Genetic Algorithms for Automated Test Assembly

Angela J. Verschoor

Samenstelling promotiecommissie:

Voorzitter/secretaris	prof. dr. H.W.A.M. Coonen
Promotor	prof. dr. W.J. van der Linden
Referent	prof. dr. P.F. Sanders (Cito, Arnhem)
Leden	prof. dr. M.P.F. Berger (Universiteit Maastricht) prof. dr. C.A.W. Glas prof. dr. H. Holling (Westfälische Wilhelms-Universität, Münster) dr. W.R. van Joolingen

ISBN: 9789058340979
Cover: Sunrise at Yavapai Point, design by Marieke Bonsma

GENETIC ALGORITHMS FOR AUTOMATED TEST ASSEMBLY

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. W.H.M. Zijm,
volgens het besluit van het College van Promoties
in het openbaar te verdedigen
op donderdag 26 april 2007 om 15.00 uur

door

Angela Jose Verschoor
geboren op 15 februari 1961
te Assen

Dit proefschrift is goedgekeurd door de promotor
prof. dr. W.J. van der Linden.

Acknowledgement

There are some similarities between the development of a test and the production of this thesis: Various phases can be distinguished in which many people played a vital role with their contributions.

First, there are the numerous members in the scientific community without whose research this thesis would not have been materialised at all. My employer, Cito, provided the opportunity to develop test assembly software and to facilitate a research project for increasing the versatility of the software.

Then there are my colleagues, both within as well as outside of Cito. Existing models and algorithms appeared sometimes to be inapplicable to their problems, and part of their questions are answered in this thesis. Without the colleagues of the POK department acting as a sounding board, it was not always easy to make the distinction between the good and not-so-good ideas for solving the new problems at hand. Regularly they found their computers still working on the simulations when they arrived Monday mornings.

My promotor, Wim van der Linden, not only provided me with helpful comments on my writing, but also inspired me to continue and finish the project. Instrumental in this continuation and finishing were also the students from Fontys, Tilburg, who assisted me with their research projects.

I would like to thank everyone involved for playing their indispensable role in the realisation of this thesis. But above all, I am obliged to Sabine for her never lasting support and patience.

Angela Verschoor

Contents

1
Introduction

Although some students may disagree, the primary goal of testing is something entirely different than merely pestering them. A test is a tool to measure their abilities. As with any other measurement instrument, its user wants the measurement to be comparable to other measurements.

This observation, which was made for the first time probably thousands of years ago, led to the development of standardised tests. A test is considered to be standardised when the test procedures are fixed in such a way that differences among testing conditions, times, and places, do not influence the scores. Every testee will be given "the same test". As this statement does not necessarily imply that every testee faces the same set of questions, accuracy of the measurement, or test reliability, is an important issue. Also, in order to establish fairness of testing and decision making based upon the scores, it is reasonable to require that the test actually measures the concept as intended, that it is valid.

In order to fulfill the validity and reliability conditions, test development requires a systematic approach, in which assembly procedures play an important role. As argued by Downing (2006), several phases in such an approach can be distinguished:

1. Project plan. All a priori decisions need to be made explicit, such as the purpose of the test and the construct hypothesized to be measured. This will provide a framework for the efficient execution of the subsequent phases.

2. Content definition. The first question to be answered is what content domain is to be tested. Validity of inferences for test scores rely

heavily on the operationalisation of the domain, and hence decisions regarding the content definition need to be taken explicitly.

3. Test specifications. *Test specifications* refers to the complete set of test characteristics. Sometimes, the terms content specifications and test specifications are used interchangeably, but in this phase more elements in the test characteristics should be considered: At a minimum, the test specifications should also include test format, test length, target difficulty / test information, the item formats to be used and the medium in which the test will be published.

4. Item development. Only after the first three phases are performed, items should be developed, or existing items should be evaluated for their suitability.

5. Item banking. Usually, systematic storage of items, including their psychometric properties, starts already during the item development phase. Two purposes are served: facilitating test assembly and storing items for future reuse.

6. Pretesting. Two types of pretesting can be distinguished, and if one wants to adhere to a "golden standard" of test development, both types should be considered. In the first pretest, sometimes also referred to as pilot test, items are evaluated for ambiguities, inconsistencies and unwanted response behaviour. If the items are approved, a second pretest is administered with the purpose of gathering psychometric data.

7. Analysis. A hypothesis, usually one that all tested items provide scores that can be described by a selected psychometric model, is tested. Depending on the purpose of the test, the analysis might include classical analysis, item response theory analysis, analysis based on multidimensionality, and analysis of potential differential item functioning. During this phase, item parameters are estimated and information on them is added to the item bank.

8. Test assembly. Several methods are used nowadays. A common practice is selection by hand, usually after item analysis either based upon classical test theory or item response theory. Larger testing programs, however, have better access to resources like sophisticated item banking systems, opening the possibility to improve their test assembly process by means of automated test assembly.

9. Test production. Although now only the lay-out of the items is officially point of consideration, it is essential in this step that the assumption that the item parameters are still valid, is not violated. It is necessary that the testees will react in the same way to the items

as in the pretest, and the best way to guarantee this is not to modify any of the items at all in this step. Apart from the test itself, also auxiliary material such as test manuals should be produced.

10. Test administration. The test should be taken under the conditions described in the test manual. Without adequate control of these conditions, it would be difficult to interpret the test scores meaningfully.

11. Scoring. Scoring, that is, applying a key to the responses, can be simple or complex, depending on the test format. After the scoring process, passing scores may be established. For many tests, grade levels are assigned to the scores using cut scores. Methods for setting cut scores can be categorised in two groups: relative or absolute methods. Relative methods use the test scores to establish a point in the score distribution, while absolute methods are based on the expected performance of a borderline testee.

12. Reporting results. Testees have a right to an accurate, timely, and meaningful report of their test performance. For some high-stakes tests, only the pass-fail result is reported, but many testing programs report total test scores, cut scores, and some relevant subscale scores. A second type of report is the technical report, serving as a justification for the decisions made and making recommendations for future improvements.

In certain cases, some of the phases are performed iteratively. Sometimes, repetition of phases is caused by incorrect or incomplete execution of previous phases. Also, during the analysis, the conclusion might be drawn to revise or to expand the test specifications, thus providing improved guidance to the test assembly process.

Automated test assembly has great advantages over manual test assembly. First, knowing that automated methods will be employed forces a rigorous definition of test specifications early on, reducing the need to repeat some phases of the test development process. But also, more importantly, the number of possible combinations of items is too large to guarantee optimal or near-optimal solutions by hand. Thus, operational costs are reduced while improving quality control. Especially item response theory provides a good framework for automated test assembly methods. On the other hand, concepts from classical test theory have proven to be more accessible for everyone involved without an expert background. The framework of classical test theory, however, is less flexible than the item response theory framework.

This thesis discusses test assembly models using a new category of optimisation methods that has recently become popular in other fields, especially for its capability to efficiently solve a wide variety of problems: genetic algorithms. With minimal assumptions regarding the optimum or

the search space, genetic algorithms mimic evolutionary processes found in biology. A population of solutions evolves towards the optimum, with the only requirement that each solution can be evaluated. Therefore, genetic algorithms have been implemented for nonlinear integer programming problems, among others. The use of these versatile algorithms leaves the possibility to introduce and solve new classes of automated test assembly models, like nonlinear optimisation problems based on classical test theory or models for the design of large-scale testsets. Implementations for solving existing and new automated test assembly models are investigated in this thesis.

First, in this chapter several key issues are introduced: classical test theory, item response theory, automated test assembly, and genetic algorithms.

1.1 Classical Test Theory

Spearman (1904) introduced the concept of measurement error in psychology. He observed that scores contain a random element, and his attempt to correct correlations for attenuation due to measurement error marks the beginning of classical test theory (CTT). In CTT, the relation between the observed score X_j of examinee j on a given test, the true score τ_j and measurement error E_j is postulated as

$$X_j = \tau_j + E_j. \tag{1.1}$$

True score τ_j is defined as the expected observed score of an examinee over repeated administrations of the test. Furthermore, because of their very nature, the measurement errors over repeated administrations are uncorrelated, as well as true scores and measurement errors, while the expected error is zero. The model in equation (1.1) can be extended to a model for a randomly selected person, formalised axiomatically by Novick (1966). Observed score and measurement error are now random due to two different sources: random sampling of a person, and random sampling from this person's distribution. If J denotes a randomly selected person, while $F_{X_j}(\cdot)$ denotes the distribution function of the observed score for a fixed examinee and $F_\tau(\cdot)$ the distribution function of the true score for the population, then the core of CTT is formed by the equations

$$\begin{aligned} X_j &\sim F_{XJ}(x;\tau_j) \\ \tau_J &\sim F_\tau(\tau). \end{aligned}$$

Essential in CTT is the assumption that persons are selected randomly, an assumption which unfortunately is violated frequently during data gathering.

The primary entity under consideration in this model is a fixed test, usually consisting of several items. Two item properties play an important role in CTT: item difficulty and item discrimination. For dichotomously scores items, the difficulty is defined as the expected score given by a randomly selected examinee from the population of interest and is denoted by π_i. Usually, its observation in a random sample from the population is referred to as the p-value p_i. Item discrimination is operationalised as the point-biserial correlation between item score and test score, for item i denoted as ρ_{it}.

An important property of a test is its reliability, which is defined as the squared correlation between true scores and observed scores denoted by $\rho^2_{X\tau}$. An approximation of the test reliability is Cronbach's α, the internal consistency of the test or the degree to which all item scores correlate. It is a lower bound on test reliability, and hence a popular approximation to this unobservable quantity. Let k be the test length, σ^2_X the test variance and σ^2_i the item variance. Cronbach's α can be written as

$$\alpha = \frac{k}{1-k}\left(1 - \frac{\sum_{i=1}^{k}\sigma_i^2}{\sum_{i=1}^{k}\sigma_i\rho_{it}}\right) = \frac{k}{1-k}\left(1 - \frac{\sum_{i=1}^{k}\sigma_i^2}{\sigma_X^2}\right). \tag{1.2}$$

Two drawbacks of the use of CTT in item banking can be observed: First, item difficulties, item discriminations and test reliability are dependent on the population. A different population would give different item and test characteristics. For example, the expected score of an item i, π_i is higher for a more able population while item discrimination and test reliability is higher for a more heterogenous population. Therefore, random sampling is essential, while results based on data collected for one population cannot be extrapolated to other populations.

A second drawback is that, because the test is the main entity in CTT, it is cumbersome to generalise item parameters collected in the context of one test to more generally usable item parameters. For example, item discrimination r_{it} cannot be considered outside the context of the test in which the item was administered, and hence cannot be regarded as a universal item discrimination parameter. When an item score is part of the test score, the value of r_{it} tends to be inflated. The shorter the test, the greater this inflation. In an attempt to define a more stable item discrimination index, Zubin (1934) introduced the item-rest correlation r_{ir}. The inflation does not occur in item-rest correlation, but a different problem is introduced: If items are positively correlated with each other, the item-rest correlation of an item tends to rise while lengthening the test with items similar to the items currently in the test.

1.2 Item Response Theory

During World War II, the rapidly rising demand for standardised tests led to a gradual shift in working procedures. Instead of constructing a test as a single entity, a trend was started to construct items separately with the prospect of reuse. While CTT lays emphasis on tests as a whole, a new theory that focused on items and provided invariant item parameters was needed. Instrumental in the development of these invariant item parameters was the disentanglement of the item parameters and person parameters, in effect separating the items from the context of the population of examinees.

The roots of item response theory (IRT) lie perhaps in Thurstone's (1925) assumption of an age related scale underlying the items of the Binet and Simon (1905) test of the mental development of children. Lazarsfeld (1950) extended Thurstone's idea with the assumption of local independence in latent trait models, accelerating the development of what was later to be called IRT. Rasch (1960) introduced a model using the logistic function as a regression function instead of the normal ogive function, thus taking advantage of the numerical properties of the logistic function. Soon, other models were formulated along the lines of the Rasch model, for example the 2- and 3-parameter models by Birnbaum (1968) and the partial credit model for polytomous items (Masters, 1982).

A central notion in IRT is the assumption that the probability of a correct response given by an examinee can be expressed as a function of the person's latent ability. This function is dependent on the item characteristics. Several models have been developed with various functions. Widely used models for dichotomous item scores are the one-, two-, and three-parameter logistic (1PL, 2PL, 3PL) models. Let ϑ denote the ability parameter of examinee j, and let a_i, b_i and c_i denote the item discrimination, the item difficulty and the guessing parameters of item i, respectively. The probability of correctly answering item i according to the 3PL model (Birnbaum, 1968) can be expressed as

$$P_i(\vartheta) = P(X_i = 1|\vartheta) = c_i + (1 - c_i)\frac{\exp(a_i(\vartheta - b_i))}{1 + \exp(a_i(\vartheta - b_i))}. \tag{1.3}$$

where X_i is the score of examinee j to item i. Adding the assumption that $c_i = 0$ for all items results in the 2PL model (Birnbaum, 1968). Adding a further assumption that $a_i = 1$ for all items gives the 1PL model, or the Rasch model.

Measurement error in IRT is expressed in terms of the standard error of measurement, or its square, the sampling variance of ϑ. When maximum likelihood estimation is used, the reciprocal of Fisher's information $I(\vartheta)$ equals the asymptotic sampling variance of the estimator of ϑ. For a given dichotomous item i it can be shown that its Fisher's information, in IRT called the item information function and denoted by $I_i(\vartheta)$, can be written

as

$$I_i(\vartheta) = \frac{\{P_i(\vartheta)'\}^2}{P_i(\vartheta)\,(1 - P_i(\vartheta))}. \tag{1.4}$$

Assuming local independence of item responses, the test information function equals the sum of information functions of the items in the test.

The model considered in the greatest part of this dissertation is the 2PL model, although other models, for example the 3PL model or several models for polytomously scored items, can generally be used without any changes or with only minor adaptations to the optimisation problems that will be discussed.

1.3 Test Assembly

As a test is a tool to measure the candidates' abilities, it comes as no surprise that a test with a low error of measurement is favoured over a similar test with a higher error of measurement. In terms of IRT this would mean that in order to minimise the sampling variance of ϑ, the test information function must be maximised.

Early attempts using this approach were undertaken by Lawley (1943) and Lord (1952) for the normal ogive model. They investigated the dependency of the information function on the parameters of the selected items. Subsequently items were selected in order to assemble a test with a desirable shape of its information function. Birnbaum (1968) introduced the concept of a target information function and proposed a more systematic method of item selection. Lacking today's computational power, selection was done by hand, making the whole procedure rather laborious. Thus, although Birnbaum's approach was theoretically important, is was very cumbersome to use it. Meanwhile, common practice was to keep using classical indices for test assembly, a practice that Embretson (2004) observed to be still widely spread by the start of the twenty-first century.

In the mean time, Gulliksen (1950) and Ebel (1967) showed that the most profitable strategy to maximise Cronbach's α was to select items with item-test correlation as high as possible, usually items with p-value close to 0.5.

Theunissen (1985) abandoned the idea of optimising the test information function as a continuous function, and concentrated on a series of distinct ability points. His idea was to assemble a test whose TIF exceeds a target function, specified on a limited number of discrete points. Usually, but not necessarily, these points show some relation with concepts like the cut-off points, and can be selected in such a way that the TIF will exceed the interpolation of the target function at intermediate points. The objective of Theunissen's model is assembling such a test with minimal effort, for example test length. Let x_i be the decision variable associated with item i,

indicating whether it is selected ($x_i = 1$) or not ($x_i = 0$), and denote the target function, evaluated in ability point ϑ_k, by T_k. In its simplest form, Theunissen's model is formulated as

$$\begin{aligned} &\text{minimise} && \sum_i x_i && (1.5)\\ &\text{subject to:} && \sum_i I_i(\vartheta_k) x_i \geq T_k && \forall k \\ & && x_i \in \{0,1\} && \forall i. \end{aligned}$$

Theunissen's model is especially suitable for high-stakes testing, where important decisions have to be made based on the test scores. Given the acceptance of a certain maximum measurement error, the purpose of the model is to assemble a test against minimal effort.

Theunissen's approach opened the possibility to employ mathematical programming methods in test assembly and his model was soon followed by others. Van der Linden and Boekkooi-Timminga (1989) maximised the TIF evaluated in a number of points, given a shape as specified in a target for the TIF, and subject to a maximum test length as well as content specifications. Their approach concurs with the traditional methods of test assembly: While it is relatively easy to specify a required test length, the fact that the test information function is an abstract concept makes its specification rather difficult. Boekkooi-Timminga (1990) extended this model to parallel test assembly, in which no overlap between the tests was allowed. The weighted deviation model by Swanson and Stocking (1993) concentrates on heavily constrained situations that might have no feasible solutions. As calculation times might sometimes be unacceptable, Armstrong, Jones, Li and Wu (1996) developed a network flow model that could be employed in special cases.

Test assembly using CTT has shown a similar development. Adema and van der Linden (1989) formulated an optimisation problem based on a linearisation of Cronbach's α. Reverting to Gulliksen's ideas, Armstrong, Jones and Wu (1992) developed an algorithm to assemble a test parallel to a seed test, while Sanders and Verschoor (1998) published two algorithms for the assembly of weakly and strongly parallel tests.

Various software packages are available today to apply these methods in practical situations like OTD (cf. Verschoor, 1991) and ConTEST (cf. Timminga, van der Linden and Schweizer, 1996). All solution methods that are currently developed can be divided in two categories: exact methods and heuristics. An example of an exact method is branch-and-bound proposed by Land and Doig (1960), which is frequently implemented in commercially available LP-solvers. There are several types of heuristic algorithms. Greedy algorithms form a common class of heuristics. A greedy algorithm always

selects the item that adds most to the test, and will never remove an item once it is selected.

A different type of heuristics that are frequently used are local search algorithms. In local search algorithms, existing solutions are iteratively improved by making small modifications. Local search algorithms are generally slower than greedy algorithms, but usually they provide better solutions. Therefore, local search algorithms are often preferred over greedy algorithms if computational power is not an issue. Genetic algorithms are members of the class of local search algorithms.

1.4 Genetic Algorithms

Genetic algorithms (GAs) form a family of general purpose search algorithms that are successfully employed to solve optimisation problems, among many others. First studied by Holland (1968, 1973, 1975), they are iterative methods inspired by an analogy with evolution theory in biology as presented by Darwin (1859). The goals of Holland's research have been twofold: modelling the adaptive processes of natural systems, and designing artificial systems that retain the important properties of these natural systems.

A population of solutions, in terms of test assembly these are candidate tests, procreate and struggle for survival. As better tests are assumed to have higher probabilities to procreate and survive, the population evolves towards better tests over the iterations. Genetic algorithms are theoretically and empirically proven to be powerful in their search for improvement in complex systems. Furthermore, they are not limited by restrictive assumptions about the search space, like continuity, the existence of derivatives, or linearity, the most important assumption being that the quality of the solutions, or candidate tests, can be evaluated.

1.4.1 An Evolution Model

Consider a group of N individuals which form a society that is subject to an evolutionary process. Note that these individuals need not be human beings, they could be anything, living or artificial. Time, an essential element for evolution, is represented by iterations in which a new generation partly replaces the old generation. Every individual has a chance to mate with another and procreate in each iteration, and has a chance to survive to the next iteration. Procreation is regarded to be a genetic process in which two mating parents create offspring bearing properties of both of them. A "survival of the fittest" process determines which individual will die and which individual will survive. Thus, each iteration consists of three

steps in which a new generation is formed: mate selection, procreation and survival (cf. Goldberg, 1989).

It is clear that evolution is modelled as a complex process. Eiben and Smith (2003) discuss this process in terms of the five elements it contains: representation, fitness, mate selection, procreation, and survival.

Representation

Each individual is represented by a string, called a *chromosome*. The set of values that the individual string positions, the *genes*, can take is called the *alphabet*. A coding and decoding scheme provides a mapping between the physical individual and its chromosome. This chromosome contains all genetic material needed to procreate, and is instrumental in the mate selection and survival process. An example of a representation scheme is based on a binary alphabet, containing the values 0 and 1, while binary variables in the physical individuals are mapped directly onto the genes.

Fitness

Usually, but not necessarily, the processes modelling mating and survival are stochastic processes controlled by the fitness of the individuals: Ones with a high fitness have a higher chance to mate and survive than individuals with a low fitness. The fitness is modelled as a *fitness function*, and provides a search direction for the evolution process. The mechanics of this process have been clarified in Holland's schema theorem, of which an outline is presented in the next paragraphs.

Mate selection

In the first step of an iteration, the mate selection, individuals are selected as parents in order to create new individuals called offspring. With a population of N chromosomes, $N/2$ pairs of parents are selected for mating. The selection chance is defined to be a function of the fitness of the individual involved. This selection process should meet two assumptions: (1) an individual with a high fitness should have a higher chance to be selected than an individual with a low fitness and (2) an individual cannot mate with itself. It is not required that an individual can mate only once. Some individuals may mate multiple times, while others might not mate at all. In the standard GA as described by Holland, the probability to be selected is proportional to the individual's fitness. Here, $N/2$ times, two individuals are drawn without replacement, with probability $P(\mathcal{S})$ for an individual $\mathcal{S}$ in population $\mathcal{G}$:

$$P(\mathcal{S}) = \frac{f(\mathcal{S})}{\sum\limits_{\mathcal{S} \in \mathcal{G}} f(\mathcal{S})}$$

where $f(\cdot)$ is the fitness function.

Procreation

After mate selection, the two mates, also called parents, procreate. Procreation is a process in which two parents create two children, called offspring. The parents pass on information about their genetic makeup to their offspring through the *crossover* operator, while the *mutation* operator introduces new genetic properties into the population. The two newly created children are initially identical to the parents, and are subsequently subject to crossover and mutation.

Crossover

The primary purpose of the crossover can be described as the exchange of information encoded in the chromosomes of the parents, while the mutation serves to introduce new information. An often-used crossover operator is *one-point* crossover, in which a position k is chosen uniformly at random between 1 and chromosome length less one, $L-1$. This position k is referred

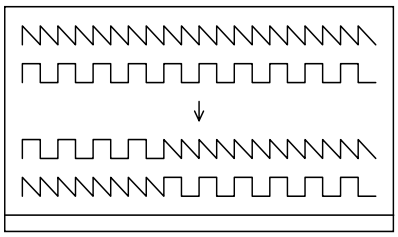

FIGURE 1.1. The One-Point Crossover

to as the crossover site. The chromosomes are cut at the crossover site, and the genes $k+1$ to L are swapped and reconnected, as shown in Figure 1.1. Other types of crossover can be defined as well, like the multiple-point crossover. Here, several crossover sites are chosen, and parts are swapped alternatingly.

A special type of crossover is *uniform* crossover, proposed by Syswerda (1989). Here, every gene is swapped between the two children with chance 0.5. Uniform crossover has advantages over one-point crossover in many applications. In mappings where neighbouring genes have no structural relation with each other, unexplored regions in the search space might be reached faster using uniform crossover, while still maintaining the necessary diversity in the population. In other mappings, several neighbouring genes might have a meaningful relation with each other, for example, they might represent one single property. Uniform crossover might disrupt these properties to the extent that exchange of information does not take place

in a systematic way, resulting in the evolution of the population to become erratic.

Mutation

While crossover provides the exchange of information, mutation introduces new information into the population. Mutation is an operator on gene level: Each gene has a probability p_μ to assume a different value in the alphabet. In case the alphabet contains only the values 0 and 1, mutation effectively flips the value of every gene with probability p_μ. Fogarty (1989) has shown that GAs are expected to perform best if $p_\mu = 1/L$.

Survival

In the last step, the population is reduced to its original size in order to form the initial population for the next iteration. This reduction process is usually a stochastic process controlled by the fitness function. Similar to mate selection, the chance of survival is a function of the individual's fitness with the assumption that an individual with a high fitness has a larger chance of survival than an individual with a low fitness. Exception to this assumption is generational survival, in which all offspring survives, and all individuals from the initial population will die.

Survival schemes can be classified according to two properties: whether they allow duplicate individuals to survive or not, and whether the best individual is guaranteed to survive or not.

If two identical individuals mate, crossover will be ineffective as both children are identical to each other as well. Only mutation will be able to modify any offspring. This is also the reason for the requirement that an individual should not be allowed to mate with itself. Allowing duplicates to survive might accelerate the search process, it might also cause the search to stop suddenly if all individuals in the population are identical. This situation is defined as *premature convergence*. Retaining a necessary level of diversity in the population is deemed necessary in order to prevent premature convergence.

Elitism is defined as the principle that the individual with the highest fitness is guaranteed to survive to the next iteration. As De Jong (1975) has pointed out, "elitism improves local search at the expense of global perspective": The search might be more efficient with an elitist survival mechanism but the risk of premature convergence is higher.

Two survival mechanisms are frequently used. In both mechanisms, duplicates are removed first. After that, in the first mechanism, a deterministic one, the N best individuals survive while in the second mechanism the probability of survival is proportional to the fitness.

Scaling

It is obvious that the relation $\forall \mathcal{S} : f(\mathcal{S}) \geq 0$ must hold in order to define all mate selection probabilities in the SGA properly. If this is not the case, either the fitness function or the relation between probabilities and fitness must be redefined. These relations between the probabilities and the fitness are usually referred to as *scaling* mechanisms. There are several scaling mechanisms, the most important being *linear scaling* and *rank based scaling* (Baker, 1985). In linear scaling, a linear transformation $P(\mathcal{S}) = a\ f(\mathcal{S}) + b$ is applied. Coefficients a and b are chosen in such a way that the probabilities for all individuals in the population are properly defined. Note that this does not imply that coefficients a and b are fixed during all iterations. As the range of fitness values in the population may vary widely during iterations, a and b may assume different values at each iteration. In rank based scaling, all individuals are sorted according to their fitness. Mate selection and survival chances $P(\mathcal{S})$ are now functions of the individual's rank within the sort order. Baker suggests to assign a user defined value MIN to the individual with the lowest fitness, $MIN + 1$ to the individual with the second lowest fitness, etc., and to define the probabilities proportional to these assigned values.

The simple genetic algorithm

One version of the genetic algorithm often used in performance analyses is called the simple genetic algorithm (SGA). Because of its simplicity, the schema theorem can easily be proven to hold for the SGA. Furthermore, the SGA can be modelled as a Markov chain, which forms the basis of more recent research on convergence. The processes implemented in the SGA are the following: a binary alphabet, mate selection chance proportional to the fitness, one-point crossover, and generational survival. Unfortunately, the SGA is rather inefficient for many optimisation problems and therefore hardly used in practical situations.

The schema theorem

The notion that the procedure of procreation and survival leads to improvement is given by Holland's schema theorem (1973, 1975). The idea behind the schema theorem is that individuals representing better solutions have certain properties in common, and that these properties are expressed in similarities between chromosomes at certain genes. The concept of a schema is an efficient way to describe similarities between individuals at certain genes. In the following outline of theorem, the SGA is assumed.

The alphabet is complemented with the "don't-care" symbol `*`, to be used to indicate indifference in gene value. Thus, 3 symbols are used: `0`, `1` and `*`, and the schema `1*0` can mean both `100` as well as `110`. It is obvious that not all schemata express the same level of similarity. For example,

the schema `011*1**` is a more specific statement about similarity than `**1****`. The *order* $o(\xi)$ of schema ξ is defined as the number of fixed positions, or positions that have either value `0` or `1`, of ξ, and the *defining length* $\delta(\xi)$ as the distance between the first and last fixed position. So the order of `1**0` is 2 and the defining length is 3. It is clear all individuals can be referred to by 3^L schemata. Each individual itself is representative of 2^L schemata. For example, `110` is representative of, amongst others, `*1*`, `1*0`, and `11*`. Thus, the whole population in a particular iteration represents at most $N2^L$ schemata.

Suppose that there are m representatives of schema ξ in population $\mathcal{G}_t$ at iteration t. This is written as $m_t(\xi)$. Let $f_t(\xi)$ be the average fitness of all representatives of ξ and let $\overline{f}_t$ be the average fitness of the whole population $\mathcal{G}_t$.

Theorem 1 *The representation of schema ξ is expected to change according to the following equation:*

$$\mathcal{E}(m_{t+1}(\xi)) \approx \{m_t(\xi)f_t(\xi)/\overline{f}_t\}\{1 - \delta(\xi)/(L-1) - o(\xi)p_\mu\}$$

Assuming a GA without crossover or mutation, thus with a reproduction scheme in which offspring is identical to the parents, ξ can be expected to gain representatives in the next generation if $f_t(\xi) > \overline{f}_t$. Mate selection depends on fitness, and in the SGA it can be expected that $m_t(\xi)f_t(\xi)/\overline{f}_t$ representatives of ξ will mate and thus will be in generation $\mathcal{G}_{t+1}$.

Selection on its own, however, creates no new material. This is done by crossover and mutation. Crossover disrupts schemata with varying probabilities, depending on their defining lengths. The smaller the defining length of a schema, the smaller the chance that the randomly selected crossover site falls within it and thus disrupts the schema. The probability that a schema will contain the crossover site is $\delta(\xi)/(L-1)$. Therefore, the upper bound on the probability of ξ losing a representative is $\delta(\xi)/(L-1)$. Note that the chance that both parents are representatives of ξ is not taken into account here and that ξ will not be disrupted in such a case even if the crossover site falls with ξ. Therefore, $1 - \delta(\xi)/(L-1)$ is an upper bound on the probability of schema ξ surviving crossover.

Furthermore, every gene has a mutation chance of p_μ, or a chance of surviving the mutation of $1 - p_\mu$. Therefore, ξ survives mutation with probability $(1-p_\mu)^{o(\xi)}$. For small p_μ this is approximated by $1 - o(\xi)p_\mu$. Summarising, the representation of ξ will change according to the following expression:

$$\mathcal{E}(m_{t+1}(\xi)) \approx \{m_t(\xi)f_t(\xi)/\overline{f}_t\}\{1 - \delta(\xi)/(L-1) - o(\xi)p_\mu\}.$$

The conclusion is that short, low-order schemata with above average fitness receive increasing representation in subsequent iterations. These short and low-order schemata are usually called *building blocks* and form the basis of

the search direction towards the optimum. It is worth bearing in mind that the schema theorem only considers the disruptive effects of crossover and mutation. Analysis of the constructive effects in creating representatives of schemata is harder since these effects largely depends on the constitution of the population. Bridges and Goldberg (1987), however, formulated a variant of the schema theorem that takes into account that crossover also can create representatives of schemata. Furthermore, under simplifying assumptions, Spears and De Jong (1999) showed that the expected number of representations of a schema that are destroyed is equal to the expected number of representatives that are created.

One can typically observe the following behaviour in representation of schema ξ. If at first the representation is very small with only individuals with high fitness, the representation can be expected to grow rapidly as representatives with high fitness have a greater chance to replace non-representatives with lower fitness than vice versa. But at the same time this causes the average fitness $\overline{f}_t$ to rise, and thus the growth rate to decrease until $f_t(\xi) = \overline{f}_t$. From that moment on, the representatives are expected to be gradually replaced by individuals that represent competing schemata with higher average fitness. The representation declines again as the population evolves into a region with higher fitnesses, and in some cases the schema might even become extinct in the population.

Summarising, the schema theorem shows that the average fitness of the population can be expected to rise over the iterations. The population then evolves towards regions in the search space with high fitness values. Therefore, genetic algorithms can be used as optimisation algorithms without the need for assumptions about certain properties of optima, such as derivatives being zero or unimodality.

1.4.2 Optimisation

The efficiency of genetic algorithms, however, depends largely on their implementation. They are processes that must be carefully balanced in order to arrive at the actual optimum. Two common threats exist: premature convergence, or getting stuck at a local optimum, and *epistasis*. Epistasis, as defined by Davidor (1991), is a condition in which some of the genes do not have any direct influence on the fitness. In that case, many chromosomes tend to have the same fitness. All these individuals have equal chance to procreate and survive, and if this is the case for the whole population, it is clear that for all schemata ξ represented in it $f_t(\xi) = \overline{f}_t$. As there are no individuals with higher or with lower fitness, representatives of a schema with above-average fitness do not replace representatives of schemata with below-average fitness, except if mutation creates an individual with a different fitness. Apparently, the search direction towards the optimum has been lost.

Three elements of optimisation

Optimisation problems, such as ATA models, contain three elements: *decision variables*, *objective function* and *restrictions*. The decision variables in ATA models indicate whether items are selected in the test or not and are the core of the model. The decision variables are mapped onto the genes of the chromosomes, while candidate tests – combinations of items – are the individuals in the genetic population represented by these chromosomes. The mapping is straightforward: Assuming a binary alphabet, every variable takes its own place in the chromosome string.

The fitness function and objective function are closely related: During the GA the fitness function is maximised. The assumption that the maximum of the fitness function coincides with the optimum of the problem, determined by the objective function and restrictions, must be made. In case of unrestricted maximisation problems, it usually suffices that the objective function and fitness function are the same. For minimisation problems, a simple transformation can be devised in such a way that the minimum of the objective function coincides with the maximum of the fitness function. Only two weak assumptions regarding the fitness function, and hence regarding the objective function, are needed: (1) the fitness function can be evaluated for all individuals that can be generated through the crossover and mutation operators, and (2) a scaling scheme can be devised to define the probabilities for mate selection and survival properly. Linearity, for example, is not required, and the absence of such assumptions reveals the true strength of GAs as optimisation algorithms.

Few problems, however, are unrestricted. In most problems, restrictions divide the entire search space into two regions: the feasible space and infeasible space. Ultimately, individuals from the infeasible space must be rejected. GAs should be designed in such a way that either infeasible individuals are not created at all or that they are rejected sooner or later. There are a few options to design such algorithms:

- Rejection of infeasible individuals. This approach is simple, and works reasonably well in cases where the feasible search space is convex and constitutes a large part of the whole search space, that is, in problems with just a few restrictions. In more complex problems, too many individuals might be generated that appear to be infeasible, and in such a case only an occasional individual will be accepted.

- Use of special mappings and operators to guarantee feasible solutions. For some problems, it could be a complicated task to design a special representation such that every chromosome maps onto a feasible solution and vice versa. It would involve a tailor-made implementation whereby each individual restriction has to be taken into account and frequently special crossover and mutation operators would be needed as well. Achterkamp (1993) took this approach in test as-

sembly. She noticed that in the standard mapping, fixed test length restrictions are frequently violated by crossover and mutation. In order to overcome this phenomenon she devised a genetic algorithm in which crossover and mutation only produced tests of the required length. Thus, fixed length restrictions were made redundant. She defined a mapping in which the genes were mapped directly onto the items in the test: the first gene indicating the first item in the test, etc. This representation does not use a binary alphabet but one that contains all item numbers in the pool. This approach works very well for simple test assembly models but additional restrictions are complicated to incorporate.

- Use of a repair algorithm as presented by Orvosh and Davis (1993). This algorithm constructs a feasible solution given any individual. But it has the drawback of sometimes repairing many individuals into the same feasible solution. Thus, since different chromosomes are mapped onto the same solution, these are given equal fitness function value, resulting in epistasis.

- Use of a penalty function for infeasible solutions. Allow any offspring to be generated, but use a fitness function based on a relaxation of the optimisation model. For objective function $c(x)$ and penalty function $g(x)$, fitness function $f(x)$ can be defined as $f(x) = c(x) - g(x)$. The major question is how the penalty function should be designed. If the penalty is too low, the optimum and maximum of the fitness function do not coincide, and the maximum of the fitness function is located in the infeasible space. Intuitively, the penalty should be kept as low as possible, just above the limit at which the optimum of the model and the optimum of the fitness coincide. Too high a penalty might cause premature convergence, as valuable information contained in the genes of the infeasible solutions gets extinct too quickly.

  In order for the evolution process to evolve towards the feasible space, a necessary condition can be given. For all infeasible solutions having directly neighbouring feasible solutions, there must be at least one such neighbour with a higher fitness. Should this not be the case the process might evolve further away from the feasible region. The penalties, however, should not be chosen too high. If infeasible solutions are eliminated too quickly, diversity in genetic information might be lost, resulting in premature convergence. Le Riche et al. (1995) confirm the effectiveness of this so-called *minimal penalty rule*. The use of penalty functions attracts another problem: Find a penalty function that is robust under a wide variety of optimisation problems as they occur in practical situations. This is why it is difficult to implement this rule. A hypothesis formulated by Richardson et al. (1989) may provide some help: "Penalties which are functions of the

distance from feasibility are better performers than those which are merely functions of the number of violated constraints."

For violated restriction $\sum_i a_i x_i \leq b$, the penalty function can be defined as $g(x) = \lambda\left(\sum_i a_i x_i - b\right)$, that is, directly proportional to the distance to the feasible space. Coefficient λ is called the penalty multiplier. In many practical cases, a simulation study must be used to determine appropriate values for the penalty multipliers. Furthermore, Siedlecki and Sklanski (1989) show that "the genetic algorithm with a variable penalty coefficient outperforms the fixed penalty factor algorithm." Such a genetic algorithm has a penalty updating scheme with the following outline: Consider τ consecutive iterations, each with their best individual. Thus, τ individuals are considered, which are possibly identical. If all these best individuals appear to be infeasible, raise the penalty multiplier by multiplying it by $1+\varepsilon$: It seems to be plausible that the optimum for the fitness function is located in the infeasible region, which is a sign that the penalties are too low. If all these solutions are feasible, lower the multiplier λ by multiplying it by $1-\delta$, since the penalty might be too high to avoid the risk of premature convergence. In other cases, that is, if some individuals are feasible and others are infeasible, leave the penalty multiplier unchanged.

1.4.3 Stopping Criteria

One element of GAs as optimisation methods has not been considered yet: we may assume that the population evolves towards the optimum, but does it actually reach it? This question was answered by Goldberg and Segrest (1987), and Eiben, Aarts and van Hee (1991) using Markov chain analysis. Their approach was to consider the SGA as a Markov chain. They analysed steady states, and the conditions under which these contained an optimal solution. Thierens and Goldberg (1994) showed that the convergence in the SA can be estimated for sufficiently large populations. The chance that the population contains the optimal solution is asymptotically given by $P_{opt}(t) = 1 - \left\{1 - (1 - 0.5e^{-t/L})^L\right\}^N$. Suzuki (1993) derived a lower bound of the probability of the fittest individual being included in the steady state solutions. From this lower bound, an upper bound on the number of iterations required to find the optimal solution with certainty δ can be calculated. Aytug and Koehler (1996) showed that this upper bound is $\log(1-\delta)/\log(1-{p_\mu}^{LN})$. Although these analyses are theoretically very important, they provide us with no practical stopping rule as they cannot be generalised to most of the GAs that are implemented for practical problems.

Alternatively, the GA can be stopped when it can be shown that the best individual is within a small bandwidth from the optimum. A detection mechanism, for example, one comparable to Lagrangian relaxation, is

absent. Therefore, other criteria must be used. The simplest rule is to stop after a fixed number of iterations in the hope that a solution within the required bandwidth has been found.

The schema theorem, however, provides some insight in the average behaviour of a GA. It states that as time passes, building blocks with above average fitness will increase in representation. It does not tell us anything about the individual building blocks in the next generation. The GAs' heuristic nature suggests that it might not be safe to use stopping rules like "stop when no significant improvement has been found in n iterations", since the process may be temporarily stuck in a local maximum. Therefore, early research concentrated on criteria based on the extent of convergence within the population. Recognising convergence on the level of individual genes is rather straightforward. By its very definition, convergence manifests itself in a large number of converged genes. Convergence of a gene is therefore be defined as the largest percentage of the population having the same value. *Bias* is defined as the average convergence across genes. This definition thus introduces a measure of the overall extent of convergence. A rule can be devised to stop when the bias has reached a value high enough to assume that the optimum has been found.

A third stopping rule can be devised by comparing the *online performance* with the *offline performance*. Online performance is defined as the average fitness of all individuals in a generation, and offline performance as the average fitness of a small number of the best individuals. If the quotient is close to 1, convergence may be assumed.

1.5 Overview

The focus in this thesis is on the implementation of genetic algorithms for automated test assembly. Traditional models are based either on linear programming methods, or on heuristics. Both approaches have their limitations.

As a rule, heuristics provide a "good" solution very quickly but are dedicated to specific models. Adapting heuristics to slightly different models can be very difficult or sometimes downright impossible. Fortunately, the improved access to computer power in recent years has diminished the need for this kind of methods.

Therefore, automated test assembly models based on linear programming methods are widely used nowadays. But these models have limitations as well: In some practical situations, nonlinear models are required that are impossible to solve by standard linear programming techniques. An example of such a situation is the development of parallel test forms in which a limited overlap is allowed. Although these models can be formulated in a linear way, this alternative formulation would introduce so many dummy

variables and restrictions that sometimes no optimum will be found within reasonable time. Models based on methods like genetic algorithms may be able to provide a solution.

In Chapter 2, a GA for a basic ATA model is presented. This model can be solved efficiently by LP methods, which thus provides a possibility for evaluating the efficiency of genetic algorithms. As the proposed GA is a heuristic as well, an implementation must be designed that solves the models involved best. A simulation study was performed to find the variant that solves ATA models most efficiently.

In Chapter 3, a test assembly model based on classical test theory is proposed. A key issue in CTT is Cronbach's α, a nonlinear entity. Adema and van der Linden (1989) proposed an efficient linearised model, and this model is compared to a model optimising α in a more direct way using an approximation of α. Using this approximation, a new model can be defined to assemble a test that exceeds a target α against minimal effort.

The subject of Chapter 4 is the assembly of testsets: a group of related test forms that may have different test specifications, and a restricted overlap. The basic genetic algorithm introduced in Chapter 2 has a serious drawback: Parallel test assembly is an open invitation for epistasis. An alternative implementation introducing the concept of seasons is proposed, which diminished the epistatic nature of the algorithm.

Chapter 5 explores the limits of automated test assembly. Especially in large scale on-going testing programs, an even use of items is crucial. When not restricted, ATA models select the best items, while security issues prevent reuse of these items in the short term. The quality of the produced tests will decrease, as was shown for two large scale testing programs: the State Examinations for Dutch as a Second Language in the Netherlands, and the Unified State Examinations in the Russian Federation. Several strategies to harness the test assembly were evaluated on their ability to prevent depletion of the item banks.

2

IRT Test Assembly Using Genetic Algorithms

2.1 Introduction

Test assembly has been an important issue in the field of educational testing for many years. Birnbaum (1968) introduced the concept of a target test information function, and proposed to select items in such a way that the target is approximated as closely as possible. His idea caused a gradual shift from using classical test theory towards using item response theory for test assembly. Because of lack of computational power, item selection was initially done by hand, which made Birnbaum's approach theoretically attractive but not very useful in practice. Feuerman and Weiss (1973) were the first ones to use mathematical programming methods in a psychometric context. However, it took even then more than a decade before Theunissen (1985) brought Birnbaum's ideas into practice. Computerised test assembly methods using item selection from an item bank have since then become increasingly popular. Van der Linden (2005) gives a comprehensive overview of the optimisation models that have been developed and the different techniques to solve these models.

All test assembly models have several elements in common: *decision variables*, *objective function*, and *restrictions*. The decision variables indicate whether or not items are selected in a test and thus form the core of the model. The objective function expresses a property of the test that the test assembler has defined as desirable. Examples of objectives are the number of items in the test or the test information function. The first objective is minimised, while the latter is maximised. The restrictions are the condi-

tions that the test has to comply with. Examples of these restrictions are the test length, content specification, or interitem relations such as enemy sets.

This paper discusses a new class of optimisation methods that has become popular in recent years for its capability to efficiently solve nonlinear integer programming problems: genetic algorithms. Mimicking evolutionary processes in biology, solutions mate and compete for survival. Through these processes, the population evolves towards the optimum. Unlike most other methods, they assume no information on the structure of the problem to be solved and are therefore amongst the most versatile optimisation methods available.

In this paper, two genetic algorithm for two widely used test assembly models are presented. Solutions found with these genetic algorithms are compared with solutions obtained from other algorithms.

2.2 Models for Test Assembly

2.2.1 IMIN Model

Theunissen (1985) proposed a test assembly model based on IRT. Given the purpose of the test, for example, a decision based upon an examination that has to be made, a measurement precision is required. This precision is usually formulated as a maximum standard error of measurement at certain ability levels. Instrumental to this formulation is the test information function (TIF). The purpose of the model is to assemble a test that exceeds a target for the TIF at a series of prespecified ability points, against minimal "effort". Examples of this effort are test length, administration time, or production costs. The function defining this effort forms the objective function, that will be minimised. The TIF target, defined at a series of ability points, forms the basis for the restrictions in the model.

The IMIN model presented here is a slight reformulation of Theunissen's model. The IMIN model expresses the wish of the test assembler to assemble a test that uses minimal resources as stated in (2.1), given restrictions in (2.2) that force the TIF to exceed the target, and subject to restrictions in (2.3) regarding other resources than the one used in the objective function, to classification restrictions in (2.4), and to restrictions on item level in (2.5), as proposed by Theunissen (1996):

$$\text{minimise} \quad \sum_i q_{i0} x_i \tag{2.1}$$

$$\text{subject to:} \quad \sum_i I_{ik} x_i \geq T_k \qquad \forall k \tag{2.2}$$

$$\sum_i q_{in} x_i \geq Q_n \qquad \forall n \tag{2.3}$$

$$C_m^\ell \leq \sum_i c_{im} x_i \leq C_m^u \qquad \forall m \tag{2.4}$$

$$p_r(x) = 1 \qquad \forall r \tag{2.5}$$

$$x_i = \begin{cases} 1, \text{ item } i \text{ in the test} \\ 0, \text{ else} \end{cases} \qquad \forall i.$$

Variable x_i denotes the decision variable indicating whether item i is selected or not. I_{ik} is the item information at ability point ϑ_k, while T_k is the target information at this point. Coefficient q_{in} is the resource parameter of the item indicating how much of resource n is needed if the item is in the test, c_{im} is a classification parameter having value 1 if item i belongs to category m and 0 otherwise. Q_n, C_m^ℓ and C_m^u are the resources required for the test, and number of items representing the classification categories, respectively. Equation (2.5) expresses relations on item level, the interitem relations. Common examples of interitem relations are enemy sets and friend sets. More general relations can be formulated as well, using Boolean operators $\vee$, $\wedge$ and $\neg$. According to De Jong and Spears (1989), these can be transformed into restrictions based on differential payoff functions, denoted by $p_r(x)$. In Appendix A, an example of the transformation from interitem relations to restrictions based upon differential payoff functions is elaborated.

The IMIN model is suitable for situations in which it is important to restrict the standard error of measurement to a certain maximum, for example, in high-stakes testing where important decisions have to be made, based on the outcome of the test. Given the acceptance of a maximum measurement error, the purpose of the model is to assemble a test with minimal use of resources of a specific kind. Thus, the IMIN model expresses the wish of the test assembler to assemble a "minimum" test with respect to a resource function, for example test length, subject to a TIF target, restrictions limiting other available resources, classification restrictions and interitem restrictions.

2.2.2 IMAX Model

Van der Linden and Boekkooi-Timminga (1989) described a maximin model suitable for situations that are more resource-driven. The model reflects the

wish to assemble the best test, that is, one with minimal error of measurement, given limited resources, such as test length or administration time, given a target shape of the TIF, and subject to classification restrictions and interitem relations, similar to the restrictions in the IMIN model. Here, the objective is defined as the minimum ratio between the TIF and its target across the ability points, while the objective should be maximised.

The IMAX model presented in (2.6) – (2.10) is a reformulation of the maximin model in such a way that the similarities and differences with the IMIN model become apparent:

$$\begin{aligned}
&\text{maximise} && y && && (2.6)\\
&\text{subject to:} && y \leq \frac{\sum_i I_{ik} x_i}{T_k} && \forall k && (2.7)\\
& && \sum_i q_{in} x_i \leq Q_n && \forall n && (2.8)\\
& && C_m^{\ell} \leq \sum_i c_{im} x_i \leq C_m^{u} && \forall m && (2.9)\\
& && p_r(x) = 1 && \forall r && (2.10)\\
& && x_i = \begin{cases} 1, & \text{item } i \text{ in the test} \\ 0, & \text{else} \end{cases} && \forall i. &&
\end{aligned}$$

The purpose of the model is to maximise the information in the test at that ability point for which the ratio between reached and target is minimal. Thus, the total information of the test is maximised while adhering to the target TIF shape as much as possible.

2.3 Genetic Algorithms

Eiben and Smith (2003) discuss five important elements of genetic algorithms: representation, mate selection, recombination, survival, and fitness. In the case of ATA models, solutions are candidate tests, and these tests are represented by chromosomes that are strings with genes assuming value 0 or 1. The gene at position i is directly related to decision variable x_i. In iteration t, the population of tests evolves into the population of iteration $t+1$ through three steps: mate selection, recombination, and survival. With population size N, $N/2$ pairs of parents are selected that create two children. The probability that a test is selected as a parent is assumed to be a monotonically increasing function of the test's fitness. A usual choice is to assume that the probabilities are proportional to the fitnesses. Initially, the two newly created child-tests are identical to their parents, and a recombination process consisting of crossover and mutation modifies these

children. Crossover can be seen as the exchange of items between the two children. With one-point crossover, a cut position is selected randomly, after which the items in one part of the pool are exchanged. With uniform crossover, a decision to exchange or not is randomly taken for each item in the pool. Mutation can be seen as the introduction or removal of items: Each gene has a small chance p_μ to flip its value, or each item in the pool has a chance either to be added to the test or to be removed from the test. After creation of N children, a survival mechanism reduces the population to its original size N. It is assumed that the chance of survival is a function of the fitness, similar to the chance of being selected as a parent.

2.3.1 Fitness

Since Holland (1975) has shown that for certain assumptions regarding the genetic algorithm, the population evolves towards the maximum of the fitness function, the objective and fitness function are closely related. For unrestricted problems, the fitness function may be chosen identical to the objective. ATA models, however, are always restricted. A usual choice to process restrictions is defining a penalty function for all infeasible solutions. Richardson, Palmer, Liepins, and Hilliard (1989) have shown that penalties that are functions of the distance to the feasible region are expected to perform best. Therefore, for violated restriction $\sum_i a_i x_i \leq b$, the penalty function can be defined as $g(x) = \lambda \left(\sum_i a_i x_i - b\right)$, while for objective $c(x)$ the fitness function is defined as $f(x) = c(x) - g(x)$. Coefficient λ is called the penalty multiplier. The use of penalty functions is versatile in the sense that many restrictions can be added without the need to redesign the implementation.

The proposed fitness function for the IMAX model is composed of the minimal ratio between the TIF and its target, and of penalty terms for each violated restriction:

$$f(x) = \min_k \left\{ \frac{\sum_i I_{ik} x_i}{T_k} \right\} - g(x) \tag{2.11}$$

$$\begin{aligned} g(x) = {} & \lambda \sum_n h\left(\sum\nolimits_i q_{in} x_i - Q_n \right) \\ & + \mu \sum_m h\left(C_m^{\ell} - \sum\nolimits_i c_{im} x_i \right) + \mu \sum_m h\left(\sum\nolimits_i c_{im} x_i - C_m^{u} \right) \\ & + \varphi \sum_r 1 - p_r(x) \end{aligned} \tag{2.12}$$

$$h(u) = \begin{cases} u, & u > 0 \\ 0, & u \leq 0 \end{cases} .$$

The values of λ, μ and φ, however, should be chosen carefully. Le Riche, Knopf-Lenoir, and Haftka (1995) have shown that the best rule is to assign values as low as possible, just above the values for which the maximum of the fitness is located in the infeasible region, and called this rule the minimum penalty rule. A high penalty might cause a situation in which infeasible solutions are removed very quickly. But invaluable genetic information, combinations of items that might be crucial to optimal and near-optimal tests, might be lost before this information is passed on to feasible offspring. Premature convergence, a situation in which all diversity in the population has been lost, might be the result.

The range of values that are too low, however, is unknown and may vary widely from restriction to restriction, as is illustrated in the example given in Appendix B. Therefore, Siedlecki and Sklanski (1989) suggest to use a variable penalty scheme with the following outline: Consider τ consecutive iterations, each with the test having the highest fitness. Thus, τ, possibly identical, tests are considered. If all these tests appear to be infeasible, raise the penalty multiplier by multiplying it by $1+\varepsilon$, since it seems to be plausible that the current optimum of the fitness function is infeasible. If all these tests are feasible, lower the multiplier λ by multiplying it by $1-\delta$, since a penalty too high might incur premature convergence. In all other cases, leave the penalty multiplier unchanged.

Note that the fitness function in (2.11) yields negative values when the penalty is larger than y. If a proportional probability scheme is used for mate selection or survival, the fitness function should be altered. Therefore, the fitness function is redefined as

$$f(x) = \frac{\min\limits_k \left\{ \frac{\sum_i I_{ik} x_i}{T_k} \right\}}{1 + g(x)} \tag{2.13}$$

in order to prevent the occurrence of negative values. An alternative option is to assume a different probability scheme. Two such schemes are widely used, linear scaling and rank based scaling. In linear scaling, the probabilities are proportional to a linear transformation of the fitness. In rank based scaling, all tests are sorted according to their fitness, after which each test is assigned a rank number. The test with the lowest fitness is assigned a user defined value MIN, the second-lowest $MIN+1$, etc. The probabilities are then defined to be proportional to these rank numbers.

The ratio between the chances of selecting the best and the worst test can be regarded as a kind of pressure on the population: The greater this pressure, the less the opportunity for tests with low fitnesses to pass on their genes to offspring. A pressure too high might lead to premature convergence, while a pressure too low might slow down the evolutionary process. With proportional selection, this pressure may vary widely over the iterations. The advantage of the scaling mechanisms is that the pressure can be controlled precisely.

2.4 Simulation Studies

Simulations consisting of three phases were conducted. In the first phase, some preliminary questions had to be answered, such as choice of the crossover operator, mate selection, and survival strategy. In the second phase, a number of variable penalty schemes were evaluated. In the third phase, a stopping rule was devised and a comparison was made with a linear programming method in order to test the GA for its usefulness.

Two different problems of the IMAX model were used, which both were based upon the same item pool consisting of 500 items with parameters simulated according to the two-parameter model with $\log(\alpha) \sim N(0, 0.4)$ and $\beta \sim N(0, 1)$. All items were coded with respect to two different classifications. The first classification consisted of four categories labelled as 101, 102, 103 and 104. These categories were filled with 250, 125, 85 and 40 items, respectively. The second classification contained ten categories labelled as 201, ..., 210, which contained equal numbers of items.

The simplest test assembly problem, Problem 1, had four restrictions related to the target for the TIF: $T_{\vartheta=-1.5} = 4, T_{\vartheta=-0.5} = 8, T_{\vartheta=0.5} = 8, T_{\vartheta=1.5} = 4$ and one resource restriction: $\sum_i x_i \leq 40$. No content restrictions based on the classification structure described above were used. Problem 2 was an extension of Problem 1, in which 40 content restrictions were added. From each combination of two categories, the first from the

range 101, ..., 104, and the second from the range 201, ..., 210, exactly one item was required in the test. In addition, Problem 2 had two interitem relations that exclude combinations of items that were observed to be included earlier in the vast majority of good tests constructed without these relations. Problem 2 had 47 restrictions in total.

2.4.1 First-Phase Simulations

In the first phase, some preliminary questions were answered:

- Is there a difference in performance between the variable and the fixed penalty schemes?
- Which crossover operator should be used: one-point or uniform crossover?
- Which strategy for mate selection should be used: proportional-to-fitness, linear scaling, or rank based scaling?
- Which strategy for survival should be used: a deterministic one allowing the best N individuals to survive or selection proportional to the fitness?

All results mentioned below are based on 400 tests assembled for each condition. The standard case is a variable penalty scheme with $\tau = 20$, $\delta = 0.11$ and $\varepsilon = 0.08$, together with uniform crossover, mate selection proportional to fitness, and deterministic survival. The current best solutions at certain iterations, and the iteration in which this solution was found, were reported. Sometimes, a good solution was found after which a long period of no improvement followed. The algorithm was stopped at 8000 iterations, after which the feasible solution with the highest fitness as well as the iteration at which it was found were reported. This was also the procedure for Problem 2, except that the process was stopped after 32000 iterations. In all cases, an initial population of 100 tests was randomly generated with each gene having a chance of $\frac{40}{500}$ of assuming the value 1.

Elitism, the principle that the best solution is guaranteed to survive to the next iteration, was used throughout the simulations. For the variable penalty schemes, the elitist strategy should be commented upon. As the variable penalty scheme modifies the penalty function at every τ iterations, the best solution might not remain the best during multiplier update. In that case, the elite position shifts towards a different solution. But when the best solution was feasible at the update, a copy of it was preserved outside the actual algorithm and reported as the optimal solution if necessary. In Table 2.1, the progress of the algorithm is shown for the base line settings for Problems 1 and 2.

TABLE 2.1. Best Fitness and Iteration when Found during Optimisation

	Fitness		Found at Iteration	
Iteration	M	SD	M	SD
		Problem 1		
250	1.969	0.169	4.3	2.0
500	4.563	0.029	440	51
1000	4.583	0.025	760	180
2000	4.592	0.022	1200	460
4000	4.597	0.019	2000	1000
8000	4.601	0.017	3300	2200
		Problem 2		
250	3.636	0.078	240	6.9
500	3.712	0.078	400	68
1000	3.741	0.070	680	220
2000	3.758	0.066	1100	500
4000	3.771	0.063	1900	1100
8000	3.781	0.058	3300	2300
16000	3.792	0.055	6500	4800
32000	3.805	0.052	13200	10000

A fitness value of 1.969 in Table 2.1 means that the average fitness of the best feasible solution at iteration 250 was 1.969 for Problem 1. Since penalty function $g(x)$ in (2.12) had value 0 for all feasible solutions, the fitness function value $f(x)$ was equal to y in (2.6). Thus, at iteration 250, the TIF of the current best test for Problem 1 exceeded, on average, $I_{\vartheta=-1.5} = 7.876, I_{\vartheta=-0.5} = 15.752, I_{\vartheta=0.5} = 15.752, I_{\vartheta=1.5} = 7.876$. Note that all replications found feasible solutions for both problems before iteration 250.

It can be seen that in iteration 250, the current best test was found at approximately iteration 4, and thus that until iteration 250, no feasible test with higher information was found. During this period, the population evolved into the infeasible region. After some time, the population evolved back into the feasible region again, but now into a part of it with much higher fitnesses. Feasible solutions with high objective function values were found again.

The relatively large standard deviation of the iterations at which the best solutions were found is an indication of the sometimes long intervals between improvements.

In the following sections, only the final results are given. The fitness during the iterations is not shown, but a graphical display of the best solutions and the iterations at which they were found is given instead for some conditions.

Penalty

Four conditions were investigated: one variable penalty scheme with $\tau = 20$, $\delta = 0.11$, and $\varepsilon = 0.08$ and three fixed penalty schemes: low with $\lambda = 0.004, \mu = 0.006, \varphi = 0.01$; medium with $\lambda = 0.02, \mu = 0.03, \varphi = 0.05$; high with $\lambda = 0.1, \mu = 0.15, \varphi = 0.25$. The rationale behind these schemes was the assumption that the optimal values for a fixed penalty scheme should follow the minimal penalty rule, while the variable penalty scheme follows the minimal penalty rule automatically. The penalty multiplier values converge to the optimal penalty multiplier values. At the end of the simulations for the variable penalty scheme, the average penalty multiplier for Problem 1 was $\lambda = 0.02$. For Problem 2, the multipliers were $\lambda = 0.02$, $\mu = 0.03$, and $\varphi = 0.05$. In real-world situations, however, these optimal values are not known in advance and a scheme with somewhat different multipliers would be chosen. Therefore, the low and high schemes were considered as well.

TABLE 2.2. Best Fitness and Iteration when Found for Different Penalty Schemes

	Fitness		Found at Iteration	
Penalty	M	SD	M	SD
		Problem 1		
Variable	4.601	0.017	3300	2200
Low	1.866	0.142	2.9	1.4
Medium	4.583	0.132	3400	2200
High	4.550	0.049	5300	2000
		Problem 2		
Variable	3.805	0.052	13200	10000
Low	–	–	–	–
Medium	3.721	0.079	19000	9000
High	3.501	0.129	23800	7000

Table 2.2 presents the final results for the penalty schemes. The multipliers of the low penalty scheme were clearly too low since the solutions that were presented as optimal appeared to be infeasible. The optima for the fitness and objective did not coincide. For Problem 1, feasible solutions were found only in the very first iterations and reported here. After those first iterations, the population moved well into the infeasible region towards the point where the best fitness was found. The low scheme did not find any feasible solution at all for Problem 2.

On the other hand, the medium scheme seemed to be a proper strategy for a fixed penalty scheme. It should be noted, however, that in 8 cases the medium penalty scheme failed to find any feasible solution for Problem 2 within 32000 iterations. All simulations produced a feasible solution for

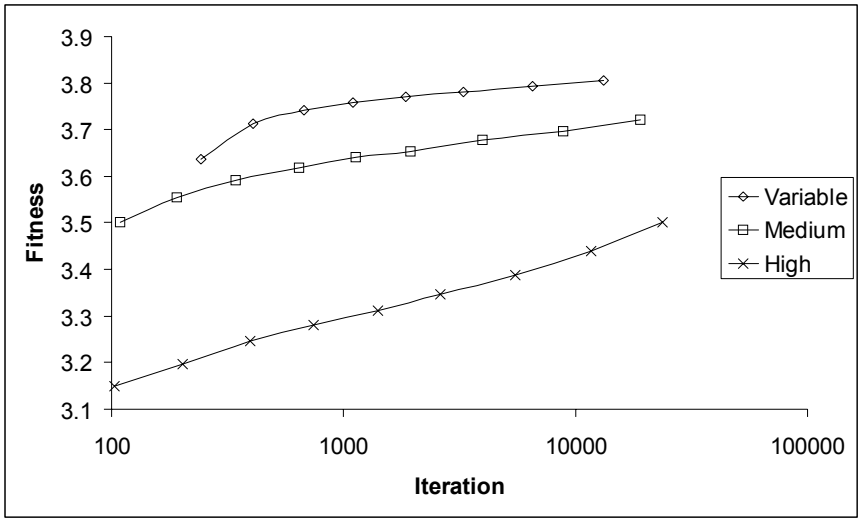

FIGURE 2.1. Increase of Fitness for Different Penalty Schemes (Problem 2)

the high scheme but the average fitness was lower. So the medium scheme might have had penalty multiplier values that were slightly too low.

Although the results suggest that the differences between the variable and medium schemes were not very great, the superiority of the variable penalty scheme becomes clear in Figure 2.1, where the current best fitness is shown as a function of the iteration at which it was found. Combining the data in Table 2.2 and in Table 2.1 shows that, even while the medium penalty scheme was the best fixed penalty scheme in this study, the fitness found at approximately 19000 iterations was comparable with that found at approximately 500 iterations with the variable penalty scheme. For the high penalty scheme, the performance was even considerably worse. For the variable penalty scheme, the solutions found within an average of 240 iterations were better than those found at 24000 iterations for the high scheme.

Crossover

Two crossover operators were considered: uniform crossover and one-point crossover. Since neighbouring items in the bank have in general no relationship with each other, neighbouring genes in the chromosomes do not form building blocks. Thus, it might be assumed that uniform crossover would propagate at least as fast as the one-point crossover while maintaining the necessary variation in the population.

It can be inferred from Table 2.3 that the differences in fitness were relatively small. However, since the optimisation process was rather slow,

TABLE 2.3. Best Fitness and Iteration when Found for Different Crossover Operators

	Fitness		Found at Iteration	
Crossover	M	SD	M	SD
		Problem 1		
Uniform	4.601	0.017	3300	2200
One-point	4.591	0.022	5000	2100
		Problem 2		
Uniform	3.805	0.052	13200	10000
One-point	3.772	0.071	16900	9600

a large difference in number of iterations could be observed. This can also be seen in Figure 2.2 or by combining the data in Table 2.3 and in Table 2.1. One-point crossover reached a fitness of 3.772 in an average of 16900 iterations while uniform crossover reached the same result in just over 1900 iterations. Therefore, uniform crossover was chosen.

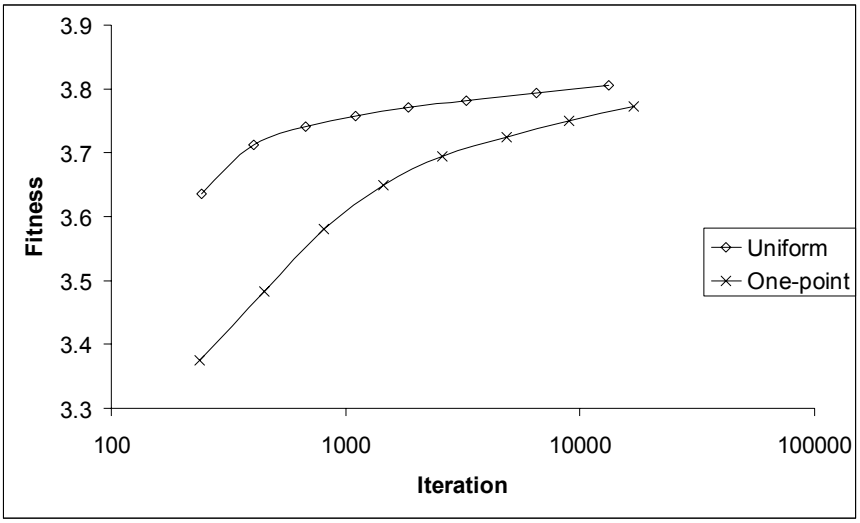

FIGURE 2.2. Increase of Fitness for Different Crossover Operators (Problem 2)

Scaling

The base line condition we used involved mate selection proportional to fitness. This condition was compared with two variants of the scaling techniques mentioned earlier: an aggressive variant in which the worst test was not selected at all and a more conservative variant where the worst test did

have a chance to be selected. So five different conditions were investigated in total:

1. Selection proportional to fitness.
2. Linear scaling (Linear 1). In each iteration, the transformation coefficients were determined such that the worst test did not procreate.
3. Linear scaling (Linear 2). The transformation coefficients were determined such that the worst test had a chance half as high to be selected as the best test.
4. Rank based scaling (Rank 1). The user defined constant MIN had the value 0. This meant that the worst test was not selected.
5. Rank based scaling (Rank 2). MIN had value 100, so that the chance of being selected for the worst test was half as high as for the best test.

TABLE 2.4. Best Fitness and Iteration when Found for Different Scaling Types

	Fitness		Found at Iteration	
Scaling	M	SD	M	SD
		Problem 1		
Proportional	4.601	0.017	3300	2200
Linear 1	4.594	0.023	3900	2200
Linear 2	4.597	0.018	3400	2200
Rank 1	4.595	0.022	3700	2200
Rank 2	4.600	0.014	3400	2200
		Problem 2		
Proportional	3.805	0.052	13200	10000
Linear 1	3.784	0.064	15600	9900
Linear 2	3.805	0.052	14500	9900
Rank 1	3.791	0.060	14700	9600
Rank 2	3.805	0.052	15000	10100

As reported in Table 2.4, differences in performance between scaling techniques were minimal. Especially the difference between the conservative scaling variants and proportional selection appeared to be negligible. This is also illustrated in Figure 2.3 for Problem 2. It should be noted that the high selection pressure provided by the large differentiation between the good and bad individuals was counterproductive. Mate selection should be more explorative so that the genetic material of individuals of low fitness has a chance of being recombined into superior offspring. The conclusion is

that the choice between the more explorative alternatives did not seem to have a large influence on the optimisation process. Therefore the simplest technique, proportional selection, was chosen.

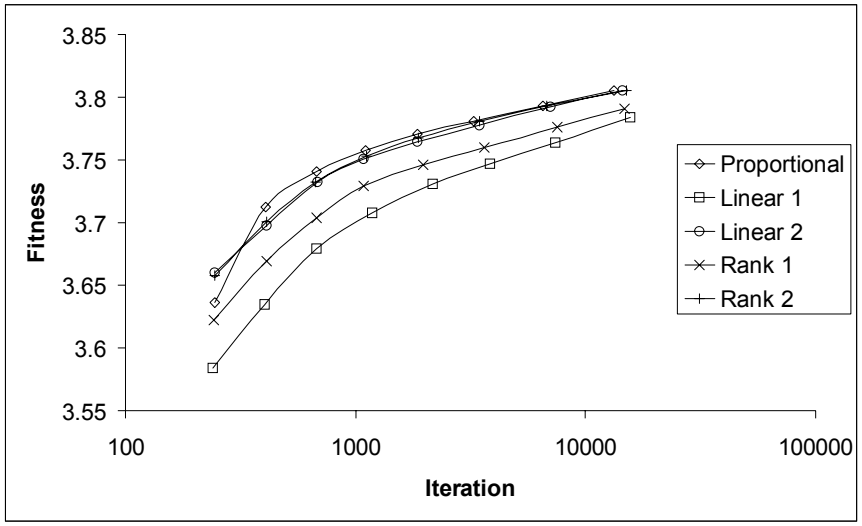

FIGURE 2.3. Increase of Fitness for Different Scaling Techniques (Problem 2)

Survival

The first issue regarding survival is whether identical tests should be allowed to survive or not. Allowing duplicates to survive to the next iteration might result in the situation that a highly fit test generates many duplicates, even to the extent that the whole population consists of duplicates only, and no search direction can be found. Creation of duplicate chromosomes was therefore not allowed in any of the strategies, thus retaining a necessary level of variety in the population.

All survival strategies started with the removal of newly created duplicates. Thereafter, two conditions were possible: a deterministic scheme with survival of the N best solutions and a stochastic one that used a linear scaling technique where the chance of survival for the best test was twice as high as for the worst test. The rationale behind this scaling procedure was that after only a few iterations the population might have converged to such an extent that the ratio between the highest and lowest fitness would be close to 1. In that case, all individuals would have about equal chance to survive and the selection pressure would disappear. The linear scaling magnified the small differences in fitness and maintained the pressure.

As can be inferred from Table 2.5 and Figure 2.4, the stochastic strategy was clearly less efficient than the deterministic strategy. Therefore, only the deterministic strategy was used in the remainder of this study.

TABLE 2.5. Best Fitness and Iteration when Found for Different Survival Schemes

	Fitness		Found at Iteration	
Survival	M	SD	M	SD
		Problem 1		
Deterministic	4.601	0.017	3300	2200
Stochastic	4.473	0.048	6300	1500
		Problem 2		
Deterministic	3.805	0.052	13200	10000
Stochastic	3.612	0.103	24300	7300

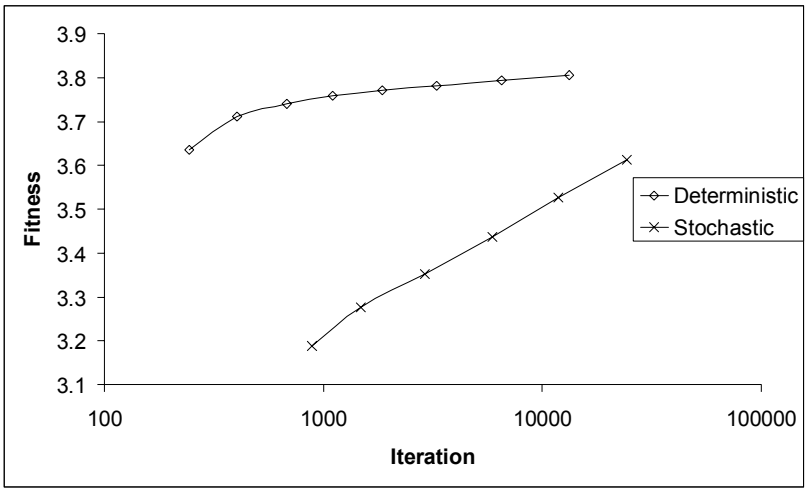

FIGURE 2.4. Increase of Fitness for Different Survival Schemes (Problem 2)

2.4.2 Second-Phase Simulations

In the second phase, the variable penalty scheme was investigated, using the best strategies found in the first phase. The question to be answered was what values τ, δ, and ε gave a good performance. Six variable penalty schemes were investigated. Three iteration cycles with $\tau = 10$ (fast), $\tau = 20$ (moderate), and $\tau = 40$ (slow) were combined with two adaptation schemes with $\delta = 0.11, \varepsilon = 0.08$ (high), and $\delta = 0.03, \varepsilon = 0.02$ (low). The algorithm was stopped after 16000 iterations for Problem 1 and 32000 iterations for Problem 2. As with the preceding simulations, the best solution and the iteration at which it was found were reported.

TABLE 2.6. Best Fitness and Iteration when Found for Different Variable Penalty Schemes

	Fitness		Found at Iteration	
Penalty Scheme	M	SD	M	SD
		Problem 1		
Fast/Low	4.603	0.016	5600	4500
Moderate/Low	4.605	0.014	5600	4100
Slow/Low	4.605	0.013	6200	3600
Fast/High	4.601	0.017	5400	4400
Moderate/High	4.603	0.015	5300	4400
Slow/High	4.606	0.013	5200	4400
		Problem 2		
Fast/Low	3.811	0.050	14300	10300
Moderate/Low	3.829	0.040	12800	9700
Slow/Low	3.829	0.039	13100	9800
Fast/High	3.791	0.057	15400	9900
Moderate/High	3.805	0.052	13200	10000
Slow/High	3.818	0.049	13400	9800

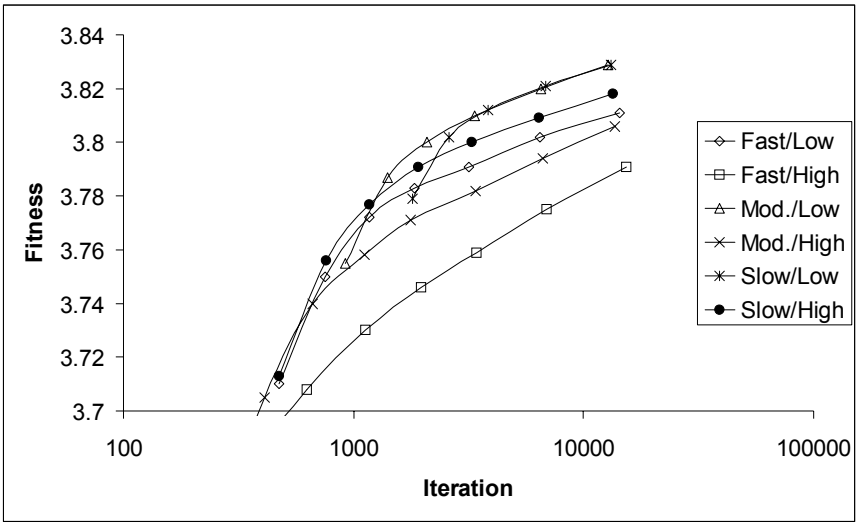

FIGURE 2.5. Increase of Fitness for Different Variable Penalty Schemes (Problem 2)

Table 2.6 shows that, in general, the slow iteration cycles performed somewhat better than the fast and moderate cycles. Similarly, the low adaptation schemes seemed to win from the high adaptation schemes. The slow-high scheme performed somewhat better than the slow-low scheme.

Although this was not strictly true for Problem 1, it should be noted that this much simpler problem needed considerably fewer iterations. Therefore, the slow-low penalty scheme seemed to be the most appropriate choice for both cases. One phenomenon should be considered, however. As can be seen in Figure 2.5, the slow-low and moderate-low schemes tended to produce feasible solutions rather late in the process while the other schemes produced feasible solutions earlier. Especially in combination with very quick stopping rules, this might lead to the unsatisfactory situation that no solution is found at all. In such cases, it might be wise to consider one of the other schemes or to modify the stopping criterion to accommodate this phenomenon.

2.4.3 Third-Phase Simulations

For the third phase, a new stopping rule was developed. Until now, the simulations were based on a fixed stopping rule. After a fixed number of iterations the process was stopped and the best solution so far was considered. Before the simulations of phase three were conducted, optima were evaluated using CPLEX (2002) as a reference. For Problem 1, the optimal value was 4.620, for Problem 2, 3.920. However, before a comparison could be made, a better stopping rule had to be devised.

Various rules were examined for their usefulness, either for a quick or a more thorough optimisation. The simplest rule was similar to the fixed rule used before: stop after a fixed number of iterations. As more complex models might need more iterations to converge, the stopping rule should be a function of the number of items in the bank L and the total number of restrictions K. Stopping after LK iterations would give fast results while the deviations from the reported optima might be acceptable in many practical applications. Two further stopping rules were considered for applications that require results better than those found by the LK-rule. One was stopping after $LK \log(LK)$. As this rule would need considerably more iterations, we studied a compromise between the two rules and stopped after $LK \log(K)$ iterations.

Other rules were based on the convergence of the optimisation process. Two types of convergence could be distinguished. The first type was based on the ratio between online and offline performance. Online performance is defined as the average fitness of all tests in a generation, and offline performance as the average fitness of a small number of the best tests. This small number is usually defined as 5 - 10% of the population. In the previous simulations it was observed that the ratio started at about 0.70, rising to approximately 0.995 at the moment that the current best test was approximately 2 - 5% from the optimum.

The second type of convergence was based on the creation of new feasible tests. If, for a certain period of time, no feasible tests were created that

would rank somewhere in the top of current best tests, then the algorithm was stopped. The intervals between improvements could be estimated from the previous simulations. We assumed that finding an improved test within the next few iterations was proportional to the number of iterations, and that the time needed to find an improvement was independent from times needed for other improvements. And though improvements are generally easier to find in early phases of the process, we assumed that the improvement density was constant over populations with equal average fitness. Under these assumptions, the improvements can be modelled as a Poisson process, whereby the interval between the iterations at which the previous improvements were found and the report iterations in the previous simulations was used as an indication of the expected time needed to find the next improvements. Furthermore, we assumed that an improvement in the n best tests was approximately n times more likely than an improvement in the single best test. We may therefore derive from interpolations in Table 2.1, that stopping when no change had been observed in the 10 best tests in $\frac{LK}{10}$ iterations, would find a solution within 2% from the optimum. Both stopping rules based on convergence, however, needed a fixed rule in case the criterion was never met.

In total, 5 stopping rules were examined:

1. Stop after LK iterations.
2. Stop after $LK \log(K)$ iterations.
3. Stop after $LK \log(LK)$ iterations.
4. Stop when the online - offline performance ratio is 0.995.
5. Stop when no change in the 10 best tests is found within $\frac{LK}{10}$ iterations.

Three problems were considered: Problem 1 and 2 used in the earlier simulations and Problem 3, based on an item pool for Turkish reading comprehension. The pool contained 205 items calibrated with the OPLM model by Verhelst, Glas and Verstralen (1995). The items were coded with respect to five themes: home, public situations, work related, study, and other. The categories contained 43, 115, 11, 37, and 3 items, respectively. Problem 3 involved an equal standard error of measurement at three critical ability levels as well as a content balancing over the five categories, requiring 5, 15, 3, 5, and 2 items. Together with a test length restricted to 30 items, Problem 3 consisted of 9 restrictions.

It can be inferred from Table 2.7 that the stopping rules based on convergence were outperformed by the fixed stopping rules. Frequently, a lower fitness was reached after, on average, a larger number of iterations. Moreover, since the standard deviation of the number of iterations was rather high, the required calculation time became unpredictable. The stopping

TABLE 2.7. Best Fitness and Number of Iterations for Different Stopping Rules

	Fitness		Iterations	
Rule	M	SD	M	SD
		Problem 1		
1	4.599	0.019	2500	
2	4.606	0.016	4024	
3	4.615	0.008	19560	
4	4.597	0.021	2700	760
5	4.602	0.019	3100	700
		Problem 2		
1	3.827	0.041	23500	
2	3.843	0.035	90478	
3	3.852	0.033	236522	
4	3.813	0.055	28900	33100
5	3.832	0.040	45200	28500
		Problem 3		
1	11.483	0.013	1845	
2	11.484	0.012	4054	
3	11.486	0.009	13875	
4	11.483	0.014	1300	55
5	11.484	0.013	2600	600

rules based on a criterion of convergence were therefore not considered any further.

TABLE 2.8. Best Fitnesses and Percentages of Deviation from the Optimum

Rule	Problem 1	Problem 2	Problem 3
LK	0.4%	2.4%	$< 0.1\%$
$LK\log(K)$	0.3%	2.0%	$< 0.1\%$
$LK\log(LK)$	0.1%	1.7%	$< 0.1\%$
Optimum	4.620	3.920	11.488

Assuming that a solution within a 5%-bandwidth from the optimum should be found, stopping rule 1 will generally be sufficient for the three instances that were investigated, although the deviations presented in Table 2.8 do not hold for individual cases, but for averages. When a solution within 2% from the optimum is preferred, stopping rule 3 could be chosen for Problem 2.

2.5 IMIN Model

The proposed fitness function for the IMIN model is similar to the fitness function of the IMAX model. Since the objective function has to be minimised while the fitness function has to be maximised, a transformation is needed that at the same time prevents the occurrence of negative fitness values:

$$f(x) = \left(\frac{1}{1+\sum_i q_{i0}x_i}\right)\left(\frac{1}{1+g(x)}\right) \tag{2.14}$$

$$\begin{aligned} g(x) = {} & \kappa\sum_k h\left(T_k - \sum\nolimits_i I_{ik}x_i\right) + \lambda\sum_n h\left(Q_n - \sum\nolimits_i q_{in}x_i\right) \\ & +\mu\sum_m h\left(C_m^{\ell} - \sum\nolimits_i c_{im}x_i\right) + \mu\sum_m h\left(\sum\nolimits_i c_{im}x_i - C_m^{u}\right) \\ & +\varphi\sum_r 1 - p_r(x)\,. \end{aligned}$$

2.5.1 Epistasis and the IMIN Model

When using the test length as the objective, there are many solutions with equal objective function values. Since all feasible solutions with the same objective function value will have the same fitness, epistasis is the result. Then a GA cannot discriminate between solutions, and the process might stagnate because of lack of a search direction. An obvious way to reduce the epistasis is to allow small differences in fitness. This way every solution has a unique fitness. These differences, however, should not be assigned randomly to the solutions but should have some meaningful value in order to accelerate the optimisation process.

Given two feasible tests, it is easier to retain feasibility when removing an item from the more informative test than from the less informative test. Thus, a reward γ should be assigned to the solutions for each surplus unit of information they have and the fitness function in (2.14) should be replaced by

$$f(x) = \left(\frac{1}{1+\sum_i q_{i0}x_i}\right)\left(\frac{1+\gamma\sum_k h\left(\sum_i I_{ik}x_i - T_k\right)}{1+g(x)}\right). \tag{2.15}$$

The reward should not be too high, though. If it is so high that adding an item to a feasible test compensates for the loss in fitness, the optimum for the fitness function does not coincide with the optimum for the problem. If the removal of an item results in a feasible test, it should always be more profitable than the reward for keeping it in the test.

2.5.2 Simulations

In order to investigate the usefulness of the reward scheme, simulations were performed. As a first step, several reward schemes were simulated using three problems based on the same item bank as the problems for the IMAX problems: Problems 4, 5 and 6 respectively. Problem 4 had four restrictions on the TIF with $T_{\vartheta=-1.5} = 12, T_{\vartheta=-0.5} = 24, T_{\vartheta=0.5} = 24, T_{\vartheta=1.5} = 12$. For Problem 5, two sets of classification restrictions were defined: For categories 101, ..., 104, the maximum percentages of items were 55, 30, 20 and 10, while for categories 201, ..., 210, the maximum percentages were 20. The same principle was used for Problem 6 but 20 - 30% of the items had to be in categories 101, ..., 104, while for categories 201, ..., 210 the ranges were 5 - 15%. Note that the classification restrictions in the IMIN model needed a small modification to accommodate the use of fractions of the test length instead of a fixed number of items. Therefore, restrictions in (2.4) were replaced by

$$C_m^{\ell} \leq \frac{\sum_i c_{im} x_i}{\sum_i x_i} \leq C_m^{u} \qquad \forall m. \tag{2.16}$$

A fixed stopping rule was applied: 8000 iterations for Problem 4 and 32000 iterations for Problems 5 and 6. Furthermore, a slow and low adaptation scheme was used.

TABLE 2.9. Best Fitness and Iteration when Found for Different Reward Schemes

	Fitness		Test Length		Found at Iteration	
Reward	M	SD	M	SD	M	SD
			Problem 4			
0	0.6641	0.0054	25.30	0.62	1600	2100
0.0001	0.6742	0.0033	24.17	0.38	2700	2000
0.001	0.6752	0.0025	24.15	0.36	2600	2000
0.01	1.0632	0.0026	44.06	0.23	730	1200
			Problem 5			
0	0.6636	0.0063	25.36	0.72	6300	7600
0.0001	0.6734	0.0038	24.26	0.44	8900	8800
0.001	0.6744	0.0029	24.23	0.42	6500	8100
0.01	1.0636	0.0016	44.02	0.14	2000	3800
			Problem 6			
0	0.6489	0.0045	27.05	0.54	5400	6400
0.0001	0.6571	0.0026	26.11	0.32	10600	9000
0.001	0.6583	0.0019	26.11	0.31	11000	9600
0.01	1.0414	0.0016	46.02	0.15	3400	5300

From Table 2.9 it can be seen that the reward scheme effectively prevented epistasis and improved the performance of the process. Just the fact that a reward scheme was used seemed important, as long as the order of magnitude for γ was appropriate. It can clearly be seen that too high a value of γ caused the optimal solutions for the fitness and the objective to be different. Therefore, $\gamma = 0.0001$ was used in the subsequent simulations.

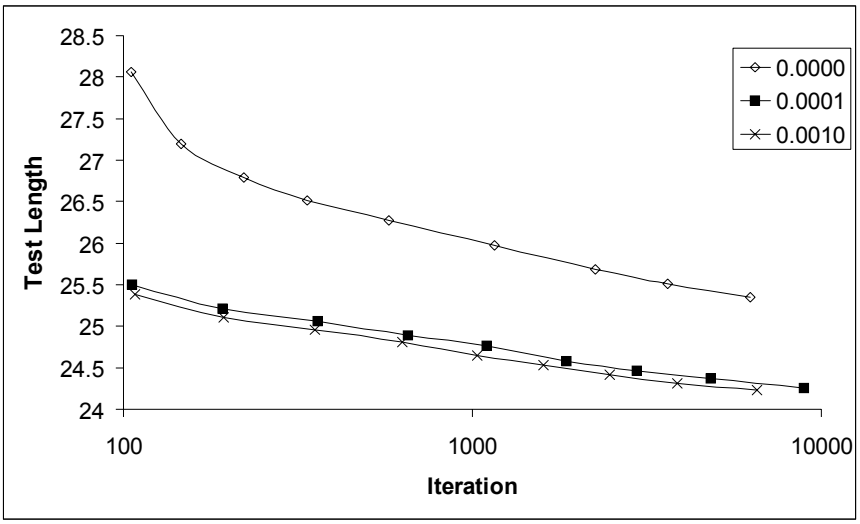

FIGURE 2.6. Decrease of Test Length for Different Reward Schemes (Problem 5)

In the second phase stopping rules were investigated. Two fixed stopping rules were used: The first one stopped the process after LK iterations for a rapid optimisation, the second after $LK\log(LK)$ iterations for a more thorough optimisation.

From Table 2.10 it can be concluded that for all problems investigated, the average test lengths were within a 2% bandwidth from the optimum. Note that this holds for the average test lengths only but not for individual replications as only integer values could occur. For Problem 5, the fast stopping rule found a test of 24 items in 54% of the cases, while in 46% of the cases the best test length was 25. This was well outside the 2% bandwidth. On the other hand, if one would accept a 25-item test, Figure 2.6 suggests that in many cases this would have been found well before the stopping criterion.

2.6 Conclusion

The results presented in this paper show which implementation of genetic algorithms solved the presented test assembly problems best. Important

TABLE 2.10. Best Test Lengths and Percentages of Deviation from the Optimum

		Test Length		Deviation
Rule	Iterations	M	SD	
		Problem 4		
LK	2500	24.35	0.48	1.4%
$LK\log(LK)$	19560	24.14	0.34	0.6%
Optimum		24		
		Problem 5		
LK	9500	24.46	0.50	1.9%
$LK\log(LK)$	87011	24.10	0.30	0.4%
Optimum		24		
		Problem 6		
LK	9500	26.19	0.39	0.7%
$LK\log(LK)$	87011	26.07	0.26	0.3%
Optimum		26		

issues in optimisation, like convergence criteria and speed, are theoretically very hard to deal with. Therefore, simulation studies were used to determine appropriate parameter settings and to demonstrate the usefulness of genetic algorithms for solving test assembly problems.

In general, variable penalty schemes with longer iteration cycles ($\tau = 40$), and low adaptation ($\delta = 0.03, \varepsilon = 0.02$), together with uniform crossover, more explorative mate selection and a rigorous survival scheme gave good results. Stopping after LK iterations yielded a feasible solution within 2% of the optimum for nearly all problems.

Being very versatile methods, GAs can successfully be employed to solve test assembly problems. Although they are in general not extremely efficient for problems that can be solved by traditional methods, their strength lies in the ease to accommodate restrictions that appear to be cumbersome for other methods, like interitem relations or for solving nonlinear models.

3

An Approximation of Cronbach's α and its Use in Test Assembly

3.1 Introduction

Test assembly modelling for classical test theory has been a somewhat neglected field. It concerns the selection of items from a pool in such a way that the reliability of the resulting test is maximised under conditions like a fixed test length, a taxonomic make up, and other demands. Assembling a test with maximum reliability involves nonlinear models, which are difficult to solve using traditional optimisation techniques. Adema and van der Linden (1989) were the first to circumvent this difficulty by formulating a linearisation of the reliability of the test. Armstrong, Jones and Wang (1994) took a different approach by formulating the problem as a network flow model. Both models can be solved using standard integer linear programming methods, but are prone to yielding suboptimal solutions or are applicable in specific contexts only.

In this paper, two test assembly models are introduced that use an approximation of Cronbach's α as an operationalisation of the reliability of the test. Utilising optimisation techniques like genetic algorithms (GAs), introduced by Holland (1975), it is possible to solve this class of models efficiently without loss of generality as in the network flow models of Armstrong et al.

GAs optimise problems by emulating principles from evolution theory in biology. A population of solutions, represented by chromosomes, is subject to an evolutionary process. Every chromosome in the population has a chance to mate with another chromosome, creating offspring through a re-

combination process. Subsequently, a "survival of the fittest" rule is applied in such a way that the population size remains constant.

Since solutions obtained through GAs cannot be proven to be optimal either, simulation studies are conducted to investigate the solutions found, and these solutions are compared with solutions obtained for a model similar to Model II of Adema and van der Linden.

3.2 CMAX Model

In classical test assembly a key notion is the reliability ρ_{XT}^2 of the to-be-assembled test or, more specifically, its lower bound Cronbach's coefficient α, given by

$$\alpha = \frac{k}{k-1}\left\{1 - \frac{\sum_i \sigma_i^2}{\sigma_X^2}\right\} = \frac{k}{k-1}\left\{1 - \frac{\sum_i \sigma_i^2}{\left(\sum_i \rho_{it}\sigma_i\right)^2}\right\} \tag{3.1}$$

where k is the test length, σ_i^2 is the variance of item i, and ρ_{it} the correlation between the item score and test score. Using equation (3.1), a test assembly model can be formulated for situations in which the test assembler wishes to assemble a test with maximum α while adhering to test specifications regarding the use of limited resources like test length or test time, a desired difficulty range, a taxonomic makeup and restrictions at the item level:

$$\text{maximise} \quad \alpha = \frac{\sum\limits_i x_i}{\sum\limits_i x_i - 1}\left\{1 - \frac{\sum\limits_i \sigma_i^2 x_i}{\left(\sum_i \rho_{it}\sigma_i x_i\right)^2}\right\} \tag{3.2}$$

$$\text{subject to:} \quad P^\ell \leq \frac{\sum\limits_i \pi_i x_i}{\sum\limits_i x_i} \leq P^u \tag{3.3}$$

$$\sum_i q_{in} x_i \leq Q_n \qquad \forall n \tag{3.4}$$

$$C_m^\ell \leq \sum_i c_{im} x_i \leq C_m^u \qquad \forall m \tag{3.5}$$

$$p_r(x) = 1 \qquad \forall r \tag{3.6}$$

$$x_i = \begin{cases} 1, & \text{item } i \text{ in the test} \\ 0, & \text{else} \end{cases} \qquad \forall i.$$

Variables x_i are the decision variables indicating whether an item is selected or not. Coefficient π_i denotes the difficulty of item i, for dichotomously scored items defined as the expected score given by a randomly selected testee from the population of interest. P^ℓ and P^u define the range in which the average item difficulty, in classical test theory also called test difficulty,

should be. The restrictions in (3.4) put a limit to the use of certain resources, for example, the number of items in the test or the time allotted to take the test. Q_n is the total amount of the resources the test assembler is willing to spend, while coefficients q_{in} are the resource parameters of item i. These parameters indicate how much of a certain type of resources, for example, test time, will be spent if the item is in the test. As can be derived from the Spearman-Brown formula for test lengthening, the easiest way to maximise Cronbach's α is to use more items in the tests as long as items can be found that leave that leave other test properties equal, possibly ad infinitum. So, resource restrictions in (3.4) prevent this situation. Next to these, the restrictions in (3.5) define a desired taxonomic makeup of the test. Coefficients c_{im} are the classification parameters having value 1 if item i belongs to category m and 0 otherwise, while C_m^{ℓ} and C_m^{u} define the desired range of items in the classification categories. A third group of restrictions in (3.6) concerns with the item level. These are called interitem relations. When building large item pools, it is almost inevitable that this type of relations is needed. Common examples of interitem relations are enemy sets and testlets. Enemy sets are groups of mutually excluding items while testlets are groups of mutually including items. Theunissen (1996) has shown that sometimes other, more complicated relations are necessary. These relations can be formulated as well, using Boolean operators $\vee$, $\wedge$ and $\neg$. Function $p_r(x)$ denotes the differential payoff function for relation r. An example of how an interitem relation can be transformed into a restriction based on the payoff function is elaborated in Appendix A. Note that, in general, these restrictions are nonlinear, as can be seen in the example.

In practical situations, a problem arises as (3.2) can only be evaluated after the test has been administered. Furthermore, even if (3.2) could be evaluated, it would result in a nonlinear model that would need special techniques to be solved. Therefore, Adema and van der Linden (1989) proposed their Model II in which the objective in (3.2) is replaced by

$$\text{maximise} \quad \sum_i r_{it} x_i \tag{3.7}$$

where r_{it} is the point-biserial correlation between item score i and the score of the test in which the item was pretested. Their model is based on Gulliksen's observation (1950) that, in general, in (3.1), the sum of item variances varies less than the test variance, and so α can be expected to depend more on the latter. This effect is also verified by Ebel (1967) and suggests that it is more profitable to select items with a high r_{it} than with a low one, and thus that it is most efficient to use (3.7) as the objective.

Adema and van der Linden's model has the advantage that it is linear, and thus can be optimised by the use of integer linear programming techniques. In its original form, however, the model has a drawback. During test assembly, the test to be assembled is not known, and neither are the corresponding r_{it}'s. This drawback can be overcome by the assumption

that the r_{it}'s of the newly assembled test will be very similar to the r_{it}'s of previous pretests.

But in general the item pools used for test assembly consist of items originating from different test forms. If the test forms used in building the item pool vary in length, items from shorter test forms are favoured without proper ground for the following reason given by Guilford (1954, p.439):

> When an item is correlated with the total score of which it is a part, the value of r_{it} tends to be inflated. The shorter the test, the greater this inflation is likely to be. Even if all items correlated actually zero with what the total score measures, and if all item variances were equal, each item would correlate to the extent of $1/\sqrt{n}$, where n is the number of items.

In order to overcome this inflation, item discrimination indices are needed that are independent from test contexts such as test length and test variance. Zubin (1934) faced this problem by concentrating on the item-rest correlation r_{ir}. This would mean that the items would be correlated with slightly different remainders. A larger drawback, however, is the fact that if items are positively correlated with each other, coefficient r_{ir} is not invariant to test length either, as Guilford (1953) noted. In that case, items originating from longer tests are favoured over items from shorter tests, all other things being equal.

Consider the point-biserial correlations found in earlier analyses, corrected for unique item variances as proposed by Henrysson (1963):

$$d_{it} = \sqrt{\frac{k}{k-1}} \frac{r_{it} s_X - s_i}{\sqrt{s_X^2 - \sum_i s_i^2}}. \tag{3.8}$$

Coefficient d_{it} is claimed to be invariant to test length. In order to verify this claim, a simulation was conducted. An item pool consisting of 500 items was used with simulated IRT-parameters according to the two-parameter model with $\log(\alpha) \sim N(0, 0.4)$ and $\beta \sim N(0, 1)$. A population of 100,000 simulated candidates with $\vartheta \sim N(0.07, 0.35)$ was used to generate item scores. From this dataset, several tests with varying lengths were defined and analysed. In Table 3.1, the r_{it}, r_{ir}, and d_{it} for item 1 are given for each of the test lengths.

It is clear that even for tests of length 80 the inflation of r_{it} and r_{ir} is still large, while even for short tests the d_{it} remains relatively stable. Therefore, d_{it} can be regarded as an item discrimination index, relatively free from the context of the test in which it was originally included, and hence can be denoted by d_i. Henrysson (1962, 1963) argues that its square, d_i^2, is an estimate of the item communality in a factor analysis and that using d_i instead of r_{it} removes spurious preference towards items analysed in short tests. Substituting d_i for r_{it} in (3.7) overcomes the effects of the context of the pretests, but prediction of α of the newly assembled test remains

TABLE 3.1. Analysis of r_{it}, r_{ir} and d_{it} for Item 1

Items	r_{1t}	r_{1r}	d_{1t}
1..5	0.477	0.040	0.152
1..10	0.311	0.050	0.145
1..20	0.256	0.074	0.149
1..40	0.214	0.102	0.151
1..80	0.184	0.119	0.149
1..500	0.155	0.143	0.149

cumbersome. Calculation of new point-biserials using (3.8) is not possible since the test variance is not known.

Now, consider the test variance to be formulated as

$$\sigma_X^2 = \sum_i \sigma_i^2 + \sum_{i \neq j} \mathrm{Cov}(X_i, X_j).$$

From the assumption of a one-factor model, it follows that d_i can be regarded as the factor loading of item i, and hence that $\forall i, j : \mathrm{Cov}(X_i, X_j) = d_i \sigma_i d_j \sigma_j$. Therefore, the following relation can be derived as an estimate of Cronbach's α for a newly assembled test:

$$\alpha^* = \frac{\sum\limits_i x_i}{\sum\limits_i x_i - 1} \left\{ 1 - \frac{\sum\limits_i \sigma_i^2 x_i}{\sum\limits_i \sigma_i^2 x_i + \sum\limits_{i \neq j} d_i \sigma_i d_j \sigma_j x_i x_j} \right\}. \tag{3.9}$$

Using (3.9), the CMAX model is formulated as

$$\text{maximise} \quad \frac{\sum\limits_i x_i}{\sum\limits_i x_i - 1} \left\{ 1 - \frac{\sum\limits_i \sigma_i^2 x_i}{\sum\limits_i \sigma_i^2 x_i + \sum\limits_{i \neq j} d_i \sigma_i d_j \sigma_j x_i x_j} \right\} \tag{3.10}$$

$$\text{subject to:} \quad P^\ell \leq \frac{\sum\limits_i \pi_i x_i}{\sum\limits_i x_i} \leq P^u$$

$$\sum_i q_{in} x_i \leq Q_n \qquad \forall n$$

$$C_m^\ell \leq \sum_i c_{im} x_i \leq C_m^u \qquad \forall m$$

$$p_r(x) = 1 \qquad \forall r$$

$$x_i = \begin{cases} 1, & \text{item } i \text{ in the test} \\ 0, & \text{else} \end{cases} \qquad \forall i.$$

As the objective in the CMAX model (3.10) is nonlinear, traditional methods based upon integer linear programming are unsuitable. On the

other hand, Verschoor (2004) has shown that genetic algorithms (GA) are capable of solving this class of optimisation problems efficiently.

3.3 Genetic Algorithms

Genetic algorithms are iterative processes in which a population of N solutions mate, procreate, and are subject to a survival of the fittest-rule in order to survive to the next iteration. Eiben and Smith (2003) discuss these processes in terms of a few relevant components that can be distinguished:

Representation. Every solution is represented by a string, called a chromosome, consisting of genes. Every decision variable in the CMAX model, indicating whether an item is selected or not, is mapped onto a gene. Thus, the chromosome length is equal to the number of items in the pool, denoted by L.

Mate selection. This is assumed to be a stochastic process controlled by the fitness of the solutions: the higher the fitness, the higher the probability to be selected. Usually, the probability is chosen to be proportional to the fitness.

Recombination. This is a process in which two newly created tests, that are initially identical to their parents, exchange items through the crossover operator. A common choice for test assembly models is uniform crossover, in which every gene, or item, has a probability of 0.5 to be exchanged between the tests. A further operator in the recombination process is mutation. Mutation can be seen as the introduction of genetic information: Each gene has a small chance p_μ to flip value. In terms of test assembly this means that each item in the pool has a small chance to be added to the test or to be removed from it.

Survival. After creation of N children, the population is reduced to its original size. Verschoor (2004) has shown that removing all duplicate solutions, and further removing as many solutions with the lowest fitness as needed, is an effective survival mechanism for automated test assembly.

Fitness. Fitness is modelled as a fitness function. Holland (1975) has proven in his schema theorem that under assumptions of selection chances being proportional to the fitness, the population evolves towards regions with high fitness.

But does evolution towards high fitnesses mean that GAs will converge to the optimum of the fitness function? This question was answered by Goldberg and Segrest (1987), and Eiben, Aarts and van Hee (1991) in the

field of Markov chain analysis. Their approach was to model the GA as a Markov chain. They analysed their steady states, and for which classes of genetic algorithms these steady states contain an optimal solution. Thierens and Goldberg (1994) showed that for these classes the convergence can be estimated for sufficiently large populations. The chance that the population in iteration t contains the solution with optimal fitness is asymptotically given by $P_{opt}(t) = 1 - \left\{1 - (1 - 0.5e^{-t/L})^L\right\}^N$. Evaluating this expression for a given problem shows that the GA will converge to the optimal solution, but at the same time it is clear that a given solution cannot be proven to be optimal.

The fitness function should have a relation with the objective. For unrestricted problems, the fitness function may be chosen identical to the objective. ATA models, however, are always restricted. A usual choice to process restrictions is defining a penalty function for all infeasible solutions. Richardson, Palmer, Liepins, and Hilliard (1989) have shown that penalties that are functions of the distance to the feasible region are expected to perform best. Therefore, for violated restriction $\sum_i a_i x_i \leq b$, the penalty function can be defined as $g(x) = \lambda\left(\sum_i a_i x_i - b\right)$, while the fitness function is defined as $f(x) = \frac{\alpha^*}{1+g(x)}$. Coefficient λ is called the penalty multiplier. A penalty function is versatile in the sense that new restrictions can be added without the need to redesign the implementation.

Summarising, the proposed fitness function for the CMAX model $f(x)$ is defined by α^*, compensated with a penalty function $g(x)$ dependent on the rate at which the restrictions are violated:

$$f(x) = \frac{\alpha^*}{1 + g(x)} \tag{3.11}$$

$$\begin{aligned} g(x) =& \kappa\ h\left(P^\ell - \frac{\sum_i \pi_i x_i}{\sum_i x_i}\right) + \kappa\ h\left(\frac{\sum_i \pi_i x_i}{\sum_i x_i} - P^u\right) \\ &+ \lambda \sum_n h\left(\sum\nolimits_i q_{in} x_i - Q_n\right) \\ &+ \mu \sum_m h\left(C_m^\ell - \sum\nolimits_i c_{im} x_i\right) + \mu \sum_m h\left(\sum\nolimits_i c_{im} x_i - C_m^u\right) \\ &+ \varphi \sum_r 1 - p_r\left(x\right) \end{aligned}$$

$$h(u) = \begin{cases} u, & u > 0 \\ 0, & u \leq 0 \end{cases} .$$

Coefficients κ, λ, μ and φ are the penalty multipliers. A major question is what value the penalty multipliers should have. If the penalty is too low, the maximum of the fitness function is located in the infeasible space and the population will evolve to an infeasible solution. Penalties should, however, not be chosen too high. If infeasible solutions are eliminated too quickly, diversity in genetic information might be lost, in extreme cases to the extent even that crossover is almost ineffective and evolution will take place only through mutation. This situation is called premature convergence. Le Riche, Knopf-Lenoir, and Haftka (1995) confirm the effectiveness of the so-called *minimal penalty rule*: The penalty should be kept as low as possible, that is, just above the – unknown – limit at which the optimum of the problem and the optimum of the fitness coincide.

In concordance with the findings of Siedlecki and Sklanski (1989), a dynamic penalty adaptation can be used to establish the optimal values for the penalty multipliers. Such a genetic algorithm has a penalty updating scheme with the following outline: Consider for τ consecutive iterations the tests with the highest fitness. If for all these τ, possibly identical, tests all resource restrictions are met, multiply λ by $1-\delta$ since there is a risk that λ is too high, causing premature convergence. If for all tests some resource restrictions are violated, multiply λ by $1+\varepsilon$, since this is an indication that the population is evolving towards infeasible solutions. Leave λ unchanged in all other cases, that is, that for some tests all resource restrictions are met while for other tests there are violated restrictions. The same rule is followed for the other penalty multipliers κ, μ and φ. Usually, τ has a fixed value in the order of magnitude of 10 - 40 iterations, while δ and ε have values in the order of magnitude of 0.02 - 0.10.

3.3.1 Simulations

Simulations were conducted in two phases. In the first phase, the dynamic penalty scheme was investigated while in the second phase several stopping rules were considered.

The question to be answered in the first phase is what values should τ, δ and ε have in order to give a good performance. Specifically, can a fixed set of values be found or should they depend on the complexity of the model, expressed in the number of restrictions?

To answer this question, three different test assembly problems were used. Each problem was based upon the same item pool used earlier in the simulations of the d_i-coefficients. Furthermore, all items were coded with respect to three different classifications. The first classification consisted of 4 categories labelled as 101, 102, 103, and 104. All items were uniformly distributed over these categories. The second classification consisted of 4 categories labelled as 201, 202, 203, and 204. These categories were filled with approximately 250, 125, 85 and 40 items, respectively. The third clas-

sification contained categories labelled as 301, ..., 310, which contained approximately equal numbers of items.

The simplest test assembly problem, Problem 1, had one restriction related to the target test difficulty $P^{\ell} = 0.6 \leq \frac{\sum_i \pi_i x_i}{\sum_i x_i}$ and one resource restriction $\sum_i x_i \leq 40$. Problem 2 was an extension of Problem 1 with 16 content restrictions were added. From each combination of two categories, the first from the range 101, ..., 104, and the second from the range 201, ..., 204, a minimum of two items and a maximum of three items were required in the test. Problem 3 had 40 content restrictions. From each combination of two categories, the first from the range 201, ..., 204, and the second from the range 301, ..., 310, exactly one item was required in the test.

Six variable penalty schemes were investigated. Three iteration cycles with $\tau = 10$ (fast), $\tau = 20$ (moderate), and $\tau = 40$ (slow) were combined with two adaptation schemes with $\delta = 0.11, \varepsilon = 0.08$ (high), and $\delta = 0.03, \varepsilon = 0.02$ (low). The algorithm was stopped after 4000 iterations for Problem 1, 8000 iterations for Problem 2, and 32000 iterations for Problem 3. Each condition was repeated 400 times. In Table 3.2, the best feasible

TABLE 3.2. Best α^* and Iteration when Found for Different Penalty Schemes

	α^*		Found at Iteration	
Penalty Scheme	M	SD	M	SD
		Problem 1		
Fast/Low	0.8492		400	130
Moderate/Low	0.8492		450	220
Slow/Low	0.8492		420	190
Fast/High	0.8492		250	90
Moderate/High	0.8492		330	90
Slow/High	0.8492		410	140
		Problem 2		
Fast/Low	0.8316	0.0007	3500	2200
Moderate/Low.	0.8317	0.0006	3500	2100
Slow/Low	0.8316	0.0007	3800	2000
Fast/High	0.8316	0.0007	3300	2300
Moderate/High	0.8316	0.0006	3100	2300
Slow/High	0.8317	0.0006	3200	2200
		Problem 3		
Fast/Low	0.8017	0.0028	21700	7700
Moderate/Low	0.8021	0.0027	21400	7600
Slow/Low	0.8025	0.0024	21100	7600
Fast/High	0.8009	0.0035	22700	6900
Moderate/High	0.8011	0.0035	21300	8100
Slow/High	0.8018	0.0028	21900	7600

solutions are reported as well as the iterations at which they were found. Note that feasible solutions were found in all replications.

In Table 3.2 it can be seen that, for example, for the fast and low adaptation scheme applied on Problem 3, an average α^* of 0.8017 was found in an average of 21700 iterations. The conclusion can be drawn that the more complex the problem in terms of the number of restrictions, the better a low and slow adaptation scheme seemed to perform. As the number of iterations that were needed tended to grow with the complexity, the low and slow scheme could be selected in all cases without great loss of efficiency for the less complex problems.

In the second phase the question was if the stopping rule could be improved. Two stopping rules, both based on the number of items and the number of restrictions in the problems, were investigated. A faster rule was to stop after LK iterations, where L is the number of items in the pool and K the total number of restrictions in the problem. The second rule, a slower one that generally gives better solutions, was to stop after $LK\log(LK)$ iterations. A slow and low penalty adaptation scheme was used for these simulations.

Before conducting the simulations, the problems were formulated for MINTO (Murtagh, 1988), a commercially available package for integer nonlinear programming. As with genetic algorithms, MINTO does not give the optimal solution, but similarly to genetic algorithms presents the best solution it has found. It does give, however, an upper bound on α^*. As the performance of MINTO vastly improves by adding problem specific knowledge, some experiments were performed with sequentially adding the restrictions to the problem, and the lowest upper bound found in these experiments was presented as the upper bound.

TABLE 3.3. Best α^* and Iteration when Found for Different Stopping Rules and Deviation from Upper Bound

		α^* Reported			
Rule	Iterations	M	SD	Upper Bound	Deviation
		Problem 1			
LK	1000	0.8492		0.8492	–
$LK\log(LK)$	6908	0.8492			–
		Problem 2			
LK	9000	0.8318	0.0005	0.8321	0.2%
$LK\log(LK)$	81945	0.8321	0.0002		$< 0.1\%$
		Problem 3			
LK	21000	0.8030	0.0022	0.8057	1.7%
$LK\log(LK)$	208998	0.8054	0.0005		0.2%

In all 400 replications for the three problems, both stopping rules found a feasible solution. In Table 3.3, the average and standard deviation of the α^* of the best feasible solutions, taken over the repetitions, are reported as well as the upper bounds on α^* according to MINTO, and the deviations in terms of extra items needed to extend the test in order to reach the upper bound on α^*, according to the Spearman-Brown formula for test lengthening.

If the criterion is to stop as soon as a solution is found within a 2%-bandwidth from the upper bound, the first stopping rule will generally be sufficient. But note that the criterion can only be validated by external means such as MINTO. Within the genetic algorithm an upper bound is not known.

3.4 Comparison of CMAX and Model II

With the simulated item pool described above, the CMAX model and Model II of Adema and van der Linden were compared in order to investigate whether the models give different solutions. For this purpose, Problems 1 and 3 were adapted for both test assembly models. Problem 1 for CMAX, combined with the $LK\log(LK)$-stopping rule, was compared to the adapted Problem 1 for Model II. Furthermore, Model II was slightly modified to accommodate the use of d_i instead of r_{it}. Objective (3.7) was replaced by

$$\text{maximise} \quad \sum_i d_i x_i. \tag{3.12}$$

Both models found a feasible test designated as the best one, and for these two tests, new response datasets for the simulated population were generated and analysed. The Problem 1 for CMAX and Problem 1 for Model II

TABLE 3.4. Comparison of Model II and the CMAX Model

Model	$\sum d_i$	α^*	α	Deviation
		Problem 1		
Model II	13.95	0.8492	0.8509	1.3%
CMAX	13.95	0.8492	0.8509	1.3%
		Problem 3		
Model II	12.03	0.8034	0.8051	1.1%
CMAX	12.00	0.8057	0.8072	1.0%

have the same optimum. After analysis of the new response data for this optimal test, Cronbach's α was observed to be 0.8509, while α^* was 0.8492, a slight underestimation of 1.3%, in the order of magnitude of half an item

difference in test length. For Problem 3, the same procedure was followed. CMAX and Model II have different optimal solutions in the case of Problem 3. The CMAX optimum has an α^* is approximately 1.5% better than the Model II optimum according to the Spearman-Brown formula.

For these solutions too, new response datasets were generated and analysed. As with Problem 1, α^* was smaller than the observed α. In first instance this might suggest that α^* is a lower bound on Cronbach's α, and therefore on the test reliability, but no evidence for such a claim can be found. But nonetheless, differences are relatively small, so that a new test assembly model becomes feasible.

3.5 CMIN Model

The CMIN model (3.13) expresses the wish to assemble a test that has a minimal use of resources, given a threshold reliability α_T, a target difficulty range, a set of resource restrictions, content balancing, and interitem relations. It can be applied in situations that require a test with a reliability at least as high as the target α_T, to be assembled against minimal cost, expressed by, for example, test length. It is formulated as

$$
\begin{aligned}
\text{minimise} \quad & \sum_i q_{i0} x_i && (3.13)\\
\text{subject to:} \quad & \alpha^* \geq \alpha_T \\
& P^\ell \leq \frac{\sum_i \pi_i x_i}{\sum_i x_i} \leq P^u \\
& \sum_i q_{in} x_i \geq Q_n && \forall n \\
& C_m^\ell \leq \sum_i c_{im} x_i \leq C_m^u && \forall m \\
& p_r(x) = 1 && \forall r \\
& x_i \in \{0,1\} && \forall i.
\end{aligned}
$$

Note that as α^* is not a lower bound on α, the observed α need not be larger than α_T.

As with the fitness function of the genetic algorithms used to solve the CMAX problems, the fitness function for the CMIN model is based on the objective and a penalty function to accommodate the threshold restriction, the difficulty restrictions, content balancing, and interitem relations:

$$f(x) = \left(\frac{1}{1+\sum_i q_{i0}x_i}\right)\left(\frac{1}{1+g(x)}\right) \tag{3.14}$$

$$\begin{aligned} g(x) =& \kappa\ h\left(\alpha_T - \alpha^*\right) + \kappa\ h\left(P^\ell - \frac{\sum_i \pi_i x_i}{\sum_i x_i}\right) + \kappa\ h\left(\frac{\sum_i \pi_i x_i}{\sum_i x_i} - P^u\right) \\ &+ \lambda \sum_n h\left(Q_n - \sum\nolimits_i q_{in}x_i\right) \\ &+ \mu \sum_m h\left(C_m^\ell - \sum\nolimits_i c_{im}x_i\right) + \mu \sum_m h\left(\sum\nolimits_i c_{im}x_i - C_m^u\right) \\ &+ \varphi \sum_r 1 - p_r\left(x\right). \end{aligned}$$

Especially when using the test length as the objective, there are many solutions with equal objective function values. Since all feasible solutions with the same objective function value will have the same fitness, *epistasis* is the result. Epistasis is described as the situation in which some genes have no direct influence on the fitness. If the population evolves towards a situation in which many solutions have the same fitness, the GA cannot make a distinction between these solutions. The result may be that the evolution will stagnate because of lack of a search direction. An obvious way to reduce the epistasis is to allow small differences in fitness so that every solution has a unique fitness. But these differences should not be assigned randomly to the solutions, they should have a meaningful value in order to accelerate the optimisation process instead.

It is easy to see that, given two feasible tests, it is easier to retain feasibility while removing an item from the test with the highest α^* than from the test with the lowest α^*. Therefore, a reward γ could be assigned to each solution for each surplus unit of α^*, and the fitness function in (3.14) could be replaced by

$$f(x) = \left(\frac{1}{1+\sum_i q_{i0}x_i}\right)\left(\frac{1+\gamma h(\alpha^* - \alpha_T)}{1+g(x)}\right). \tag{3.15}$$

The reward, however, should not be too high. If it is high, adding an item compensates for the increase in objective function value and loss in fitness. In that case, the optimum of the fitness function does not coincide with the optimum of the objective function. If the removal of an item results in a feasible solution, it must always be more profitable than the reward for the surplus of reliability caused by keeping the item in the test.

3.5.1 Simulations

In order to investigate the CMIN model, simulations were conducted. Three problems were simulated using the same item pool as for the CMAX problems: Problems 4, 5, and 6, respectively. Problem 4 had one restriction on the threshold reliability: $\alpha_T = 0.83$, and one restriction on the target test difficulty: $P^\ell = 0.6 \leq \frac{\sum_i \pi_i x_i}{\sum_i x_i}$. For Problem 5, two sets of classification restrictions were defined. The maximum percentage of items in categories 201, ..., 204 was 55, 30, 20, and 10, while for categories 301, ..., 310 the maximum percentages were 20. Problem 6 had 40 content restrictions, similar to Problem 3. From each combination of two categories, the first from the range 201, ..., 204, and the second from the range 301, ..., 310, a maximum of one item was required in the test. At the same time, the threshold α_T was lowered to 0.80 as a test of $\alpha^* \geq 0.83$ appeared not to be feasible in combination with these content restrictions. Note that for Problem 5, the classification restrictions in the CMIN model needed a small modification in order to accommodate percentages of test length instead of numbers of items:

$$C_m^\ell \leq \frac{\sum_i c_{im} x_i}{\sum_i x_i} \leq C_m^u \qquad \forall m. \tag{3.16}$$

Two fixed stopping rules were applied, the first rule stopped after LK iterations and the second rule after $LK \log(LK)$ iterations. Furthermore, a slow and low adaptation scheme with $\tau = 40$, $\delta = 0.03$ and $\varepsilon = 0.02$ was used.

TABLE 3.5. Best Solutions for the CMIN Models

Rule	Iterations	α^*	Fitness∗100	Test Length
		Problem 4		
LK	1000	0.8306	3.4404	28.07
$LK \log(LK)$	6908	0.8303	3.4484	28
		Problem 5		
LK	9000	0.8309	3.3336	29
$LK \log(LK)$	81945	0.8309	3.3336	29
		Problem 6		
LK	21000	0.8020	4.0008	24
$LK \log(LK)$	208998	0.8020	4.0008	24

Only for Problem 4 and the fast stopping rule, the optimal test length of 28 items was occasionally not found and a test of length 29 was assembled. In all other cases the optimal test lengths were found, since MINTO confirmed the test lengths of 28, 29, and 24 items, were optimal indeed.

3.6 Conclusions

While Cronbach's α is an important and easy to understand concept, it cannot be evaluated during test assembly. Current test assembly models do not attempt to evaluate α, although they try to optimise it indirectly. This circumvention leads necessarily to suboptimal solutions.

This paper introduces an approximation of α, α^*, that can be evaluated during test assembly. An assumption of a unidimensional item pool, in terms of a one-factor model, is needed. However, this is often an implication of building item pools based on classical test theory. As the resulting test assembly model, CMAX, is nonlinear, genetic algorithms are used. Simulations for three various problems show that, in general, solutions within a 2%-bandwidth from upper bounds could be found.

Comparison of the CMAX model with Adema and van der Linden's Model II shows the benefit of this approach. A second advantage of α^* is that a new test assembly model becomes available: the CMIN model minimises the resources needed for the test given a minimum α^*. The CMIN model finds its justification in those situations in which circumstances dictate the assembly of a test with a specific internal consistency, or higher.

4

Automated Assembly of Testsets: Fit in all Seasons

4.1 Introduction

Since Theunissen (1985) showed how to apply mathematical programming methods in test assembly by formulating a target test information defined at a number of discrete ability points, this topic has gained both theoretical as well as practical attention. Testing agencies try to reduce costs when producing their test forms by constructing item banks, from which items are selected. Usually, no single test forms are produced from item banks, but testsets, consisting of various test forms designed for several purposes. A model for the assembly of parallel test forms, in the sense that test characteristics such as test information function (TIF) are identical, is reported by Boekkooi-Timminga (1990). A related paper on a heuristic approach is Ackerman (1989), while Armstrong, Jones and Wu (1992) concentrated on the assembly of tests similar to an existing seed test using a network flow model. Sanders and Verschoor (1998) took a somewhat different approach by using a greedy heuristic to minimise the distance between the item parameters in the different test forms.

4.2 Test Assembly in Item Response Theory

A key notion in item response theory (IRT) is that of a latent scale on which both the item difficulties and the test takers' abilities are defined. An IRT model defines the probability on an item score obtained by test

takers as a function of the latent ability. A widely used IRT model is the two parameter logistic (2PL) model for dichotomous items, with score 0 for a wrong answer and 1 for a correct answer. The 2PL model assumes the probability on a correct answer for item i and a candidate with ability ϑ to be

$$P(X_i = 1|\vartheta) = \frac{\exp(a_i(\vartheta - b_i))}{1 + \exp(a_i(\vartheta - b_i))} \tag{4.1}$$

where a_i is referred to as the discrimination parameter and b_i as the difficulty parameter of item i. A usual way to build item banks from which tests are assembled, consists of two phases. First, items are constructed and, through pretests, data are gathered to estimate the item parameters. In the next phase, the items are selected for the tests while the item parameters are considered to be known without any error. Purpose of this selection is the assembly of a set of tests with minimal error of measurement, against minimal effort as expressed in, for example, test length.

A statistical property of an item is Fisher information, also called the item information function, which for the 2PL model (Hambleton and Swaminathan, 1985) is given by

$$I_i(\vartheta) = \frac{a_i^2 \exp(a_i(\vartheta - b_i))}{(1 + \exp(a_i(\vartheta - b_i)))^2}. \tag{4.2}$$

The variance of the ability estimator $\widehat{\vartheta}$ is asymptotically equal to the inverse of the TIF value evaluated at ϑ. Under the assumption of local independence between item scores, the TIF is the sum of the information functions of the items in the test:

$$I(\vartheta) = \sum_i I_i(\vartheta). \tag{4.3}$$

As a general aim of tests is to measure candidates abilities as accurately as possible, the variance of $\widehat{\vartheta}$ should be minimised and, hence, the TIF should be maximised. Note that not only the TIF has a relation with the error of measurement, also the difficulties of the items has a relation with the TIF. As the item information function reaches its maximum at $\vartheta = b_i$, the joint difficulties of the items control the shape of the TIF.

Models for parallel test assembly can be described in a rather straightforward way: Assuming an item pool of size L, assemble J tests with the same test characteristics but with no items in common. Boekkooi-Timminga (1990) was the first to formulate such a model, using decision variables x_{ij} to indicate whether item i is selected for test j. The model formulated by van der Linden and Adema (1998), which allowed other restrictions in their model as well, is the basis of the PARIMAX model formulated below. Let variables x_{ij} denote the decision variables indicating whether an item is selected in test j or not. I_{ik} is the item information in ability point ϑ_k, while T_{kj} is defined as the target information for test j at this point. Note

that formulating the TIF target for each test separately gives the possibility to specify different characteristics for each test. Coefficients q_{in} are the resource parameters of item i, indicating how many resources are needed to use it in the test, c_{im} the classification parameters having value 1 if item i belongs to category m and 0 otherwise. Q_{nj}, C^{ℓ}_{mj} and C^{u}_{mj} are the desired use of resources and number of items in the classification categories for test j, respectively:

$$\text{maximise} \quad y \tag{4.4}$$

$$\text{subject to:} \quad y \leq \frac{\sum_i I_{ik} x_{ij}}{T_{kj}} \qquad \forall k, j \tag{4.5}$$

$$\sum_i q_{in} x_{ij} \leq Q_{nj} \qquad \forall n, j \tag{4.6}$$

$$C^{\ell}_{mj} \leq \sum_i c_{im} x_{ij} \leq C^{u}_{mj} \qquad \forall m, j \tag{4.7}$$

$$p_r(x) = 1 \qquad \forall r, j \tag{4.8}$$

$$\sum_j x_{ij} \leq 1 \qquad \forall i \tag{4.9}$$

$$x_{ij} = \begin{cases} 1, & \text{item } i \text{ in test } j \\ 0, & \text{else} \end{cases} \qquad \forall i, j. \tag{4.10}$$

The PARIMAX model formulated in (4.4) - (4.10) expresses the wish of the test assembler to produce a series of tests that maximise the TIF at selected ability levels, given desired shapes in the TIF targets in (4.5), and subject to various restrictions. The restrictions in (4.6) put a limit on the use of certain resources. As can be derived from the additivity of the item information functions, the easiest way to maximise the TIF is to use all items in the pool. So, the resource restrictions in (4.6) limit the resources that the tests use, for example, test length or the total time allotted for test taking. In addition, the restrictions in (4.7) define a desired taxonomic makeup of the tests. Items are classified, for example, with respect to content domains or behavioural aspects. For all these categories minima and maxima are specified. A third group of restrictions in (4.8) concerns the item level. When building large item pools, it is almost inevitable that some items form a relation with others. These relations are referred to as interitem relations. Common examples of these item level relations are enemy sets and testlets. In enemy sets, one item may give away a clue to the answer on the other items, clearly an unwanted situation. Testlets are structures within the item pool that contain several items that should stay together. An example of a testlet is a reading passage with a group of test questions. Theunissen (1996) has shown that more complex relations exist frequently. These can be formulated as well, using Boolean operators $\vee$, $\wedge$ and $\neg$. These relations can be transformed into restrictions based on dif-

ferential payoff functions, denoted by $p_r(x)$, as proposed by De Jong and Spears (1989). A last group of restrictions in (4.9) is added in order to stipulate the requirement of non-overlap, that is, an item is allowed to be used in no more than one test.

The purpose of the model is to maximise the information in the tests at those ability points for which the ratio between the TIF and its target is minimal. Thus, the total information of each test is maximised while adhering to the preferred TIF shapes as much as possible.

These models can be large for programs such as the NIVOR testing program for Dutch as a Second Language, produced by Cito (de Jong, 1998). For each of the four language skills Reading, Writing, Listening and Speaking comprehension, testsets had to be developed for four levels of mastery. The first test was a placement test to assign an incoming learner to a course level. Then the progress of the learners was monitored during the courses by a number of tests. Finally, a certification test concluded the various courses. The placement tests were relatively short and had a broad and flat TIF in order to determine the appropriate course level. The monitoring tests had increasing difficulty to reflect the growing proficiency during the courses. These were short tests with narrow TIFs. The certification tests aimed at deciding whether the candidate had passed or failed the appropriate mastery level as accurately as possible. For this purpose, for each skill and each level an item bank was built with sizes ranging from 400 to 600 items, from which the tests were assembled. In those cases, standard integer linear programming techniques should be applied carefully.

In order to control calculation times, Armstrong (1992) proposed a heuristic for the assembly of parallel test forms. As a first step, a group of items is selected into a seeding test. In practical situations, this might be an existing test. Thereafter, parallel forms are created by minimising the distance between the item parameters of the new forms and the parameters of the seed test. The problem is modelled as a network flow problem. This approach, however, might impose rather strict demands on the item bank from which the items are drawn: Numerous items must be present, identical both in classification as well as in psychometric parameters. A more flexible approach in which items are selected in such a way that their parameters do not have to be identical, but in which the overall test characteristics are, would be welcome. Moreover, incorporation of other types of restrictions is generally not possible with network flow models. On the other hand, Verschoor (2004) has shown that genetic algorithms (GAs) can be applied successfully to test assembly problems while still retaining a large degree of flexibility.

The mapping between decision variables x_{ij} and the chromosomes in a GA is straightforward: Each variable forms a gene with alphabet $\{0, 1\}$. Furthermore, Verschoor (2004) has found that a population of 100 individuals, mate selection proportional to fitness, uniform crossover, and a survival scheme in which the best unique individuals survive, is an efficient

combination to solve the class of test assembly models. The fitness function is comprised of the objective function and a penalty function related to the restrictions, in such a way that only positive values are possible:

$$f(x) = \frac{y}{1+g(x)} = \frac{\min\limits_{k,j}\left\{\frac{\sum_i I_{ik}x_{ij}}{T_{kj}}\right\}}{1+g(x)} \tag{4.11}$$

$$\begin{aligned} g(x) = {} & \lambda \sum_{n,j} h\left(\sum\nolimits_i q_{in}x_{ij} - Q_{nj}\right) \\ & + \mu \sum_{m,j} h\left(C^{\ell}_{mj} - \sum\nolimits_i c_{im}x_{ij}\right) + \mu \sum_{m,j} h\left(\sum\nolimits_i c_{im}x_{ij} - C^{u}_{mj}\right) \\ & + \varphi \sum_{r,j} 1 - p_r\left(x\right) + \xi \sum_{i} h\left(\sum\nolimits_j x_{ij} - 1\right) \end{aligned}$$

$$h(u) = \begin{cases} u, & u > 0 \\ 0, & u \leq 0 \end{cases} .$$

Coefficients λ, μ, φ and ξ denote the penalty multipliers that are determined dynamically similar to the strategy of Siedlecki and Sklanski (1989). Consider for τ consecutive iterations the individuals with the highest fitness. If for all these τ individuals all resource restrictions are met, multiply λ by $1-\delta$. If for all individuals some resource restrictions are violated, multiply λ by $1+\varepsilon$. Leave λ unchanged in all other cases, that is, that for some individuals all resource restrictions are met while for other individuals there are violated restrictions. The same rule is followed for the other penalty multipliers μ, φ and ξ.

Variations to the PARIMAX model can easily be formulated in cases, for example, where a limited overlap is allowed. Define test overlap V_{jl} as the maximum number of items to be included simultaneously in tests j and l. Next to the test overlap, the item exposure must be controlled to prevent items from appearing in too many tests. Define W_i as the maximum number of tests that item i is allowed to appear in. Replace the restrictions in (4.9) by

$$\sum_i x_{ij}x_{il} \leq V_{jl} \qquad \forall j,l \tag{4.12}$$

$$\sum_j x_{ij} \leq W_i \qquad \forall i. \tag{4.13}$$

Penalty function $g(x)$ is modified in order to reflect the incorporation of the overlap and exposure restrictions. Note that the restrictions in (4.12) are nonlinear, but can be linearised by the introduction of a dummy variable z_{ijl} for each combination of item i and pair of test forms j and l, and related restrictions:

$$\sum_i z_{ijl} \leq V_{jl} \qquad \forall j, l \tag{4.14}$$

$$z_{ijl} \geq x_{ij} + x_{il} - 1 \qquad \forall i, j, l. \tag{4.15}$$

This, however, would result in the growth of the complexity of the model to such an extent that for sizeable item pools it is questionable that problems based on this linearisation can be solved within reasonable time limits.

4.3 Compact Coding

As can be seen above, using the conventional model formulations implies the introduction of many variables as well as restrictions on overlap control. This causes a significant increase in problem size compared to single test assembly. Various approaches have been studied in other fields to investigate the efficiency of alternative representations (Hornsby and Pollack, 2001; Rothlauf and Goldberg, 2003; Toussaint, 2005), leaving no firm conclusion whether alternative representations improve the performance. If certain restrictions are made redundant, the performance might be improved.

If no overlap between the tests should be allowed at all, there is an alternative problem formulation that uses a special representation scheme for the chromosomes. Define a coding scheme that maps gene x_i to the test assembly model:

$$x_i = \begin{cases} 0, \text{ if item } i \text{ is not selected} \\ 1, \text{ if item } i \text{ is selected in test 1} \\ 2, \text{ if item } i \text{ is selected in test 2} \\ 3, \text{ etc.} \end{cases}.$$

Now, the chromosome length is equal to the size of the item pool, one gene mapping onto a single item and vice versa, while a mapping directly based on the decision variables x_{ij} would cause the chromosome length to be the number of items times the number of test forms to be assembled. On the other hand, the alphabet should accommodate all possible decisions regarding an item: values 1, ..., J to designate the test form for which the item is selected, or 0 in case the item is not selected at all. In this way, overlap is not possible and the corresponding restrictions in (4.9) are redundant. The other restrictions in (4.4) - (4.8) remain unchanged.

Even though it may be expected that a GA based on compact coding would consume considerably more time than for a similar single test assembly problem using a comparably large item pool, compact coding might deliver results faster than traditional coding. Especially in the early phase of the optimisation process, compact coding might have a distinct advantage over traditional coding as the majority of restrictions have become redundant.

4.4 Epistasis

Since the fitness is based upon y, and thus on the combination of ϑ_k and test j for which y is minimal, it will be no surprise that the fitness function is epistatic if no precautions are taken. All feasible solutions that have the same minimum test in common have the same fitness. The individual with the highest fitness, and thus with the highest minimum test, will have the greatest probability to procreate and to survive. Within a few iterations all individuals will be based upon the same minimum test having only differences in the other tests. From that iteration on, all individuals have the same fitness. With all differences located in the non-minimum tests, a search direction for improvement has been lost, and the population has converged to a local maximum.

An obvious way to reduce the epistasis is to allow small differences in fitness. This way, every solution will have a unique fitness and the search direction will be restored. These differences should have some meaningful value in order to propagate the optimisation process.

Consider two different feasible solutions with equal fitness. These two candidate testsets share the same minimum test, while differences are located in (at least one of) the non-minimum tests. It is easy to see that, given these two testsets, it is easier to improve the minimum test, and hence the fitness, and retain feasibility by an item migrating from a high non-minimum test to the minimum test than by migrating from a low non-minimum test. Even if there is no direct reason to do so, the fitness function should be based on at least one non-minimum test in order to favour higher non-minimum tests.

Two strategies can be considered to overcome the epistasis:

- The use of a reward scheme that involves the non-minimum tests in the fitness. For the PARIMAX model, this reward scheme could be devised as

$$f(x) = \frac{y + \gamma \sum_j y_j}{1 + g(x)}$$

whereby restrictions in (4.5) are replaced by

$$\begin{aligned} y &\leq y_j && \forall j \\ y_j &\leq \frac{\sum_i I_{ik} x_{ij}}{T_{kj}} && \forall k, j. \end{aligned}$$

Solutions that have high non-minimum tests will get a higher reward than those with relatively low non-minimum tests. The purpose of a reward scheme is to favour those individuals that have a higher chance to produce offspring with more favourable objective function values. When the non-minimum tests have a high TIF, it will be easier for crossover and mutation to establish an exchange of items so that the minimum test is improved, than when the non-minimum tests have a low TIF. The reward, however, should be chosen carefully. A small increase in the TIF of the minimum test, and therefore in objective, should be more profitable than an increase in TIF of the non-minimal tests.

- The use of several alternating fitness functions in successive iterations. A cycle of iterations is formed in which all fitness functions are evaluated sequentially. If the cycle is very short, only individuals that do well according to all fitness functions will survive. This approach can be regarded as the introduction of seasons, analogous to seasons in biology. Individuals that do well in all seasons have a larger chance of survival than individuals that thrive in one season but encounter problems in the other. For multiple test assembly, a cycle of two seasons can be used. In the odd iterations, the original fitness function as described in (4.11) is evaluated while in the even iterations a fitness function is evaluated that also takes the non-minimum tests into account. Define this alternative fitness as

$$f^*(x) = \frac{\sum_j y_j}{1 + g(x)}.$$

In effect, individuals that have a high TIF for the minimum test as well have a high information function for the other tests will be favoured. This approach has the advantage of being more robust than the reward scheme, since too high a reward could interfere with the optimisation process. The disadvantage, however, is that in half of the iterations a fitness function is evaluated that has only an indirect bearing on the objective function. Therefore, the process might slow down somewhat and more iterations are needed to reach good solutions.

4.5 Simulations

Simulations have been conducted in order to investigate the effectiveness of the various approaches in preventing epistasis and solving the test assembly problems. Two questions have to be answered: What strategy is most effective in solving test assembly problems: reward, seasons, or a combination thereof? Second, does compact coding accelerate the optimisation process in case of non-overlap?

To answer these questions, three different test assembly problems were used. All three problems were based upon an item pool consisting of 500 items with simulated parameters according to the two-parameter model with $\log(\alpha) \sim N(0, 0.4)$ and $\beta \sim N(0, 1)$. All items were classified on two different dimensions. The first dimension consisted of 4 categories labelled as 101, 102, 103 and 104. These categories were filled with approximately 250, 125, 85 and 40 items, respectively. The second dimension contained categories labelled as 201, ..., 210, which contained approximately equal numbers of items.

The simplest test assembly problem, Problem 1, involved the assembly of two parallel tests. For each test, four restrictions on the shape of the TIF ($\vartheta_1 = -1.5$, $\vartheta_2 = -0.5$, $\vartheta_3 = 0.5$, $\vartheta_4 = 1.5$ with $T_{\vartheta_1} = 4$, $T_{\vartheta_2} = 8$, $T_{\vartheta_3} = 8$, $T_{\vartheta_4} = 4$) and one resource restriction, $\sum_i x_{ij} \leq 40$, were defined. No content restrictions based on the classification structure described above were used and item overlap was not allowed. Problem 2 was an extension of Problem 1, in that it involved assembly of three parallel tests with a similar TIF as in Problem 1 with zero overlap. All tests had 40 classification restrictions. From each combination of two categories, the first from the range 101, ..., 104, and the second from the range 201, ..., 210, exactly one item was required.

Furthermore, the two coding schemes were combined with the four combinations of epistasis prevention strategies, resulting in eight different conditions. The optimisation process was stopped after $LK \log(LK)$ iterations, where L is the number of decision variables, and K is the total number of restrictions, excluding the overlap restrictions. Verschoor (2004) has shown that in general for test assembly models this stopping criterion gives satisfactory solutions. Thus, Problem 1 was stopped after 92103 iterations while Problem 2 was stopped after 750592 iterations, after which the best solution found so far was presented. For Problem 1, 400 replications were performed. For Problem 2, this number was 200. In the dynamic penalty adaptation scheme, τ was chosen to be 80, while δ was set equal to 0.03, ε to 0.02, while reward γ was equal to 0.0001.

The average best fitnesses and their standard deviations are presented in Table 4.1. As all strategies gave feasible solutions in all replications and all fitnesses were based on feasible solutions, it follows that $f(x)T_{\vartheta_k}$ is a lower bound on the TIFs of solution x realised at ϑ_k. Therefore, $\frac{1}{\sqrt{f(x)T_{\vartheta_k}}}$ is an

TABLE 4.1. Best Fitness Function Values

	Best Fitness			
	Problem 1		Problem 2	
Strategy	M	SD	M	SD
None-Binary	3.911	0.025	3.105	0.038
None-Compact	3.836	0.050	3.022	0.072
Reward-Binary	3.919	0.018	3.101	0.037
Reward-Compact	3.844	0.054	3.016	0.085
Seasons-Binary	3.914	0.017	3.101	0.022
Seasons-Compact	3.902	0.011	3.123	0.029
Both-Binary	3.869	0.029	3.101	0.021
Both-Compact	3.899	0.011	3.122	0.023

upper bound on the standard error of measurement at ϑ_k. Note that this argument does not hold for the strategies involving a reward: In the best solutions on which the data in Table 4.1 are based, a total reward in the order of magnitude of $0.001 - 0.002$ was observed, and this reward should be subtracted from the fitness before estimations of the TIFs can be made.

Furthermore, it can be seen that there is no strategy that is clearly best overall. For Problem 1, a relatively small problem, compact coding offered no advantages over binary coding. For Problem 2, a larger problem, compact coding performed better in combination with either the seasons approach or with both seasons and reward scheme.

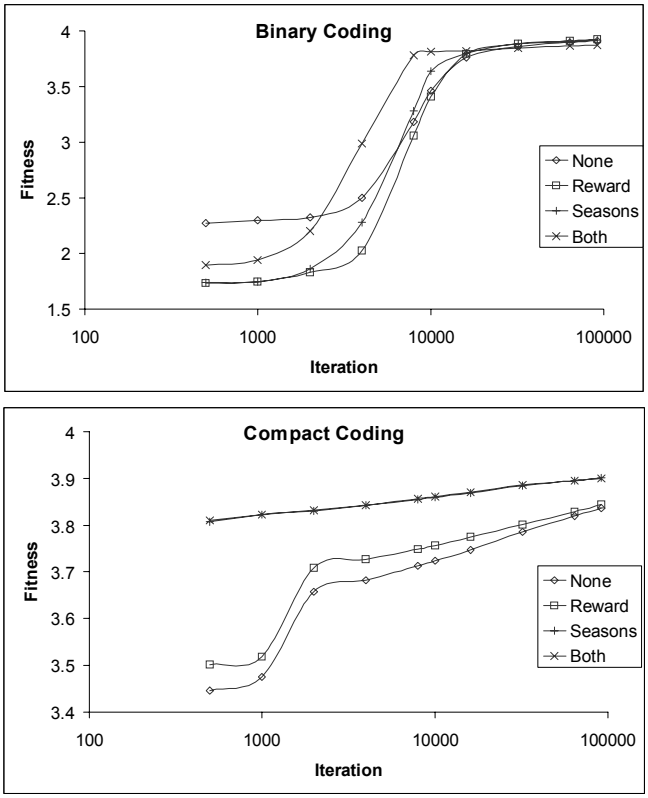

FIGURE 4.1. Average Best Fitness during Optimisation after Reaching Feasibility (Problem 1)

Figure 4.1 shows the average fitness of the best solution during the iterations. It can be seen that compact coding gave good solutions much faster than binary coding. While in Problem 1 all strategies yielded feasible solutions within 500 iterations, the average fitness for the strategies that involved seasons was over 3.8. For binary coding, these fitness values were reached after 16000 iterations.

Note that for compact coding the performance of the seasons approach coincided almost entirely with the combination of both approaches. Apparently, adding a reward scheme to the seasons approach had no effect for compact coding. It is interesting to see that this was different for binary coding. In the first 10000 iterations, the combination performed better than either single approach, after which the other strategies caught up and eventually performed somewhat better.

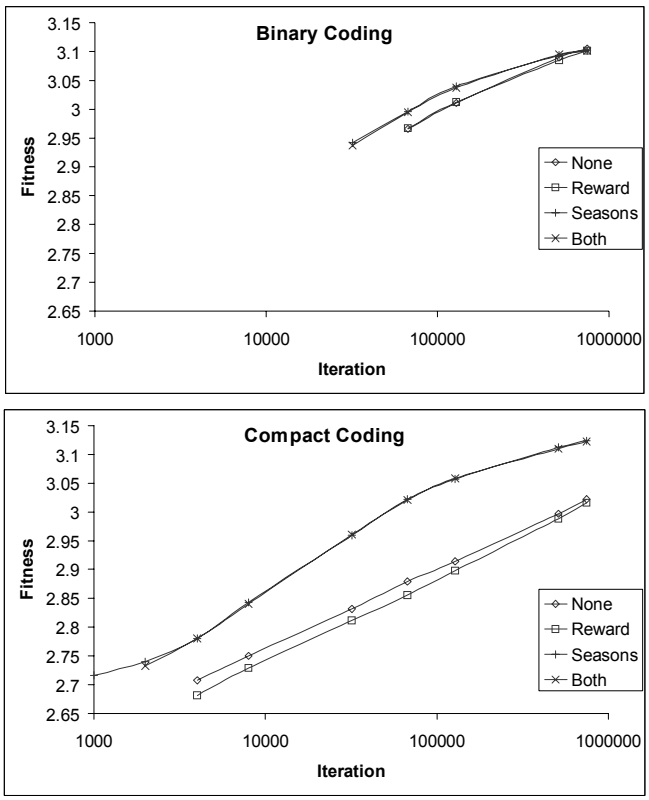

FIGURE 4.2. Average Best Fitness during Optimisation after Reaching Feasibility (Problem 2)

Figure 4.2 shows that for Problem 2 the difference between binary coding and compact coding was even larger than for Problem 1. The GA with compact coding found feasible solutions within 2000 iterations for the strategies involving seasons, but it lasted 32000 iterations before all replications of the binary coding with seasons found a feasible solution, and it took 64000 iterations without seasons. From that point, however, the best fitnesses that were found for the seasons approach were of the same order of magnitude as those for the compact coding scheme. For the approach without seasons, binary coding produced slightly better solutions than compact coding at this stage. Note that for Problem 2, the results for the seasons approach coincided almost fully with those for the combination of approaches. This could be observed not only for compact coding, as with Problem 1, but also for binary coding. Even more, also the performance for using no scheme coincided with the reward scheme in case of binary coding. In case of com-

pact coding, the reward scheme performed even slightly worse than using no scheme at all.

4.5.1 A Testset with Overlap

The third problem in the simulation studies concerned the assembly of three tests with different characteristics, similar to the case of the monitoring tests in the NIVOR testing program. All three tests had length 40 and were subjected to the same content restrictions: Category 101 had to be represented by 15 items, category 102 and 103 by 10 items each, and category 104 by 5 items. The tests had an increasing difficulty in order to represent a growth in ability level over the test administrations. For each test, a TIF target was defined at three ability points with $T_{\vartheta_1} = 8$, $T_{\vartheta_2} = 8$, $T_{\vartheta_3} = 4$, while the ability points varied for each test. For test 1 these were: $\vartheta_1 = -1.5$, $\vartheta_2 = -0.5$, and $\vartheta_3 = 0.5$; for test 2: $\vartheta_1 = -1.0$, $\vartheta_2 = 0$, and $\vartheta_3 = 1.0$, and for test 3: $\vartheta_1 = -0.5$, $\vartheta_2 = 0.5$, and $\vartheta_3 = 1.5$. The test overlap was restricted to a maximum of four items and each item was allowed to appear in at maximum two tests.

Since Problem 3 allowed for overlap between the tests, compact notation could not be used. Thus, four epistasis prevention strategies were simulated: the reward scheme, the seasons approach, both, and using no strategy. Similar to Problem 1 and Problem 2, the optimisation process was stopped after $LK\log(LK)$ iterations, where L is the number of decision variables, and K is the total number of restrictions, excluding the overlap restrictions. Problem 3 was stopped after 588424 iterations, after which the best solution found so far was reported. For each strategy, 200 replications were performed. In the dynamic penalty adaptation scheme, τ was chosen to be 80 while δ was 0.03 and ε was 0.02.

The average best fitnesses and their standard deviations are presented in Table 4.2.

TABLE 4.2. Best Fitness Function Values for Problem 3

	Best Fitness	
Strategy	M	SD
None-Binary	4.413	0.106
Reward-Binary	4.449	0.116
Seasons-Binary	4.495	0.032
Both-Binary	4.482	0.045

From Figure 4.3 it can be seen that the strategies did not differ very much in effectiveness. The strategies involving seasons performed somewhat worse during the first 10000 iterations, after which they performed somewhat better than the strategies without seasons.

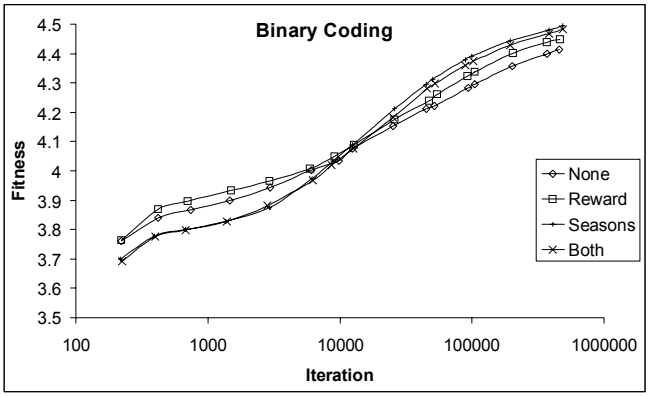

FIGURE 4.3. Average Best Fitness during Optimisation after Reaching Feasibility (Problem 3)

The question which strategy prevents epistasis best for the assembly of testsets cannot be answered simply. If the optimisation is stopped after a relatively large number of iterations, as was the case in our simulation study, using a reward scheme seems to be the best choice for relatively simple models like Problem 1. For more complicated models, such as Problem 2, compact coding combined with either the seasons approach or both seasons and reward schemes, seemed to be the favourable choice. In cases where overlap restrictions do not allow the use of compact coding, such as in Problem 3, the seasons approach seemed to perform best.

From a practical point of view, however, the stopping criteria seemed to be rather strict, and it might be preferrable to give a reasonably good solution at an earlier moment. When the optimisation is stopped at an earlier moment, compact coding combined with seasons or with both the seasons and reward schemes seemed to be the best choice.

4.6 Conclusions

A model for the assembly of testsets was proposed in this study. The idea behind the model is the observation that in some testing programs, sets of test forms are developed all originating from a common item pool, and where a limited amount of overlap is allowed. The test specifications may vary between test forms, as each may serve a different purpose within the testing program. It is shown that a genetic algorithm is capable of solving these models efficiently. Especially if a limited overlap is allowed, the resulting test assembly models are either nonlinear or need many extra dummy

variables and restrictions, which may cause algorithms traditionally used in the field, based upon integer linear programming, to fail.

Should no overlap between tests be allowed, a compact coding using a more elaborate alphabet improves the performance significantly, especially when combined with an approach that mimics seasons, evaluating different fitness functions in consecutive iterations. Compact coding is an effective method in preventing epistasis inherent to the model formulations of the assembly of testsets.

5

Preventing the Bankruptcy of an Item Bank

5.1 Introduction

The popularity of computer based testing (CBT) and automated test assembly (ATA) has created various opportunities for testing agencies to improve the cost-effectiveness of their "silverware" by intensifying the use of test items over a prolonged period of time. For this improved use, testing agencies have been investing considerable amounts of money into the development of item banks. These investments are justified by the foreseen extended use of the items in these banks. Therefore, it is not surprising that attention has recently been paid to the problem of item bank management to safeguard long-term item bank quality, and thus to maximise the return on the investments that have been made.

Recent research by Way and Steffen (1998), Belov and Armstrong (2004) and Ariel (2005) focused either on item bank design or on training and selection of item writers to construct items with favourable characteristics. In some testing programs, however, the possibilities to implement these measures might be limited, for example, by budget restrictions. Moreover, optimisation techniques, whether performed by hand or by using ATA models and accompanying software, will select the most favourable items that are available. Therefore, these methods put a pressure on the item construction process to construct more items of even better quality than before. As test assembly methods continue to select the best available items, improvement of the quality of newly constructed items will not prevent this problem. The quality of the tests might improve but depletion of high-quality items

in the item bank continues to occur. In fact, as ATA models are likely to be more efficient in identifying favourable items than manual optimisation techniques, this pressure on item construction tends only to increase when ATA is introduced.

Therefore, in order to maintain a uniform quality of an item bank, test assembly must be harnessed in such a way that the quality of the items used in the tests does not exceed the expected quality of newly constructed items or previously used items that are returned to the bank. Several strategies to harness the assembly of tests are proposed in this paper and a simulation study is conducted to evaluate these strategies in the context of two testing programs for which item bank management is a vital issue: the Unified State Examinations (USE) for secondary education in the Russian Federation, and the State Examinations for Dutch as a Second Language (DSL) in the Netherlands.

5.1.1 The Russian Unified State Examinations

Started in 2001, the USE program is gradually being introduced in the Russian Federation. Nationwide, schools for secondary education are in the process of adapting to the school leaving and university entrance examinations in the program, providing their students with test results comparable all over the country (Russian Federation Ministry of Education and Science, 2005). The Federal Institute for Pedagogical Measurement (FIPI) develops the examinations. Approximately 700,000 students have been tested in 2006, and this number is likely to grow to over 1,500,000 annually as of 2012.

Since the Russian Federation extends over nine time zones, two issues are important for the success of the USE program: test equivalence and item security. It has been decided that 100 parallel test forms will be developed yearly, while each form will be administered to roughly equal numbers of students. A limited overlap between the test forms is acceptable in the USE program. Currently the item banks contain approximately 3000 - 4000 items that have been calibrated under the OPLM model, that is, a 2PL model with integer discrimination indices (Verhelst, 1995). The item banks are planned to be expanded annually with 1000 calibrated items until the target size of 10000 items has been reached.

The parameter distributions of the items in the current USE item banks for Russian and Physics can both be approximated by $\log(\alpha) \sim N(1, 0.36)$ and $\beta \sim N(0, 0.28)$. Analysis of current data has shown that the ability of the population is approximately normally distributed with $\mu = 0.01$ and $\sigma = 0.23$.

5.1.2 Dutch as a Second Language Examinations

While the purpose of the USE testing program is to give an advise on possible further education for a relatively broad population, the tests of the DSL program can be characterised best as certification tests. Each year, approximately 5000 DSL-learners take an examination on either level 1 or 2 of four different language skills: Reading, Writing, Listening, and Speaking. Cito has been commissioned to produce the Listening and Speaking examinations three times a year. The test lengths for these examinations are 40 and 15 items, respectively. Starting in 2007, a new system of examinations will be implemented, giving candidates higher flexibility in taking the examinations. For each skill and level combination, an item bank is developed that should contain 1000 - 2000 items, from which six parallel examinations will be assembled annually. In this case, the test forms should be parallel according to Samejima (1977). Contrary to the USE testing program, no overlap is allowed.

In order to assemble the examinations, ATA methods will be used. The goal is to assemble shorter examinations, preferrably with no more than 30 - 35 items for Listening, while the TIFs will be required to be equal to those of the current examinations. Every year approximately 100 new items will be constructed, pretested, and added to the banks. In addition, items will not be allowed to be used again within two years after administration. In this way, the addition and reuse of items will compensate for the loss of the items that will be used in the examinations or will become outdated.

The current item bank for Listening at level 1 contains 767 items, while the parameter distribution can be approximated by $\log(\alpha) \sim N(1.5, 0.28)$ and $\beta \sim N(-0.17, 0.24)$. As it was decided that the cut-off point on the raw score scale was at 28 out of 40, while simultaneously it was defined to be $\vartheta = 0.089$ on the ability scale of a reference test, the majority of the items are relatively easy in order to accommodate the combination of cut-off points. At the same time, the distribution of the target population was observed to be approximately normally distributed with $\mu = 0.12$ and $\sigma = 0.28$.

5.2 The Dynamics of Item Bank Management

It is assumed that the item bank can be split into two separate item pools: an active item pool from which items may be selected, and a passive item pool containing items that were previously used and for security reasons should not be used again for a certain period of time. Then, the dynamics involved in item bank management can be modelled as two processes that interact between the two pools. The first process comprises of the construction and pretesting of new items, and, if applicable, of the return of items to the active pool. The items from this process form the input of the active

pool. The second process is an output process. After items have been used in a test, they are stored in the passive pool for possible future reuse. In the remainder of this paper, the term item pool refers to the active item pool, unless stated otherwise.

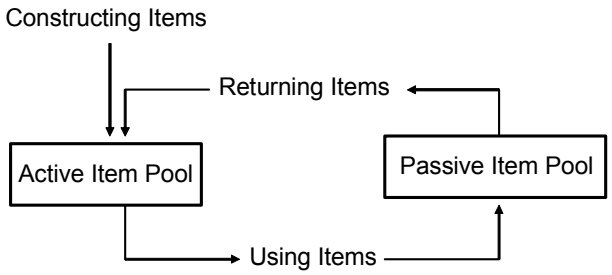

FIGURE 5.1. The Processes of Item Bank Management

Three distributions can be distinguished: the parameter distribution of items from the input process, of those from the output process, and of those in the item pool. Note that the last distribution depends on the other two.

It is clear that the parameter distribution in the input process must at least match that of the output process to avoid depletion and to maintain an item pool that can supply items according to the test specifications over an extended period of time. Strategies for item bank management could be said to be successful if they control the input and output processes such that a stable pool is maintained.

As strategies to improve the input are expected to influence the quality of the tests only, and not to prevent depletion of high-quality items, the input distribution is assumed to be stable throughout this study. Therefore, it suffices to regard either the quality of the tests or the distribution of the items in the item pool to conclude whether an item bank management strategy is successful or not.

5.3 Item Bank Depletion

Before we discuss strategies to prevent item bank depletion, it is necessary to identify the precise nature of the depletion. Which types of items are selected first, and which types of items tend to remain behind when no precautions are taken? For commonly used ATA models, it is easy to determine which items are most favourable. A look at the objective function of the maximin model (van der Linden and Boekkooi-Timminga, 1989) reveals that highly discriminating items are more likely to be selected than lowly discriminating items.

A simple simulation of a large-scale test assembly problem, resembling the case of the Physics examinations in the USE program, confirms the favourableness of highly discriminating items. From an item pool consisting of 10000 items with parameter distribution of $\log(\alpha) \sim N(1, 0.36)$ and $\beta \sim N(0, 0.28)$, a number of tests was assembled. Annual groups of 100 tests were assembled sequentially for a target population with ability distribution $\vartheta \sim N(0.01, 0.23)$. Each test contained 30 items and required a maximum overlap of three items with any other test within its group, and no overlap with other tests. Each item should appear in no more than two tests. The PARIMAX model was used to formulate the test assembly problem involved. The test information function (TIF) target was defined to be equal at three ability points $\vartheta_1 = -0.34$, $\vartheta_2 = 0.0$, and $\vartheta_3 = 0.34$. Equations (5.1) through (5.6) specify the test assembly problem used. $I_i(\vartheta_k)$ is the information function value of item i at point ϑ_k, while x_{ij} is the decision variable indicating whether item i is selected in test j:

$$\begin{aligned}
\text{maximise} \quad & y && && (5.1)\\
\text{subject to:} \quad & y \leq \sum_i I_i(\vartheta_k) x_{ij} && \forall k, j && (5.2)\\
& \sum_i x_{ij} \leq 30 && \forall j && (5.3)\\
& \sum_i x_{ij} x_{il} \leq 3 && \forall j, l && (5.4)\\
& \sum_j x_{ij} \leq 2 && \forall i && (5.5)\\
& x_{ij} = \begin{cases} 1, & \text{item } i \text{ in test } j \\ 0, & \text{else} \end{cases} && \forall i, j. && (5.6)
\end{aligned}$$

The tests were assembled through the DOT 2005 software (Verschoor, 2005), while the depletion of the item bank was investigated through the parameter distribution of the items remaining in the pool. These specifications were observed to require between 1500 and 1600 items annually, thus an item pool of 10000 items could support the test demand for six years before new items had to be added or previously used items had to be returned. The assembled tests were evaluated using two criteria: the height of the TIF values on the three ability points and the average standard error of measurement for the target population:

$$\omega = \int_{-\infty}^{\infty} \frac{g(\vartheta)}{\sqrt{I(\vartheta)}} d\vartheta. \tag{5.7}$$

Table 5.1 contains the TIF values at the specified ϑ's, averaged over the 100 tests per year, as well as the average ω-values. Next to the average TIF

TABLE 5.1. Average TIF-values and ω-values (USE Physics)

Year	$I(\vartheta_1)$		$I(\vartheta_2)$		$I(\vartheta_3)$		ω	
	M	SD	M	SD	M	SD	M	SD
1	91.7	1.3	122.8	6.9	91.5	1.4	0.102	0.001
2	66.5	0.6	95.7	1.0	66.6	0.6	0.118	0.001
3	51.6	0.3	64.5	0.0	51.6	0.3	0.136	0.000
4	45.8	0.5	49.1	0.7	45.1	0.3	0.148	0.001
5	26.6	0.0	29.8	0.0	26.6	0.0	0.192	0.000
6	25.5	0.1	28.1	0.0	25.6	0.1	0.196	0.000

values, the standard deviations are reported, giving an indication of to what extent the tests can be assumed to be parallel. It can be clearly seen that over six years, the average height of the TIF decreases to approximately 25% of the average TIF in the first year. At the same time, the average ω almost doubled. Thus, it could not be maintained that the tests in the last year were parallel to the tests in the first year, and signs of depletion were obvious.

Figure 5.2 shows the distributions of the a- and b-parameters of the available items at the start and directly after assembly of the tests for the second, fourth and sixth year. It can be observed that items with high discrimination parameters are favoured, thus depleting the pool with respect to these items. This effect is in line with similar findings for computerised adaptive tests, as were found, for example, by Lord (1980). As a result, the lower discriminating items tended to remain, from which only low-informative tests could be assembled. A second observation is that no significant depletion with respect to item difficulty occurred. For every difficulty level a reasonably large choice of items remained available during the entire test assembly process.

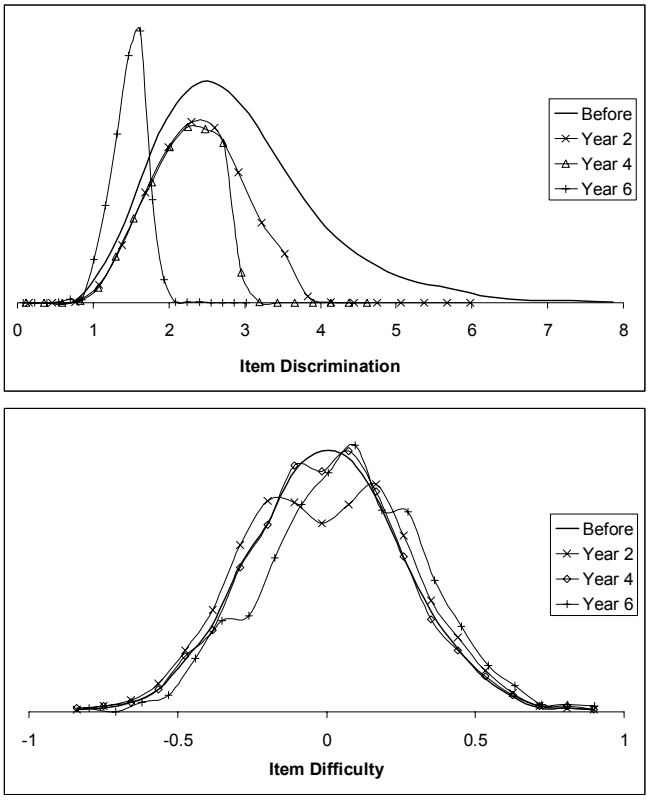

FIGURE 5.2. Parameter Distributions (USE Physics)

A second simulation, resembling the Listening examinations of the DSL program, was conducted. The current examinations have, on average, a TIF value of 122.6 at the cut-off point. The test specifications require a TIF target defined at two ability points. The first point is the cut-off point at which a TIF target of 122.6 should be reached, and the second point was chosen at -0.15, with an equal target. This results in a TIF that will be high in the interval ($-0.15 \leqslant \vartheta \leqslant 0.089$), and a cut-off score at approximately 70% of the maximum raw score.

Thus, for the simulations the target was defined to be 122.6 at $\vartheta_1 = -0.15$ and $\vartheta_2 = 0.089$, while the test length was minimised. Test overlap was not allowed. Equations (5.8) through (5.12) specify the test assembly problem:

$$\begin{aligned}
&\text{minimise} && y && && (5.8)\\
&\text{subject to:} && y \geq \sum_i x_{ij} && \forall j && (5.9)\\
& && \sum_i I_i(\vartheta_k) x_{ij} \geq 122.6 && \forall k, j && (5.10)\\
& && \sum_j x_{ij} \leq 1 && \forall i && (5.11)\\
& && x_{ij} = \begin{cases} 1, \text{ item } i \text{ in test } j \\ 0, \text{ else} \end{cases} && \forall i, j. && (5.12)
\end{aligned}$$

Every year 100 items were added to the item bank drawn from the target parameter distribution: $\log(\alpha) \sim N(1.5, 0.28)$ and $\beta \sim N(0, 0.24)$. At the same time, every year 50 items were removed because they were no longer considered to be appropriate. Although in the DSL program it is allowed to reuse items after two years, reuse was not allowed in the simulations at all.

TABLE 5.2. Average TIF-values and ω-values (DSL)

Year	$I(\vartheta_1)$		$I(\vartheta_2)$		ω		Length
	M	SD	M	SD	M	SD	
1	130.2	3.5	129.2	1.0	0.143	0.004	12
2	125.1	1.2	125.1	1.4	0.124	0.002	17
3	131.9	3.4	125.6	0.4	0.108	0.002	22
4	124.6	1.9	123.6	1.1	0.107	0.001	27
5	136.4	4.4	124.7	0.7	0.102	0.002	30

In Table 5.2, it can be seen the test length rose rapidly over the years, while the average TIF values remained rather stable. At the same time, the average standard error of measurement, ω, dropped significantly over the years, despite the fact that the TIF at the specified abilities was almost constant. These signs indicate that large differences in information might be found at extreme abilities, and hence that the tests might not be parallel at these ability levels. If the differences between the TIFs are deemed unacceptable, it might be wise to specify the target at more ability points in order to decrease these differences. But even then, it might be expected that the test length will have to rise over time in order to maintain stable

values for both the TIF and ω.

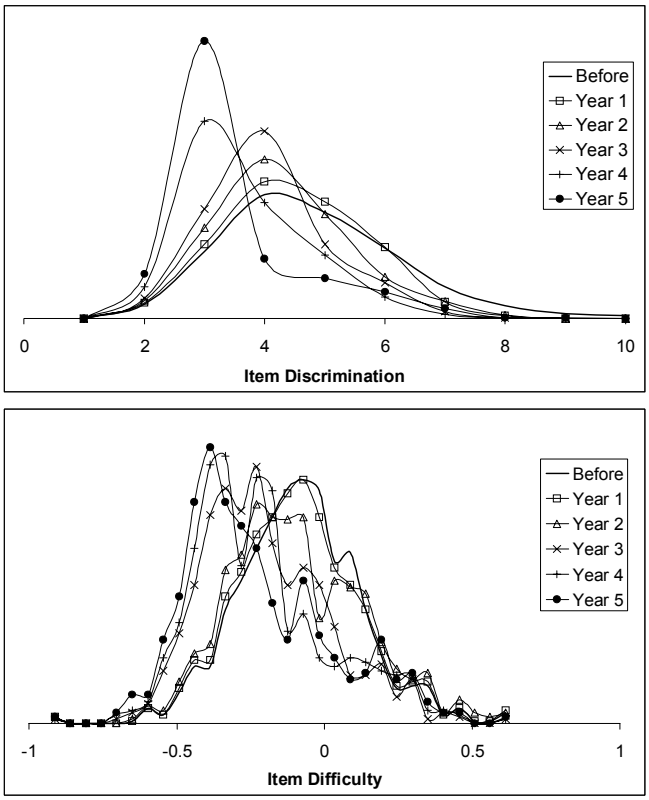

FIGURE 5.3. Parameter Distributions (DSL)

Figure 5.3 shows the distribution of the a- and b-parameters of the available items at the start and after test assembly each year. Also for the DSL simulations, items with high discriminations were favoured and tended to be selected early on. Contrary to the absence of depletion with respect to item difficulty in the simulations for the USE program, Figure 5.3 shows that the test assembly process had a tendency to select the somewhat more difficult items, that is, those with a b-parameter near the cut-off ability. Despite this tendency, still sufficient items from all difficulty levels remained available.

5.4 Item Bank Management Strategies

Item bank management strategies should prevent the depletion of item banks with respect to highly discriminating items. The following four actions could be considered:

Stratification. The first strategy is stratification of the item pool according to the discrimination parameters of the items. By adding restrictions to the test assembly model, the number of selected items from every stratum can be restricted to be proportional to the number of available items in the pool. Thus, an even use of items from various discrimination categories will be enforced. As a result it would be plausible that the parameter distribution in the output process is equal to the parameter distribution in the pool and the latter distribution remains stable.

Subdivision. The second strategy is a random subdivision of the entire item pool into several pools. Each year a pool will be selected to serve as the current pool. The size and number of these pools can be determined by observing how many items are needed for solutions without any item bank management. Each current pool should have this size in order to fulfill the demands for test assembly each year, and the number of years that the testing program could be supported before depletion will occur is thus equal to the number of pools that can be constructed. In every item bank, however, lowly informative items, which do not contribute substantially to the information in the tests, can be found. If the pool sizes are increased, more of these lowly informative items can be left unused. More highly informative items would be selected, resulting in higher-informative tests. Reversing this argument reveals the dilemma that test assemblers are confronted with: In order to assemble more informative tests, the pool sizes should be increased, shortening the life span of the bank.

An alternative subdivision, through parallel item pool assembly, could be considered if differences between the tests appear to be unacceptably high when a random subdivision is used.

Shadow pool. The third item bank management strategy is the big-shadow-test method proposed by van der Linden (2005). In this method, the items are selected either directly in the current tests or in a shadow test. The items in the shadow test are preserved and after test assembly they are returned to the item pool. In the current application the big shadow test can be considered as a shadow item pool.

Similar to the decision on the pool sizes in case of random subdivision, a decision must be made on the size of the shadow item pool. The fewer items are accepted to be left unused, the larger the shadow pool

should be. Once the size of the shadow pool has been determined, the restrictions can be assumed to be proportional to the sizes of the current tests, while no overlap between the shadow pool and the current tests is allowed.

Alternative IRT model. The fourth item bank management strategy is using an alternative IRT model that does not allow for any differences in item discrimination: the Rasch model. When no differences in item discrimination are recognised, then by definition no depletion of highly informative items could occur, thus avoiding the management problems completely. There is a remark to be made, however. It may be observed in practice that using the Rasch model may result in the rejection of 10 - 15% of the available items during calibration due to poor item fit. Many of those items would fit the 2PL-model because the only deficiency is that their discriminations are too high or too low for the Rasch model.

5.5 Simulations

A series of simulations was performed to investigate which strategy prevents depletion most efficiently. Three scenarios were used for the two testing programs. Scenario 1 and 2 were used to investigate the proposed item bank management strategies for the USE program and were based on the previously simulated item bank. Scenario 1 represented the situation in the near future, when the item bank is still under development. At the start, 5000 items were available, to which every year 1000 new items were added. Scenario 2 represented a situation in the longer future, without structural item development. All 10000 items were available at the start, while no replacement or extension was assumed. In both scenarios, 100 tests were assembled annually: Each test consisted of 30 items, while the test overlap was restricted to three, and the item exposure restricted to two. Overlap between tests of different years was not allowed. The TIF target was defined to be equal at three ability points $\vartheta_1 = -0.34$, $\vartheta_2 = 0.0$, and $\vartheta_3 = 0.34$.

Scenario 3 simulated the DSL program. At the start of the simulations, 767 calibrated items were available, to which 100 items were added annually to the item bank drawn from the target parameter distribution: $\log(\alpha) \sim N(1.5, 0.28)$ and $\beta \sim N(0, 0.24)$. At the same time, 50 items were removed annually to represent that they were considered to be outdated. Every year, six tests with no overlap were assembled.

The aim of these simulations was to investigate which strategy prevents depletion most efficiently. Therefore, the following conditions were studied:

- No action was undertaken with respect to depletion. This condition served as the base line condition.

- The items were stratified according to their discrimination. In the item bank for the USE program, the range of discrimination parameters was observed to be [1..10]. Five strata were defined:

Stratum		Available Items
1	$\alpha \geqslant 6$	253
2	$\alpha = 5$	564
3	$\alpha = 4$	1573
4	$\alpha = 3$	3624
5	$\alpha \leqslant 2$	3986

Furthermore, four restrictions for every test were added to the test assembly problem in (5.1) - (5.6). In each test, a maximum number of 1, 2, 5 and 10 items were allowed from strata 1 through 4, respectively. As items from stratum 5 were expected to be selected only when no alternatives were available, they could be selected freely. The stratification restrictions were formulated as

$$\sum_i c_{im} x_{ij} \leq C_m \qquad \forall m, j$$

whereby coefficient c_{im} indicates the stratification, having value 1 if item i belongs to stratum m, and value 0 otherwise. C_m is the maximum number of items from stratum m in the tests.

In the DSL bank, the discrimination parameters varied from 2 to 10 and the item bank was stratified using 6 categories:

Stratum		Available Items
1	$\alpha \geqslant 8$	65
2	$\alpha = 7$	101
3	$\alpha = 6$	264
4	$\alpha = 5$	468
5	$\alpha = 4$	550
6	$\alpha \leqslant 3$	318

Five restrictions per test were added to (5.8) - (5.12), imposing maxima on the number of items to be selected from each stratum. As the test lengths were not fixed, these maxima were expressed as percentages of the realised test length, rounded off upwards. (3%, 6%, 15%, 25% and 30% were allowed in strata 1 through 5, respectively.) Selection of items from stratum 6 was not restricted. These restrictions were formulated as

$$\sum_i c_{im} x_{ij} \leq \left\lfloor C_m \sum_i x_{ij} \right\rfloor \qquad \forall m, j$$

whereby C_m is the maximum fraction of items from stratum m in the tests.

- The items were selected from a random subdivision that was slightly larger than the total number of items needed. For USE, every year 1666 items were randomly drawn from the available items, in order to form the current pool. For DSL, the number of items needed was not fixed as the corresponding test specifications were based on a minimisation model. The goal was, however, to assemble tests with TIFs comparable to the existing tests, using a maximum of approximately 30 items each. The aim, therefore, was to use no more than approximately 180 items per year. Based on this situation, the size of the item pools was determined to be 200 items.

- The idea behind the shadow pool method is the distribution of characteristics of the items in the tests is equal to that of the items reserved for future use. Therefore, the shadow pool should contain the vast majority of the items remaining available. The size of the shadow pool was determined at 90% of the remaining items. Thus, the size for year i was $0.9(5000 - 500i)$ for Scenario 1 and $0.9(10000 - 1500i)$ for Scenario 2. For Scenario 3, not the size of the shadow pool had to be restricted, but the target for the TIF had to be determined. The target was $122.6 * 6 * 0.9 * N_i/180$, where N_i denotes the size of the item pool in year i.

- Items were calibrated under the Rasch model. Use of this model resulted in the rejection of a total of 1318 items in the USE bank, and of 103 items in the DSL bank.

For Scenario 1, the results of the simulations are presented in Tables 5.3 and 5.4. The average ω-values in Table 5.3 showed a moderate increase for the Rasch model and the stratification approach, while they remained almost constant for the subdivision and shadow pool methods. In Table 5.4, the minimum TIF values at the three specified ϑ's are presented, averaged over the 100 tests assembled for each year, together with the standard deviation observed across these minima. In accordance with the increase in ω for the Rasch model and the stratification approach, the minimal TIF values showed a decline. Because of the fact that 1318 items had to be rejected for the Rasch model, in year 6 only 77 parallel tests could be assembled instead of the required 100 tests.

The results for Scenario 2 are shown in Tables 5.5 and 5.6. The differences were somewhat larger than for Scenario 1. In Scenario 1, every year a number of items was added according to the original parameter distribution, thus replenishing the highly discriminating items somewhat while Scenario 2 lacked this replenishment. Also in Scenario 2, the subdivision and shadow pool methods showed an output of tests with almost constant properties

TABLE 5.3. Average ω-values for Scenario 1

Year	None	Rasch	Stratification	Subdivision	Shadow
1	0.101	0.146	0.134	0.136	0.136
2	0.129	0.147	0.135	0.136	0.136
3	0.142	0.148	0.142	0.136	0.137
4	0.152	0.151	0.136	0.138	0.137
5	0.155	0.154	0.144	0.137	0.137
6	0.150	0.155$\star$	0.147	0.135	0.135

Note: $\star$: Only 77 tests could be assembled

TABLE 5.4. Average Minimum $I(\vartheta)$ for Scenario 1

Year	None		Rasch		Strat.		Subdiv.		Shadow	
	M	SD	M	SD	M	SD	M	SD	M	SD
1	78.6	0.8	45.1	0.1	51.2	0.1	50.7	0.4	51.8	0.4
2	56.4	2.7	44.5	0.1	51.2	0.3	51.5	0.5	51.4	0.4
3	47.5	0.6	44.0	0.2	49.5	0.4	51.6	0.6	50.8	0.4
4	41.1	0.5	42.6	0.2	50.9	1.2	49.7	0.6	51.2	0.5
5	40.0	0.5	41.2	0.2	47.1	1.5	50.3	0.5	50.5	0.5
6	42.2	0.6	40.3$\star$	0.2	43.3	1.0	52.0	0.6	52.0	0.5

Note: $\star$: Only 77 tests could be assembled

over the years. Note that for both scenarios, the Rasch method did not only fail to assemble the required number of tests in the last year but the height of the TIFs and the ω-values were also unfavourable compared to the other methods. All poorly fitting items were removed, this involved not only items that had a low a-parameter in the 2PL-model and thus would normally be ignored but also items with a high a-parameter for the 2PL-model, which would have contributed substantially to the information in the tests if they had been available.

The simulation results for Scenario 3 are presented in Tables 5.7 and 5.8.

The average ω-values are given in Table 5.7. For all strategies, except subdivision, the ω-values tended to decrease, a sign that the tests might not be parallel over the years. Contrary to Scenarios 1 and 2, the shadow pool method for Scenario 3 showed a slight deterioration. This phenomenon can be explained by the use of the test assembly models: In minimisation models, the test length is not fixed but must be estimated instead. Yet, this number of items was used as the basis to determine the size of the shadow pool, or in case of minimisation models, the target for the TIF of the shadow pool. In hindsight, the numbers of items used annually appeared to

TABLE 5.5. Average ω-values for Scenario 2

Year	None	Rasch	Stratification	Subdivision	Shadow
1	0.102	0.146	0.133	0.136	0.136
2	0.118	0.146	0.134	0.136	0.136
3	0.136	0.147	0.135	0.136	0.136
4	0.148	0.149	0.137	0.138	0.136
5	0.192	0.153	0.141	0.137	0.137
6	0.196	0.165$\star$	0.163	0.135	0.138

Note: $\star$: Only 77 tests could be assembled

TABLE 5.6. Average Minimum $I(\vartheta)$ for Scenario 2

Year	None		Rasch		Strat.		Subdiv.		Shadow	
	M	SD	M	SD	M	SD	M	SD	M	SD
1	91.0	1.2	45.3	0.0	51.2	0.0	50.7	0.4	52.4	0.4
2	65.5	0.6	45.1	0.0	51.2	0.1	51.5	0.5	52.0	0.5
3	51.2	0.1	44.6	0.0	51.4	0.1	51.6	0.6	51.8	0.4
4	45.2	0.3	43.8	0.0	51.4	0.3	49.7	0.6	51.4	0.4
5	26.6	0.0	41.9	0.1	50.2	1.2	50.3	0.5	50.8	0.5
6	25.5	0.0	36.7$\star$	0.1	38.2	0.4	52.0	0.6	49.5	0.5

Note: $\star$: Only 77 tests could be assembled

be smaller than the estimated 200 and an improved estimation might have improved the performance of the shadow pool method. On the other hand, the subdivision method appeared to be more robust against estimation errors for item usage.

5.6 Discussion

Long-term use of items is one of the main reasons for which testing agencies make use of item banks. At the same time, the use of automated test assembly methods poses a threat to the life span of these banks. In this paper it is shown that item banks can be depleted rapidly if no counter measures are taken. Therefore, it is important for the design of an item bank to use methods for prevention of depletion from the start. In the two testing programs discussed in this paper, the reason of depletion was the tendency to select highly discriminating items. Various item bank management methods were considered, two of which appeared to effectively prevent depletion of highly

TABLE 5.7. Average ω-values for Scenario 3

Year	None	Rasch	Stratification	Subdivision	Shadow
1	0.143	0.110	0.121	0.113	0.117
2	0.124	0.110	0.118	0.113	0.114
3	0.108	0.108	0.110	0.114	0.109
4	0.107	0.102	0.109	0.110	0.109
5	0.102	0.099	0.102	0.115	0.109

TABLE 5.8. Average Test Lengths for Scenario 3

Year	None	Rasch	Stratification	Subdivision	Shadow
1	12.0	27.0	20.5	20.3	20.8
2	17.5	27.0	21.7	20.8	22.5
3	22.0	28.7	23.8	20.3	25.3
4	26.7	30.7	25.7	22.2	25.0
5	30.2	35.0	29.3	19.7	24.7

discriminating items: random subdivision of the bank and the shadow pool method. Of these two, random subdivision of the item bank into several item pools, and subsequently selecting one pool as the active pool to assemble the tests from, is the simplest method. If the test specifications do not contain many restrictions, using this method ensures an output of tests of constant quality over the years. Although test specifications with larger numbers of restrictions were not needed in the programs under consideration, and hence were not considered in this paper, it remains to be seen whether this method still performs well.

The shadow pool method performed at least equally well for maximin models. On the other hand, for minimisation models the shadow pool method seemed to be less robust than the random subdivision of the item bank into pools.

6
Epilogue

The wish to construct high-quality tests is the raison d'être of the test assembly models that have been developed over the last two decades. Different practical situations may need different models as well as different algorithms to solve these models. Predictably, large-scale testing programs ask for models of growing complexity, resulting in more complicated and time consuming item selection algorithms. As the concepts of item response theory, such as the test information function, are rather abstract, models using classical test theory are often preferred in practical situations, resulting in nonlinear test assembly models.

Current algorithms, however, are usually based on branch-and-bound methods and not always suitable for solving large-scale or nonlinear models. Therefore, local search algorithms such as genetic algorithms may prove to be a viable alternative, even if they fail to give a proof of optimality. A strong point of genetic algorithms is that they are extremely robust with respect to the structure of the problems, that is, they are capable of solving large-scale integer nonlinear programming problems. In a genetic algorithm, a population is formed by solutions that mate, get offspring, and struggle for survival. A solution, that is, a possible test, or in the case of the models studied in Chapters 4 and 5, a set of tests, is represented by a chromosome. In each iteration, pairs of solutions are selected to mate and reproduce. The crossover operator passes on genetic information encoded in the chromosomes from the parents to the children, while the mutation operator modifies the chromosomes of the children slightly. Thus, it is likely that the children have some resemblance to both their parents. A survival mechanism reduces each new population, which temporarily consists of

both parents and children, to its original size. Both mate selection and survival are controlled by the fitness of the solutions. Quantifying this fitness is an important element in the implementation of a genetic algorithm.

The main topic of this thesis is to design genetic algorithms for automated test assembly. For such algorithms to be successful, they should be able to solve classes of problems that arise in practical testing situations better than current methods.

Two basic test assembly models, together with a genetic algorithm, were proposed in Chapter 2. The focus was mainly to show that a genetic algorithm can successfully be implemented for test assembly problems. It was shown that a combination of mate selection proportional to the fitness of the solutions, uniform crossover, survival of the fittest solutions, and a dynamic penalty scheme to incorporate restrictions gave the best results. Comparison with a commercially available solver showed that for nearly all problems acceptable solutions were found.

In Chapter 3, a model based on classical test theory was presented. Solving the proposed nonlinear CMAX model with a genetic algorithm had two advantages: First, Cronbach's α of the proposed test could be estimated directly. Second, solutions with a higher α than those found by the often-used Model II of Adema and van der Linden could be found. The purpose of the CMAX model is to assemble a test with α as high as possible, under restrictions of, for example, a fixed test length and a given content domain. With this model, one of the advantages of genetic algorithms became fully apparent. While a genetic algorithm for the CMAX model, which might be considered an alternative to Model II, produced solutions that were better than those produced for Model II, we were able to propose the CMIN model for the assembly of a test as short as possible for a given α. The CMAX model would be useful in situations where consequences of wrong decisions demand a test that exceeds a minimum α, for example, in high-stakes testing.

Large-scale test assembly is the main topic of Chapter 4. Frequently, testing programs are faced with the task to develop sets of test forms where each test form serves a slightly different purpose and a limited overlap between the forms is allowed. Especially the requirement of a limited overlap results in either a nonlinear model or the introduction of numerous dummy variables and corresponding restrictions. In the case of the Russian Unified State Examination testing program as described in Chapter 5, an admittedly very large example, the overlap specifications led to the use of approximately 100,000,000 dummy variables and equally many restrictions. Unfortunately, the models formulated for these testsets turned out to be epistatic, that is, the fitness function did not make a distinction between solutions and as soon as a feasible solution to the problem had been found, the genetic algorithm lost an incentive to search for improvements. However, evaluating two different fitness functions in consecutive iterations proved to be an effective method to prevent epistasis. To continue the analogy with

biological processes this approach could be considered as the modelling of seasons in an evolutionary process.

The use of large-scale problems, such as in the Unified State Examination, may risk overusing the item pools built by the testing programs, thus wasting the investments made. Therefore, the limits to large-scale test assembly were explored in Chapter 5. Long-term sequential assembly of test forms from an item pool may cause its depletion, not entirely unlike the effects that can be observed in computerised adaptive testing: The highly discriminating items are selected first, resulting in depletion in the case of test assembly, or resulting in overexposure in the case of computerised adaptive testing. Item selection algorithms should be harnessed in order to prevent these adverse effects. We found that division of the item bank into active pools that could be used consecutively was the best method to prevent item bank depletion.

One of the advantages of genetic algorithms is their capability to optimise nonlinear programming problems, and in extension to this thesis the question arises for which other item selection problems genetic algorithms can be used. Next to the models for item selection using classical test theory or item response theory that have been developed, optimisation models for generalisability theory have been developed, assuming random selection of for example, items and raters. Optimal selection of raters for a given test might improve interrater reliability, subject to restrictions that each team of raters give approximately equal severity of scoring. Simultaneous selection of items and raters might further improve the reliability of the testing procedures. As could be expected, the objective functions related to these procedures are nonlinear, hence genetic algorithms might be the proper tool to solve such models. A second application in which genetic algorithms may be successfully employed is the construction of optimal designs. The objective functions related to for example, A-optimality and D-optimality are nonlinear, while the associated designs are subject to a wide variety of practical restrictions similar to the restrictions in test assembly models.

Computerised adaptive testing might be regarded as the ultimately optimal item selection method: All information gathered during test administration is used to dynamically select items. Eggen (2004) has shown that a significant reduction of test length can be accomplished with a computerised adaptive test, compared to a conventional, or linear, test with equal measurement error. But for some test purposes for example, certification tests, the difference between a computerised adaptive test and an optimally assembled linear test could be very small, while practical considerations, such as item security, suggest to use a linear test instead. Therefore, it can be foreseen that computerised adaptive tests will not fully replace linear tests.

Genetic algorithms are not the only class of local search algorithms. Two other classes show great resemblance to genetic algorithms and thus might successfully solve test assembly problems.

The first class is simulated annealing. An often-used analogy to describe simulated annealing is the process of forging iron. Heating causes the atoms to move more fiercely while a careful cooling process will freeze the atoms in a strong structure. A repeated process of balancing heating and cooling builds an optimal structure in which the atoms represent the decision variables of the optimisation problem.

The second class is neural networks, also frequently described in terms of an analogy: Synapses learn to react to certain stimuli in order to recognise certain patterns. As the decision variable values of optimal and near-optimal solutions form patterns, the objective function teaches the synapses the proper reaction to these patterns. Neural networks are frequently used for solving optimisation problems. Similar to genetic algorithms, these problems need not be linear. All three classes share the property that a tailor-made implementation might be needed for acceptable performances, and thus also neural networks and simulated annealing could be considered for nonlinear item selection problems.

A last consideration is the application of test assembly models and accompanying algorithms in the practice of testing programs. Many testing agencies have extensive item bank management systems nowadays, in which it would be relatively easy to incorporate optimisation software. Designers of item banking software have a choice of roughly two options: either to employ ready-made commercially available optimisation software or to design dedicated software. Several solvers using branch-and-bound methods are available, and these solvers could either be used directly by specialists or a user interface for the definition of the test specifications and interpretation of the outcome could be developed. On the other hand, if designers would prefer to implement genetic algorithms into their item banking systems, only one option will be available: to design the dedicated software needed. Next to the user interface, also the algorithms must be developed. Cito has chosen for the latter option, and the resulting software has been used in all studies in this thesis.

Samenvatting (Summary in Dutch)

Als er één ontwikkeling aan te geven is die de evolutie van toetsconstructie over de afgelopen twee decennia karakteriseert, is het wel de toegenomen inzet van computers: Diverse modellen voor geautomatiseerde toetsconstructie werden geformuleerd, terwijl tegelijkertijd geavanceerde systemen werden ontwikkeld voor de opslag van toetsvragen, samen met hun psychometrische en inhoudelijke kenmerken.

Aan de andere kant kan men in het veld van optimaliseringsmethoden de ontwikkeling zien dat gebruik van z.g. local search algoritmen toeneemt, soms in plaats van traditionele algoritmen zoals branch-and-bound algoritmen. Een klasse van deze local search algoritmen wordt gevormd door genetische algoritmen. Geïnspireerd door processen in de natuur imiteren genetische algoritmen processen uit de evolutietheorie: Een groep van oplossingen, in geval van toetsconstructieproblemen zijn dit mogelijke toetsen, of voor de modellen in hoofdstuk 4 en 5 sets van toetsen, vormt een populatie die in een aantal iteraties evolueert naar een regio van de oplossingsruimte waarin ook het optimum van het probleem zich bevindt. De oplossingen worden gerepresenteerd door chromosomen. In iedere iteratie worden ouderparen geselecteerd voor het krijgen van nageslacht dat eigenschappen van de ouders erft. De crossover-operator geeft informatie opgeslagen in de chromosomen van de ouders door aan de kinderen, terwijl de mutatie-operator zorgt voor lichte aselecte afwijkingen. Hierdoor is het waarschijnlijk dat kinderen gaan lijken op beide ouders. Vervolgens zorgt een 'struggle for survival'-mechanisme ervoor dat de nieuwe populatie, nu bestaande uit ouders en kinderen, weer wordt teruggebracht tot de originele grootte. Het principe van survival-of-the fittest wordt vormgegeven door het feit

dat goede oplossingen een hogere kans op nageslacht en overleving hebben dan minder goede oplossingen. Hierdoor ontstaat een flexibel optimaliseringsalgoritme waarvoor men onder bepaalde voorwaarden diverse bewijzen van convergentie kan leveren. Het theoretische belang van deze bewijzen is echter groter dan het praktische belang aangezien implementaties waarvoor een convergentiebewijs kan worden gegeven in de regel dermate traag zijn dat ze in de praktijk slechts zelden worden gebruikt.

In dit proefschrift komen beide ontwikkelingen samen: Het voortschrijdende gebruik van itembanken doet de vraag naar steeds complexere toetsconstructiemodellen toenemen, waarvoor de huidige oplosmethoden niet altijd geschikt zijn. Het ligt dan voor de hand te onderzoeken in hoeverre genetische algoritmen kunnen worden ingezet voor toetsconstructieproblemen.

Wil men kunnen beweren dat genetische algoritmen met succes kunnen worden gebruikt, dan zullen zij voor een belangrijke klasse van praktijkproblemen goede oplossingen moeten kunnen produceren. Hoofdstuk 2 laat zien dat genetische algoritmen in staat zijn acceptabele oplossingen te vinden voor problemen die ook op een andere manier kunnen worden opgelost. Daarbij rijst wel de vraag welke implementatie het meest efficiënt is. Er werd door ons gevonden dat een combinatie van paringskansen evenredig aan de fitness van de oplossingen, een uniforme crossover, het laten overleven van alleen de beste oplossingen, en het gebruik van een dynamische boetefunctie voor het overschrijden van voorwaarden de beste resultaten gaf. In vrijwel alle onderzochte toepasingen werd een acceptabele oplossing gevonden.

In toepassingen waar de itemresponstheorie niet in gebruik is, wordt veelal de voorkeur gegeven aan toetsconstructiemodellen gebaseerd op klassieke toetstheorie. In hoofdstuk 3 wordt het niet-lineaire CMAX model voorgesteld, waarin een benadering voor Cronbach's α wordt geoptimaliseerd. Weliswaar is een goed model in de vorm van Model II van Adema en van der Linden voorhanden, maar bij gebruik van het CMAX model is het echter mogelijk een betere benadering te geven voor Cronbach's α, terwijl tevens oplossingen kunnen worden gevonden met een hogere α dan bij Model II. Deze benadering maakt een tweede model mogelijk, namelijk het CMIN model. Dit model gaat uit van de situatie waarin een zo kort mogelijke toets met een zekere α is gewenst. Het CMIN model is met name geschikt voor high-stakes testing, waarbij het belang van beslissingen een niet te grote foutmarge dicteert.

Hoofdstuk 4 behandelt het probleem van grootschalige toetsconstructie, waarbij uit een verzameling items een grote samenhangende set van toetsen moet worden geconstrueerd. Iedere toets in deze set heeft een eigen doel en eigen specificaties, waarbij tussen sommige toetsen wel en tussen andere toetsen geen overlap wordt toegestaan. Speciaal de eisen rond de overlap zijn problematisch voor de bestaande optimaliseringsmethoden: De overlap-voorwaarden veroorzaken de introductie van dummy-variabelen en

bijbehorende extra restricties, en voor sommige problemen is dit een groot aantal. Ter illustratie: in het geval van de Russische centrale staatsexamens dat beschreven is in hoofdstuk 5 – een zeer groot probleem – zijn circa 100.000.000 dummy-variabelen nodig. Als alternatief kan worden gekozen voor een niet-lineaire modelformulering die door genetische algoritmen kan worden opgelost. De modellen voor de constructie van de sets van toetsen zijn echter wel epistatisch: Zodra er een toegelaten oplossing wordt gevonden verliest het genetische algoritme iedere aansporing tot verbetering waardoor de optimalisering vroegtijdig stopt. Het afwisselend evalueren van twee verschillende fitness functies bleek een effectieve strategie om epistase te voorkomen. Men kan deze aanpak beschouwen als het voorkomen van seizoenen in de natuurlijke processen die door een genetisch algoritme worden geïmiteerd.

Met grootschalige problemen zoals de Russische centrale eindexamens loopt men risico roofbouw te plegen op de itembanken waardoor gepleegde investeringen teniet gedaan kunnen worden. Derhalve werden in hoofdstuk 5 de grenzen aan geautomatiseerde sequentiële toetsconstructie verkend. Langdurig grootschalige toetsconstructie kan leiden tot uitputting van de itembank: Indien er geen beperkingen aan de toetsconstructie worden opgelegd, worden hoogdiscriminerende items in het algemeen sneller geselecteerd waardoor de kwaliteit van de toetsen die later worden geconstrueerd beduidend lager is dan van eerder geconstrueerde toetsen. Het is daarbij goed te bedenken dat een dergelijke uitputting zich niet alleen bij het gebruik van genetische algoritmen voordoet, maar bij alle optimaliseringsmethoden, inclusief handmatige procedures. Enkele strategieën ter voorkoming van deze uitputting werden met elkaar vergeleken. Het opdelen van de gehele itembank in een aantal delen, waarbij in iedere periode een deel wordt gebruikt voor toetsconstructie, bleek een eenvoudige en doeltreffende maatregel.

Appendix A
Interitem Relations

The transformation from a Boolean expression, such as an interitem relation, to a numeric function called payoff function, involves a few rather simple steps without the need to use conjunctive normal form (CNF) or the introduction of dummy variables and restrictions, which are needed for linear restrictions. The use of CNF may be time consuming and may result in many linear restrictions, needing much extra computer power to solve by them. The introduction of dummy variables and restrictions may also increase the complexity substantially. On the other hand, Smith (1979) has shown that nonlinear restrictions based on payoff functions can be efficiently solved by genetic algorithms.

Let z be a Boolean variable, and $f(z)$ its associated payoff, having value 1 if z is true, or less if false. Then the following transformation rules can be used in order to transform a Boolean expression into a numerical restriction:

$$\begin{aligned} f(\neg z) &= 1 - f(z) \\ f(z_1 \wedge z_2) &= \min\{f(z_1), f(z_2)\} \\ f(z_1 \vee z_2) &= \max\{f(z_1), f(z_2)\}. \end{aligned}$$

De Jong and Spears (1989) have shown that the performance of genetic algorithms can be improved by transformation into differential payoff functions, substituting the minimum by the average in the transformation rules. However, the only types of expressions that can successfully be transformed are the ones in which the $\neg$-operator has the scope of simple variables.

Consider the following example of an interitem relation:

$$\begin{array}{c}\text{If item 1 is selected without item 2,}\\ \text{then select item 3 but not item 4}\end{array} \tag{A.1}$$

Define z_1, ..., z_4 as the Boolean variables associated with the selection of items 1, ..., 4, respectively, and let $\rightarrowtail$ denote the if-then operator. The relation in (A.1) can then be formulated as the following Boolean expression:

$$(z_1 \wedge \neg z_2) \rightarrowtail (z_3 \wedge \neg z_4). \tag{A.2}$$

The first step towards a restriction based upon the differential payoff function of relation (A.1) is the elimination of the $\rightarrowtail$-operator. `condition A` $\rightarrowtail$ `clause B` can be standardised as $\neg$ `condition A` $\vee$ `clause B`. Thus, standardisation of the expression in (A.2) results in

$$\neg(z_1 \wedge \neg z_2) \vee (z_3 \wedge \neg z_4). \tag{A.3}$$

The scope of the $\neg$-operator can be reduced by the application of De Morgan's laws. Therefore, we rewrite the expression in (A.3) as

$$\neg z_1 \vee z_2 \vee (z_3 \wedge \neg z_4). \tag{A.4}$$

The next step is the transformation of the expression in (A.4) into decision variables x_1, ..., x_4. Substitute z_i by x_i, $\neg z_i$ by $1 - x_i$, $\vee$ by maximum, and $\wedge$ by minimum. The expression in (A.4) is thereby transformed into

$$\max\{1 - x_1; x_2; \min\{x_3; 1 - x_4\}\} = 1. \tag{A.5}$$

De Jong and Spears argued that left hand sides of expressions such as in (A.5) would only be evaluated to 1 in case it is met or to 0 if it is violated, while a noninteger expression indicating the extent of violation would improve the performance of a genetic algorithm. This type of restrictions gives the genetic algorithm a direction to search for a solution that meets the restriction.

Therefore, define the differential payoff function of relation (A.1) by substituting the average for the minimum:

$$p_r(x_1, ..., x_4) = \max\left\{1 - x_1; x_2; \frac{x_3 + 1 - x_4}{2}\right\}.$$

Hence, the restriction in (A.5) can be replaced by

$$\max\left\{1 - x_1; x_2; \frac{x_3 + 1 - x_4}{2}\right\} = 1. \tag{A.6}$$

The restriction in (A.6) can replace the interitem relation in (A.1) in the test assembly problem. Note that a restriction of this type is nonlinear and that a linear formulation using one single restriction is generally not possible. For genetic algorithms, however, the only condition for restrictions is that they can be evaluated, and linearity is no requirement.

Appendix B
Penalty Functions

Using a penalty function is a common approach to deal with infeasible solutions. This approach allows any offspring to be generated but uses a fitness function based on a relaxation of the optimisation problem. A necessary condition for convergence to a feasible solution can be given: For all infeasible solutions having directly neighbouring feasible solutions, there must be at least one such neighbour with a higher fitness.

In order to proof this claim, consider a simple neighbourhood of a solution, defined by all solutions that differ just one item from it, that is, either an item is added to or removed from it. Consider an infeasible solution with at least one feasible neighbour. This means that there must exist at least one item whose addition or removal yields a feasible solution, that is, a solution in which the penalty function as formulated in (2.12) gives $g(x) = 0$. At the same time, the objective function value must change by such a quantity that the value of the fitness function in (2.11) for the feasible solution is higher than the fitness of the solution under consideration.

Now consider an IMAX problem with only one restriction on the desired shape of the TIF. Dummy variable y can be eliminated from the problem, resulting in the objective function

$$\text{maximise} \qquad \frac{\sum_i I_i x_i}{T}.$$

Further, the problem has only one resource restriction

$$\sum_i q_i x_i \leq Q.$$

For all infeasible solutions in the neighbourhood of a feasible solution it holds that there is at least one item j whose removal yields a feasible solution. Let $\mathcal{S}$ be such a solution. Let $y(\mathcal{S})$ be the objective function value for $\mathcal{S}$ and let $Q^*(\mathcal{S})$ be $Q - \sum_i q_i x_i$, the value by which the resource restriction is violated. The fitness of $\mathcal{S}$ is given by $f(\mathcal{S}) = y(\mathcal{S}) - \lambda Q^*(\mathcal{S})$. By removing item j, the objective function value will decline to $y(\mathcal{S}) - \frac{I_j}{T}$. Since this solution is feasible, its fitness is equal to the objective function value. The penalty multiplier λ must be chosen such that

$$\forall \mathcal{S} : \exists j : y(\mathcal{S}) - \lambda Q^*(\mathcal{S}) < y(\mathcal{S}) - \frac{I_j}{T}$$

or

$$\forall \mathcal{S} : \exists j : \lambda Q^*(\mathcal{S}) > \frac{I_j}{T}.$$

For a given solution $\mathcal{S}$, the choice of item j is obvious: it is the item in the test whose information at the chosen value of ϑ is lowest while $q_j > Q^*(\mathcal{S})$. Item j, however, can be different for each solution $\mathcal{S}$, and in the worst case it is the item with the highest information in the item pool that must be removed in order to reach feasibility:

$$\forall \mathcal{S} : \lambda Q^*(\mathcal{S}) > \max_j \frac{I_j}{T}.$$

In general, finding an infeasible solution with the smallest $Q^*(\mathcal{S})$ causes a problem. One can imagine that for some restrictions $Q^*(\mathcal{S})$ might be infinitesimally small, causing λ to go to infinity. This can be guaranteed not to happen only for certain resource restrictions. If the restriction has only integral coefficients, for example, in case of a maximum-number-of-items restriction, it holds that $\forall \mathcal{S} : Q^*(\mathcal{S}) \geq 1$. It can be inferred that λ is bounded by

$$\lambda > \max_j \frac{I_j}{T}.$$

When adding other restrictions and reverting to the original definition of the neighbourhood as the collection of all solutions that can be created in one single iteration, the situation becomes less clear. But here the same principle holds: although $Q^*(\mathcal{S})$ can be very small, removal of highly informative items might be the only way to reach the feasible region. In this case, the lower bound of λ becomes excessively high. If infeasible solutions are eliminated too quickly, the process might not pass to feasible solutions with higher fitnesses, resulting in premature convergence.

Siedlecki and Sklanski (1989) suggest a dynamic penalty scheme of the following form: Consider τ consecutive iterations and their best solutions. If all these solutions appear to be infeasible, raise the penalty multiplier. If all these solutions are feasible, lower the multiplier. In all other cases, leave the penalty multiplier unchanged.

References

Achterkamp, M. (1993). Toetsconstructie met behulp van genetische algoritmen (test construction with genetic algorithms). Master's thesis, University of Twente.

Ackerman, T. (1989). An alternative methodology for creating parallel tests using the IRT-information function. Paper presented at the annual meeting of the National Council on Measurement in Education, San Francisco, CA.

Adema, J. & van der Linden, W. (1989). Algorithms for computerized test construction using classical item parameters. *Journal of Educational Statistics*, 14, 279–290.

Ariel, A. (2005). *Contributions to Test-Item Bank Design and Management*. PhD thesis, University of Twente.

Armstrong, R., Jones, D., Li, X., & Wu, I.-L. (1996). A study of network-flow algorithm and a noncorrecting algorithm for test assembly. *Applied Psychological Measurement*, 20, 89–98.

Armstrong, R., Jones, D., & Wang, Z. (1994). Automated parallel test construction using classical test theory. *Journal of Educational Statistics*, 19, 73–90.

Armstrong, R., Jones, D., & Wu, I. (1992). An automated test development of parallel tests from a seed test. *Psychometrika*, 57, 271–288.

Aytug, H. & Koehler, G. (1996). Stopping criteria for finite length genetic algorithms. *INFORMS Journal of Computing*, 8, 183–191.

Baker, J. (1985). Adaptive selection methods for genetic algorithms. In J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and their Applications* (pp. 101–111).: Pittsburgh: Carnegie-Mellon University.

Belov, D. & Armstrong, R. (2004). A monte carlo approach for item pool analysis and design. Paper presented at the annual meeting of the National Council on Measurement in Education, San Diego.

Binet, A. & Simon, T. (1905). Méthodes nouvelles pour le diagnostic du niveau intellectual des anormaux. *L'Année Psychologique*, 11, 191–244.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In F. Lord & M. Novick (Eds.), *Statistical Theories of Mental Scores*: Reading, MA: Addison-Wesley.

Boekkooi-Timminga, E. (1990). The construction of parallel tests for IRT-based item banks. *Journal of Educational Statistics*, 15, 129–145.

Bridges, C. & Goldberg, D. (1987). An analysis of reproduction and crossover in a binary-coded genetic algorithm. In J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 9–13).: Hillsdale, NJ: Lawrence Erlbaum Associates.

Cronbach, L. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16, 279–334.

Darwin, C. (1859). *The Origin of Species by Means of Natural Selection*. London: John Murray.

Davidor, Y. (1991). Epistasis variance: a viewpoint on GA-hardness. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms* (pp. 23–35).: San Mateo, CA: Morgan Kaufmann.

de Jong, J. (1998). *NIVOR toetsen 1998 handleiding*. Arnhem: Cito. in Dutch; Test Manual.

De Jong, K. (1975). *Analysis of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

De Jong, K. & Spears, W. (1989). Using genetic algorithms to solve NP-complete problems. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 124–132).: San Mateo, CA: Morgan Kaufmann.

Downing, S. & Haladyna, T. (2006). *Handbook of Test Development*. Mahwah, NJ: Lawrence Erlbaum Associates.

Ebel, R. (1967). The relation of item discrimination to test reliability. *Journal of Educational Measurement*, 4, 125–128.

Eggen, T. (2004). *Contributions to the Theory and Practice of Computerized Adaptive Testing*. PhD thesis, University of Twente.

Eiben, A., Aarts, E., & van Hee, K. (1991). Global convergence of genetic algorithms: a Markov chain analysis. In H.-P. Schwefel & R. Männer (Eds.), *Parallel Problem Solving from Nature I* (pp. 4–12).: Berlin: Springer.

Eiben, A. & Smith, J. (2003). *Introduction to Evolutionary Computing*. Berlin: Springer.

Embretson, S. (2004). The second century of ability testing: some predictions and speculations. *Measurement*, 2, 1–32.

Feuerman, F. & Weiss, H. (1973). A mathematical programming model for test construction and scoring. *Management Science*, 19, 961–966.

Fogarty, T. (1989). Varying the probability of mutation in the genetic algorithm. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 104–109).: San Mateo, CA: Morgan Kaufmann.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading, MA: Addison-Wesley.

Goldberg, D. & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. In J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 1–8).: Hillsdale, NJ: Lawrence Erlbaum Associates.

Guilford, J. (1953). The correlation of an item with a composite of the remaining items in a test. *Journal of Psychological Measurement*, 13, 87–93.

Guilford, J. (1954). *Psychometric Methods*. New York: McGraw-Hill, 2nd edition.

Gulliksen, H. (1950). *Theory of Mental Tests*. New York: Wiley.

Hambleton, R. & Swaminathan, H. (1985). *Item Response Theory: Principles and Applications*. Boston: Kluwer-Nijhoff.

Henrysson, S. (1962). The relation between factor loadings and biserial correlations in item analysis. *Psychometrika*, 27, 419–424.

Henrysson, S. (1963). Correction of item-total correlations in item analysis. *Psychometrika*, 28, 211–218.

Holland, J. (1968). *Hierarchical description of universal spaces and adaptive systems.* Technical Report ORA projects 01252 and 08226, Ann Arbor: University of Michigan.

Holland, J. (1973). Genetic algorithms and the optimal allocations of trials. *SIAM Journal of Computing*, 2, 88–105.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Hornsby, G. & Pollack, J. (2001). The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computing* (pp. 600–607).: IEEE Press.

ILOG, Inc. (2002). *CPLEX 8.1.* Mountain View, CA: ILOG. Software and User's Manual.

Land, A. & Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28, 185–199.

Lawley, D. (1943). On problems connected with item selection and test construction. *Proceedings of the Royal Society of Edinburgh*, 61-A, 273–287.

Lazarsfeld, P. (1950). The logical and mathematical foundation of latent structure analysis. In S. Stouffer (Ed.), *Measurement and prediction*: Princeton, NJ: Princeton University Press.

Le Riche, R., Knopf-Lenoir, C., & Haftka, R. (1995). A segregated genetic algorithm for constrained structural optimization. In L. Eschelbaum (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 558–565).: San Mateo, CA: Morgan Kaufmann.

Lord, F. (1952). The relation of the reliability of multiple-choice tests to the distribution of item difficulties. *Psychometrika*, 17, 181–192.

Lord, F. (1980). *Applications of Item Response Theory to Practical Testing Problems.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Masters, G. (1982). A Rasch model for partial credit scoring. *Psychometrika*, 47, 149–174.

Murtagh, B. (1988). *MINTO.* Sydney, NSW: Macquarie University. Software and User's Manual.

Novick, M. (1966). The axioms and principal results of classical test theory. *Journal of Mathematical Psychology*, 3, 1–18.

Orvosh, D. & Davis, L. (1993). Shall we repair? genetic algorithms, combinatorial optimization and feasibility constraints. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 650).: San Mateo, CA: Morgan Kaufmann.

Rasch, G. (1960). *Probabilistic Models for some Intelligence and Attainment Tests.* Copenhagen: Nielsen and Lydiche.

Richardson, J., Palmer, M., Liepins, G., & Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 191–197).: San Mateo, CA: Morgan Kaufmann.

Rothlauf, F. & Goldberg, D. (2003). Redundant representations in evolutionary computation. *Evolutionary Computation*, 11, 381–415.

Russian Federation Ministry of Education and Science (2005). *Analytical Report on National Examinations in the System of Educational Quality Assessment.* Moscow: Author.

Samejima, F. (1977). Weakly parallel tests in latent trait theory with some criticisms of classical test theory. *Psychometrika*, 42, 193–198.

Sanders, P. & Verschoor, A. (1998). Parallel test construction using classical item parameters. *Applied Psychological Measurement*, 22, 212–223.

Siedlecki, W. & Sklanski, J. (1989). Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 141–150).: San Mateo, CA: Morgan Kaufmann.

Smith, G. (1979). *Adaptive Genetic Algorithms and the Boolean Satisfiability Problem.* unpublished manuscript.

Spearman, C. (1904). The proof and measurement of association between two things. *American Journal of Psychology*, 15, 72–101.

Spears, W. & De Jong, K. (1999). Dining with GAs: operator lunch theorems. In W. Banzhaf & C. Reeves (Eds.), *Foundations of Genetic Algorithms 5* (pp. 85–101).: San Francisco, CA: Morgan Kaufmann.

Suzuki, J. (1993). A Markov chain analysis on a genetic algorithm. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 146–153).: San Mateo, CA: Morgan Kaufmann.

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

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9).: San Mateo, CA: Morgan Kaufmann.

Theunissen, T. (1985). Binary programming and test design. *Psychometrika*, 50, 411–420.

Theunissen, T. (1996). *Combinatorial Issues in Test Construction.* PhD thesis, University of Amsterdam.

Thierens, D. & Goldberg, D. (1994). Convergence models of genetic algorithm selection schemes. In Y. Davidor (Ed.), *Parallel Problem Solving from Nature - PPSN III* (pp. 119–129).: Berlin: Springer.

Thurstone, L. (1925). A method of scaling psychological and educational tests. *Journal of Educational Psychology*, 16, 433–451.

Timminga, E., van der Linden, W., & Schweizer, D. (1996). *ConTEST 2.0: A decision support system for item banking and optimal test assembly.* Groningen: iec ProGAMMA. Software and User's Manual.

Toussaint, M. (2005). Compact genetic codes as a search strategy of evolutionary processes. In A. Wright, M. Vose, K. De Jong, & L. Schmidt (Eds.), *Foundations of Genetic Algorithms 2005* (pp. 75–94).: Berlin: Springer.

van der Linden, W. (2005). *Linear Models for Optimal Test Design.* New York: Springer.

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

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

Verhelst, N., Glas, C., & Verstralen, H. (1995). *One-Parameter Logistic Model (OPLM).* Arnhem: Cito. Software and User's Manual.

Verschoor, A. (1991). *Optimal Test Design.* Arnhem: Cito. Software and User's Manual.

Verschoor, A. (2004). *IRT Test Assembly Using Genetic Algorithms.* Measurement and Research Department Reports 2004-4, Arnhem: Cito.

Verschoor, A. (2005). *DOT 2005.* Arnhem: Cito. Software for Automated Test Assembly.

Way, W. & Steffen, M. (1998). Strategies for managing item pools to maximize item security. Paper presented at the annual meeting of the National Council on Measurement in Education, San Diego.

Zubin, J. (1934). The method of internal consistency for selecting test items. *Journal of Educational Psychology*, 25, 345–356.