Measurement and Research Department Reports 2004-6

Infeasibility in Automated Test Assembly Models: A Comparison Study of Different Methods

Hiddo A. Huitzing
Bernard P. Veldkamp
Angela J. Verschoor

Measurement and Research Department Reports 2004-6

Infeasibility in Automated Test Assembly Models: A Comparison Study of Different Methods

Hiddo A. Huitzing University of Groningen
Bernard P. Veldkamp University of Twente
Angela J. Verschoor Citogroep

Citogroep
Arnhem, 2004

Abstract

Several techniques exist to automatically put together a test meeting a number of specifications. In an item bank, the items are stored with their characteristics. A test is constructed by selecting a set of items that fulfills the specifications set by the test assembler. Test assembly problems are often formulated in terms of a model consisting of restrictions and an objective to be maximized or minimized. A problem arises when it is impossible to construct a test from the item pool that meets all specifications, that is, when the model is not feasible. Several methods exist to handle these infeasibility problems.

In this paper, test assembly models resulting from two practical testing programs were reconstructed to be infeasible. These models were analyzed using methods that either forced a solution (Goal programming, Multiple-Goal programming, Greedy Heuristic), that analyzed the causes (Relaxed and Ordered Deletion Algorithm, Integer Randomized Deletion Algorithm, Set Covering and Item Sampling), or that analyzed the causes and used this information to force a solution (Irreducible-Infeasible-Set Solver). Specialized methods like the Integer Randomized Deletion Algorithm, and the Irreducible-Infeasible-Set-Solver performed best. Recommendations about the use of different methods are given.

Keywords: goal programming, infeasibility analysis, integer programming, test assembly.

1. Introduction

Test construction is an important step in high stakes educational measurement. The purpose of test construction is to assemble tests from a pool of items that meet the specifications developed by test committees. Because of the importance the test results of an admission test, a final exam, or the SAT (Scholastic Aptitude Test) might have for the examinee's life, the test construction algorithms have to be developed carefully and quality control is necessary. In order to guarantee quality several Automated Test Assembly (ATA) models have been developed.

Most ATA models are based on mathematical programming techniques. A commonly used objective is to measure the ability level of the examinees as precisely as possible, under restrictions defined by the test specifications. These restrictions might specify the content of the test, the item type, time limit, gender-neutral orientation of the items, minority orientation, or total word count. ATA models have been developed for Paper & Pencil tests (Adema, Boekkooi-Timminga, & Van der Linden, 1991, Armstrong, Jones, & Wang, 1995, Van der Linden, & Adema, 1998, Van der Linden & Boekkooi-Timminga, 1989, Veldkamp, 2002), for Computerized Adaptive Tests (Stocking & Swanson, 1993, Van der Linden, 2000, Van der Linden & Reese, 1998, Veldkamp & Van der Linden, 2002), for Mastery Tests (Vos, 1999), and for Multi-Stage Tests (Luecht & Nungester, 1998,2000).

All applications of ATA deal with Linear Programming (LP) models with binary decision variables x_i, $i = 1, \ldots, I$, and a number of constraints, indicated by $j = 1, \ldots, J$. Here the index i refers to items, and I is the total number of items available in the pool. The choice $x_i = 1$ indicates that item i is included in the test, where $x_i = 0$ describes the choice that it is not. In matrix vector notation, where a bold printed variable or parameter represents a vector or matrix, such a LP model can be written as follows

$$\text{Maximize } \boldsymbol{c}'\boldsymbol{x} \tag{1.1}$$

Subject to

$$\boldsymbol{A}\boldsymbol{x} \begin{matrix} \geq \\ = \\ \leq \end{matrix} \boldsymbol{b} \tag{1.2}$$

$$x_i \in \{0, 1\}, i = 1, \ldots, I, \qquad \boldsymbol{x} = (x_i, \ldots, x_I) \tag{1.3}$$

where $\boldsymbol{c}$ is the objective coefficients vector, and c_i represents the relative importance of item i with respect to the other items according to the objective function. The matrix of constants $\boldsymbol{A}$ has elements a_{ij}, $i = 1, \ldots, I$, and $j = 1, \ldots, J$, and $\boldsymbol{b}$ is the vector of

parameters b_j, $j = 1, \ldots, J$. The parameters b_j are called the bounds of their respective constraints. Decision variables are usually written at the left side of an inequality or equality sign, and the bounds at the right side. This general model will be used throughout this paper.

When some of the constraints in 1.2 contradict each other no test can be constructed. The test assembly model is then said to be infeasible, and causes for this infeasibility have to be found. Several methods have been developed to analyze infeasibility in mathematical programming models (Chinneck, 1993, 1997; Chinneck & Dravnieks, 1991; Greenberg, 1987, 1992; Huitzing, in press a, in press b; Timminga, 1998). The topic of this paper is to describe which approaches have been developed to deal with infeasible test assembly problems, and how they behave in practice for problems of large size.

First, the problem of infeasibility is explained. Then, different methods for analyzing causes of infeasibility are introduced. Two test assembly problems illustrate the methods. Finally, the methods are discussed and recommendations about their use are given.

2. Infeasibility in Linear Programming Test Assembly Models

Disregarding typing errors in writing the demands into a mathematical model, there are two main reasons of infeasibility. The first is a contradiction between the demands. For example, wanting to assemble a test that consists of ten items, while also requesting that for the three specified content categories, say mathematics, geography and biology, at most three items are selected. The second main reason of infeasibility is a deficient item bank. As a very simple example, consider the previous test assembly model, where a test of ten items has to be assembled, containing at least three categories (and no constraints on the number of items of each category), while the item bank only has items on biology and mathematics. A deficient item bank appears in a mathematical model either as a contradiction between the number of decision variables and the constraint bounds, i.e., the item bank is too small, or as a contradiction between the coefficients in the constraints and the constraint bounds, i.e., the item bank cannot satisfy the specific item characteristic demands. In the first case it is clear that the item bank has to be extended, while in the second case, the quality of the items in the item bank is deficient. From a mathematical point of view, all types of infeasibility are of the same form, and are caused by a contradiction between one or several groups of constraints.

In the short term, to solve infeasibility some constraints have to be violated, i.e., they have to be adjusted or deleted. This means that either the bound has to be relaxed,

or the coefficients in the constraint have to be adjusted. As a long-term strategy, more decision variables (i.e., items) have to be added to the model (i.e., the item bank). If, in the short term, a test has to be assembled anyhow, a feasible model has to be 'forced'. As the item characteristics, expressed as the elements of the matrix $\boldsymbol{A}$, are usually considered as fixed, only adjusting the constraint bounds can force a feasible model. A small illustration is the following set of constraints:

$$\sum_{i=1}^{6} x_i = 3 \quad (2.1)$$

$$x_1 + x_2 + x_3 \leq 1 \quad (2.2)$$

$$x_4 + x_5 + x_6 \leq 1 \quad (2.3)$$

$$x_i \in \{0, 1\},\ i = 1, \ldots, 6. \quad (2.4)$$

It is obvious that Constraints 2.1, 2.2 and 2.3 together are in conflict. Possibilities to force a feasible model include to enlarge the bound of either Constraint 2.2 or Constraint 2.3 to 2, or to set the bound of Constraint 2.1 to 2. In those cases the original constraints are said to be violated.

In practice most infeasibilities will be caused by an item bank deficiency. To repair the infeasibility, the item bank should be updated, i.e., expanded with new or better-suited items. However, before that can be done, the causes of infeasibility should become clear.

3 Theory of Infeasibility Analysis

A first step in the analysis of infeasibility is to detect and pinpoint the causes. In the literature, there exist a number of concepts that can elucidate causes of infeasibility. An important definition is the following. An Irreducible Infeasible Set of constraints (IIS) (Chinneck & Dravnieks, 1991; Chinneck, 1997; Timminga, 1998; Huitzing, in press a) is a minimal set of constraints that together are infeasible, but for which every subset of constraints is feasible. Thus taking one or more constraints out of an IIS ensures that the remaining set of constraints is feasible again (for a proof see Chinneck & Dravnieks, 1991). A small illustration at the end of this paragraph will perhaps clarify the mathematical formulation. An infeasible model can have several IISs, which can also overlap, i.e., a constraint can belong to more than one IIS. A Minimal Cardinality IIS Set

Cover (MCISC) (e.g., Amaldi, 1994; Chinneck, 1997, 2000; Huitzing, in press b) is set of constraints in which all IISs of the infeasible model are represented by at least one constraint. Removing even one constraint from a MCISC means that one or more IISs are not represented in the MCISC anymore. There can be several MCISCs in an infeasible model, but they all share the next feature. Removing a MCISC from an infeasible model makes the remaining set of constraints feasible again. This follows from the fact that removing a MCISC is the same as removing one or more constraints from every IIS in the infeasible model. Each MCISC has a complementary set of constraints, called a Maximum Feasible Set of constraints (MFS), which is a maximum number of constraints that together are still feasible, while adding another constraint (of the original infeasible model) would make the set infeasible again.

To elucidate these concepts and their potential use in practice, we will present an example. Say we want a test that fulfills the following five demands

(3.1) the test should contain not more than three items;
(3.2) the test should contain two items on history;
(3.3) the test should contain two items on mathematics;
(3.4) the test should contain one item on geography;
(3.5) in a test with an item on geography only one item on mathematics is allowed.

Moreover, suppose that all items have one subject only, but that there are sufficient items on all subjects. The above model is infeasible. Already Demands 3.1, 3.2 and 3.3 are together infeasible. Removing any of these three demands from the set {3.1, 3.2, 3.3} makes the remaining set of two constraints feasible again. Thus, {3.1, 3.2, 3.3} is an IIS. Another IIS is {3.3, 3.4, 3.5}. If we want to make the Model 3.1-3.5 feasible again, we must remove a number of demands. Of course, a minimal number of demands to remove is preferred. From both IISs at least one demand must be removed, and 3.3 seem obvious choices. A MCISC is {3.3}, as it covers both IISs, and its complementary MFS is {3.1, 3.2, 3.4, 3.5}.

4. Methods

Different methods that are able to either bypass the infeasibility problem or analyze the causes are described in the literature.

4.1. Forcing methods

Forcing methods force a solution to the infeasible model by violating the constraints. These methods result in a test, but the test does not meet all the specifications. Forcing methods are sometimes referred to as repairing methods. However, the term repairing is not ambiguous, as it also implies that true errors are really mended, while in fact a solution is forced by violating the constraints, therefore we prefer the term 'forcing methods'.

4.2. Goal Programming Models

While a standard LPTA model cannot give a solution in the case of infeasibility, Goal-Programming (GP) models (Swanson & Stocking, 1993; Timminga, 1998) can help. One such model is the Weighted Deviations Model (WDM) (Swanson & Stocking, 1993; Stocking & Swanson, 1993; Stocking, Swanson & Pearlman, 1993). The general form of a goal programming is

$$\text{Minimize} \sum_{j=1}^{J} v_{+j}\, d_{+j} + \sum_{j=1}^{J} v_{-j}\, d_{-j} \tag{4.1}$$

Subject to

$$\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{d}_{+} - \boldsymbol{d}_{-} \begin{matrix} \geq \\ = \\ \leq \end{matrix} \boldsymbol{b} \tag{4.2}$$

$$x_i \in \{0, 1\}, \qquad i = 1, \ldots, I, \tag{4.3}$$

$$d_{+j}, d_{-j} \geq 0, \qquad j = 1, \ldots, J, \tag{4.4}$$

where d_{+j} and d_{-j} are the amounts by which the constraint are violated positively or negatively, called the deviation variables, whose weighted sum has to be minimized (i.e., the goal function) and the non-negative coefficients v_{+j} and v_{-j} are the weights. In an optimal solution of a GP model, for every constraint j at most one of d_{+j} and d_{-j} will be positive when the bound is violated, i.e., for a '≥' constraint j, d_{+j} will be positive if constraint is violated, while in the case of a '≤' d_{-j} will be positive. For an equality constraint either d_{+j} or d_{-j} will be positive if the equality is not satisfied. The weights v_{+j} and v_{-j} represent the importance of not violating constraint j.

4.3. Multi-Objective Goal Programming

Adapting the constraints by means of a goal programming model results in a feasible region. However, sometimes the test assembler is not only interested in obtaining a set

of feasible solutions, but also has other wishes, such as, e.g., maximizing the test information function. This can be achieved by varying the coefficients v_{+j} and v_{-j}, usually by trial-and-error, or by adding a second objective function,

Maximize $\boldsymbol{c'x}$ (4.5)

which is the original objective function of the Model 1.1–1.4. Using a goal-programming model with two objective functions, also called a Multi-Objective Goal-Programming (MOGP) model (Nemhauser & Wolsey, 1988; Nering & Tucker, 1993; Veldkamp, 1999), one has the following choices. Either solve the model using only Model 4.1-4.4, and when a feasible region has been forced by relaxing some of the constraints, solve the model again now with Equation 4.5 as objective function, or solve the goal programming model using a combination of the two objective functions, where each objective is given a weight representing its importance. The drawback of the first choice is that, after solving Model 4.1-4.4, the obtained feasible region often consists of one or a few feasible solutions only, which gives very little room for the second objective function. The shortcoming of the second choice is that combining the objective functions into one usually leads to larger violations of the original constraints bounds. In this paper multi-objective goal-programming models apply a combination of the objective functions.

4.4. Greedy Heuristics

Most used in test assembly are greedy heuristics: start with an initial item and add a next best suited item according to the wishes of the test assembler until a certain prefixed number of items is reached. In the case of infeasibility, these heuristics systematically allow the constraints to be violated in a more or less controlled way. This is done by setting weights representing the cost of violating a constraint, and minimizing the sum of these costs when choosing a next item for the test, effectively introducing a goal function. For example, the WDM and the Normalized Weighted Absolute Deviation Heuristic (NWADH) (Luecht, 1998) are usually solved with greedy heuristics (the WDM can also be solved with a LP solver). Both WDM and NWADH have successfully been used in test assembly for several years. However, while they do deliver a solution even if the model was infeasible, they do not focus on the causes of infeasibility, but allow constraint bounds to be relaxed 'in the case of need'. Moreover, they do not present the test assembler a tool to fix the causes for future tests. In this paper a modified version of the heuristic WDM will also be investigated.

The original objective function, which is to maximize the test information, is also a constraint in the WDM with Heuristic solver. Setting its bound as a very large number the Heuristic (while seeking to minimize the deficit between the value of the test information in the constraint and the constraint bound) maximizes the test information.

5. Analyzing methods

While GP models offer a forced solution in the case of infeasibility, they do not analyze its causes. Some methods described in the literature are more focused on finding causes of infeasibility such as IISs, MFSs or MCISCs. Information about the causes can help a test assembler to decide whether additional items have to be added to the item bank, or whether the test assembly model has to be reformulated.

5.1. Deletion Algorithms IRDA and RODA

An algorithm capable of distilling an IIS out of an infeasible LPTA model is the deletion algorithm (Chinneck & Dravnieks, 1991; Huitzing, in press a). The deletion algorithm works as follows. An arbitrarily ordered set of constraints is checked for feasibility. If the set is not feasible then the first constraint of the ordered set is temporarily excluded, and the remaining set is checked for feasibility. If this new set is feasible then the temporarily excluded constraint is restored in the set, otherwise it is definitively excluded. The next step is to temporarily remove the next constraint of the ordered set, check the remaining set for feasibility, and restore the constraint in the case of feasibility or delete in the case of infeasibility. This is done for all the constraints in the original ordered set. The remaining set of constraints is then an IIS. This algorithm is usually programmed in commercial software packages such as CPLEX (ILOG, 2001) to detect an IIS, and will be called the Relaxed and Ordered Deletion Algorithm (RODA).

The RODA programmed in CPLEX only uses the so-called relaxed LP model, where all binary variables are relaxed into being real variables that can take values between 0 and 1. Usually integer LP models (i.e., the variables must take on integer values) are first solved for a relaxed version, and, by means of an algorithm (e.g., the branch-and-bound algorithm; Nemhauser & Wolsey, 1988; Nering & Tucker, 1993), integer values for the variables must be found. It is, however, possible that while the relaxed model is feasible, the integer model is not. A DA that only analyzes the relaxed version of the LPTA model will then fail.

Because a deletion algorithm makes use of a given order in the set of constraints, running the deletion algorithm twice on the same set will always give the same result. However, an infeasible model can have a number of different IISs. Randomizing the

order of the constraints may yield a different IIS. A version of the deletion algorithm that first randomizes the order of constraints and respects the binary variables, is programmed in NuzLight (Huitzing, 2002), and will be called the Integer and Randomized Deletion Algorithm (IRDA).

5.2. Set Covering and Item Sampling Method

It was argued that a good tool for infeasibility analysis is the use of IISs. Although algorithms exist that are able to distill an IIS out of an infeasible model, this can be quite a time-consuming job. Feng (1999) showed that another option is to sample a large number of points in the solution space, i.e., the vectors $\boldsymbol{x}$, and using a set-covering model (Boneh, 1984; Papadimitriou & Steiglitz, 1982; Wolsey, 1998), to find one or more IISs. Huitzing (in press b) implemented the idea of item sampling and set covering (SCIS) to test assembly models and LPTA models in particular. Also, it was showed that SCIS can also be used to find MCISCs, and, probably most appealing to test assemblers, MFSs, i.e., the largest set of constraints of the original model that together is still feasible. The remaining set of constraints, after deleting an MFS in an infeasible model, form an MCISC, which can be solved by means of goal programming model.

The first decision when using SCIS regards the sampling of the items. In LPTA the variables are usually binary, either an item is selected or not. Therefore, at each replication, a vector of zeros and ones of size I is sampled, where each one means that the corresponding item has been selected for the test, and a zero otherwise. Such a replication corresponds to a draw of a set of items from the item bank, i.e., a test. A test consisting of, say, 500 items out of an item bank of 1000 items when the test length is 40 will not satisfy many constraints for that test, and will not give us much information. Therefore we set the probability that a variable gets the value one equal to the number of items demanded for the test divided by the number of items in the item bank, increasing the probability that the constraint representing the test length will be satisfied. The probability that the other constraints will be satisfied is now also higher.

In the next step, this set of selected items is checked whether it satisfies the constraints of the LPTA model. If the set of selected items satisfies all constraints, we have a feasible test. But in the case of infeasibility this will not occur. One sample vector only represents one sampled set of selected items. But by sampling a large number of such sets of selected items, information on which constraints are never satisfied and which constraints are relatively easily satisfied can be obtained. The general form of a SCIS algorithm can now be given

1. Generate a large number of random points in $\{0,1\}^I$ with the aim to cover all of the combinations. Any such point represents a set of selected items, i.e., a test;
2. Check each test for all individual constraints: construct a matrix P where each row z represents a test, and each column j a constraint. If the test z does not satisfy constraint j then $p_{jz} = 0$, otherwise 1.
3. To find an IIS solve the following set covering problem,

$$\text{Min}\{\boldsymbol{e}^T\boldsymbol{y} \mid \boldsymbol{P}\boldsymbol{y} \geq \boldsymbol{e}, \boldsymbol{y} \text{ binary}\} \quad (5.1)$$

where $\boldsymbol{e}$ is a vector with elements 1, and $y_j = 1$ if element j, $j \in \{1,\ldots,J\}$, is selected, otherwise it is 0. Supposing that all combinations have been sampled, the solution to Model 5.1 is minimal set of constraints that makes the whole model infeasible, which is a smallest cardinality IIS (i.e., a smallest IIS, where smallest refers to the number of constraints in the IIS). The solution may not be unique, as a number of smallest IISs may exist.

The solution space is the set of all possible solutions, i.e., all possible combinations of selecting items from an item bank. Each constraint divides this solution space into two subspaces, i.e., one subspace where the constraint is satisfied and one where it is not. In theory the number of subspaces of a LPTA model with J constraints is 2^J. For example, a model of 100 constraints may have $1.26\ 10^{31}$ subspaces. Huitzing (submitted) showed that in practice this number is much lower. This is important because infeasibility is caused by a contradiction between constraints, and to be certain to find the causes (i.e., the IISs) of the infeasibility, all subsets should be found.

Regarding the first step of the SCIS, where it states, "sample with the aim to cover all subsets", some remarks are in place. In the SCIS, however, a subset is found by whether it contains a (sampled) combination of selected items (i.e., the vector of size I of ones and zeros). Some subsets created by the constraints may never be detected if they do not contain any integer coordinates. On the other hand, an item bank often has more items than constraints, and several distinct combinations of selecting items may lay in a same subset of the solution space (and thus do not give new information on the causes of infeasibility). Moreover, while sampling the 0-1 vectors representing the selected sets of items out of an item bank is relatively cheap in computer time, the set covering algorithm, i.e., calculating the solution to Model 5.1, can also take large amounts of computer time for larger matrices P. Therefore not all combinations of selecting items from an item bank need to be sampled, but a 'large number with the aim to cover all combinations'.

The solutions of the SCIS algorithm, i.e., the IISs, should be seen as indications of IISs. A rule of thumb is to start with 1000 times the number of items in the item bank as the number of replications. Then solve the Model 5.1 with the generated matrix *P*, and then check whether the found IIS is a true IIS. If its components, i.e., the constraints, are feasible together, then start again (possibly with a higher number of replications).

5.3. Combined

In combined methods, the causes of infeasibility are analyzed and solved. In the first step an analyzing method is applied. In the second step, a solution to the cause is forces, based on an infeasibility measure.

5.4. IIS-Solver

A combined method focused on infeasibility caused by IISs, is the IIS-Solver (Huitzing, in press a). This heuristic works as follows. If a model is infeasible, an IIS is found by means of either IRDA or RODA, as explained in the previous section. Remember that in order to make an IIS feasible again, at least one bound must be relaxed. The IIS-Solver proceeds by calculating a so-called 'measure of infeasibility' for each constraint in the IIS, where the measure of infeasibility is a comparing tool to make a choice between constraints. Huitzing (in press a, see the section on 'Possible Loss Functions') shows that a good measure of infeasibility is the following one

$$m_z(b_{j_z}^F) = \begin{cases} w_j \left(\sum_{z \in Z} b_{j_z}^F \right) / \, |b_j| & \text{if } b_j \neq 0, \\ w_j \left(\sum_{z \in Z} b_{j_z}^F \right) / \, \delta & \text{if } b_j = 0, \end{cases} \tag{5.2}$$

where $w_j \geq 0$ are the weights of the constraints; $0 < \delta < \min_{y \in \{1,\ldots,J\}} \{b_y: b_y \neq 0\}$; z is the index of the IIS and $b_{j_z}^F$ is the amount by which a constraint must be relaxed (all constraints are actually rewritten as '<' or '≤' constraints). To find $b_{j_z}^F$, Model 5.3–5.6 is solved for each constraint in the IIS

$$\text{Min} \quad b_{j_z}^F \tag{5.3}$$

$$\text{s.t.} \quad \boldsymbol{a_j x} \leq b_j \quad \text{for all } j \in \text{IIS}_z,\ j \neq j_z \tag{5.4}$$

$$\boldsymbol{a_{j_z} x} \leq b_{j_z} + b_{j_z}^F \tag{5.5}$$

$$x_i \in \{0, 1\},\ i = 1, \ldots, I, \qquad \boldsymbol{x} = (x_i, \ldots, x_I) \tag{5.6}$$

In the IIS-Solver only one constraint is relaxed at a time. The choice for the constraint to relax can be done automatically (e.g., by using 5.2) or by hand. Once the IIS under inspection has been repaired, the whole model is solved again. If it is still infeasible, a next IIS is sought and repaired as before, until the whole model is feasible again. Then the IIS-Solver will solve the modified model with the original objective function. The IIS-Solver can also be set to respect hard constraints.

6. Theoretical Evaluation of the Methods

It is interesting to compare the different methods and state our expectations beforehand. Because the goals of forcing methods and analyzing methods differ, they will be dealt with separately.

6.1. Forcing Methods

Forcing methods patch up the infeasibility by relaxing the constraint bounds. Criteria must be chosen to compare the methods. Obvious choices are the number of violated constraints, the sum of constraint bound violations, the value of the original objective function (maximize the test information function), and the time needed to force a solution. Only looking at one of these criteria is not sufficient as they are interdependent. Obviously, by allowing all constraints to be violated with no penalty and maximizing the test information function, all items in the item bank will be selected, scoring high on the criterion of 'value of the original objective function'. On the other hand, this would result in high costs in terms of constraint bound violations and in terms of the number of constraints violated.

When a goal-programming model is applied, the weighted sum of violations is minimized. Application of this method entirely focuses on constraints. Violation of constraints has to be minimized. In the Multi-Objective goal-programming model, the amount of information in the resulting test is also taken into account.

The third method does not concentrate on how to model the problem, but on the algorithm applied for solving the model. When using a goal-programming model with a greedy heuristic all constraints are open to relaxation and a local search at each iteration for a next best item to add to the test takes place. But, what might have been a good decision an in earlier iteration, may have negative influence on later choices. The advantage of greedy heuristics lies in computer time needed to find a solution. However, the question remains whether they reach an optimal solution.

Four different criteria where introduced that might be used to compare the methods. The GP method is expected to result in smallest number of violations and smallest sum of constraint violations. The MOGP is expected to result in highest value of the TIF. The greedy heuristic is expected to result in smallest computation time.

6.2. Analyzing Methods

The RODA, the IRDA and the SCIS all search for IISs. While the RODA and the IRDA make use of a deletion algorithm in an infeasible model, the SCIS takes a probabilistic approach.

We have already stated the drawback of the RODA, which applies the deletion algorithm to the relaxed LPTA model only. An infeasibility which is caused by the fact that the variables are integer will not be detected, which can be a serious drawback in the case of LPTA where the variables are binary. The reason it is used is that a relaxed LPTA model takes much less computer time than an integer LPTA model. IRDA does not suffer from these problems. However, for both RODA and IRDA it is a drawback that at each iteration when a constraint is temporarily dropped, the remaining set of constraints must be checked for feasibility, which is mathematically as difficult as solving a LP model. The necessary computer time when using RODA or IRDA can therefore rise considerably.

When a SCIS is used, a Set Covering (SC) model must be solved to find an IIS. SC models can be very hard to solve (Papadimitriou & Steiglitz, 1982; Wolsey, 1998) and take much computer time. Moreover, because of the probabilistic nature of the SCIS, it is not guaranteed that an IIS will be found. However, sampling the items should take a matter of seconds with the right software, and one can start with lower number of replications and gradually increase the number of replications if no IIS is found.

We expect the SCIS to perform better in terms of computer time, than the RODA and the IRDA. Moreover, we have some objections on whether the RODA will perform well because of the integer nature of the variables in LPTA.

In the Conclusion and Discussion we will consider our expectations of the methods and discuss their achievements.

7. Numerical Examples

The different methods were applied to two practical test assembly cases. Case 1 comes from PPON (*Periodieke Peiling van het Onderwijsniveau;* Periodical survey of educational level, CITO). PPON is a periodical assessment of the level of education in the Netherlands. It is comparable to the National Assessment of Education Program (NAEP)

in the USA. CITO (National Institute for Educational Measurement) in the Netherlands oversees this test, which is carried out for the Dutch Department of Education. *PPON Biologie* (PPON biology) measures whether students of the last year of the primary education meet certain standards set previously by a panel of experts, and is repeated every five years.

Case 2 also comes from CITO. WISCAT-pabo (CITO) is an adaptive test for measuring deficiencies in Mathematical knowledge of pabo students. Pabo is the Dutch higher education for teachers for primary education. The test serves to assign the students to the different levels of a course.

For both cases the item bank and a test assembly model were available. In practice these test assembly models are feasible, but for the purpose of this study the models were slightly altered to make them infeasible.

7.1. Settings of methods in Comparison Study

Three methods are available for forcing a solution. To apply these methods, the test assembly problems PPON Biology and WISCAT-pabo were transcribed into goal programming models. For every constraint, variables d_{+j} and d_{-j} were introduced that denote the amounts by which the constraint is violated positively or negatively. The forcing methods were implemented in NuzLight (Huitzing, 2002), open source software developed at the University of Groningen.

When the Goal Programming (GP) method is applied, the model is solved with a LP solver but without the original objective function, so as just to minimize the sum of violations. In Multi-Objective Goal programming (MOGP), the cases are solved using again a LP solver, but now the original objective function is included and has the same relative weight as the objective function of minimizing the sum of violations. The Greedy heuristic minimizes at each iteration the total additional unweighted sum of violations. In NuzLight the Greedy Heuristic is programmed in the following way. Starting with zero items, the best item (with respect to the constraints) is found by means of a LP solver and is added to the test. The decision variable representing this item is then set to one in a new constraint. This is then repeated at each iteration until the desired number of items is attained.

For analyzing the causes of infeasibility, three methods have been introduced. The Relaxed Ordered Deletion Algorithm (RODA) is implemented in CPLEX (ILOG, 2001), a commercial software package for solving mathematical programming problems. Both the Integer Randomized Deletion Algorithm (IRDA) and the Set Covering with Item Sampling (SCIS) method are implemented in NuzLight. For the SCIS method the following settings were used. The SCIS was run each time with 10,000 replications and the probability of

selecting an item was set to the test length divided by the total number of items. In this way the average test length of the sampled tests was equal to the total test length.

The IIS-Solver, the only method that analyzes infeasibility and forces a solution, uses a combination of the IRDA (all variables are integer) to find an IIS, and then the GP method to solve this IIS. If the model is not yet feasible, it repeats this process until a solution has been forced. The IIS-solver is also implemented in NuzLight.

Case 1

The PPON Biology 2001 for Grade 6 of primary education has an item bank of 417 items, of which 172 items are singletons, and 245 items are part of groups of two to fifteen items, and are called set-based items. The items have five domains. All items are dichotomous (i.e., scoring is either wrong or true), and are calibrated according to the one-parameter logistic model (OLPM), i.e., all items have an information function that is characterized by two parameters, *a*, the item discrimination and *b*, the difficulty of the item.

A typical PPON biology test assembly model will have one constraint on the test length, three constraints on the target information function, five constraints on the domains of the items, four constraints balancing the number of set-based items, and about 255 constraints defining the item sets. Moreover, there will be a number of constraints on enemy sets, i.e., items which must not appear in a same test, e.g., because they contain clues to each other, and some constraints on minimal choices of items from the larger item sets.

Two IISs were added to a problem. For two content constraints (Constraint Max11 and Constraint Max12) referring to a same set of items, the upper bound and the lower bound were interchanged. As a result the lower bound was larger than the upper bound for that set of items. The second IIS resulted from a matrix specification error. Three enemy constraints (Constraints Max11_1, Max11_2 and Max11_3) were added to the model and the lower bound of content Constraint Max11 was increased. These four constraints resulted in the second IIS.

The results of the comparison study are shown in Tables 1 and 2. For comparison purposes we give the value of the objective function (maximize TIF) of the original feasible model, which is 40.0593. To compare the different forcing methods, the four criteria introduced in the Theoretical Evaluation of Methods section are denoted in Table 1.

Table 1. Forcing Methods in PPON Case

Method	Violated constraints	Total violation	TIF value	Time
GP	2	3	16.7743	1 second
MOGP	21	33	50.2756	3 seconds
Greedy Heuristic	21	33	50.2756	16 seconds
IIS-Solver	2	3	38.9149	150 seconds

GP: Goal Programming model without the original objective function

MOGP: Goal Programming model with the original objective function

Total violation: sum of constraint bound violations

TIF value: Sum of the values of Test Information Function (measured at threepoints)

For GP, the method just violated two constraints (one constraint per IIS). Although this method provided some information about the causes of the infeasibility, it did not enable the test assembler all information to restore the problem. Moreover, the value of the Test Information Function (TIF), namely the original objective function to be maximized, was very low.

When MOGP was applied to the infeasible model, it resulted in a solution that violated 21 constraints and found a maximal value for the Test Information Function. The Greedy Heuristic found the same results, so in Case 1 both the MOGP and the Greedy Heuristic did equally well. The necessary time to solve the model with the Greedy Heuristic is distorting, as in NuzLight at each iteration of the heuristic a new model is solved with a LP solver.

The IIS-Solver correctly identified the causes of infeasibility and proposed to modify two constraints. The resulting value objective function is very close to the objective value of the original problem.

For the analyzing methods, the results are shown in Table 2.

Table 2. Analyzing Methods in PPON Case

Method	Detection		Time
	Lower bound > Upper bound	Error in Specifications	
RODA	Detected	Not detected	-
IRDA	Detected	Detected	50 seconds
SCIS	Detected	Detected	4,5 minute
IIS-Solver	Detected	Detected	150 seconds

RODA: Relaxed Ordered Deletion Algorithm as programmed in CPLEX

IRDA: Integer Randomized Deletion Algorithm as programmed in NuzLight

SCIS: Set Covering with Item Sampling method

The IIS consisting of Constraints Max113 and Max114 was found by all three infeasibility detection methods. Once the infeasibility has been pinpointed down to two inconsistent constraints, it is easy for the test assembler to rectify the error. RODA proposed changing the value of the bound of Constraint Max11, after which the relaxed model did become feasible. However, according to the model expert, this suggestion was not acceptable to test assemblers.

Only the IRDA and the SCIS detected the IIS of Constraints Max11, Max11_1, Max11_2 and Max11_3. The RODA was unable to detect this second IIS, even after following up its proposal of changing the bound of Constraint Max11 and eliminating the first IIS. While the relaxed model had now become feasible, the binary model was still infeasible, but RODA offered no information about its causes.

Because of the random nature of the SCIS and the IRDA, it is remarkable that they needed only two runs to detect both IISs. However, the time needed by SCIS is very high. This is due to the software, which is not professionally and efficiently programmed. Most of the computer time was spent on the sampling, exactly the part that should be fast to implement.

Note also that the IIS-Solver needed 150 seconds to find both IIS, but these 150 seconds include fixing the IISs.

Case 2

The WISCAT-pabo item bank contained 557 items. The items are scored dichotomously, and calibrated with the OPLM. Four content domains have been specified that are divided in several sub-domains. The test length is 40 items. Instead of assembling an adaptive test, two parallel linear versions were assembled. The resulting test assembly model consisted of 823 constraints. To make the model infeasible, it was changed in the same

way as the previous model. The lower- and upper bound of a content constraint were interchanged. Besides, a matrix specification error was introduced. Therefore, three enemy constraints were added to the model. Together with the constraint on the lower bound in the first IIS, these four constraints result in the second IIS.

The results for the comparison study are shown in Tables 3 and 4. The value of the objective function (maximize TIF) of the original feasible model is $3.10525*10^2$.

Table 3. Forcing Methods in WISCAT-pabo case

Method	Violated constraints	Total Violation	TIF value	Time
GP	4	6	$1.41982*10^2$	6 seconds
MOGP	93	153	$3.58534*10^2$	3 seconds
Greedy Heuristic	91	114	$2.36484*10^2$	70 seconds
IIS-Solver	1	1	$3.10525*10^2$	10 minutes

GP: Goal Programming model without the original objective function

MOGP: Goal Programming model with the original objective function

Total violation: sum of constraint bound violations

TIF value: Sum over both tests of the values of the Test Information Function (measured at five points)

The Greedy Heuristic resulted in a solution that violated 91 constraints. For the GP the method violated four constraints. When the objective function was taken into account, MOGP violated 93 constraints, comparable to the Greedy Heuristic, but with a much higher objective function and also a much higher total sum of constraint violations.

The IIS-Solver identified the IIS and proposed how to solve it, resulting in just one constraint violated. The IIS-solver resulted in a TIF of $3.10525*10^2$, again outperforming all other methods.

Table 4. Analyzing Methods in WISCAT-pabo case

Method	Detection		Time
	Lower bound > Upper bound	Error in Specifications	
RODA	Detected	Not detected	-
IRDA	Detected	Detected	9,5 minutes
SCIS	Detected	Detected	30 minutes
IIS-Solver	Detected	Detected	10 minutes

RODA: Relaxed Ordered Deletion Algorithm as programmed in CPLEX

IRDA: Integer Randomized Deletion Algorithm as programmed in NuzLight

SCIS: Set Covering with Item Sampling method

The RODA just identified the first error. Again the fact that it only used the relaxed model of Case 2 to search for an IIS was the reason it did not detect the second error. Moreover, after readjusting the first error it detected, it remained infeasible without giving any clues for the test assembler. For the IRDA, a remarkable result was obtained. It also found only one IIS. When the feasible model was altered, apparently one constraint became part of both errors. Although we did not realize it in advance, the combination of this writing error and error in the specifications matrix resulted into one IIS only. Thus, in retrospect, the RODA was even unable to detect the whole IIS. While the SCIS detected the IIS, it took a fair amount of computer time.

Note that while the IIS-Solver found the only IIS and solved it. The IIS-solver needed 10 minutes to fix a solution, most of this time was actually necessary to detect the IISs with the IRDA (9,5 minutes).

To see what would happen if another specification error was introduced, two more constraints were added, interacting with a third constraint in the model. In Tables 5 and 6 the results are displayed, where we now speak of IIS1, the IIS of the two errors introduced previously, and IIS2, the new error (which we took care not to interact with IIS1). The results are shown in Table 5 and Table 6.

Table 5. Forcing Methods in WISCAT-pabo Extended case

Method	Violated constraints	Total Violation	TIF value
GP	5	7	$1.43397*10^2$
MOGP	94	155	$3.58534*10^2$
Greedy Heuristic	92	115	$2.36484*10^2$
ISS Solver	2	3	$3.12911*10^2$

GP: Goal Programming model without the original objective function

MOGP: Goal Programming model with the original objective function

Total violation: sum of constraint bound violations

TIF value: Sum over both tests of the values of the Test Information Function (measured at five points)

Again the IIS-Solver did best in terms of TIF value versus the number of violated constraints.

Table 6. Analyzing Methods in WISCAT-pabo Extended case

Method	Detection	
	Lower bound > Upper bound	Error in Specifications
RODA	Not Detected	Not Detected
IRDA	Detected	Detected
SCIS	Detected	Detected
IIS-Solver	Detected	Detected

RODA: Relaxed Ordered Deletion Algorithm as programmed in CPLEX

IRDA: Integer Randomized Deletion Algorithm as programmed in NuzLight

SCIS: Set Covering with Item Sampling method

Also for this problem, the RODA was unable to find the second IIS because of its binary nature.

7.2 Implications of settings

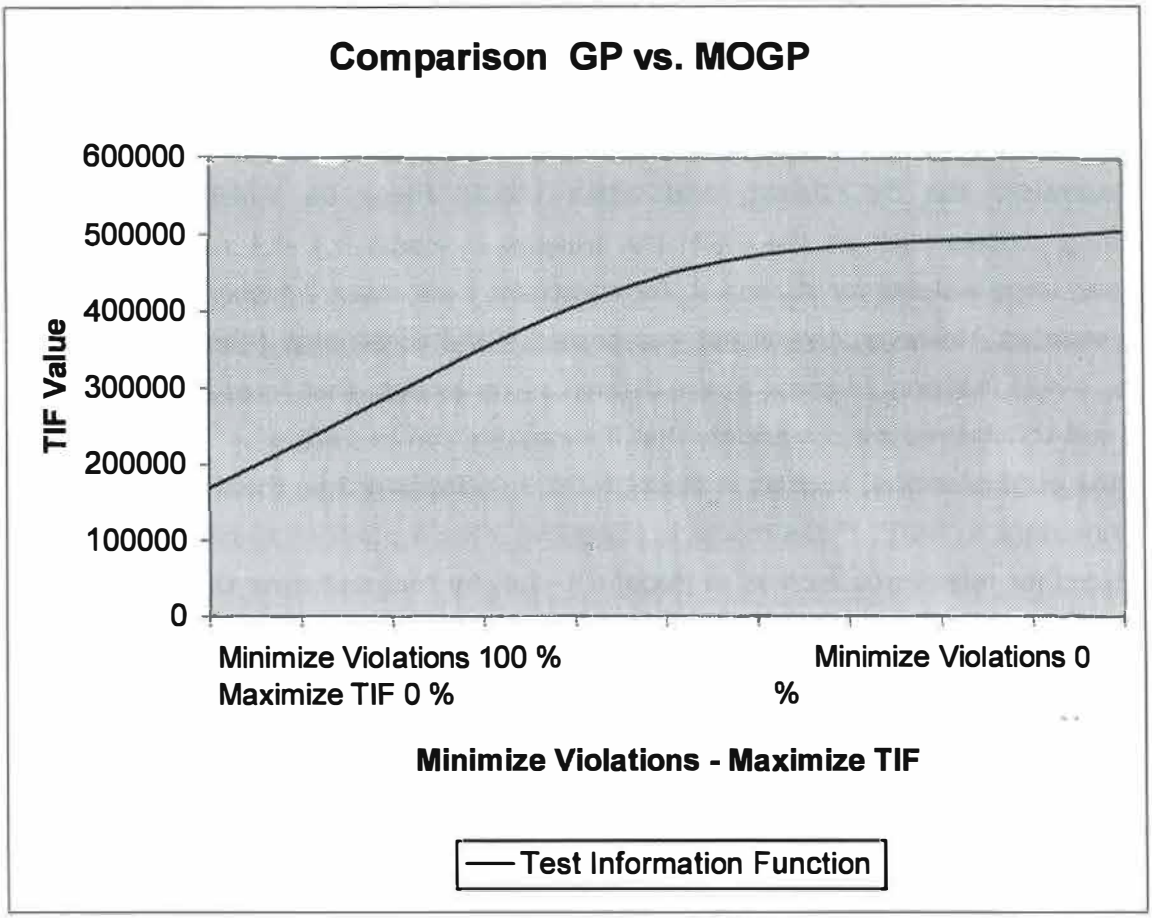
Hard and Soft Constraints

One of the drawbacks of many presently used ATA models is the difficulty to control which constraints can be relaxed, and which should never be violated. In Goal Programming models such as the WDM the weights in Model 4.1–4.4 can guide this process, and large weights for d_{+j} and d_{-j} for constraint j will make it highly unlikely that it will be violated. However, this is not guaranteed. Hard constraints (Chinneck, 1997; Timminga, 1998; Huitzing, in press a) are defined as constraints that must be met, while so-called soft constraints are constraints that if necessary can be relaxed.

If the original model, such as in Model 1.1-1.3, is feasible then there is no need to set any constraint as 'soft'. If the model is infeasible, then a priori all constraints should be considered for relaxation, such as in Model 4.1–4.4. By trial-and-error the weights for d_{+j} and d_{-j} can be set such that an acceptable test is found. In a situation where there is no time for repair, the test assembler can decide that some constraints should never run the risk of being violated, and set them as 'hard'. An example of a hard constraint is, e.g., the test length, and an example of a soft constraint is constraint referring to the IRT difficulty parameter values representing item difficulty, which are only estimates anyway.

7.3. Additional comparison GP and MOGP

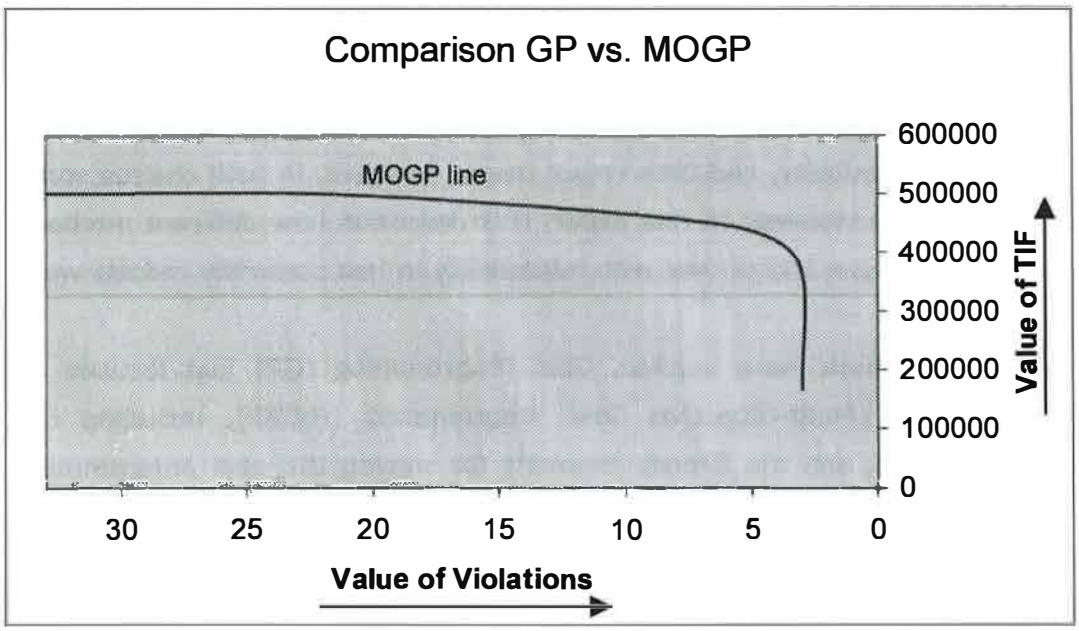
The main difference between GP and MOGP lies in the objective function. In GP the original objective function is not taken into account, while in MOGP it is included in the model. The weight of the objective function in both methods is zero for the GP method and 1 for the MOGP. It is interesting to see what would happen if we gradually increase the importance of the objective function, that is, if we gradually set the weight of the original objective function higher. In Figure 1 a graph is depicted with on the x-axis the relative importance (0 to 100 percent) of the goal function 'Minimize Violations', where its complement is the relative importance (100 to 0 percent) of the original objective function 'Maximize TIF', and on the y-axis the value of the TIF. Note that if the weight of the original objective function is zero we have the situation as in GP.



TIF: Test Information Function * 1000

Figure 1. Setting Weights on Conflicting Objective Functions

In Figure 2, the two conflicting objectives are shown again. On the x-axis the sum of the constraint bound violations and on the other axis the value of the TIF are shown. An arrow below the axis shows to the direction of its objective function. A lower total sum of violations entails a lower TIF, and it is to the test assembler to make a choice here, when using a Multi-Objective Goal Programming method along the line shown in Figure 2.



TIF: Test Information Function * 1000

Violations: Exceeding the Constraint Bounds

⟶ : arrow denoting direction of objective function

Figure 2. Results of Conflicting Objective Functions

Two things are noteworthy. First note that the MOGP line does not go all the way to the origin of the quadrant. The MOGP line never reaches the zero of the value of violations because the model is infeasible: if we want to force a solution, we must violate a number of constraints. Moreover, the constraint concerning the test length is set as hard thus, once we force a solution, there will always be a number of selected items, ensuring that the value of the test information function will never be zero too. Second note that the MOGP line goes straight up in value of the TIF for a same number of constraint violations. Indeed, at the right end of the MOGP line we have the situation as in GP. The original objective value plays no role here. But adding the original objective function while giving a very large relative weight to not violating the constraint bounds induces the solver to search for a best solution within the set of feasible solution defined by first meeting the goal function. This solution is actually the solution found by the IIS-Solver, and by again changing the relative weights of the original objective function can also be found by the Greedy Heuristic.

8. Conclusion and Discussion

In the case of infeasibility of test assembly models two choices are at hand, either directly 'force' a solution, without knowing the causes of the infeasibility, or first find and analyze the causes of infeasibility, and then repair these. However, in both choices some of the constraints must be violated. In this paper, it is described how different methods that are developed to analyze and or deal with infeasibility in test assembly models work in practice.

Three forcing methods were applied. Goal Programming (GP) just focuses on violations of constraints. Multi-Objective Goal Programming (MOGP), including the original objective function, and the Greedy Heuristic for solving the goal programming model are examples of methods where not only the violations must be kept to a minimum, but also where the value of the test information function is taken into account.

About differences in performance some expectations were formulated in the Theoretical Evaluation section. As expected, Goal Programming (GP) minimized the number of violations and the total sum of violations. The method was able to find causes of infeasibility for both cases, and forced a solution by a minimal violation of the constraints. MOGP was expected to result in the highest value for the TIF. For the PPON example, the Greedy Heuristic performed as good as the MOGP. For the WISCAT example, the MOGP performed best w.r.t. this criterion. Our expectations about necessary computer time were not met. For the larger Case of WISCAT-pabo, the MOGP surpassed the Greedy Heuristic by a large extent. The Greedy Heuristic performed worst instead of best. Probably, this is due the inefficient programming of the algorithm.

Analyzing methods include the deletion algorithm (Chinneck & Dravnieks, 1991), and the Set Covering with Item Sampling (SCIS) method (Huitzing, in press b), which search for one or more Irreducible Infeasible Subsets (IIS) of constraints. The deletion algorithm can be implemented in a deterministic way, the Relaxed and Ordered Deletion Algorithm (RODA) implemented in CPLEX (ILOG, 2001), or a stochastic way, the Integer and Random Deletion Algorithm (IRDA) implemented in NuzLight (Huitzing, 2002).

As expected, IRDA outperformed RODA, because the latter only looked at the relaxed model (i.e., the binary variables are relaxed into taking values between zero and one). This greatly frustrated the search for IISs. The SCIS is a completely different method, based on entirely different solving techniques. For the two cases at hand, the SCIS detected the IISs in the both Cases. To our disappointment the computer time needed to sample the tests was by far too high to be practical. Again, this is mostly due to the software that is not efficiently programmed.

One combined method was evaluated. The IIS-Solves first analyzes the problem, and then forces a solution. The results of the IIS-Solver can be compared with both forcing and analyzing methods. The IIS-Solver performed as well as, or even better than

the GP method when it comes to minimum number of violated constraints and minimum total violation. The resulting TIF value comes very close to the TIF value of the original models without infeasibility. However, the IIS-Solver needs much more time than the other forcing methods. When the IIS-Solver is compared with the analyzing methods it should be mentioned that the IIS-Solver builds on the results of IRDA. Since the IRDA was very successful in identifying IISs, this method also performed well. Because of this, the IIS-Solver is recommended. It is applicable both for analyzing the model and for forcing a solution, besides it performed very well in the comparison study.

However, these conclusions are drawn for two specific cases, and generalizations of these conclusions have to be made carefully. Also, all methods 'able to deal' with infeasibility do violate more or less the original bounds set by the test assembler, even if this is done in an automatic way embedded in the model, such as is the case for goal programming methods. However, not all methods are clear about how they violate constraints, other than that a weighted sum of the violations must be minimized. The problem then becomes how to set the weights of the constraints, which is a matter of trial-and-error.

The original objective of the test assembler, such as maximizing the test information function on a certain scale, can influence the number of violated constraint. Moreover, using an overall approach instead of a heuristic can have advantages in terms of a smaller number of violations. If, instead of forcing methods, an analyzing method was chosen, the source of one of the infeasibilities, namely an upper bound larger than a lower bound for a same set of items caused by a typing error, could by easily identified and adjusted.

All methods have drawbacks as well as advantages. Being faster, an important issue when a solution must be found in a real-time online examination, usually entails more unnecessary violations, and the more specific a method analyzes the infeasibility, the more computer time and interaction with the test assembler is needed. Which method is used depends thus on the goals of the test assembler or settings in which a test must be assembled.

References

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

Amaldi, E. (1994). *From finding maximum feasible subsystems of linear systems to feedforward neural network design.* Ph.D. thesis no. 1282, Département de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

Armstrong, R.D., Jones, D.H., & Wang, Z. (1995). Network optimization in constrained standardized test construction. *Application of Management Science, 8,* 189-212.

Boneh, A. (1984). Identification of redundancy by a set-covering equivalence. *Operational Research '84,* 407-422.

Chinneck, J.W. (1993). Infeasibility analysis using MINOS. *Computers and Operations Research, 21*, 1-9.

Chinneck, J.W. (1997). Feasibility and viability. In T. Gal & H.J. Greenberg (eds.), *Advances in sensitivity analysis and parametric programming* (pp. 14.1-14.41). Boston, MA: Kluwer Academic Publishers.

Chinneck, J.W. (2000). *Fast heuristics for the maximum feasible subsystem problem.* (Technical Report SCE-00-02). Ottawa, Canada: Carleton University, Systems and Computer Engineering.

Chinneck, J.W., & Dravnieks, E.W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing, 3*, 157-168.

Feng, J. (1999). *Nonlinear redundancy: where is the information.* Unpublished master thesis, University of Waterloo, Waterloo, Canada.

Greenberg, H.J. (1987). Intelligent analysis support for linear programs. *Operations Research Letters, 6*, 249-255.

Greenberg, H.J. (1992). Enhancements of ANALYZE: A computer-assisted analysis system for linear programming. *ACM Transactions on Mathematical Software, 19 (2),* 233-256.

Huitzing, H.A. (in press a). Solving infeasible linear programming test assembly models. *Journal of Educational Measurement.*

Huitzing H.A. (in press b). Using Set Covering with Item Sampling to analyze infeasibility of linear programming test assembly models. *Applied Psychological Measurement.*

Huitzing, H.A. (2002). NuzLight, Free academic computer software for LPTA models, using different solvers, Groningen: University of Groningen. http://www.oprit.rug.nl/huitzing/Research.html

ILOG, inc. (2001). *CPLEX 6.6* [Computer program and manual]. Incline Village, NV: ILOG.

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

Luecht, R.M., & Nungester, R.J. (1998). Some practical examples of Computer - Adaptive Sequential Testing. *Journal of Educational Measurement, 35,* 229-249.

Luecht, R.M., & Nungester, R.J. (2000). Computer-Adaptive Sequential Testing. In W.J. van der Linden & C.A.W. Glas (Eds.). *Computerized adaptive testing: Theory and practise* (pp. 117-128). Boston, MA: Kluwer Academic Publishers.

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

Nering, E.D., & Tucker, A.W. (1993). *Linear Programs and related problems*, Boston: Academic Press, cop.

Papadimitriou, CH.H., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity.* Englewood Cliffs, NJ: Prentice Hall.

Stocking, M.L., & Swanson, L. (1993). A method for severely constrained item selection in adaptive testing. *Applied Psychological Measurement, 17,* 277-292.

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

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

Timminga, E. (1998). Solving infeasibility problems in computerized test assembly. *Applied Psychological Measurement, 22*, 280-291.

Van der Linden, W.J. (2000). Constrained Adaptive Testing with shadow tests. In W.J. van der Linden and C.A.W. Glas (Eds.) *Computerized adaptive testing: Theory and practice* (pp. 27-53) Boston, MA: Kluwer Academic Publishers.

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

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

Van der Linden, W.J., & Reese, L.M. (1998). A model for optimal constrained adaptive testing. *Applied Psychological Measurement, 22,* 259-270.

Veldkamp, B.P. (1999). Mulit-Objective Test Assembly Problems. *Journal of Educational Measurement, 36*, 253-266.

Veldkamp, B.P. (2002). Multidimensional constrained test assembly. *Applied Psychological Measurement, 26,* 133-146.

Veldkamp, B.P., & Van der Linden, W.J. (2002). Multidimensional constrainted adaptive testing. *Psychometrika, 67,* 575-588.

Vos, H.J. (1999). Application of Bayesian decision theory to sequential Mastery Testing. *Journal of Educational and Behavioral Statistics, 24,* 271-292.

Wolsey, L.A. (1998). *Integer programming*. New York, NY: Wiley cop.