat all constraints equivalently mathematically, but to allow differential weighting to reflect relative importance in the test construction process, provides additional desirable features. There are a number of extensions to the basic algorithm that are important to its further practical use in test development.

The most important extension deals with item sets. Item sets occur when a group of items is associated with a single stimulus, as for example with a reading passage followed by questions about the passage. Test developers typically have certain constraints that apply to the stimulus (e.g., "exactly one science passage"; "no more than two passages with gender appeal"; etc.), while other constraints, including the statistical ones, apply to individual items or to the test as a whole. They typically select the passages first,

---

[2]The algorithm was actually implemented on an IBM-compatible 386 PC running at 16 mz and took 14 seconds. This translates to approximately 0.8 seconds on the 3090.

hoping they will be able to meet item and test constraints from the items associated with those passages.

One approach to the problem of item sets is to mimic this process: select the stimuli according to the stimulus constraints, and then select items from the reduced item pool. This approach suffers from the obvious risk that item and test constraints will not be met because of inadequacies in the reduced pool. The approach we adopted was to explicitly incorporate simultaneous set and item selection into the algorithm. As each item is considered for selection, the program determines whether selection of the item would "bring in" its parent stimulus (i.e., does it have a stimulus and, if so, has it already been included because of a prior item selection?). Any relevant set constraints are then tested and considered in the computation of the weighted sum of deviations. If more than one set corresponding to a set constraint will be included in the test, then the algorithm also attempts to automatically balance (i.e., make approximately equal) the number of items selected from each set.

A second extension to the basic algorithm allows it to consider constraints on conventional (as opposed to IRT) item statistics. This is done by forming desired frequency distributions for deltas and r-biserials, and constraining the number of items that may be chosen from each interval in the distribution. No change to the program that implements the algorithm is actually needed, since these constraints are -- like all other linear constraints -- merely bounds on the number of items having specified properties.

Another by-product of the underlying methodology is the easy ability to incorporate non-linear constraints (for example, an upper and lower bound on

the standard deviation of the item difficulties or discrimination parameters). These could be included by performing the necessary computations as each item is considered and, again, measuring the deviation from the specified bounds.

Another future extension of this work is to correct for a potential problem that is similar to (but not identical with) a phenomenon noted by Ackerman. Because we are indifferent about where between the lower and upper target functions the solution falls, the resulting information function is allowed to have "bumps" -- that is, its first derivative changes sign more than once. In our case this problem is bounded (by the upper and lower target bounds), but nonetheless potentially troubling. It could be easily corrected by adding a (non-linear) constraint on the number of these sign changes.

A final extension is to apply this algorithm in the context of computerized adaptive testing. Most current algorithms for implementing adaptive testing (see, for example, Lord (1980), Chapter 10) control the content of an adaptive test through the means of tables specifying the features of the items to be administered. That is, a table is constructed that directs that the first item to be administered to all examinees should have one particular feature, the second item should have another feature, and so forth. This arrangement of tables to control adaptive test content works well when the content structure is relatively simple. If the content structure is complex, that is, each new item selected is subject to many different constraints in addition to constraints on its statistical properties, the table mechanism may become difficult, if not impossible, to implement. The algorithm proposed here may provide the required flexibility in the adaptive testing context also and research on this possibility is currently underway.

We believe that the algorithm has great potential for operational test assembly. The algorithm appears promising for immediate use by large-scale testing programs that produce parallel forms for mass administration, using predetermined content and statistical specifications, similar to the program that was the source of the data for this experiment. In addition, more innovative test assembly procedures, like the generation of multiple parallel forms that are computer-assembled and delivered at the convenience of the user, are compatible with the abilities of the algorithm to build test editions that simultaneously satisfy numerous constraints.

References

Ackerman, T. (1989, March). An alternative methodology for creating parallel test forms using the IRT information function. Paper presented at the 1989 NCME annual meeting, San Francisco.

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

Adema, J. J. (1989). Implementations of the Branch and Bound method for test construction problems (Research Report 89-6). Enschede, The Netherlands: Department of Education, University of Twente.

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

Boekkooi-Timminga, E. (1988). A cluster-based method of test construction (Research Report 88-3). Enschede, The Netherlands: Department of Education, University of Twente.

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

Fletcher, R. (1987). Practical methods of optimization (2nd ed.). New York: Wiley.

IMSL (1989). International Mathematics and Statistics Library. Houston: Math Library, IMSL Corporation.

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

van der Linden, W. J. (Ed.). (1987a). IRT-based test construction. Enschede, The Netherlands: Department of Education, University of Twente.

van der Linden, W. J. (1987b). Automated test construction using minimax programming. In W. J. van der Linden (Ed.), IRT-based test construction. 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 Report 69.3). Princeton, NJ: Educational Testing Service.

Wingersky, M. S. (1983). LOGIST: A program for computing maximum likelihood procedures for logistic test models. In R. K. Hambleton (Ed.), Applications of item response theory. Vancouver, BC: Educational Research Institute of British Columbia.

Table 1: Complete Specifications for a 25-Item Test of Written English.

1. No more than 5 items from any author. For example,

Author	Number of items
Jones	0 - 5
Smith	0 - 5
etc.	

2. The correct answer must appear in each possible position roughly an equal number of times. For 5-choice items, this might appear as

Position	Number of items
1	3 - 7
2	3 - 7
3	3 - 7
4	3 - 7
5	3 - 7

3. The position of the underlined portion of the sentence must appear with roughly equal frequency. For example,

Position	Number of items
beginning	5 - 9
middle	5 - 9
end	5 - 9
all	5 - 9

4. The average sentence length must be between 2.5 and 3.5 printed lines.

5. The type of sentence must be balanced. For example,

Type	Number of items
simple	12 - 14
complex	3 - 5
compound	3 - 5
compound/complex	3 - 5

6. The subject matter must be balanced in detail. For example

Subject	Number of items
business	8 - 13
domestic politics	0 - 2
earth science	0 - 2
health science	0 - 2
humanities	0 - 2
natural science	0 - 2
social science	0 - 2
technical	0 - 2
world affairs	0 - 2

7. Writing problems must be balanced. For example,

Writing Problem	Number of items
agreement	3 - 7
grammatical construction	3 - 7
rhetorical construction	3 - 7
diction	3 - 7
idiom	3 - 7
logical predicate	3 - 7
negation	3 - 7
parallelism	3 - 7
verb form	3 - 7

Table 2: Five Programming Methods Applied to Solve the Relative Information Model With Realistic Test Data.

Method[1]	Final Test Information at θ=-1	θ=0	θ=.5	θ=1	Constraints Satisfied?	CPU Seconds[2]
Method 1	4.138	6.941	6.650	4.938	No: n and 1 range constraint	3.78
Method 2	4.139	6.955	6.715	5.154	No: 3 range constraints	4.06
Method 3	3.391	6.326	6.318	5.078	Yes	4.59
Method 4	3.747	5.917	5.911	4.716	Yes	137.02
Method 5	3.896	5.399	5.521	4.743	Yes	1200.35

[1] Method 1: Single linear solution, crude rounding of decision variables.
Method 2: Single linear solution, improved rounding of decision variables.
Method 3: Single linear solution, optimal rounding of decision variables.
Method 4: First integer solution.
Method 5: Second integer solution.

[2] The CPU was a very fast IBM Model 3090-300S mainframe. Relative times as well as absolute times are important in the comparison.

Table 3: Three Programming Methods Applied to Solve the Absolute Information Model With Realistic Test Data.

Method[1]	Final Test Information at θ=-1 (2.80)[3]	θ=0 (3.50)	θ=.5 (3.60)	θ=1 (3.40)	Constraints Satisfied?	CPU Seconds[2]
Method 1	2.82	3.52	3.63	3.46	No: 7 range constraints	2.04
Method 3	2.78	3.30	3.35	3.30	No: 6 range constraints	4.52
Method 4	2.90	3.71	3.87	3.68	Yes	122.26

[1] Method 1: Single linear solution, crude rounding of decision variables.
Method 3: Single linear solution, optimal rounding of decision variables.
Method 4: First integer solution.

[2] The CPU was a very fast IBM Model 3090-300S mainframe. Relative times as well as absolute times are important in the comparison.

[3] Values of the target information function are listed in parentheses.

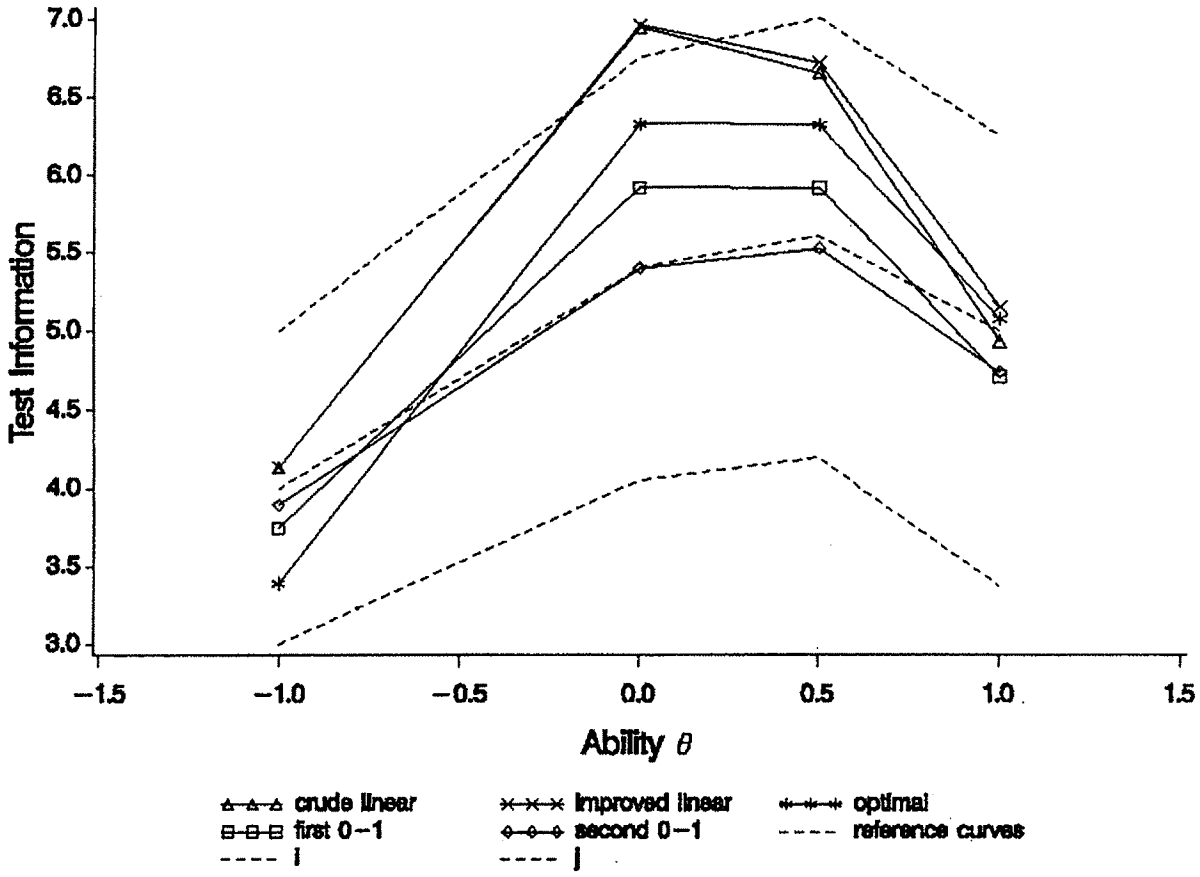

Figure 1. Five methods to solve the relative information model with realistic simulated data.

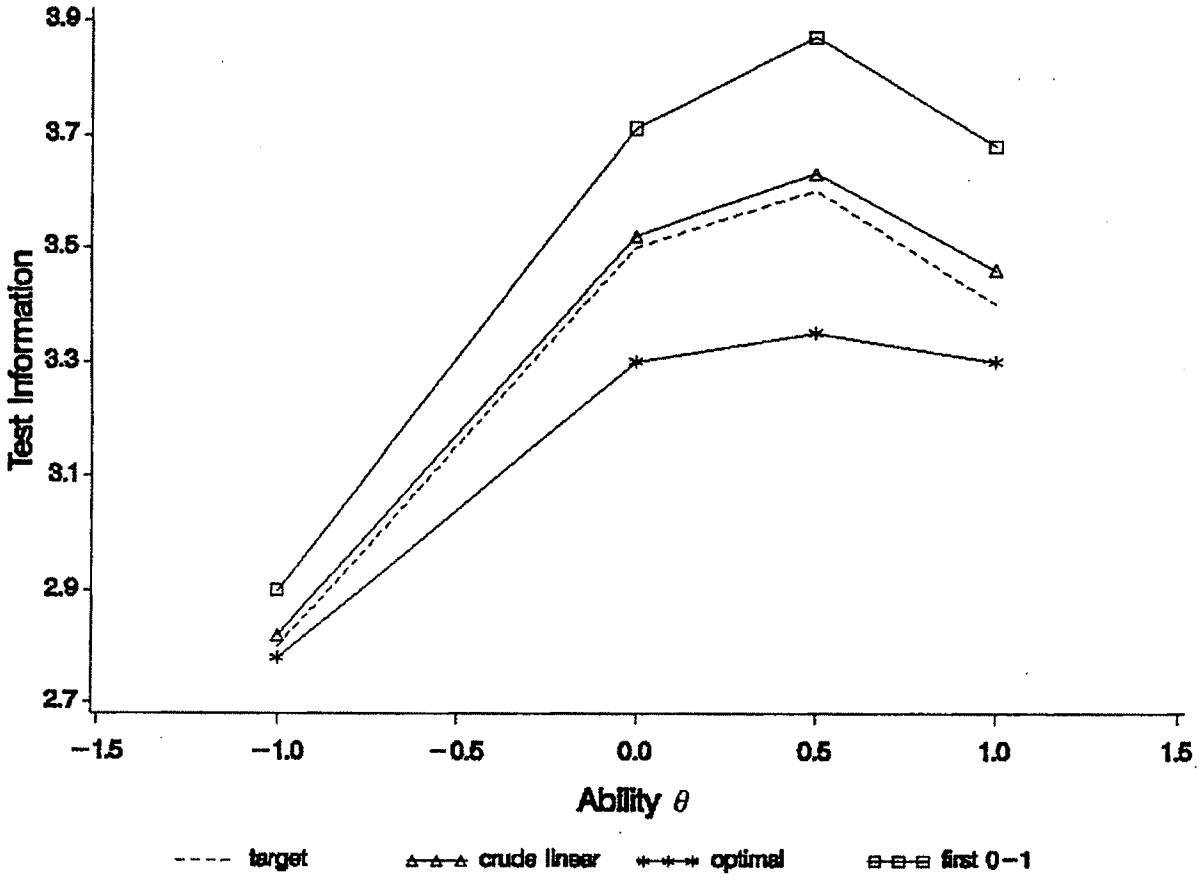

Figure 2. Three methods to solve the absolute information model with realistic simulated data.

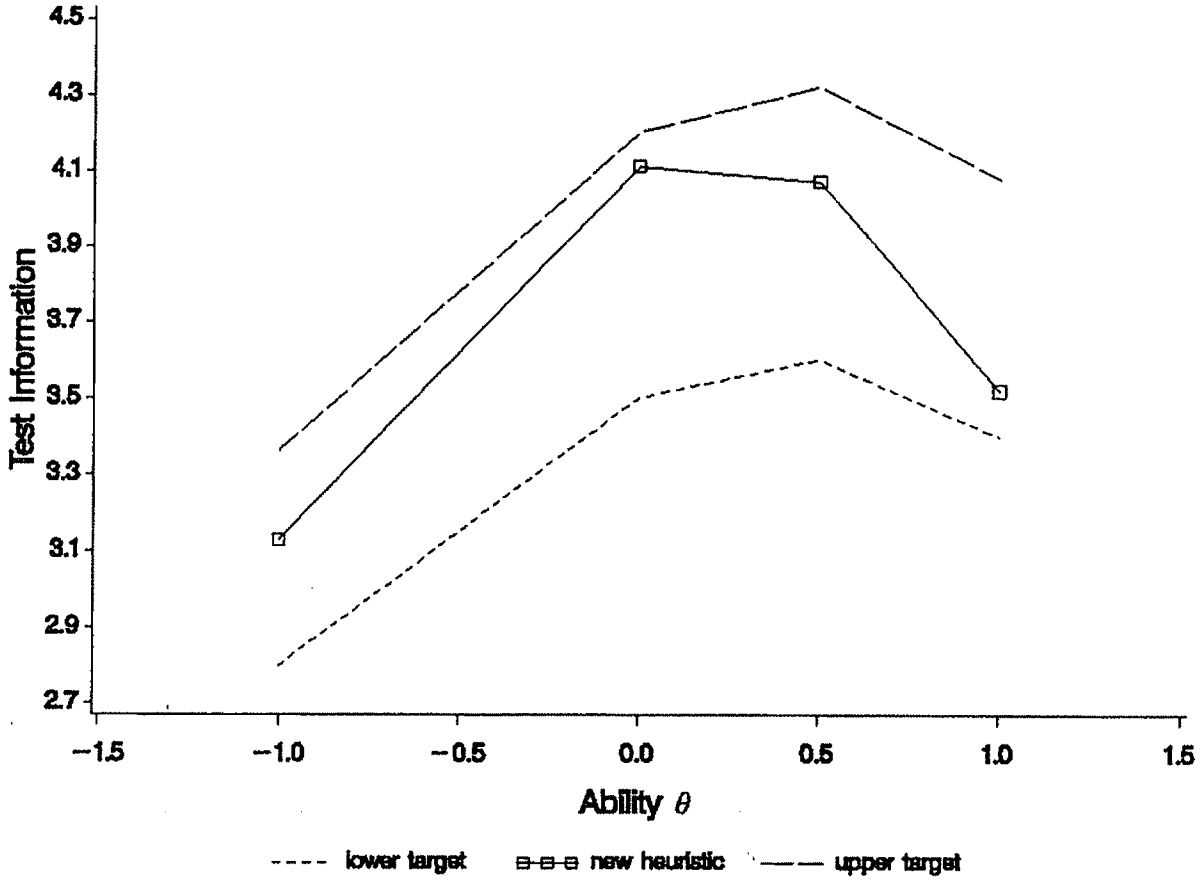

Figure 3. Successive item replacement heuristic method applied to realistic simulated data (solid line). Upper and lower targets are plotted with large and small dashes respectively.