Report Nr 03

Various mathematical programming approaches toward item selection

J.G. Kester

Cito

VARIOUS MATHEMATICAL PROGRAMMING APPROACHES TOWARD ITEM SELECTION

by

J.G. Kester

Report Nr 03

Project 'Optimal Item Selection'

CITO

Arnhem, the Netherlands (1988)

General Introduction

The purpose of Project 'Optimal Item Selection' is to solve a number of issues in automated test design, making extensive use of optimization techniques. To this end, there has been close cooperation between the project and, among others, the department of Operations Research at Twente University. In each report, one or several theoretical issues are raised and an attempt is made to solve them. Furthermore, each report is accompanied by one or more computer programs, which are the implementations of the methods that have been investigated. The texts of these programs were included in the original thesis report, but will not be included in this version. In due time, requests for these programs can be sent to the project director.

T.J.J.M. Theunissen
project director.

Summary

This study concerns the item selection problem, which is a problem from test theory, where items are chosen in order to design a test that fulfils certain demands in the best possible way. This item selection problem can be formulated as a mathematical programming problem. The derivation of this is based on the so-called Rasch model.

In previous work on this subject a large area already has been covered. In this report the earlier results are examined in order to get a good picture of the explored and unexplored fields.

After that some new methods to solve the problem are described, all dealing with situations where cost minimization is the objective, whereas some methods also take care of a special category division for the selected items.

Finally the results of a number of experiments are reported, incorporating the most important algorithms. Some conclusions about the use of these methods are summarized.

Contents

Preface

The study about the item selection problem in this report, which serves as my master thesis Applied Mathematics at the University of Twente, is part of a CITO-project performed by the OPD-department. Its results will be incorporated in a larger project by several Dutch institutes.

I do not claim to give a complete coverage of the problem area. However I hope to have divided the item selection problem into clear-cut sub-areas, some of which I will treat extensively.

I would like to thank the staff members of OPD for making it possible for me to work on the item selection problem in good atmosphere, and especially P.Sanders, T.Theunissen and H.Verstralen for their advices. Furthermore I am indebted to S.Baas from the University of Twente who was my supervisor and who supported me with ideas and took care that I stayed on the right track.

14-th of April 1988,
CITO Arnhem,
J.G. Kester.

0 General Introduction

CITO in Arnhem is the National Institute for Educational Measurement. It has 320 staff members, of which 18 find their work at OPD, a Dutch abreviation for Research and Psychometric Services, the research department of CITO.

Test Service Systems

I worked on the CITO-project Optimal Item Selection, of which the results will be used in the long-term project Test Service Systems. The goal of this latter project, that is executed by several Dutch institutes, is to come to a system where a user, e.g. a teacher who wants to test his pupils, can get a test that fulfils his demands simply by giving some details to the computer and waiting a few minutes.

Optimal Item Selection

Such a system should be achieved with large item banks, i.e. collections of items for testing purposes, and with computer software that selects the best items for the test out of these banks: the Optimal Item Selection process.

So the problem that is faced is the following. Construct and implement a computer program that is able to design a test by selecting from an item bank those items that meet certain specified conditions in the best possible way and in a reasonable time.

In this report my contribution to the project will be described. First I will give some necessary information about the underlying test theory (section 1). Secondly I will derive a mathematical model for the optimal item selection problem (section 2). After that there will be the exact problem formulation that is the subject of this master thesis (section 3). Then I will discuss the previous work on this subject, done by my predecessors at CITO (section 4). Then it is time to report on my own work: some methods to solve the item selection problem when looking mainly at costs (section 5/7), followed by the results of various experiments with those methods (section 8), and by some remarks on the complexity of the algorithms (section 9). I will finish this report with a number of conclusions and recommendations for further study (section 10).

The listings of the written software and examples of an item bank and problem file can be found in the appendices.

1 Introduction test theory

In this section I will give an explanation about the underlying test theory. Most of it can be found in the first five sections of a book by F.M. Lord [11]. Here I will give the most important ideas and definitions.

Dichotomous items

Today, testing is a very common way to determine a person's ability. Tests are used e.g. at schools, job selections and military examinations. In this study a test is supposed to consist of a number of dichotomous items, i.e. items that can be answered in only two ways: right or wrong.

Item response function

Now if one wants to design a test that gives most information about someone's ability, one has to know the response behaviour of that person on the items of that test. Hence the item response function is defined as the probability that someone with ability θ answers the item correctly:

$$p(\theta) = c + (1-c)/(1+e^{-D*a(\theta-b)}) \qquad (1.1)$$

This three-parameter logistic function was introduced by Birnbaum. In general θ is assumed to have a value between -3 and 3. In this function e is the mathematical constant 2.7182818.. and D is a positive constant, which will be taken equal to one here.
The parameters a,b and c have the following meaning.

Discriminating power

Parameter a represents the discriminating power of the item, i.e. the degree in which item response varies with ability. More specific: a is the slope of the curve at θ=b.

Difficulty parameter

Parameter b is the difficulty parameter of the item. It has the same scale as the ability θ and it determines the "relative position" of the curve on the X-axis: as b increases, i.e. the more difficult the item becomes, the further the curves moves to the right.

Guessing parameter

Parameter c is the guessing parameter of the item, i.e. the probability that someone with an absolute lack of ability ($\theta \rightarrow -\infty$) answers the item correctly. One can think of a multiple choice question with five possible answers, where c=0.2.

The meaning of these parameters for the item response function is illustrated in figure 1.1.

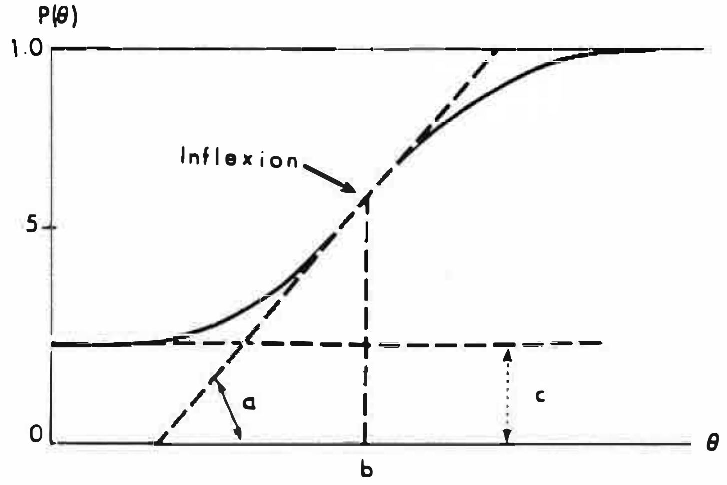

figure 1.1 An item response function

Rasch model

After substituting a=1 and c=0 in (1.1) the Rasch model arises, which will be used throughout this report:

$$p(\theta) = 1/(1+e^{-(\theta-b)}) \qquad (1.2)$$

Now suppose one has a test with n items. For every ability θ one can compute a 95%-confidence interval for the expected score, i.e. the expected number of correct answers, a person with ability θ will achieve. When this is done for several values of θ between -3 and 3 one can obtain a 95%-confidence belt like in figure 1.2. Here the test contains 80 items.

When the test results are known and for every person with known ability the score is marked in figure 1.2, at most 2½% of the persons should be above the belt and at most 2½% below it.

However one wants to say something about a person's ability based on his score. Suppose someone has a score of x0, then a 95%-confidence interval for his ability is given by (θ_1,θ_2).

Information function

According to Birnbaum the information function $I(\theta,x)$ for a score x is inversely proportional to the square of the length of the asymptotic confidence interval for estimating the ability θ from the score x. Here asymptotic means that the number of items n goes to infinity.

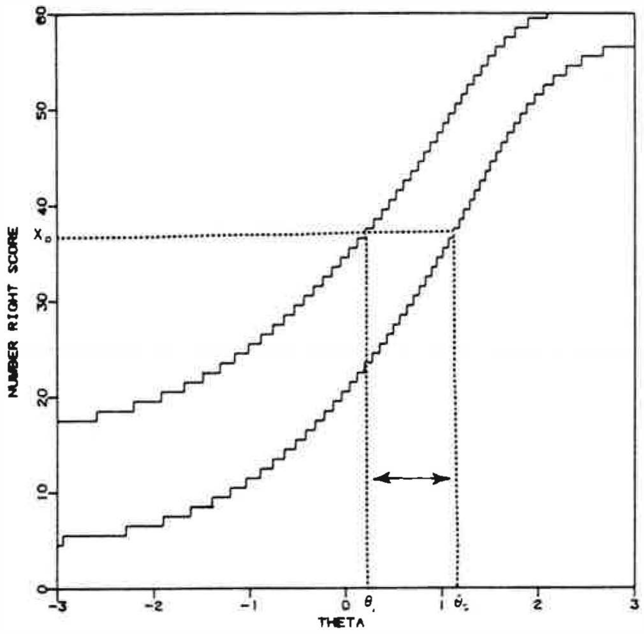

figure 1.2 Determination of a 95%-confidence interval for θ

Item information function

Now one can derive the item information function,

$$I(\theta,u_i) = (P_i')^2/(P_i(1-P_i)) \qquad (1.3)$$

where $u_i=1$ if item i is answered correctly and $u_i=0$ otherwise.

Test information function

An upperbound to the information that can be obtained from a test is given by the test information function

$$I(\theta) = \sum_{i=1}^{n}(P_i')^2/(P_i(1-P_i)) \qquad (1.4)$$

It is clear that the test information function is simply the sum of the individual item information functions. This feature is illustrated in figure 1.3.

After substituting the Rasch function (1.2) into (1.3) one finds:

$$I(\theta,u_i) = e^{-(\theta-b_i)}/(1+e^{-(\theta-b_i)})^2 =$$

$$= 1/(e^{-(\theta-b_i)}+2+e^{(\theta-b_i)}) \qquad (1.5)$$

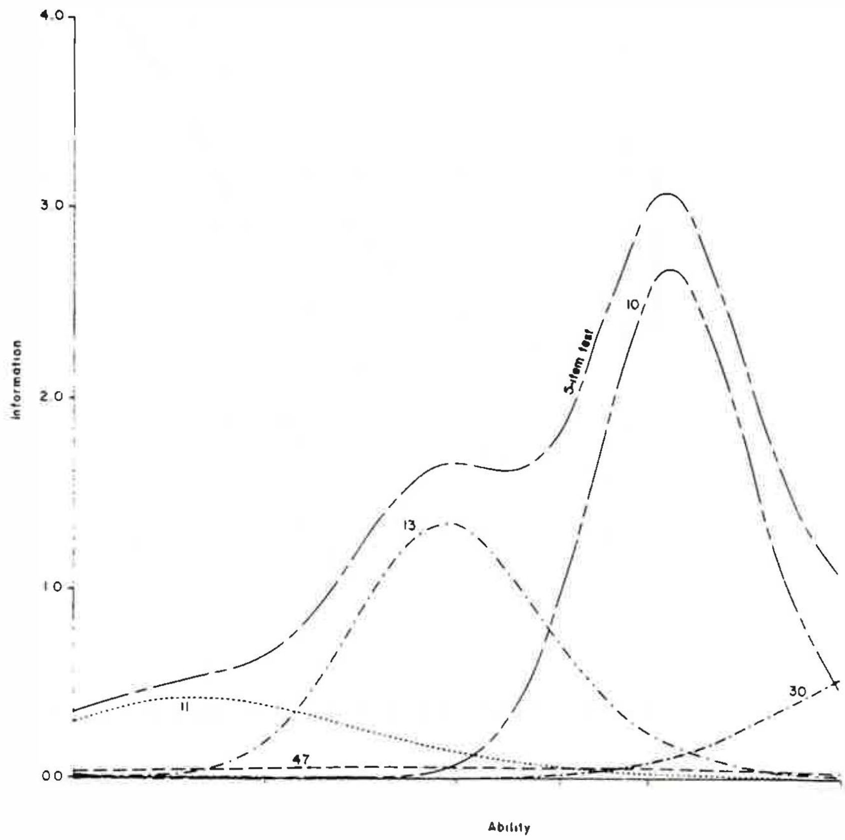

figure 1.3 A test information function composed of five item information functions

The item information function has its maximum value for $\theta=b_i$: $I(b_i,u_i)=0.25$. One can easily see this by thinking of an intelligent girl Mary with ability approximately equal to 2. Now it makes no sense to give Mary an easy item with difficulty -2, because she will probably answer this item correctly, i.e. when this item is answered one can not say much more about Mary's ability. So this easy item provides very little information. However if Mary is given an item with difficulty 2, then the chances of a right and wrong answer are approximately equal: this item gives much information about Mary's ability. Of course more than just one item is required to get some more definite information about Mary's ability, but items of difficulty 2 contribute much more to this than items of difficulty -2.

Target information function

The last function that is to be defined is the target information function. This function gives at every ability level the amount of information that is desired for a test. For instance if one wants to design a test that will separate the good half of a class from the bad half, one should gain much information at $\theta=0$, as in figure 1.4.

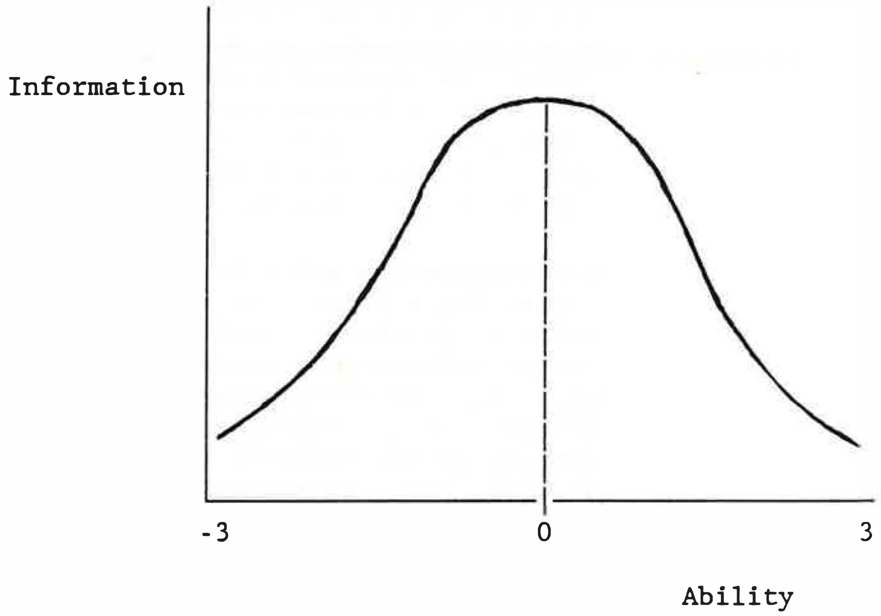

figure 1.4 Example of a target information function

This was in short the underlying test theory. For more details or derivations I refer the reader to Lord [11]. This abstract should suffice to understand the rest of this report.

2 Mathematical modeling

In this section I am going to derive a mathematical model for the item selection problem, making use of the theory from the previous section.

Suppose a desired test is specified by a target information function so that at every ability level θ it is known how much information is required. For practical reasons the target information function will be specified at a finite number of ability points: Verstralen [16] has proved that a specification at three to five different points is sufficient to fix information functions. In this study I will look at target information functions specified only at the points {-3,-2,-1,0,1,2,3} or at a subset of these points.

Now suppose one wants to design a test from an item bank containing n items. Let x_j be defined by $x_j=1$ if item j is selected in the test and $x_j=0$ otherwise. Further let the target information function be specified at m points $\theta_1,..,\theta_m$: the information points, and the required information at those points is $I(\theta_i)$ i=1..m. Finally let $I(\theta_i,j)$ be the value of the information function of item j at ability θ_i (=information point i). Now the problem is to find a test for which the test information function exceeds the target information function at the specified points. Since a test information function is the sum of the individual item information functions, this condition can be translated into the following inequalities:

$$\begin{array}{l} I(\theta_1,1)*x_1 + I(\theta_1,2)*x_2 + \ldots\ I(\theta_1,n)*x_n >= I(\theta_1) \\ \ldots \\ I(\theta_m,1)*x_1 + I(\theta_m,2)*x_2 + \ldots\ I(\theta_m,n)*x_n >= I(\theta_m) \end{array} \qquad (2.1)$$

$$x_j \in \{0,1\} \quad j=1..n$$

Positive costs

One can assign a positive cost c_j to every item j expressing the eagerness to have item j in the test. A high cost for instance can be given to an item that has been selected recently in another test, and a low cost can be given to an item of which the parameters are not so well-known, i.e. a bad-calibrated item: it should be included in some more tests in order to become more sure about its parameters.

Stochastic approach

From this last remark it follows that the values $I(\theta_i,j)$ are not known with 100% certainty. However in this report I will assume otherwise, since the deterministic constraints (2.1) become considerably more complicated when they are made stochastic. Moreover, a stochastic approach will probably not be relevant, also for practical purposes.

Mathematical programming problem

With the costs c_j and the definitions $a_{ij}=I(\theta_i,j)$ and $b_i=I(\theta_i)$ for $i=1..m$ and $j=1..n$ this results in the following mathematical programming problem with cost minimization.

$$
\begin{aligned}
\text{(P)} \quad & \min \sum_{j=1}^{n} c_j * x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} * x_j >= b_i \qquad i=1..m \qquad (2.2) \\
& x_j \in \{0,1\} \qquad j=1..n
\end{aligned}
$$

Note that with $c_j=1$ for $j=1..n$ the number of items is minimized.

Multi-dimensional knapsack problem

In literature problem (P) is known as the multi-dimensional 0-1 knapsack problem. This is a NP-hard problem, so when the number of items becomes large it becomes very difficult to find the optimal solution for this problem. Therefore there is a great need for algorithms that can give optimal or near-optimal solutions in a reasonable time. The search for those algorithms was started by other persons already and I will follow in their footsteps, as outlined in the next section.

As will be seen later, the item selection problem is not fully covered by this mathematical model. There are certain demands that can not be described by constraints as in (P), but for a lot of applications the model is sufficient.

3 Problem formulation

In this section I will give the formal problem description, which will be used as the starting point for this thesis.

Boomsma [4], Gademann [5] and Razoux Schultz [12] have investigated and elaborated several algorithms for test construction from a large item bank. Among those various methods one can distinguish two mainstreams: an exact approach, in which the optimal solution is pursued, and a heuristic approach: problem-specific algorithms that yield near-optimal solutions. In the studies of Gademann and Razoux Schultz more emphasis was put upon extra demands which have to be imposed on the tests in view of practical usibility.

This all leads to the following problem formulation. Making use of the already obtained insights and previously developed methods one has to investigate whether there is at least one best method. Such a method should unify known principles and possible new concepts into practically useful algorithms.

Hereby one should take the following points into account.
(a) Next to the already introduced constraints there are also extra demands concerning the partitioning of items into certain categories.
(b) Because of the size of the problem one should make use of the special structure as much as possible.
(c) Future users must be accounted for with every algorithm considered.

4 Previous work

In this section I am going to describe previous work on the subject. Boomsma [4], Gademann [5] and Razoux Schultz [12] all had their own way of approaching the item selection problem. As already stated, one can make a distinction between exact and heuristic methods. Now here I will explain the main principles of both approaches and evaluate some of the algorithms on their practical usibility.

4.1 The exact approach

The exact approach of the item selection problem consists of those methods where one tries to obtain an exact solution for problem (P), i.e. the real optimum. This was done by Boomsma [4] and Gademann [5] in two manners. First there is a branch and bound method by Balas and secondly there is a continuous approach, in which the (0,1)-constraints are relaxated and the resulting problem is solved with the Simplex method. Although these Simplex-orientated methods do not always lead to the optimum, I will nevertheless consider them in this section because of the exact way in which the problem is approached. They are called quasi-exact methods and are studied here with a linear and a quadratic objective function.

4.1.1 Balas' algorithm

Boomsma [4] describes a branch and bound algorithm as designed by Balas. It can be found in Syslo, Deo and Kowalik [13]. Test results showed that the optimum to (P) is found, but also that computation time gets huge when the number of items n becomes greater than 100. Since the item selection is supposed to take place at item banks with 300 to 1000 items, I propose to let this method rest until the computing capacity is sufficient for coping with the extensive calculations needed for the Balas' algorithm.

4.1.2 Continuous approach with linear objective function

The idea behind the continuous approach is the following. Replace in problem (P) the (0,1)-constraints by upper- and lowerbounds on the variables x_j. The result is a linear relaxation (RP) of problem (P).

Linear relaxation

$$\text{(RP)} \quad \min \sum_{j=1}^{n} c_j * x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} * x_j >= b_i \qquad i=1..m \qquad (4.1)$$

$$x_j >= 0 \qquad j=1..n$$

$$x_j <= 1 \qquad j=1..n$$

Problem (RP) can be solved exactly with the Simplex method. Suppose this gives a solution with objective function value z(RP). This solution will probably not be a feasible solution for (P), since some variables x_j may have a non-integer value. It can be proved however that an optimal solution for (RP) contains at most m variables with a non-integer value. Since m is the number of information constraints there will never be more than seven non-integer variables.

Integer solution

A feasible solution for (P) can be derived from the (RP)-solution by rounding every variable x_j with $0<x_j<1$ off to one. One can easily see that this indeed gives a feasible solution for (P) and because of the relatively small number of non-integer variables this is a near-optimal solution. Let z(P) be the objective function value of the optimal solution for (P) and z(RRP) the objective function value corresponding with the rounded-off solution, then it follows that:

$$z(RP) <= z(P) <= z(RRP) \qquad (4.2)$$

In other words z(RP) is a lowerbound on the unknown optimal objective function value z(P) and gives an indication of how good, i.e. how near-optimal, the rounded-off objective function value z(RRP) actually is.

Land and Doigh algorithm

This relaxation idea was used by Boomsma [4] and proceeded by Gademann [5]. For the Simplex part of the job they used the Land and Doigh algorithm (LANDO) with a few minor adjustments to make it appropriate for this type of problem.

Multiple objective functions

Gademann extended Boomsma's program by making it work with multiple objective functions. Suppose there are linear objective functions $f_k(x) = \Sigma_j \, c_{jk}*x_j$ for k=1..p. Now the linear multiple objective function F is given by $F(x) = \Sigma_k \, \mu_k*f_k(x) = \Sigma_k \, \Sigma_j \, \mu_k*c_{jk}*x_j$.

It is obvious that this multiple objective function is not essentially different from the single objective function of (P), but this approach makes it easier to give priorities to certain goals by an appropriate choice of the weight-vector μ.

The main disadvantage of this approach can also be found in the use of the weights. A lot of experience with the algorithm is required to be able to assign values to the weight-parameters μ_k in a fast satisfactory way.

The flowchart of the multi-objective algorithm by Gademann can be found in figure 4.1.

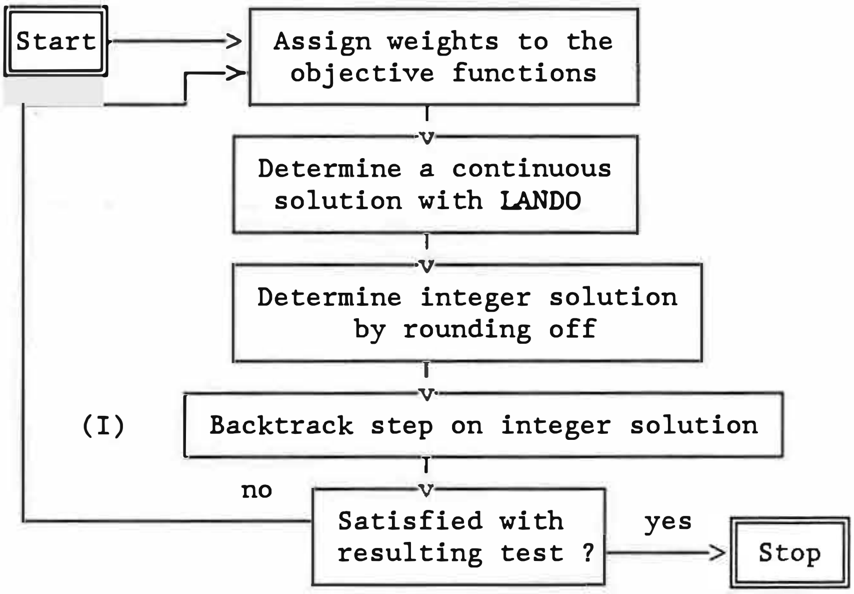

Figure 4.1 Flowchart of the multi-objective algorithm

Ad (I) The backtrack step means that the integer solution is checked on the presence of a redundant item, i.e. an item that can be omitted (corresponding variable is set to 0) without violating the information constraints.

In section 8 there will be some test results for this algorithm.

4.1.3 The multiple quadratic approach

Gademann [5] also pays attention to problems with linear constraints and multiple quadratic objective functions. These problems have the following form.

Quadratic objective functions

$$(QP) \quad \min \sum_{k=1}^{p} \mu_k * f_k(x)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} * x_j >= b_i \qquad i=1..m \qquad (4.3)$$

$$x_j \in \{0,1\} \qquad j=1..n$$

Here $f_k(x) = \Sigma_l \Sigma_j d_{klj} * x_l * x_j + \Sigma_j c_{kj} * x_j$ is a quadratic objective function for k=1..p. The weights μ_k k=1..p have the same interpretation as in the linear case.

In case the multiple objective function is strictly convex, Gademann shows that the algorithm of Wolfe can optimally solve the relaxed version of this problem. The general idea behind this algorithm is that an optimal solution has to satisfy a number of conditions: the Kuhn-Tucker conditions. Now finding a solution vector x that satisfies these conditions becomes a problem with linear constraints and a linear objective function. This can be solved with the Simplex method. For more details I refer to Gademann [5].

However there are a few practical problems. First the resulting linear programming problem has 4n+2m variables and n+m restrictions. Since n will be quite large it can last very long before Simplex has solved this problem, even on a big mainframe computer. Secondly one can place questionmarks at the rounding-off procedure. There will no longer be at most m variables resulting from Simplex with non-integer value. In some cases all m variables can have a non-integer value. Now rounding off to an integer solution will probably lead to a big gap between the optimal solution for (QP) and the integer solution found.

Test results in [5] have shown that for small problems (20 items) the Wolfe algorithm can give good solutions for (QP). For larger problems it can not be used in practice yet, because of the already mentioned size of the resulting linear programming problem. Still this is an important method, since quadratic objective functions enable the processing of logical restrictions.

Logical restrictions

A logical restriction is a condition on a test of the form: if item 1 is selected, then item 2 should not be selected and vice versa. This restriction can be dealt with in a quadratic objective function like in (4.3), by setting $d_{k12}=d_{k21}=H$, where H is a large positive constant, for k=1..p.

Until there is no sound way to settle with logical restrictions, this quadratic objective function approach can not be ignored. However it still needs a lot of care and dedication to become of practical use for the item selection problem.

4.2 The heuristic approach

Among the heuristic approaches, I consider those algorithms that give a good feasible, though not necessarily optimal, solution for (P) in relatively short time. The main feature of the heuristic algorithms is the relative weight of the exactness of the solution against the computation time. One tries to get a near-optimal solution for (P), but an increasing accuracy imposed on the item selection process will lead to increasing computation times.

Heuristic algorithms for the item selection problem were developed by Boomsma [4] and Razoux Schultz [12]. I will consider the Surrogate method by Boomsma and the algorithms Mindev and Twoitems by Razoux Schultz.

4.2.1 The Surrogate method

In the Surrogate method problem (P) is reduced to the surrogate problem (SP).

Surrogate problem

$$\text{(SP)} \quad \min \sum_{j=1}^{n} c_j * x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \mu_i * \left(\sum_{j=1}^{n} a_{ij} * x_j - b_i \right) >= 0 \qquad (4.4)$$

$$x_j \in \{0,1\} \qquad j=1..n$$

So the multiplier vector μ reduces the different original information constraints from problem (P) to one surrogate constraint. Now of course the problem is how to choose a multiplier vector μ, so that when the surrogate constraint is satisfied this is also true for the original information constraints. Boomsma shows that such a surrogate constraint is obtained by taking for μ the vector of the dual variables corresponding with the optimal solution for (RP).

Since at that moment there was no fast method available to obtain these dual variables, Boomsma makes use of an iterative method by Gavish and Pirkul [6] to find good multipliers.

The flowchart of the resulting heuristic, that finds a good solution for (P), can be found in figure 4.2. It should be noted however that Boomsma studied problem (P) with all $c_j=1$, i.e. he was only minimizing the number of items. This makes problem (SP) very easy to solve: continue to select that item j which maximizes $\Sigma_i \mu_i * a_{ij}$ until the selected coefficients add up to $\Sigma_i \mu_i * b_i$.

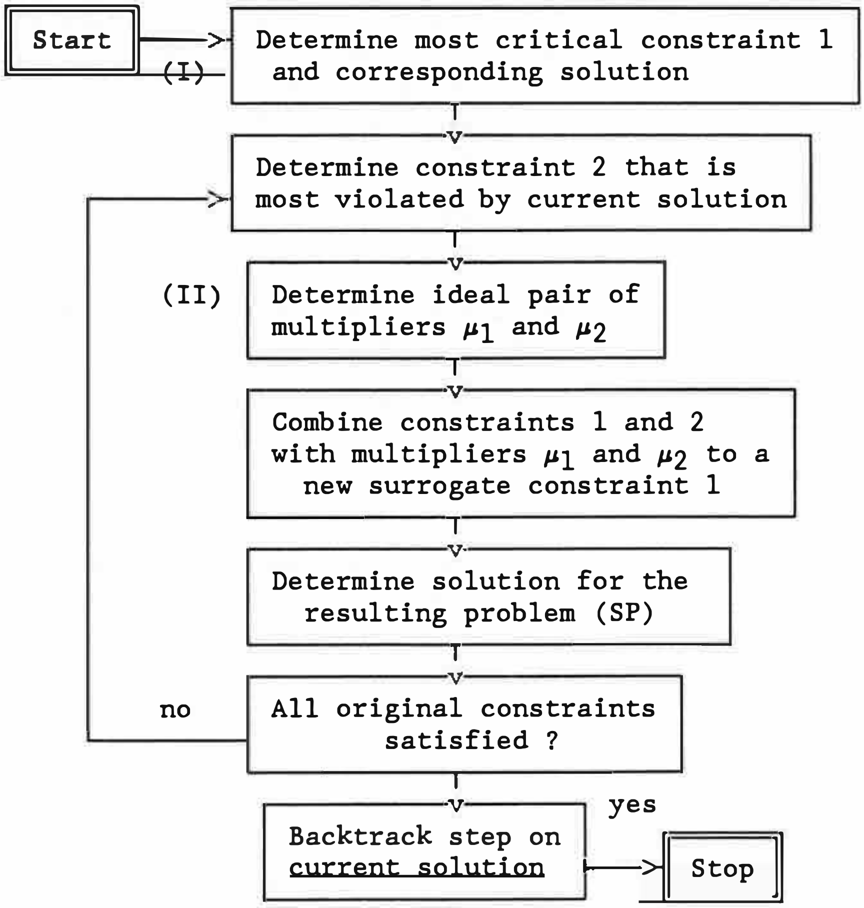

Figure 4.2 Flowchart of the Surrogate method

Ad (I)

The most critical constraint is determined as follows. Solve the problems (P(i)) for i=1..m. These problems are:

$$(P(i)) \quad \min \sum_{j=1}^{n} c_j * x_j$$
$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} * x_j >= b_i \qquad (4.5)$$
$$x_j \in \{0,1\} \quad j=1..n$$

Let z(i) be the optimal objective function value of problem (P(i)). If $z(i^*) = \max z(i)$ then i^* is the most critical constraint, and $z(i^*)$ the objective function value of the corresponding solution.

Ad (II) The ideal pair of multipliers is determined iteratively by searching for a $\mu>0$ so that the solution $S(1,\mu)$ to

$$\min \sum_{j=1}^{n} c_j * x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{n} (a_{1j}+\mu * a_{2j}) * x_j >= b_1+\mu * b_2 \qquad (4.6)$$

$$x_j \in \{0,1\} \qquad j=1..n$$

satisfies both constraint 1 and 2. Note that here $\mu_1=1$ and $\mu_2=\mu$.

After some testing Boomsma made an adjustment, in which the loop in the flowchart of figure 4.2 is traversed only once. If the constraints are not satisfied a filling-up procedure is called that selects extra items until all information constraints are satisfied.

In [4] Boomsma gets good test results with this Surrogate method. However the more recent study by Razoux Schultz [12] shows better heuristics which make this method a bit outdated. Moreover for practical use there should be the possibility to work with costs. This can be arranged in this method, but then it becomes a lot more difficult to find optimal solutions for the (0,1)-knapsack problems (4.5) and (4.6), which would lead to the use of less accurate heuristics.

4.2.2 The clusterpoint method

The clusterpoint method was developed by Razoux Schultz [12]. It is a heuristic for problem (P) with $c_j=1$ for $j=1..n$. Razoux Schultz discovered that in a solution for an item selection problem the selected items have difficulty parameters that are always close to one or two values: the clusterpoints. He made use of this feature by constructing an algorithm that first determines two near-optimal clusterpoints, i.e. clusterpoints close to the best possible clusterpoints, and then selects items with difficulty parameter close to those points until all constraints are satisfied.

The algorithm was implemented in the computer program Twoitems of which the flowchart can be found in figure 4.3.

Ad (I) The clusterpoints b1 and b2 are optimal if the function

$$G(b1,b2) = \min_{i=1..m} \{ [f(\theta_i-b1) + f(\theta_i-b2)] / I(\theta_i) \}$$

is maximal. Here θ_i is the ability level corresponding with information point i and $I(\theta_i)$ the target information value at information point i, i=1..m. The function f is the item information function (1.5).

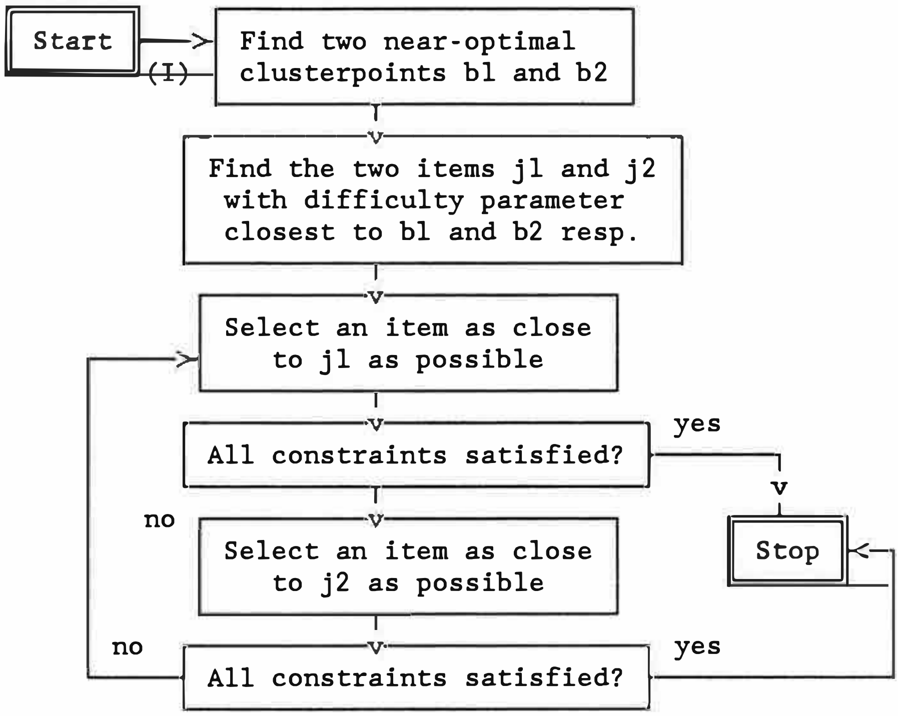

Figure 4.3 Flowchart of the clusterpoint algorithm Twoitems

In words this search for optimal clusterpoints proceeds as follows. The information point where the total information from two items with difficulty parameters b1 and b2 divided by its target information value is minimal is called the critical information point. Now one searches for the pair (b1,b2) that maximizes this relative information at the corresponding critical information point.

Razoux Schultz showes that this searching can be done in a fast way by a stepwise approximation, making use of some special properties of the item selection problem. For further details I wish to refer to his report [12].

Note that in order to get the best results with this algorithm, the items should be ordered within the bank according to increasing difficulty. This is no essential restriction, so for reasons of convenience I will asume all item banks in this report to be ordered that way.

The test results in [12] for Twoitems are very good. In a few seconds this method gives a very sharp upperbound to the minimum number of items that is required to satisfy the information constraints. Because this algorithm works exclusively with costs $c_j=1$ for $j=1..n$, there is no direct practical use for it, but it can very well be used as part of a more general algorithm to get a proper idea about the number of items involved in the selection process.

4.2.3 The minimum deviation method

The minimum deviation method is a heuristic by Razoux Schultz [12] for the positive cost problem (P). It is an extension of an algorithm by Boomsma [4], which was only suited for the cases with $c_j=1$ for $j=1..n$. It was implemented in the computer program Mindev and the working of it can best be understood by inspecting the flowchart in figure 4.4.

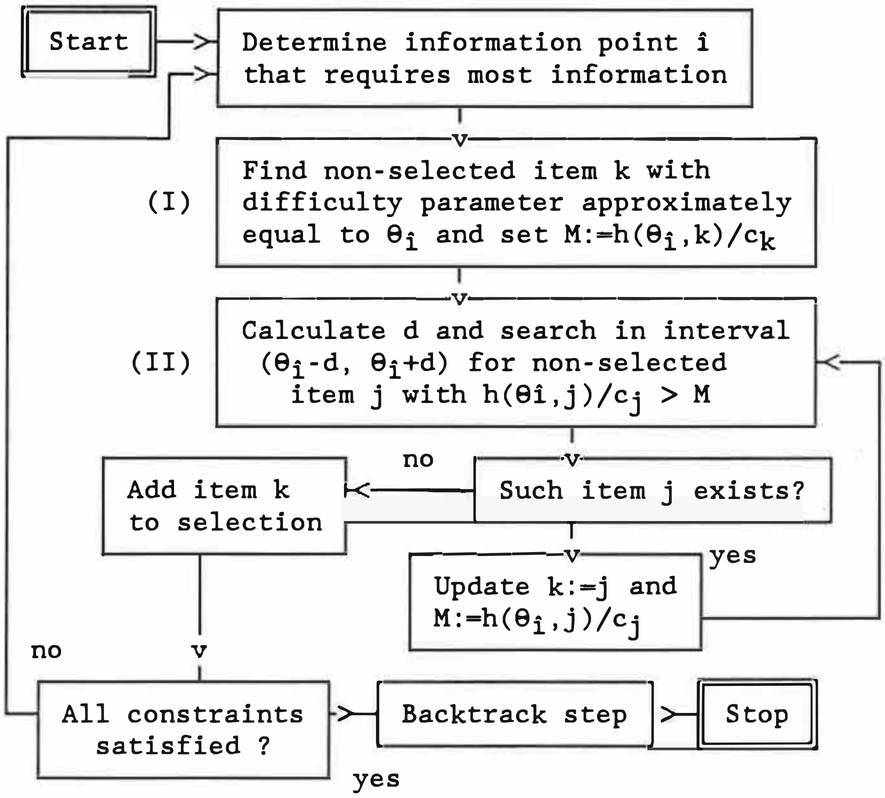

Figure 4.4 Flowchart of the minimum deviation method

Ad (I) Define c_{min} = min c_j and $h(\theta_i,j)$ the value of the information function of item j at ability level θ_i (=information point i) for i=1..m and j=1..n. In this algorithm one searches for the item j which maximizes $h(\theta_i,j)/c_j$. Since $M=h(\theta_i,k)/c_k$ for a non-selected item k, it follows that $h(\theta_i,j)/c_j >= M \rightarrow h(\theta_i,j) >= M*c_{min}$. Now the ordering of the item bank can be used.

Ad (II) By means of the inverse of the information function (1.5) a search interval for the optimal item j can be determined. Let $d=\ln[a+\sqrt{(a^2+1)}]$ with $a=\frac{1}{2}(M*c_{min})$, then the search interval is given by (θ_i-d, θ_i+d). Only items with difficulty parameter in this interval can produce the optimal item j. When during the search M gets larger, the interval gets smaller and so this process will yield the optimal item j.

For a more detailed explanation I again refer to Razoux Schultz' report [12].

Mindev gives good test results. It is very fast and the obtained solution forms a pretty sharp upperbound to the optimal solution for (P). Though usually it does not produce the optimal solution, this method is very appropriate for a first quick estimate of the costs. In section 8 Mindev will be tested elaborately in order to compare it with more accurate but slower algorithms.

5 Working with categories

Sometimes a test has to be designed with items concerning different subjects. For instance a Physics test should cover the topics Electricity, Magnetism and Nuclear Physics. Moreover these topics should be in the test at certain proportions, for instance half of the test should deal with Electricity, a third should cover Magnetism and the rest should be about Nuclear Physics.

In [5] Gademann gives a method to deal with such category division. He formulates a quadratic objective function:

$$\min f(x) = \min \sum_{k=1}^{s} \left(DF_k * \sum_{j=1}^{n} x_j - \sum_{j \in S_k} x_j \right)^2 \qquad (5.1)$$

Here DF_k = desired fraction of selected items referring to subject k, for k=1..s; S_k = set of items in the item bank referring to subject k, for k=1..s. So with this objective function the sum of the quadratic deviations between the desired and actually determined numbers of items for each subject is minimized.

Since the quadratic multiple objective approach in [5] is not yet suitable for practical use and the test results are not that good, I had to search for alternatives. Here I will discuss three different approaches. Two of them are based on the algorithm Mindev and one works with the Simplex method.

5.1 The Simplex approach

The Simplex method can be used for working with categories by adding extra constraints to the problem. When defining DF_k and S_k as above, appropriate constraints could be:

$$DF_k * \sum_{j=1}^{n} x_j - 1 <= \sum_{j \in S_k} x_j <= DF_k * \sum_{j=1}^{n} x_j + 1 \qquad k=1..s$$

This would lead to the following set of constraints:

$$\sum_{j \in S_k} x_j - DF_k * \sum_{j=1}^{n} x_j >= -1 \qquad k=1..s$$

$$DF_k * \sum_{j=1}^{n} x_j - \sum_{j \in S_k} x_j >= -1 \qquad k=1..s \qquad (5.2)$$

Although this seems to be a solid way to solve the category division, there are a few disadvantages. First when a solution obtained with the Simplex method is rounded off to an integer solution by setting all x_j with $0<x_j<1$ equal to one, it may occur that one or more of the constraints out of (5.2) are violated. Now this is not the end of the world, there could still result a good category division, but it may not be as sharp as aimed for.

The second disadvantage concerns the exactness of the integer solution. In section 4.1.2 was shown that when the number of constraints increases, also the number of non-integer variables in the solution of the linear relaxation of (P) increases. After rounding off, the integer solution tends to be less exact. So the more category constraints of the type (5.2) are added, the less exact the final integer solution becomes.

Those two disadvantages can be taken care of partly by using instead of (5.2) the constraints:

$$\sum_{j \in S_k} x_j >= DF_k * N \qquad k=1..s \qquad (5.3)$$

Here N is the minimal number of items required to satisfy the information constraints, so the objective function value to (P) with $c_j=1$ for j=1..n. Now rounding off will not give problems, and the number of extra constraints has been halved. N needs to be known in advance, which could be accomplished by using the fast accurate heuristic Twoitems from section 4.2.2.

Since adding extra constraints to the Simplex method will lead to both higher computation times and less exact integer solutions I decided to give priority to finding good and fast heuristics to solve this category division problem. Therefore I did not implement the above approach in an algorithm. However this does not mean that this approach is not appropriate for implementation. When the heuristics I will descibe in the next sections, would not satisfy in future practice, this idea can still be used in an effort to produce a good algorithm.

5.2 The Split-up approach

This method is based on the principle of dividing problem (P) into s subproblems, where s is the number of categories one is working with. These subproblems (P_k) have the following form:

Subproblem (P_k)

$$(P_k) \qquad \min \sum_{j \in S_k} c_j * x_j$$

s.t.

$$\sum_{j \in S_k} a_{ij} * x_j >= DF_k * b_i \qquad i=1..m \qquad (5.4)$$

$$x_j \in \{0,1\} \qquad j \in S_k$$

Note that the subsets S_k form a partition of the item collection {1..n} and that $\Sigma_k DF_k * b_i = b_i$. Now all items selected on the basis of these subproblems put together form a feasible solution for the original problem (P), and furthermore the proportions between the categories will be approximately equal to the desired ratios.

In general the resulting objective function value will be greater than z(P), because of the compensation effects that can arise at the selection process of problem (P), i.e. an information shortage for one category can be compensated by an information overspill for another category. This compensation effect can partly be accomplished, using this approach, by performing a backtrack step on the resulting solution. However this will make the resulting proportions less desirable, so one can question whether a cost reduction is worth the inaccuracy.

I used the algorithm Mindev from section 4.2.3 to solve subproblems (P_k). I chose this algorithm because of its speed and ease of applicability in this situation. It is not the algorithm that gives lowest costs, but here the main interest is a good category division. The flowchart of the algorithm can be found in figure 5.1.

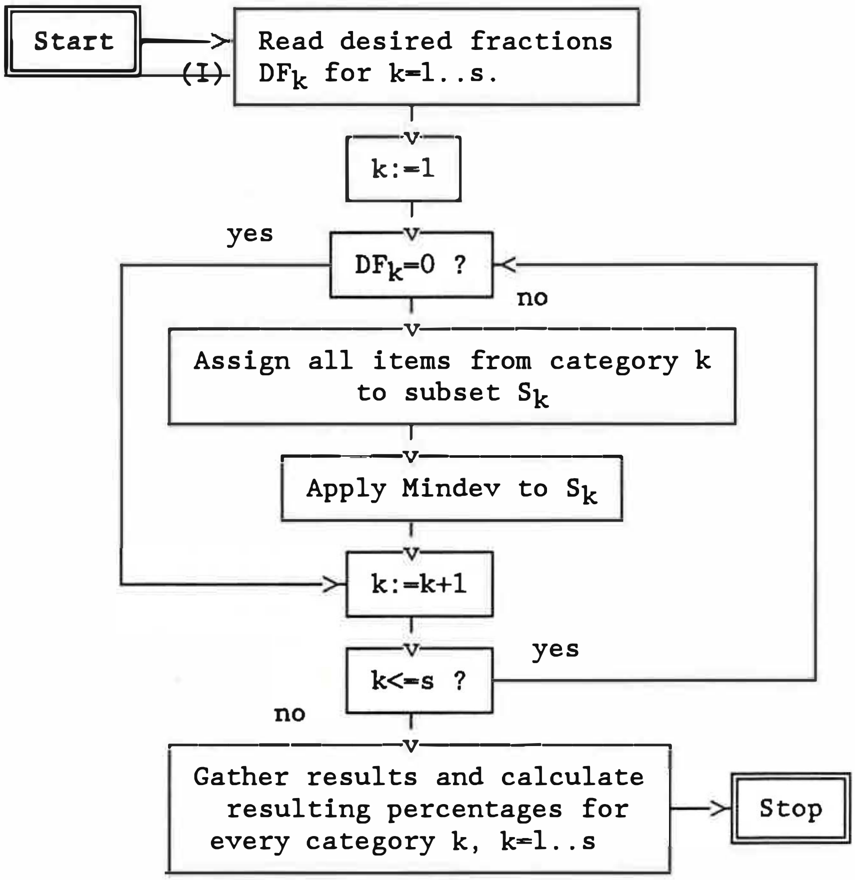

Figure 5.1 Flowchart of the algorithm Split-up

Ad (I)

While reading the desired percentages DF_k, k=1..s, there is a check whether this input is correct, i.e. whether $\Sigma_k DF_k = 100$. A second problem may arise when percentages are given that lead to infeasible solutions, simply because there are not enough items of some category in the bank. This problem is related to the problem of specifying the target information values too high and will be discovered by the algorithm. In those cases a user just has to start from scratch again, that means specifying new target information values or percentages. This experience will probably make him richer.

The above method was tested extensively and some of the results can be found in section 8, where a comparison is made with other algorithms.

5.3 The Direct approach

With this approach problem (P) is not divided into subproblems, but solved as a whole by the algorithm Mindev. In order to get a nice category division there needs to be an adaptation in Mindev.

Define s(j) = the category of item j, j=1..n; N(k) = the number of items from category k selected so far, k=1..n; N = the total number of items selected so far. Now in the steps (I) and (II) of the Mindev flowchart in figure 4.4 an item j needs to be not only non-selected, but also has to meet the following requirement:

$$N(s(j)) <= \text{round}(DF_{s(j)}*N) \qquad (5.5)$$

Here "round" represents the rounding-off function. This function is not absolutely necessary, but it weakens the demand somewhat, which will probably lead to more cost-friendly solutions than with the Split-up approach. Since Mindev contains a backtrack step the proportions found will be less close to optimal than those of Split-up. The test results in section 8 will shine more light on both methods.

6 The Top5 algorithm

This algorithm is a heuristic method based on an article by David J. Gonsalvez e.a. [8]. In this article an algorithm to solve the multicovering problem with heuristics is presented, together with a way to construct a confidence interval for the optimal solution for this problem. The multicovering problem has the following form.

Multicovering problem

$$\text{(MCP)} \quad \min \sum_{j=1}^{n} c_j * x_j$$
$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} * x_j >= b_i \qquad i=1..m \qquad (6.1)$$
$$x_j \in \{0,1\} \qquad j=1..n$$

The difference with problem (P) is that here $a_{ij} \in \{0,1\}$ for i=1..m and j=1..n. However article [8] proved to be applicable to the item selection problem too, after a few minor changes. Later on I will deal with the confidence intervals, but first I want to focus on the algorithm in its adapted form.

6.1 The algorithm

The algorithm consists of two parts. In the first part problem (P) is solved several times with different item selection criteria. The best five solutions are picked out and the corresponding criteria are put in a special set: the top5. In the second part of the algorithm problem (P) is solved again several times, but now at every iteration (= selection of one item) a selection criterion is randomly chosen out of the top5. The best solution resulting from parts 1 and 2 will be the final solution. The solution of problem (P) goes according to the flowchart in figure 6.1.

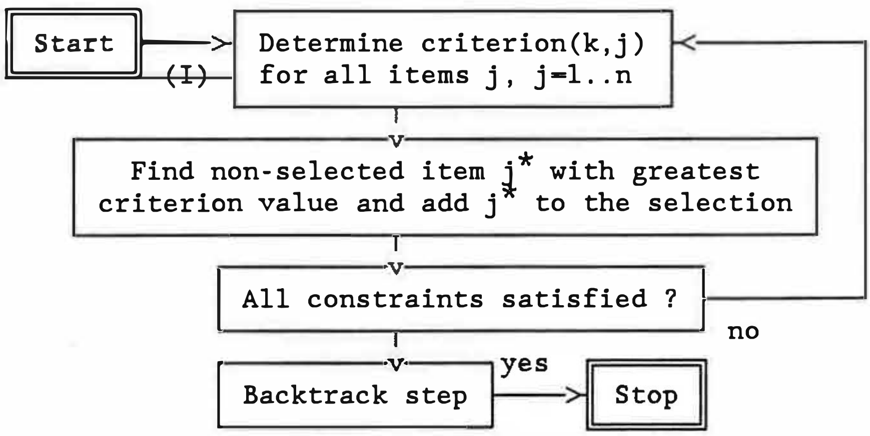

Figure 6.1 Flowchart of solution procedure in Top5 algorithm

Ad (I)

In [8] ten possible criteria are given. Since this article is based on the multicovering problem, I had to investigate whether those criteria had any relevance for the item selection problem and whether there could be established any other suitable criteria, not mentioned in [8]. This lead to the following criteria.

criterion 1: $1/c_j * \sum_{i=1}^{m} a_{ij}$

criterion 2: $1/c_j * \sum_{i=1}^{m} [(b(i)^2 * a_{ij})/rsum(i)]$

criterion 3: $1/c_j * \sum_{i=1}^{m} [a_{ij}/space(i)]$

criterion 4: $1/c_j * \sum_{i=1}^{m} b(i)^2 * a_{ij}$

criterion 5: $1/c_j * \sum_{i=1}^{m} [(b(i)^2 * a_{ij})/space(i)]$

criterion 6: $1/c_j * \sum_{i=1}^{m} [a_{ij}/rsum(i)]$

criterion 7: $1/c_j * \ln[\sum_{i=1}^{m} 1/(\sum_{i=1}^{m} 1 - \sum_{i=1}^{m} a_{ij})]$

criterion 8: $1/c_j * \ln[\sum_{i=1}^{m} b(i)/(\sum_{i=1}^{m} b(i) - \sum_{i=1}^{m} a_{ij})]$

Hereby it should be noted that each summation for i=1 to m only applies to the violated constraints. Furthermore b(i) is the information still needed at information point i, $rsum(i) = \Sigma_j a_{ij}$, where the summation over j is performed with respect to every non-selected item j, and finally space(i) = rsum(i) - b(i). This means that both b(i) and rsum(i) change continuously during the selection process, and therefore all criterion values have to be calculated anew at every step. I assume rsum(i) and space(i) always to be greater than zero, else there would be no feasible solution for (P).

Regarding criteria 7 and 8 a computational problem arises when the arguments of the logarithm get smaller than one. In those cases the criterion values are set equal to $1/c_j$. In the original multicovering case those problems would not arise.

Finally with "criterion(k,j)" in figure 6.1 is meant the criterion value of criterion k for item j, k=1..8, j=1..n.

The flowchart of the complete Top5 algorithm can be found in figure 6.2.

Ad (I)

This random choosing can be done in a weighted and in an unweighted version. I worked with weights, i.e. gave the criterion that scored best in part 1 a greater chance to be chosen than number two etc.. The values of these weights are more or less arbitrary and can be found best through experience.

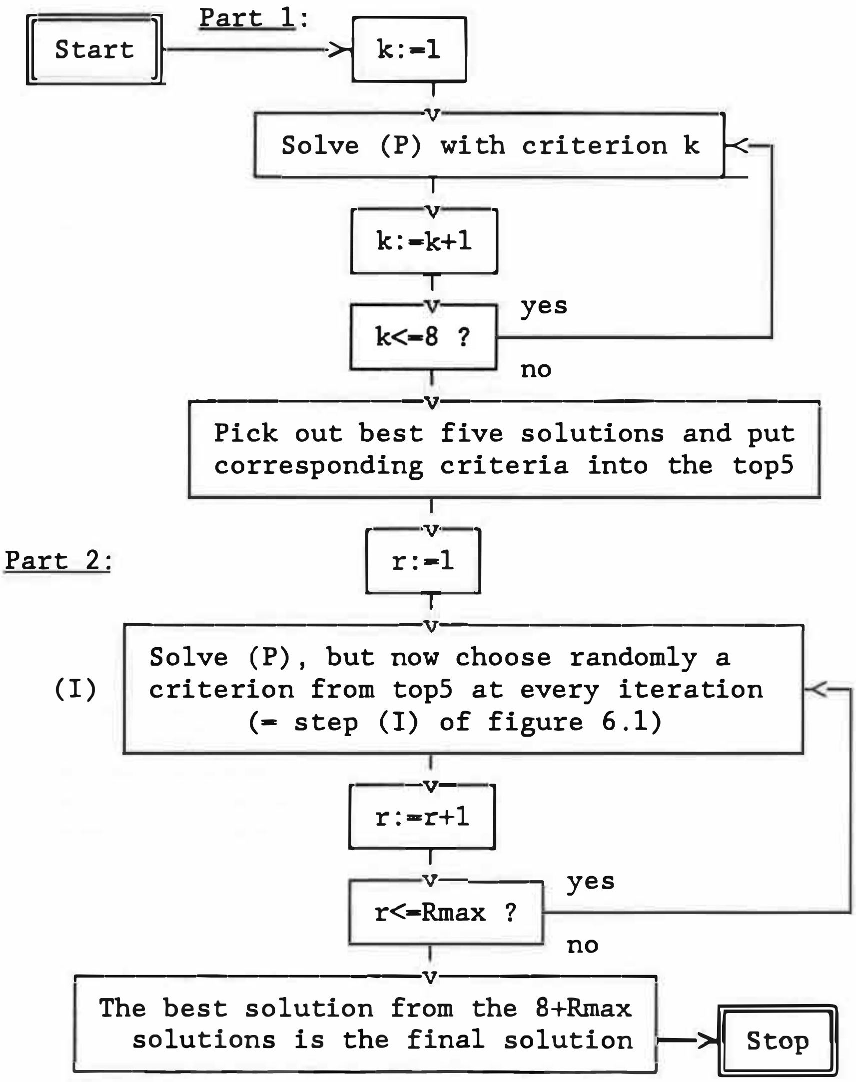

Figure 6.1 Flowchart of the complete Top5 algorithm

Experiments performed with the algorithm from figure 6.2 exhibited two striking features. First, the first part of the algorithm gives the same three criteria as the best in almost all cases. Those criteria are the numbers 5,2 and 4, in that order. Secondly it takes a very long time, sometimes more than 30 minutes for a problem with 300 items, to solve (P) with this algorithm. This is due to the fact that (P) is solved not just once, but altogether 8+Rmax times.

This made me introduce the following changes into the algorithm.

- The first part is skipped. I directly start with the second part of the algorithm, with a top3 instead of a top5, consisting of criteria 5,2 and 4.
- The weights for this top3 are set 0.50, 0.30 and 0.20. Rmax is set equal to 3.

These changes lead to more reasonable computation times, as can be seen in the description of the test results in section 8.

6.2 A confidence interval for the optimal solution

In [8] a method is given to construct a confidence interval for the optimal solution for (P), in the light of the heuristic solutions from the second part of the Top5 algorithm. The idea is based on an article by B.L. Golden and F.B. Alt [7] and stems from the following line of thought.

Suppose there are S independent solutions for (P) obtained by one or more heuristics. The corresponding objective function values z_i are bounded from below by the unknown optimal objective function value z(P), i=1..S. Now the distribution of the z_i approaches a Weibull distribution with z(P) as the location parameter. This distribution generally has the following shape.

Weibull distribution

$$F_x(x0) = Prob(x<=x0) = 1 - exp(-[(x0-a)/b]^c)$$
$$\text{with } 0<=a<=x0,\ b>=0 \text{ and } c>=0 \qquad (6.2)$$

Here a is the location parameter, so in this case a=z(P), b is the scale parameter and c is the shape parameter.

Suppose that from now on the z_i are arranged in increasing order with z_1=v. It can be derived from (6.2) that $F_{zi}(a+b)$ = 1-exp(-1). From this it follows:

$$\begin{aligned} Prob(v<=a+b) &= 1 - Prob(v>a+b) = \\ &= 1 - Prob(z_i>a+b,\ i=1..S) = \\ &= 1 - (1-F_{z1}(a+b))* \dots *(1-F_{zS}(a+b)) = \\ &= 1 - exp(-S) \end{aligned}$$

$$\text{Or: } Prob(v-b<=a<=v) = 1 - exp(-S) \qquad (6.3)$$

In other words: (v-b,v) is a 100*(1-exp(-S))% confidence interval for the optimal objective function value of problem (P). So the intention now is to get an estimate of the parameter b from the objective function values z_i, i=1..S.

Maximum likelihood equations

One way could be to solve the maximum likelihood equations for the parameters a,b and c from the Weibull distribution. Those equations can be found in Johnson and Kotz [10], but are very complicated: it would take a lot of time to solve them.

Statgraphics

A second method is to use statistical software that is able to provide good estimates for the Weibull parameters. I tried the software package Statgraphics, but this package knows the Weibull distribution only by the two parameters b and c; a is assumed to be zero, which it is not in the problem considered here. I tried to solve this by substracting v from all z_i, i=1..S, but this did not lead to satisfactory results.

A third approach was suggested in [8]. Good estimates for the Weibull parameters are given by:

$$a = v-(z_2-v)$$
$$b = z_r-a \qquad (6.4)$$
$$c = \ln(-\ln(0.5))/(\ln(z_m-a)-\ln(b))$$

Here z_m is the median of $z_1..z_S$ and r=[0.63*S+1], with [z] being the largest integer less than or equal to z. In [8] the estimates from (6.3) are used as initial values for the Harter-Moore iterative procedure [9] in order obtain very good estimates for the Weibull parameters.

Since the Top5 method was already taking quite a lot of time, I decided to use the initial values from (6.3) to construct the confidence interval (v-b,v) for z(P). Note that expression (6.2) with S=Rmax=3 already provides a 95% security that z(P) is in the interval, but the estimate for b will in this case not be so accurate. The consequences of this are illustrated by the test results in section 8.

7 The Subgradient method

This method was developed according to an article by J.E. Beasley [2], which on its turn was based on an article by E. Balas and A. Ho [1]. The article [2] deals with the set covering problem (SCP).

Set covering problem

$$\text{(SCP)} \quad \min \sum_{j=1}^{n} c_j * x_j$$
$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} * x_j >= 1 \qquad i=1..m \qquad (7.1)$$
$$x_j \in \{0,1\} \qquad j=1..n$$

This is basically the same problem as the multicovering problem of (6.1) and so the main difference with (P) is again that $a_{ij} \in \{0,1\}$ for i=1..m and j=1..n. Note that here $b_i=1$ for i=1..m. This is no limitation, since in (P) every information constraint i can be divided by its target information value b_i, because all $b_i>0$, i=1..m. From now on in this section I will assume that (P) is transformed accordingly, i.e. $b_i=1$ for i=1..m.

Although the algorithm that is described in [2] is specially designed for (SCP), I adopted the general principle on which the algorithm was built. The underlying idea is as follows.

First a feasible solution for the dual problem of a relaxed version of problem (P) is determined. This serves as a lowerbound on the optimal objective function value of (P). Then this lowerbound is improved by means of subgradients. An upperbound is obtained by a heuristic that is called several times in the course of the algorithm and that makes use of the current values of the lagrange multipliers.

In order to get a clear picture of the relations between the various primal and dual problems that are used in this section, I will define those problems that have not been considered yet.

The dual problem (DRP) of problem (RP) of (4.1) is:

Dual relaxed problem

$$\text{(DRP)} \quad \min\left(\sum_{j=1}^{n} w_j - \sum_{i=1}^{m} u_i \right)$$
$$\text{s.t.} \quad w_j - \sum_{i=1}^{m} a_{ij} * u_i >= -c_j \qquad j=1..n \qquad (7.2)$$
$$u_i >= 0 \qquad i=1..m$$
$$w_j >= 0 \qquad j=1..n$$

Setting all $w_j=0$ for $j=1..n$ yields the dual problem (DP) that is regarded in behalf of this algorithm.

$$\text{(DP)}\quad \max \sum_{i=1}^{m} u_i$$
$$\text{s.t.}\quad \sum_{i=1}^{m} a_{ij}*u_i <= c_j \qquad j=1..n \qquad (7.3)$$
$$u_i >= 0 \qquad i=1..m$$

This is the dual problem of problem (RP) without the upperbounds to the variables x_j, $j=1..n$. I call this latter problem the linear relaxation (LRP) of problem (P).

Linear relaxation

$$\text{(LRP)}\quad \min \sum_{j=1}^{n} c_j*x_j$$
$$\text{s.t.}\quad \sum_{j=1}^{n} a_{ij}*x_j >= 1 \qquad i=1..m \qquad (7.4)$$
$$x_j >= 0 \qquad j=1..n$$

Now let z(Q) be the optimal objective function value of problem (Q), where Q is from the problem set {P,RP,LRP,DRP,DP}, then it follows that:

$$z(LRP) = z(DP) <= z(RP) = z(DRP) <= z(P) \qquad (7.5)$$

So a feasible solution for (DP) provides a lowerbound on the optimal objective function value of (P).

In figure 7.1 the flowchart of the algorithm can be found. The various subroutines used in this flowchart will now be explained in order to elucidate the algorithm.

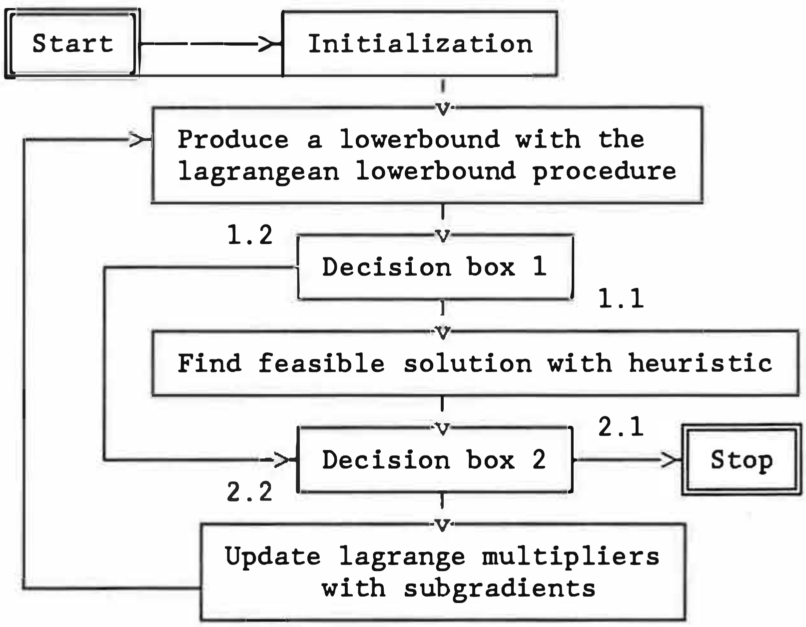

Figure 7.1 Flowchart of the Subgradient algorithm

Initialization

Both an initial feasible solution for (P) as for (DP) are required. The former serves as the first upperbound and the latter as the first lowerbound on the optimal objective function value of (P).

The first upperbound z_{ub} is obtained quickly by Mindev, while the first lowerbound z_{lb} is determined by setting for all k=1..m: $u_k := u := \min[\, c_j / \Sigma_i\, a_{ij}\,]$. This gives a feasible solution for (DP), because suppose otherwise, i.e. there is a $k \in \{1..n\}$ with $\Sigma_i\, a_{ik} * u_i > c_k$, then $u * \Sigma_i\, a_{ik} > c_k$ and hence $c_k / \Sigma_i\, a_{ik} < u = \min[\, c_j / \Sigma_i\, a_{ij}\,]$, which leads to a contradiction. So the first lowerbound is given by $z_{lb} = \Sigma_i\, u_i = m * u$.

The lagrange multipliers s_i are initiated as $s_i := u_i = u$, i=1..m.

Lagrangean lowerbound procedure

In this procedure the lagrangean lowerbound problem (LLP) is solved ,given the current values of the lagrange multipliers s_i, i=1..m. This problem is:

$$\text{(LLP)} \quad \min\Big(\sum_{j=1}^{n} [c_j - \sum_{i=1}^{m} a_{ij} * s_i] * x_j + \sum_{i=1}^{m} s_i\Big) \qquad (7.6)$$
$$\text{s.t.} \quad x_j \in \{0,1\} \quad j=1..n$$

Now define $C_j := c_j - \Sigma_i\, a_{ij} * s_i$ as the lagrangean costs, then the solution for (LLP) is found by setting $x_j := 1$ if $C_j <= 0$ and $x_j := 0$ otherwise, j=1..n. Call the resulting solution vector X. It can be proved that a new lowerbound is given by $z = \Sigma_i\, C_j * X_j + \Sigma_i\, s_i$. If $z > z_{lb}$ then update $z_{lb} := z$. In that case a sharper lowerbound has been found.

Greedy heuristic

This heuristic produces a feasible solution for (P), which also is a new upperbound. It selects items at minimal lagrangean costs till all information constraints are satisfied. It is not so much a sophisticated as a fast algorithm that makes use of the continuously changing lagrange multipliers. As a consequence it provides different solutions, which tend to get better as the lagrange multipliers are improving. The flowchart of this heuristic can be found in figure 7.2.

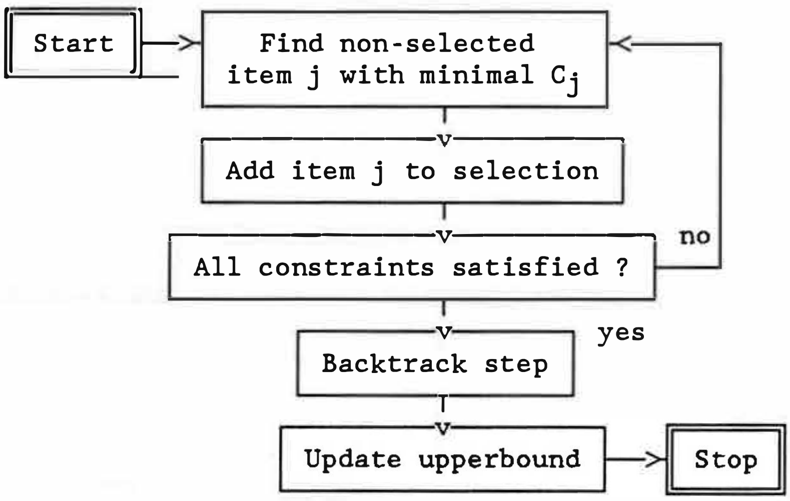

Figure 7.2 Flowchart of the greedy heuristic using lagrangean costs

Subgradients

The subgradients G_i are determined in order to improve the lagrange multipliers s_i, i=1..m. They provide a search direction for better s_i and are defined as:

$$G_i := 1 - \sum_{j=1}^{n} a_{ij} * X_j \qquad i=1..m \qquad (7.7)$$

Here $X=(X_1..X_n)^T$ is the current solution vector determined by the lagrangean lowerbound procedure.

Intuitively G_i can be seen as a kind of slack parameter for the information constraint i from problem (P). If $G_i<0$ then there is enough information at constraint i and therefore one can give less weight to this constraint: s_i can be lowered. If $G_i>0$ then there is not enough information at constraint i, so there has to be an extra emphasis on this constraint: s_i has to be enlarged. All this applies to i=1..m.

The lagrange multipliers are updated as follows. Let f be a factor that is initiated at 2 and is halved whenever there is no substantial improvement on the lowerbound for some iterations in a row. Define the stepsize T as $T := f*(z_{ub} - z_{lb}) / \Sigma_i G_i^2$, then the new lagrange multipliers become:

$$s_i := \max[\ 0,\ s_i + T*G_i\ ] \qquad i=1..m \qquad (7.8)$$

Decision boxes

Box 1

Although the greedy heuristic is fast, it can not be called at every iteration, since the number of iterations can be more than 200. Therefore it is only called about ten times during the algorithm. In box 1 the heuristic is called whenever the improvement of the lowerbound has been less than 0.01 over the last nine iterations [1.1]. Otherwise the algorithm continues with box 2 [1.2].

Box 2 Whenever the improvement on the lowerbound has been less than 0.01 over the last ten iterations, the factor f is halved. If this makes $f<0.008$ the algorithm stops [2.1]. Otherwise the lagrange multipliers are updated again [2.2].

The just described algorithm is my final version of Subgradient, the version I used in the experiments. However this does not mean that it is the best possible Subgradient method. Especially the criteria used at the decision boxes offer an opportunity for changes that might lead to improvements. I did some efforts on that area, but without significant success.

A second possibility for improvement may be the greedy heuristic. If in some easy way one could find out when the lagrange multipliers are "good", the heuristic could be called at those moments, probably leading to better solutions than in the situation where the calls take place rather arbitrary.

Further I tried another heuristic based on the Surrogate method. The lagrange multipliers s_i, i=1..m, are used in order to get problem (P) in a form like (4.4): a (0,1)-knapsack problem with one surrogate constraint. Now a feasible solution for (P) is obtained by selecting item j which maximizes $\Sigma_i\ s_i*a_{ij}/c_j$ until all information constraints are satisfied. However this search is not essentially different from the search for an item j which minimizes $(c_j - \Sigma_i\ s_i*a_{ij})$, which is done at the greedy heuristic. So it is no surprise that the results with this surrogate heuristic were almost the same as with the greedy heuristic, while the latter was a bit faster. Therefore I maintained the greedy heuristic.

Another effort I made concerned speeding up the convergence of the lagrange multipliers, by preventing a zigzag-process. According to a study by A. de Boer [3] this zigzag-effect can occur when the angle between the old multiplier vector s_k and the new one s_{k+1} is obtuse. That is whenever $<s_{k+1},s_k> < 0$, with <.,.> the Eucledian inproduct. In this case the change in the lagrange multipliers is too drastic and has to be slowed down. This can be done by setting the new lagrange multipliers vector:

$$s_{k+1} := s_{k+1} - 2*(<s_{k+1},s_k>/<s_k,s_k>)*s_k \qquad (7.9)$$

Unfortunately this adjustment leads to a deterioration instead of an improvement, however with other changes of this kind a better Subgradient method might be obtained.

In the next section this method will be tested and compared with other methods from this study.

8 Test results

In this section various methods from the previous sections are tested in order to come to a clear insight into the practical use of these methods. For this testing I always used the same set of problems: a problem file introduced by E. Timminga [14] consisting of thirteen problems with various structures. This problem file can be found in appendix I. Further I worked all the time with an item bank, generated by a program by Razoux Schultz [12], containing 300 items. The difficulty parameters are drawn from a Normal distribution with mean equal to 0 and variance equal to 2. The categories are drawn from a Discrete Uniform distribution on {1..5}. This bank can be found in appendix II.

The algorithms used for the testing are the Simplex algorithm from section 4.1.2, the algorithm Mindev from section 4.2.3, the Top5 method from section 6, which is strictly speaking a Top3 method, the algorithm Subgradient from section 7 and the two category division methods Split-up and Direct from the sections 5.2 and 5.3 respectively. I introduced two little adaptations into the Simplex algorithm. First I made it appropriate for working with the given problem file and item bank and secondly I had it perform extra backtrack steps, i.e. investigate for every item whether it is redundant or not, instead of just for one item.

The tests are performed on a Victor V286 personal computer (XT) for three different cost structures. First a low cost structure, in which favourable items, i.e. items with costs smaller than one, have unfavourable difficulty parameters, i.e. difficulty parameters in the range (-3,-0.75) or (0.75,3). So in the ordered item bank those items have low or high numbers. The second is a high cost structure: items with favourable difficulty parameter have an unfavourable cost. Finally there is a random cost structure, in which the costs vary from favourable to unfavourable throughout the item bank.

8.1 Low cost structure

In this cost structure all items have a cost of one, except for the items:
1,5,295,300: cost=0.5
10,20,30,270,280,290: cost=0.6
40,50,60,70,80,210,220,230,240,250: cost=0.7
The desired percentages for the category division can be found in the tables 8.1 up to and including 8.6.

Table 8.1 Test results on low cost structure for Simplex

Problem	Costs	Time(s)	Lowbound
1	64.2	74	64.015
2	18.6	101	18.396
3	24.6	137	24.010
4	76.0	274	75.619
5	33.1	176	31.343
6	60.7	151	60.126
7	31.2	138	30.694
8	73.2	178	72.570
9	24.6	112	24.010
10	73.2	136	72.390
11	15.5	38	15.413
12	76.0	147	75.619
13	33.1	114	31.343
Total	604.0	1776	595.548

Table 8.2 Test results on low cost structure for Mindev

Problem	Costs	Time(s)
1	64.4	17
2	19.6	5
3	24.6	5
4	78.7	19
5	33.1	5
6	62.1	14
7	31.5	4
8	73.6	19
9	24.6	5
10	72.8	19
11	15.7	5
12	78.7	19
13	33.1	4
Total	612.5	140

Table 8.3 Test results on low cost structure for Top5

Problem	Costs	Time(s)	Conf.int
1	64.2	140	(64.2 64.2)
2	19.0	98	(19.0 19.0)
3	24.6	189	(24.0 24.6)
4	76.4	412	(76.4 76.4)
5	32.6	298	(32.6 32.6)
6	60.4	254	(59.2 60.4)
7	31.2	194	(31.2 31.2)
8	72.8	223	(72.8 72.8)
9	24.6	132	(24.6 24.6)
10	73.2	271	(73.2 73.2)
11	15.5	41	(15.5 15.5)
12	77.4	238	(77.4 77.4)
13	32.6	162	(32.6 32.6)
Total	604.5	2652	-

Table 8.4 Test results on low cost structure for Subgradient

Problem	Costs	Time(s)	Lowbound
1	64.2	82	64.015
2	18.8	115	18.392
3	24.6	154	24.010
4	76.3	309	75.618
5	32.4	241	31.309
6	60.7	153	60.121
7	31.2	114	30.692
8	72.8	124	72.568
9	24.6	118	24.009
10	72.6	239	72.352
11	15.5	58	15.413
12	75.7	129	75.618
13	32.4	120	31.331
Total	601.8	1956	595.448

According to these tables, there is a clear difference between the fast algorithm Mindev and the slower algorithms Simplex, Top5 and Subgradient. Mindev on the average requires only 11 seconds to solve a problem, while the slower algorithms take 137 to 204 seconds. However the latter algorithms give solutions that are closer to optimal. Here Subgradient bears the palm with also lowerbounds that are very close to those of the Simplex method. The confidence intervals given by the Top5 algorithm are bad: at six of the thirteen problems the optimal objective function value is certainly not in the interval. This can probably be due entirely to the lack of a sufficient number of random runs Rmax. With Rmax=3 it appears to be impossible to come to a good estimate for the b-parameter of the Weibull distribution. More random runs however would lead to too high computation times.

Quadratic deviation

Before passing to the tables 8.5 and 8.6 I have to explain the term quadratic deviation (QD), that is used in these and other tables.

$$QD := \Sigma_k (DF_k - RP_k)^2 \qquad (8.1)$$

Here RP_k is the realized percentage for category k, k=1..5. These percentages can be found in the columns before the QD-column.

Table 8.5

Test results on low cost structure for Split-up

Problem	Costs	Time(s)	RP_1	RP3	RP_5	QD
1	66.8	4	30.4	40.6	29.0	1.5
2	21.5	2	34.6	38.5	26.9	33.0
3	27.8	2	30.3	39.4	30.3	0.5
4	80.8	5	29.8	40.4	29.8	0.2
5	39.8	2	31.1	40.0	28.9	2.4
6	63.5	4	31.3	38.8	29.9	3.1
7	38.8	2	31.8	40.9	27.3	11.3
8	77.4	5	30.0	40.0	30.0	0.0
9	27.8	2	30.3	39.4	30.3	0.5
10	76.4	5	30.4	40.5	29.1	1.2
11	17.3	1	31.6	36.8	31.6	15.4
12	80.8	5	29.8	40.4	29.8	0.2
13	38.1	2	30.2	41.9	27.9	8.1
Total	656.8	41	-	-	-	77.4

Table 8.6 Test results on low cost structure for Direct

Problem	Costs	Time(s)	RP_1	RP_3	RP_5	QD
1	66.2	15	27.9	39.7	32.4	10.3
2	20.5	5	32.0	40.0	28.0	8.0
3	26.5	4	34.4	37.5	28.1	29.2
4	79.1	18	30.1	38.6	31.3	3.7
5	34.5	4	27.5	40.0	32.5	12.5
6	62.2	13	30.3	37.9	31.8	7.7
7	34.4	4	30.8	38.4	30.8	3.8
8	74.9	17	29.5	38.5	32.0	6.5
9	26.5	4	34.4	37.5	28.1	29.2
10	75.1	18	30.8	38.4	30.8	3.8
11	15.8	4	29.4	41.2	29.4	2.2
12	79.1	17	30.1	38.6	31.3	3.7
13	34.8	4	30.0	37.5	32.5	12.5
Total	629.6	127	-	-	-	133.1

From the tables 8.5 and 8.6 it can be concluded that the Split-up approach comes closer to the desired percentages and in less time than the Direct approach. The latter however keeps a better eye on the costs.

8.2 High cost structure

In this cost structure the items 76 up to and including 224 have a cost of 1.5 and the other items all have a cost of one. The desired percentages for the category division are DF_1=0, DF_2=50, DF_3=0, DF_4=50 and DF_5=0. The test results can be found in the tables 8.7 up to 8.12 included.

The conclusions from the tables 8.7, 8.8, 8.9 and 8.10 are similar to those in the low cost structure. Now Mindev takes 7 seconds averagely, while the other methods require 153 to 214 seconds. From those methods again Subgradient is best. The confidence intervals by Top5 do not contain the optimal values at five problems at least.

Concerning tables 8.11 and 8.12 I can say that again Split-up comes closer to the desired percentages at less time, but the Direct approach provides solutions at lower costs.

Table 8.7 Test results on high cost structure for Simplex

Problem	Costs	Time(s)	Lowbound
1	92.0	129	91.546
2	24.0	88	22.826
3	31.0	222	29.949
4	105.0	246	104.416
5	39.0	197	38.571
6	74.5	178	73.452
7	38.0	180	36.812
8	104.0	139	103.679
9	31.0	169	29.949
10	104.0	207	103.679
11	21.0	59	20.619
12	105.0	260	104.416
13	39.0	121	38.571
Total	807.5	2195	798.485

Table 8.8 Test results on high cost structure for Mindev

Problem	Costs	Time(s)
1	92.5	14
2	23.5	1
3	31.5	2
4	107.5	16
5	39.0	2
6	73.5	4
7	39.0	1
8	104.0	16
9	31.5	2
10	104.0	16
11	21.0	4
12	107.5	16
13	40.5	3
Total	815.0	97

Table 8.9 Test results on high cost structure for Top5

Problem	Costs	Time(s)	Conf.int.
1	92.0	162	(92.0 92.0)
2	24.0	99	(24.0 24.0)
3	32.0	177	(32.0 32.0)
4	105.5	436	(105.5 105.5)
5	39.0	295	(39.0 39.0)
6	74.0	271	(74.0 74.0)
7	38.0	191	(38.0 38.0)
8	104.0	244	(104.0 104.0)
9	31.0	128	(31.0 31.0)
10	105.0	314	(105.0 105.0)
11	21.0	49	(21.0 21.0)
12	104.5	259	(104.5 104.5)
13	39.0	160	(37.0 39.0)
Total	809.0	2785	-

Table 8.10 Test results on high cost structure for Subgradient

Problem	Costs	Time(s)	Lowbound
1	92.0	105	91.544
2	23.5	102	22.826
3	31.0	155	29.938
4	104.5	238	104.414
5	39.0	318	38.504
6	73.5	167	73.440
7	38.0	115	36.810
8	104.0	127	103.679
9	31.0	115	29.942
10	104.0	250	103.561
11	21.0	51	20.619
12	104.5	148	104.415
13	39.0	103	38.571
Total	805.0	1994	798.263

Table 8.11 Test results on high cost structure for Split-up

Problem	Costs	Time(s)	RP_2	RP_4	QD
1	100.0	3	50.0	50.0	0.0
2	25.0	1	50.0	50.0	0.0
3	33.0	1	50.0	50.0	0.0
4	116.0	4	50.6	49.4	0.7
5	49.0	1	48.9	51.1	2.4
6	99.0	3	48.7	51.3	3.4
7	49.0	1	48.9	51.1	2.4
8	115.0	4	50.0	50.0	0.0
9	33.0	1	50.0	50.0	0.0
10	115.0	4	50.0	50.0	0.0
11	23.0	1	47.6	52.4	11.5
12	116.0	4	50.6	49.4	0.7
13	49.5	1	48.9	51.1	2.4
Total	922.5	29	-	-	23.5

Table 8.12 Test results on high cost structure for Direct

Problem	Costs	Time(s)	RP_2	RP_4	QD
1	99.0	8	49.3	50.7	1.0
2	24.5	1	50.0	50.0	0.0
3	31.0	1	48.4	51.6	5.1
4	116.0	9	50.6	49.4	0.7
5	43.5	2	48.8	51.2	2.9
6	97.0	6	49.3	50.7	1.0
7	42.0	1	47.6	52.4	11.5
8	114.0	9	49.4	50.6	0.7
9	31.0	1	48.4	51.6	5.1
10	114.0	9	49.4	50.6	0.7
11	22.0	2	45.0	55.0	50.0
12	116.0	9	50.6	49.4	0.7
13	43.5	2	48.8	51.2	2.9
Total	893.5	60	-	-	82.3

8.3 Random cost structure

In this random cost structure for all items the costs are drawn from a Uniform distribution on (0.1 2.1). The desired percentages are DF_1=25, DF_2=25, DF_3=25, DF_4=25 and DF5=0. The test results can be found in tables 8.13 up to and including 8.18.

Table 8.13 Test results on random cost structure for Simplex

Problem	Costs	Time(s)	Lowbound
1	30.36	182	30.326
2	4.81	116	4.678
3	8.11	205	7.767
4	37.33	349	37.006
5	16.22	273	15.884
6	27.24	225	27.193
7	16.22	227	15.884
8	37.14	267	36.975
9	8.11	159	7.767
10	37.14	243	36.975
11	3.70	71	3.547
12	37.33	243	37.006
13	16.22	200	15.884
Total	279.93	2760	276.892

Table 8.14 Test results on random cost structure for Mindev

Problem	Costs	Time(s)
1	31.50	28
2	6.48	7
3	8.40	8
4	38.39	32
5	17.30	9
6	28.56	24
7	17.30	9
8	37.04	32
9	8.40	8
10	37.04	32
11	4.38	6
12	38.39	32
13	17.30	9
Total	290.48	236

Table 8.15 Test results on random cost structure for Top5

Problem	Costs	Time(s)	Conf.int.
1	30.36	158	(30.36 30.36)
2	4.79	93	(4.79 4.79)
3	7.98	184	(7.71 7.98)
4	37.15	435	(37.15 37.15)
5	16.25	305	(16.16 16.25)
6	27.24	258	(27.24 27.24)
7	16.30	211	(16.30 16.30)
8	37.09	239	(37.09 37.09)
9	7.98	131	(7.98 7.98)
10	37.09	298	(37.09 37.09)
11	3.70	43	(3.70 3.70)
12	37.15	251	(37.15 37.15)
13	16.25	177	(16.25 16.25)
Total	279.33	2783	-

Table 8.16 Test results on random cost structure for Subgradient

Problem	Costs	Time(s)	Lowbound
1	30.36	89	30.326
2	4.79	89	4.678
3	8.11	122	7.766
4	37.33	209	37.005
5	16.22	183	15.881
6	27.24	119	27.193
7	16.22	113	15.884
8	37.04	105	36.972
9	8.11	87	7.767
10	37.04	219	36.960
11	3.70	50	3.547
12	37.28	107	37.006
13	16.22	95	15.884
Total	279.66	1587	276.869

None of the tables 8.13 / 8.16 enforces any changes in the conclusions about the algorithms. Mindev is the fastest with an average of 18 seconds, Subgradient requires 122 seconds, and Simplex and Top5 take 212 and 214 seconds respectively. The obtained costs hardly differ for the slow methods and the confidence intervals by Top5 now certainly do not contain the optimal objective function value at four problems.

Table 8.17 Test results on random cost structure for Split-up

Prob	Costs	Time(s)	RP_1	RP_2	RP_3	RP_4	QD
1	39.77	7	23.5	25.9	23.5	27.1	9.7
2	10.79	2	23.1	26.9	23.1	26.9	14.4
3	14.55	2	25.7	22.9	25.7	25.7	5.9
4	48.86	8	24.4	25.6	24.4	25.6	1.4
5	23.47	3	28.0	24.0	26.0	22.0	20.0
6	37.02	6	23.6	25.0	26.4	25.0	3.9
7	23.47	3	28.0	24.0	26.0	22.0	20.0
8	48.21	8	24.2	25.3	24.2	26.3	3.1
9	14.55	2	25.7	22.9	25.7	25.7	5.9
10	48.21	8	24.2	25.3	24.2	26.3	3.1
11	7.36	2	25.0	25.0	25.0	25.0	0.0
12	48.86	8	24.4	25.6	24.4	25.6	1.4
13	23.47	3	28.0	24.0	26.0	22.0	20.0
Tot	388.59	62	-	-	-	-	108.8

Table 8.18 Test results on random cost structure for Direct

Prob	Costs	Time(s)	RP_1	RP_2	RP_3	RP_4	QD
1	37.74	31	22.5	26.3	26.2	25.0	9.4
2	6.13	8	28.0	24.0	24.0	24.0	12.0
3	9.80	8	21.2	24.2	27.3	27.3	25.7
4	47.04	35	23.3	25.6	25.6	25.5	3.9
5	18.94	11	21.3	27.7	23.4	27.6	30.3
6	34.90	26	25.0	25.0	26.4	23.6	3.9
7	18.94	11	21.3	27.7	23.4	27.6	30.3
8	46.31	34	22.5	25.8	27.0	24.7	11.0
9	9.80	8	21.2	24.2	27.3	27.3	25.7
10	46.31	35	22.5	25.8	27.0	24.7	11.0
11	5.00	6	26.3	21.1	31.6	21.0	76.5
12	47.04	35	23.3	25.6	25.6	25.5	3.9
13	18.94	10	21.3	27.7	23.4	27.6	30.3
Tot	346.89	258	-	-	-	-	273.9

This also leads to the same conclusions as before. Split-up is the fastest and gives better proportions, while Direct provides solutions at lower costs.

9 Complexity of the algorithms

In this section I will make some remarks on the complexity of the algorithms that were tested in the previous section, in connection with the size of the item selection problem. This size depends on the number of items in the bank, n, and on the target information values, which have a direct bearing upon the number of items required to fulfil the information constraints.

Item need

Define the item need N as the number of items needed to satisfy the information constraints. Now N will only be known when an algorithm has solved (P), but that is annoying for someone who wants to predict something about the computation time before applying an algorithm. Therefore it is necessary to have an estimate of N.

N is problem-dependent. It depends on the target information values, the number of information points m and the distance d between the first and the last information point, i.e. $d=\theta_m-\theta_1$. Now if $B= \Sigma_i b_i$ is the total target information, then a good measure for N has proved to be:

$$N \approx B*(d+8)/(2*m) \qquad (9.1)$$

Of course this formula can not be used for exact calculations of N, since this also depends for instance on the cost structure, but it gives an idea of the magnitude of N, which is sufficient regarding the determination of the complexities in this section. Therefore I will treat N as a known quantity, since (9.1) gives an adequate estimate.

The number of information constraints m can be regarded as a kind of a constant. It will always be in the range {1..7}, so when I need m in the complexity calculations I will use the average value m = 4. Now the complexity of the algorithms Simplex, Mindev, Top5 and Subgradient will be determined successively.

9.1 Simplex

Theoretically this algorithm, that means the pure Simplex part of it, has an exponential complexity. In practice however it has a complexity of $O(n^3)$.

9.2 Mindev

Stepwise I will go through the algorithm and determine the complexity by counting the simple statements that have to be executed. Hereby I make no distinction between p.e. an addition or a multiplication. However I think it is sufficient for getting a general idea of the complexity of the algorithm.

Initialization: $\frac{1}{2}mn + n + m \approx n(\frac{1}{2}m + 1)$
Selection proces: $N*(4m + \frac{1}{2}n) \approx \frac{1}{2}nN$
Backtrack step: $\frac{1}{4}n^2 + nm \approx \frac{1}{4}n^2$
Reporting: $m + N \approx N$

This yields for the entire algorithm:
$n*(\frac{1}{2}m + 1 + \frac{1}{2}N + \frac{1}{4}n) + N \approx \frac{1}{4}(n^2 + 2nN)$, so the complexity of Mindev is $O(n^2 + 2nN)$.

9.3 Top5

The same procedure as with Mindev gives:

Initialization: $2m + nm \approx nm$
Random selections: $3*[3m + 2n + N(2mn+3m) + \frac{1}{4}n^2 + nm] \approx$
$\approx 6mnN + 3/4n^2$

After substitution of m=4 the total becomes:
$4n + 24nN + 3/4n^2 \approx 3/4n^2 + 24nN$, so the complexity for Top5 is $O(n^2 + 32nN)$.

9.4 Subgradient

For Subgradient stepwise determination gives:

Mindev start: $\frac{1}{4}(n^2 + 2nN)$
Initialization: $nm + mn(m+1) = mn(m+2)$
Lowerbound procedure: $f(m,N)*(2nm + 3m + n) \approx$
$\approx f(m,N)*(n(2m+1))$
Greedy algorithm: $10*[\ N(n+2m) + n(m+2)\ ] \approx 10nN$
Reporting: $n(m+1)$

Here f(m,N) represents the number of iterations in the lowerbound procedure. I assume it is of O(mN). In practice f(m,N) = mN is sufficient, however this can change drasticly if the criteria used in the decision boxes are changed.

After substitution of m=4 and f(m,N)=mN the total becomes:
$n*[\frac{1}{4}n + 10\frac{1}{2}N + mN(2m+1) + m+1 + m(m+2)] \approx n*(\frac{1}{4}n + 46\frac{1}{2}N)$, so the complexity of Subgradient is $O(n^2 + 186nN)$.

9.5 Some final remarks on the determined complexities

There are two remarks to be made on the results of the previous sections.

(i) Regarding the complexity of Subgradient and Top5, one could think that Top5 is a faster algorithm, which is in contradiction with the test results of section 8. But remember that these complexities give an idea of how computation time increases when n and N increase. They can not be seen as formulas that give the computation time for the algorithm in any situation.

(ii) It can now be explained why the Split-up approach is faster than the Direct approach. Suppose five categories are used, then the complexities for Split-up and Direct can be derived from the complexity of Mindev.

Direct: $n^2 + 2nN$
Split-up: $5*[(1/5n)^2 + 2*1/5n*1/5N] = 1/5*(n^2 + 2nN)$

So in this case the Direct approach will require five times as much computation time as the Split-up approach. Hence it follows that the more categories are used, the faster the Split-up approach becomes relative to the Direct approach.

More in general: if the principle of dividing problem (P) into subproblems is used in the algorithms Top5, Subgradient or Simplex, the computation time can be reduced drastically. However in section 8 it can be seen that this computation time reduction is at the expense of solutions at higher costs, so this should only be tried in situations where costs are not that important, like with category division. It may be a nice subject for further study.

10 Conclusions and recommendations

In this thesis several methods to solve the item selection problem have been discussed. All these methods were developed for problems with positive costs. Two of them also had the possibility of selecting items according to a certain specified category division. This last section will be used for some final remrks on the described algorithms and for a look into the future: what is there still to be done?

The test results from section 8 have shown that problem (P) from (2.2) can be solved very close to optimal by the heuristics Subgradient, Top5 and the quasi-exact Simplex method. This is done within a reasonable time. Subgradient can be regarded as the best of those algorithms, since it gives solutions at the lowest costs in the least time and moreover provides a lowerbound on the optimal objective function value which is almost as sharp as the exact solution of the relaxed problem (RP), i.e. the lowerbound given by the Simplex method.

The lowerbounds given by Top5 are disappointing. However it is very well possible that those confidence intervals are a failure simply because the number of independent solutions, that is the number of random runs in the algorithm, is not sufficient for a good estimate of the b-parameter of the Weibull distribution. It is also possible that the assumption of a Weibull distribution for the objective function values does not suit the actual situation, for one would expect that a lowerbound procedure is in some way dependent of the heuristic used, which is not the case in the Top5 method. This can only be discovered by testing the algorithm with a larger number of random runs. However this will take quite a lot of time, which makes such a method less competitive for solving the item selection problem. Still this Top5 approach is very interesting, if only because of the fact that the random runs provide better solutions than the runs where just one criterion is used. I think the Top5 method deserves further study.

The two methods that deal with the category division appear to perform well. The Split-up approach is the fastest and reproduces the desired percentages best, but the Direct approach provides solutions at lower costs. It seems a matter of taste which method is to be preferred. If however the solutions given by Direct or Split-up would give unsatisfactory results concerning the costs, one can always apply the principle of Split-up, i.e. the division of problem (P) into subproblems, to a slower but more accurate algorithm like Subgradient. In section 9.5 it was shown that this may lead to solutions at low costs in a more than reasonable time.

Now when looking at the place of the item selection problem within the Test Service Systems (TSS), what should still be done before that aspect of TSS is properly attended to? Most of the needs will probably only be discovered in the prototyping phase, but right now I can spot two of them.

The first potential need is the possibility of specifying a maximum number of items Imax one wants to have in a test. Gademann [5] shows that with Simplex this problem can be solved by adding to (P) the constraint:

$$\sum_{j=1}^{n} x_j <= Imax-2 \qquad (10.1)$$

Here Imax-2 is taken instead of Imax in order to absorb the rounding-off effect.

For the other methods the problem becomes more complicated, but it may be solved by an interaction between the user and his computer. If the resulting test does not please him because of too many items, he should lower the target information values so that the number of items in the next test will be less than Imax. This interaction will only perform well after some experience of the user with TSS.

The second need is more urgent. It is the possibility of working with logical restrictions. I already spoke of this in section 4.1.3, where a suggestion of Gademann to formulate this by means of a quadratic objective function is rejected because the corresponding algorithm can not work in practice yet. However the problem remains.

Verstralen [15] shows that logical restrictions can be transformed into a set of linear equations in the (0,1)-variables x_j, j=1..n. For instance the logical restriction "if item 1 is in the test, then item 2 should not be in the test and vice versa" can be transformed into the equation:

$$x_1 + x_2 <= 1 \qquad (10.2)$$

These equations can not be dealt with by Simplex, because of the rounding-off effect. Suppose that the solution of (RP) has $x_1=½$ and $x_2=½$, then rounding off would give $x_1=1$ and $x_2=1$, which would violate constraint (10.2).

A second problem is that a logical restriction with only 10 variables can lead to a set of 70 linear equations. This makes it unlikely that problems with logical restrictions can be solved exactly. Again an interaction between user and computer can be the solution, but that requires a lot of experience with TSS and besides, with many logical restrictions it will be an endless task. Therefore I think that the first priority for further study should be on these logical restrictions.

References

[1] Balas E. and Ho A., Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study, 1980, Mathematical Programming 12, 37-60

[2] Beasley J.E., An algorithm for set covering problem, 1987, European Journal of Operational Research 31, 85-93

[3] Boer A.H.B. de, Subgradientmethoden voor niet differentieerbare konvexe minimaliseringsproblemen, 1987, Master thesis Applied Mathematics, University of Twente, Enschede

[4] Boomsma Y., Item selection by mathematical programming, 1986, Bulletin nr. 47, CITO, Arnhem

[5] Gademann A.J.R.M., Item selection using multiobjective programming, 1987, Master thesis Applied Mathematics, University of Twente, Enschede

[6] Gavish B. and Pirkul H., Zero-one integer programming with few constraints: efficient branch- and bound algorithms, 1985, European Journal of Operational Research 22, 35-43

[7] Golden B.L. and Alt F.B., Interval estimation of a global optimum for large combinatorial problems, 1979, Naval Research Logistics Quarterly 26, 69-77

[8] Gonsalvez D.J., Hall N.G., Rhee W.T.and Siferd S.P., Heuristic solutions and confidence intervals for the multicovering problem, 1987, European Journal of Operational Research 31, 94-101

[9] Harter H.L. and Moore A., Maximum likelihood estimation of the parameters of the Gamma and Weibull populations from complete and from censored samples, 1965, Technometrics 7, 639-643

[10] Johnson N.L. and Kotz S., Continuous univariate distributions-1, 1970, John Wiley and Sons Inc., New York

[11] Lord F.M., Applications of item response theory to practical testing, 1980, Lawrence Erlbaum, Hillsdale, New Jersey

[12] Razoux Schultz A.F., Item selection using heuristics, 1987, Enschede

[13] Syslo M., Deo N. and Kowalik J., Discrete optimization algorithms, 1983, Prentice Hall Inc., Englewood Cliffs, New Jersey

[14] Timminga E., Geautomatiseerd toetsontwerp: Itemselectie met behulp van binaire programmering, 1985, Master thesis Applied Educational Measurement, University of Twente,Enschede

[15] Verstralen H., Processing of Boolean functions in the context of test construction, 1988, Preliminary draft OPD, CITO, Arnhem

[16] Verstralen H., Verkenningen in geautomatiseerde toetsconstructie, 1984, Internal Report, CITO, Arnhem

List of appendices

These appendices can all be found in a separate volume.