Stageverslag 2003-7

Parallelle Toetsconstructie met behulp van Genetische Algoritmes

Sjef Sanders

Cito groep

Inhoudsopgave:

1. Inleiding

Dit rapport is de beschrijving van een simulatiestudie naar parallelle toetsconstructie met behulp van genetische algoritmes.

De belangrijkste taak van het CITO is het samenstellen van toetsen. Een toets is een middel om een vaardigheid van een persoon te bepalen.
Er zijn verschillende manieren om een toets te construeren.
De afgelopen decennia is de interesse voor toetsconstructie door itemselectie uit een itembank sterk toegenomen. Hiervoor zijn verschillende optimalisatiemodellen ontwikkeld, die gebruik maken van verschillende oplossingsmethoden.
Sinds enkele jaren is er een nieuwe oplossingsmethode ontwikkeld, de zogenaamde genetische algoritmes.
In hoofdstuk 2 wordt een uitgebreide uitleg geven over deze genetische algoritmes, hoe deze ontstaan zijn en hoe ze weken.
In hoofdstuk 3 wordt verder ingaan op de vraag hoe de vaardigheid van een persoon geschat kan worden, in dit hoofdstuk zal de item response theorie worden besproken.
In hoofdstuk 4 worden de verschillende optimalisatiemodellen besproken die gebruikt worden bij het construeren van toetsen.
In hoofdstuk 5 worden de simulaties beschreven en in het laatste hoofdstuk de conclusies en bevindingen.

2. Genetische Algoritmen

Een genetisch algoritme is een optimalisatietechniek die gebaseerd is op het principe van de evolutietheorie: "survival of the fittest".

2.1 Biologische Basis

In 1859 legde Charles Darwin met de publicatie van zijn boek *"The origin of species"* de basis voor de huidige evolutietheorie.
Evolutie is verandering van een *populatie* in de loop van de tijd.
Voor evolutie is dus in eerste instantie een populatie nodig waarvan de individuen onderling kunnen verschillen.
De eigenschappen van een individu zijn opgeslagen in de *chromosomen*.
Chromosomen kunnen dus gezien worden als de codering van de structuur van het individu.
Bij het reproduceren, het verkrijgen van nakomelingen, vindt de evolutie plaats. De chromosomen van de nakomelingen kunnen door twee redenen verschillen van die van de ouders.
Ten eerste combineert een nakomeling het materiaal van de chromosomen van de verschillende ouders. Dit proces wordt ook wel *crossover* genoemd. Als die ouders niet gelijk zijn, kan een nakomeling een nieuwe combinatie van materiaal bevatten. Welk materiaal een nakomeling van welke ouder overneemt, wordt grotendeels bepaald door het toeval.
Ten tweede komen er in de natuur *mutaties* voor. Mutaties zijn toevallige wijzigingen in de chromosomen. Op deze manier kan een nakomeling materiaal verkrijgen dat niet in de ouders aanwezig is of zelfs nog in geen enkel individu van de huidige populatie aanwezig was.
Niet alle individuen lukt het om te overleven in de huidige omgeving en nakomelingen te verkrijgen. De kans dat een individu genetisch materiaal kan doorgeven aan nieuwe generaties is afhankelijk van hoe 'goed' het materiaal van dit individu is. Hierdoor ontstaat er een natuurlijke selectie van materiaal in de populatie. Als het een individu niet lukt om te overleven en te reproduceren, zal het materiaal van dat individu niet overgedragen worden. Als het individu materiaal bevat, dat verder niet in de populatie voorkomt, zal dat materiaal uit de populatie verdwijnen. Aan de andere kant zal het materiaal van een individu dat wel overleeft en reproduceert, wel terug komen in de volgende generatie.
Stel dat er door mutatie nieuw materiaal in de populatie komt. Als dit materiaal goede eigenschappen oplevert, waardoor het individu betere prestaties levert en dus een grotere kans heeft te overleven en te reproduceren, dan zal het materiaal in de populatie blijven en in steeds meer individuen voorkomen. Door het crossover-proces geldt dit niet alleen voor los materiaal maar ook voor combinaties ervan.

2.2 Genetische algoritmen

In 1975 kwam men voor het eerst op het idee dat, als deze biologische eigenschappen op een juiste manier omgezet zouden worden tot een algoritme, er een techniek zou ontstaan die problemen op dezelfde manier zou aanpakken als de natuur heeft gedaan, met behulp van evolutie.
Genetische algoritmen simuleren de biologische evolutie, maar er wordt niet gepoogd de biologische processen volledig te beschrijven. De algoritmen maken alleen gebruik van idealisatie van bepaalde processen in de biologische systemen

Het centrale thema van onderzoek naar genetische algoritmes is het evenwicht tussen efficiëntie en doeltreffendheid. Deze combinatie van eigenschappen is noodzakelijk om in verschillende omgevingen te kunnen overleven. Omdat in de biologie deze efficiëntie en doeltreffendheid op verschillende gebieden aanwezig is, is de interesse van ontwerpers van kunstmatige systemen in deze biologische systemen zeer gestegen.
Naast de vergelijking met de biologische systemen is ook theoretisch en empirisch bewezen dat genetische algoritmes voorzien in efficiënte en doeltreffende optimalisatiemethoden in complexe ruimten.
Gezien een gevestigde reputatie van betrouwbare benadering van problemen met betrekking tot efficiënte en doeltreffende zoekacties worden genetische algoritmes alom toegepast in de zakelijke, wetenschappelijke en technische wereld.
De reden voor het groeiend aantal toepassingen moge duidelijk zijn. De algoritmen zijn eenvoudig te programmeren, maar krachtig in hun zoektocht naar verbetering. Bovendien worden ze niet fundamenteel beperkt door restrictieve aannamen met betrekking tot de zoekruimte.
We zullen de oorzaken van deze aantrekkelijke eigenschappen en de theoretische grondslagen ervan nog verder onderzoeken. Maar eerst zullen we de thema's efficiëntie en doeltreffendheid verder bekijken aan de hand van vergelijkingen met traditionele optimalisatiemethoden.

2.3 Efficiëntie en doeltreffendheid van traditionele optimalisatiemethoden

De huidige literatuur onderscheidt drie hoofdtypen van zoekmethoden: de op calculus gebaseerde, de enumeratieve en de random typen.

2.3.1 De op calculus gebaseerde methoden

De op calculus gebaseerde methoden zijn uitgebreid bestudeerd. Deze methoden zijn ook weer onder te verdelen in twee subtypen: directe en indirecte.
Indirecte methoden zoeken locale extrema door op te lossen de verzameling van vergelijkingen, voortvloeiend uit het gelijkstellen van de gradiënt van de doelfunctie aan nul.
Directe methoden zoeken lokale extrema door te 'springen' op de functie en te bewegen in een richting, die gerelateerd is aan de lokale gradiënt. Dit is min of meer het principe van bergbeklimmen. Om het lokale extremum te vinden, moet de functie 'beklommen' worden in de steilste richting.
Het is niet moeilijk om het gebrek aan efficiëntie van deze methodes aan te tonen. Op de eerste plaats zijn beide methoden lokaal: het extremum dat gevonden wordt is het beste extremum in de buurt van het huidige punt. Dus het is lang niet zeker of er een globaal extremum gevonden wordt. De enige kans op verbetering moet dan gezocht worden in een herstart van het proces.
Een ander nadeel van deze methoden is dat ze afhankelijk zijn van het bestaan van afgeleiden en continuïteit en vaak is dit niet het geval bij optimalisatieproblemen.
De conclusie luidt dus: de op calculus gebaseerde methoden moeten verworpen worden, ze zijn onvoldoende efficiënt op onvoorwaardelijke domeinen.

2.3.2 De enumeratieve methoden

Het idee achter enumeratieve methoden is eenvoudig. Binnen een eindige zoekruimte, of een discrete oneindige zoekruimte, beschouwt het zoekalgoritme de doelfunctiewaarden in elk punt van de ruimte, één voor één.

Een voorwaarde voor het werken met genetische algoritmes is dat de natuurlijke parameterverzameling voor het optimalisatieprobleem wordt gecodeerd als een eindig lange rij tekens uit een eindig alfabet.

ii. Genetische algoritmes gaan uit van een populatie van punten en niet van één enkel punt.
Bij veel optimalisatiemethoden wordt er van een punt naar het volgende punt gegaan in een kansruimte door gebruik te maken van een overgangsregel die het volgende punt bepaalt. Deze van-punt-tot-punt methode is gevaarlijk, omdat het een perfect recept is voor het lokaliseren van valse pieken in een veeltoppige zoekruimte.

iii. Genetische algoritmes gebruiken de doelfunctie als informatiebron en geen afgeleiden of andere supplementaire kennis.
Veel zoektechnieken vereisen een hoop supplementaire informatie en toegevoegde kennis om fatsoenlijk te functioneren. Zo zijn bijvoorbeeld voor gradiënttechnieken afgeleiden nodig, al dan niet analytisch bepaald of numeriek benaderd. Genetische algoritmes hebben dit niet nodig. Voor een effectieve zoektocht naar betere structuren hebben genetische algoritmes alleen met individuele tekenrijtjes geassocieerde doelfunctiewaarden nodig.

iv. Genetische algoritmes gebruiken probabilistische overgangsregels en geen deterministische.
In tegenstelling tot andere methoden gebruiken genetische algoritmes probabilistische overgangsregels in hun zoektocht. Maar het kansgebruik houdt niet in dat de methode een random zoekmethode is. Genetische algoritmes gebruiken random keuze als een gereedschap om hun zoektocht te sturen in de richting van gebieden in de zoekruimte met een waarschijnlijke verbetering.

Deze vier verschillen samen, dragen bij tot de robuustheid van genetische algoritmes.

In het volgende hoofdstuk zullen we de verschillende stappen van het genetisch algoritme door middel van een voorbeeld toelichten.

2.6 Voorbeeld

Van een doelfunctie trachten we het maximum te bepalen (in figuur 1 is de doelfunctie getekend).
Een populatie die deze "omgeving" bewoont zou gevormd kunnen worden door een verzameling getallen tussen 0 en 2.55. De individuen (getallen) die de grootste doelfunctiewaarde opleveren zijn het best aangepast aan hun omgeving.

Het is meteen al duidelijk waarom deze methode niet voldoet aan onze eisen. Er is sprake van een gebrek aan efficiëntie.
Veel zoekruimtes zijn in praktijk vaak te groot om alle punten één voor één na te gaan.

2.3.3 Random methoden

Random zoekmethodes hebben enorm aan populariteit gewonnen, toen onderzoekers de tekortkomingen van op calculus gebaseerde en enumeratieve zoekmethoden erkenden. Maar de random schema's zullen ook buiten beschouwing gelaten moeten worden vanwege het gebrek aan efficiëntie. Op den duur zullen random zoekmethodes het naar verwachting niet beter gaan doen dan enumeratieve schema's.
Hierbij moet wel opgemerkt worden dat random methoden niet hetzelfde zijn als gerandomiseerde technieken. Het genetische algoritme is een voorbeeld van een zoekprocedure, die gebruik maakt van een gerandomiseerde techniek als gereedschap voor een uiterst exploiterende zoektocht door een gecodeerde parameterruimte. Het gebruik van random keuzes als gereedschap in een gericht zoekproces lijkt een beetje vreemd op het eerste gezicht. Maar de natuur voorziet in genoeg voorbeelden. Het is op dit punt belangrijk in te zien dat een random zoektocht niet automatisch een richtingloze zoektocht hoeft te betekenen.

De conventionele zoekmethoden zijn dus blijkbaar niet robuust. Dit betekent overigens niet dat ze niet bruikbaar zijn. Bovenstaande schema's en algoritmes zijn succesvol toegepast op verschillende terreinen. Maar blijkbaar zijn ze niet geschikt om meer complexe problemen mee aan te pakken. Zoals later zal blijken kunnen genetische algoritmen dat wel. Maar eerst is het nuttig om duidelijk te formuleren wat onze doelen zijn wanneer we spreken over het optimaliseren van een proces of functie.

2.4 Doelen van optimalisatie

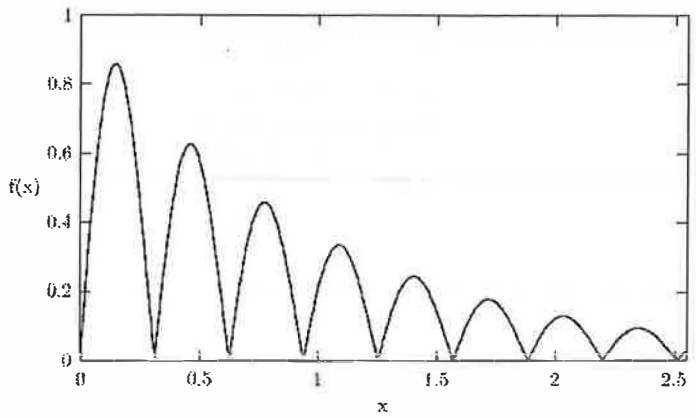
We zullen nu definitief vastleggen wat we verstaan onder het begrip optimalisatie. Optimaliseren houdt in dat er voortdurend gezocht wordt naar verbetering in de richting van een optimaal punt. Deze definitie is tweeledig. Allereerst wordt er gestreefd naar verbetering. Op de tweede plaats wordt er gezocht naar een optimaal punt. Er is dus duidelijk een verschil tussen het proces van verbetering en het bereiken van het optimum zelf.
Normaliter wordt er bij de beoordeling van een optimalisatieprocedure voornamelijk gelet op de convergentie van het proces. Er wordt dus alleen gekeken of het optimum bereikt wordt en niet naar de manier waarop dat gebeurt.
In het dagelijks leven wordt vaak niet gezocht naar de beste oplossing, wel wordt er gezocht naar een relatief goede oplossing.

Voordat we een model voor genetische algoritmes gaan bespreken, geven we eerst enkele belangrijke verschillen aan tussen genetische algoritmes en meer traditionele methode.

2.5 Verschillen

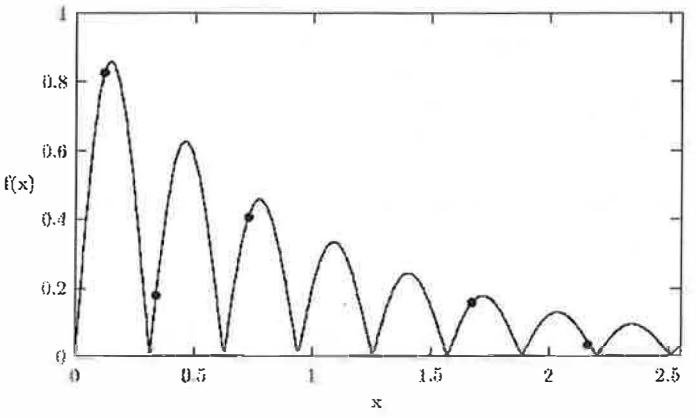
Genetische algoritmes verschillen op de volgende vier punten van de conventionele optimalisatiemethoden.

i. Genetische algoritmes gebruiken een codering van een parameterverzameling en niet de parameters zelf.



Figuur 2.1: Voorbeeld van een optimalisatieprobleem: de doelfunctie bereikt een globaal maximum voor x≈0.15.

De eerste stap in het genetisch algoritme is nu genereren van een populatie:
Er wordt een verzameling van *N* random getallen gekozen, dit is de initiële populatie P^0, bijvoorbeeld P^0={0.12, 0.34, 0.73, 1.21, 1.67, 2.16}. De omgeving en de initiële populatie zijn weergegeven in figuur 2.
Één van deze getallen zal de hoogste doelfunctiewaarde opleveren, dit zal dan het best aangepaste individu zijn. In het voorbeeld leveren de individuen in P^0 respectievelijk de volgende doelfunctiewaarden op: {0.83, 0.18, 0.41, 0.13, 0.16, 0.04}.
De doelfunctiewaarde noemen we de "*fitness*" f_i van het individu *i*. Het is dus duidelijk dat individu 0.12 het best aan zijn omgeving is aangepast, deze heeft de hoogste fitness waarde, namelijk 0.83 en zal dus vrijwel zeker in de volgende generatie voorkomen.



Figuur 2.2: De initiële populatie is weergegeven in de omgeving mbv. •. 0,12 is het best aangepast aan de omgeving.

Voordat verdergegaan kan worden met het genereren van nageslacht moet eerst de populatie gecodeerd worden zodat het genetisch algoritme er mee kan werken. Een

manier om de populatie te coderen is gebruik te maken van een *n*-bits binaire voorstelling van de getallen. Voor het hier gegeven voorbeeld zou men getallen tussen 0 en 2.55 kunnen voorstellen met behulp van 8 bits. De initiële populatie komt er dan als volgt uit te zien.

$$0.12 \rightarrow 00001100$$
$$0.34 \rightarrow 00100010$$
$$0.73 \rightarrow 01001001$$
$$1.21 \rightarrow 01111001$$
$$1.67 \rightarrow 10100111$$
$$2.16 \rightarrow 11011000$$

Nu kunnen we verdergaan met het genereren van nageslacht.

Selectie:
Als eerste moeten er, uit de begin populatie, *N* individuen geselecteerd worden om cross-over en mutatie op toe te passen.
Men zou de individuen willekeurig kunnen selecteren, maar het is beter rekening te houden met de fitness. Dit kan vrij eenvoudig op de volgende manier.
De kans dat een individu *i* uit populatie *P* geselecteerd wordt is gegeven door:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}. \qquad (2.1)$$

In woorden is de kans dat een individu geselecteerd wordt om zich voort te planten te omschrijven als de fractie van de individuele fitness ten opzichten van de totale fitness van de populatie.
In de onderstaande tabel is dit uitgewerkt voor het voorbeeld, ook is de gemiddelde en maximale fitness berekend.

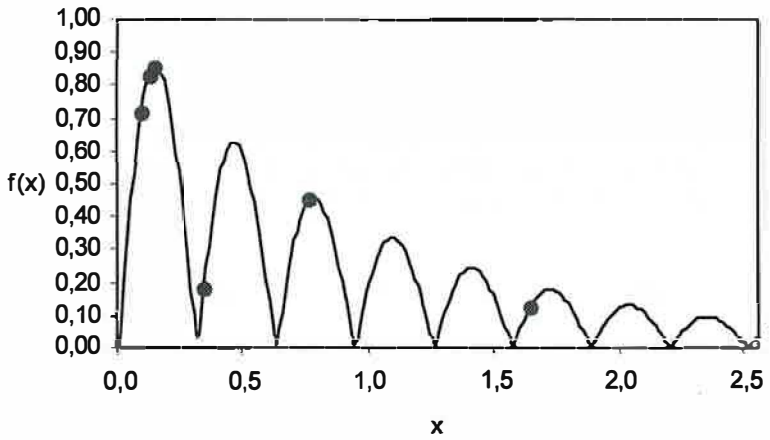
rijtje A_i	codering	f_i	p_i	aantal geselecteerd
A_1	00001100	0.83	0.47	3
A_2	00100010	0.18	0.10	1
A_3	01001001	0.41	0.23	1
A_4	01111001	0.13	0.07	0
A_5	10100111	0.16	0.09	1
A_6	11011000	0.04	0.02	0
som		1.75		
gemiddelde		0.29		
maximum		0.83		

Zoals te zien is kan het dus voorkomen dat één individu meer dan één keer geselecteerd wordt.

Crossover:
Men kan nu de "crossover" operatie definiëren. Bij deze crossover operatie worden als eerste op een random manier alle geselecteerde individuen gekoppeld, waarbij voor iedere set van twee individuen een getal *m* tussen 1 en *n*–1, het zogenaamde crossover-punt, gekozen wordt. Nu worden de nakomelingen geconstrueerd door van ieder koppel de *n*–*m* laatste bits om te wisselen. Voor het voorbeeld wordt dat dus,

koppels	crossover-punt	codering	Nieuwe populatie	f_i
A_1	5	00001|100	A'_1=00001111	0.86
A_5	5	10100|111	A'_2=10100100	0.12
A_1	2	00|001100	A'_3=00100010	0.18
A_2	2	00|100010	A'_4=00001100	0.83
A_3	3	010|01001	A'_5=01001100	0.45
A_1	3	000|01100	A'_6=00001001	0.72
som				3.16
gemiddelde				0.53
maximum				0.86

Zowel de gemiddelde als de maximale fitness zijn toegenomen (in figuur 3 zijn de nieuwe punten te zien).
Hoewel een gerandomiseerd proces deze gelukkige omstandigheden veroorzaakt, zien we toch dat het geen toeval is.



Figuur 2.3: De nakomelingen zijn weergegeven in de omgeving. 0,15 is het best aangepast aan de omgeving.

Mutatie:
Het mutatieproces is zeer eenvoudig voor te stellen. Iedere bit verandert van waarde met de kans p_m. Het volgende plaatje illustreert dit.

$$001|0|1100\} \xrightarrow{mutatie} \{001|1|1100$$

De kans p_m wordt over het algemeen zeer klein genomen (b.v. 1/1000).
Omdat de kans op een mutatie erg klein is ga ik er vanuit dat er in mijn voorbeeld geen mutatie plaatsvindt.

Wanneer nu deze twee processen zijn toegepast worden alle nakomelingen bij de originele populatie P^t gevoegd.
De nieuwe populatie P^{t+1} ontstaat nu door N individuen te selecteren uit de populatie P^t. Dit kan op verschillende manieren, men zou de beste N individuen kunnen selecteren, maar het blijkt dat er dan waardevolle informatie verloren kan gaan. Een betere manier is selectie zoals al eerder beschreven is in (2.1).

In het volgende hoofdstuk zal het genetische algoritme theoretisch onderbouwd worden.

2.7 *Wiskundige Basis*

2.7.1 Schema Stelling

Het kan op het eerste zicht nogal wonderlijk overkomen dat een wel heel eenvoudige parallel met natuurlijke evolutie een praktisch bruikbare optimalisatietechniek oplevert. In dit hoofdstuk zal geprobeerd worden met wiskundige middelen aan te tonen waarom het beschreven algoritme werkt.
Indien we het voorbeeld uit het vorige hoofdstuk opnieuw bekijken dan is het onmiddellijk duidelijk dat een individu die een bit-string heeft van de volgende vorm '11******' zeer weinig kans heeft een betere fitness te hebben dan een met een bit-string zoals '****11**' (hierin staat '*' voor een 0 of een 1). Een dergelijke representatie van een klasse bit-strings noemt men een "schema" en deze vormen de sleutel tot het begrijpen van genetische algoritmen. Voor strings van lengte *n* die bestaan uit de tekens 0 en 1 zijn er 3^n van dergelijke schema's, immers, op elk van de *n* posities kan een 0, een 1 of een '*' staan.
Men kan nu het aantal keer *m(H, t)* berekenen dat een bepaald schema *H* voorkomt in de populatie op tijdstip *t*. Indien er enkel rekening gehouden wordt met selectie als voortplantingsmechanisme kan een eenvoudige relatie tussen *m(H,t+1)* en *m(H,t)* opgesteld worden. Hiervoor wordt gebruik gemaakt van *f(H)*, de fitness van een schema. Deze berekent men als de gemiddelde fitness van de bit-strings gerepresenteerd door dit schema. Het schema '110100**' representeert bijvoorbeeld de strings '11010000', '11010001', '11010010', '11010011'. Men kan nu schrijven:

$$m(H,t+1) = m(H,t) \cdot N \cdot \frac{f(H)}{\sum_{i=1}^{N} f_i} = m(H,t) \cdot \frac{f(H)}{\overline{f}}. \qquad (2.2)$$

Hierin is $\overline{f} = \sum_{i=1}^{N} f_i / N$ de gemiddelde fitness van de populatie P^t.
Uitdrukking (2.2) maakt direct duidelijk dat een schema met een fitness groter dan de gemiddelde fitness van de populatie meer zal voorkomen in de volgende generatie en vice versa. De toename van geschikte schema's en de afname van niet geschikte zal zelfs exponentieel verlopen.
Dit is als volgt te bewijzen: laten we aannemen dat een schema *H* boven het gemiddelde licht en wel $c\overline{f}$ erboven. Onder deze aanname kunnen we uitdrukking (2.2) herschrijven tot,

$$m(H,t+1) = m(H,t) \cdot \frac{\left(\overline{f} + c\overline{f}\right)}{\overline{f}} = (1+c) \cdot m(H,t). \qquad (2.3)$$

Wanneer we starten op t=0 en een stationaire waarde voor c veronderstellen, dan krijgen we,

$$m(H,t) = m(H,0) \cdot (1+c)^t. \qquad (2.4)$$

Iedere wiskundige (en econoom) zal in uitdrukking (2.4) meteen een exponentieel verloop herkennen.
Hiermee is aangetoond dat de selectieprocedure, zoals beschreven in het vorige hoofdstuk, er wel degelijk voor zorgt dat opeenvolgende generaties beter zullen passen in hun omgeving.
Nu moet de invloed van crossover en mutatie nog bestudeerd worden. Het zijn deze mechanismen die er voor zorgen dat er variatie in de populatie gebracht wordt zodat het vinden van de gewenste oplossing waarschijnlijk wordt, ook indien deze niet in de oorspronkelijke populatie voorkomt.

Op de vraag hoe schema's zich onder crossover gedragen, kan men het best antwoorden door de volgende twee voorbeelden te bekijken:

H_1 = ***10***

H_2 = *10****0

Het is duidelijk dat schema H_1 veel robuuster is tegenover crossover dan H_2: het eerste heeft immers maar een kans van 1/7 om verbroken te worden, het tweede 6/7. Voor het al dan niet overleven van een schema is dus het maximaal aantal posities tussen de bits die vast liggen in het schema van belang, dit wordt ook wel "definiëringslengte" genoemd en genoteerd als $\delta(H)$. Voor het eerste voorbeeld geldt dus $\delta(H_1)=1$, voor het tweede $\delta(H_2)=6$. De kans dat een schema verdwijnt bij een crossover is dus: $\delta(H)/(l-1)$, met l het aantal bits (genen).
Wanneer crossover plaatsvindt via een random manier, dan kan de overlevingskans p_s, gegeven een selectiekans van p_c, op de volgende manier geschreven worden.

$$p_s \geq 1 - p_c \cdot \frac{\delta(H)}{l-1}. \tag{2.5}$$

Nu kan formule (2.3) herschreven worden tot,

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \cdot \frac{\delta(H)}{l-1} \right]. \tag{2.6}$$

Het mutatieproces verandert deze formule slechts weinig. De schemaorde *o(H)* wordt gedefinieerd als het aantal bits dat vastligt in schema *H*. In het voorbeeld is dit dus: $o(H_1)=2$ en $o(H_2)=3$. De kans dat een bit op een van de vaste posities niet zal veranderen is $(1-p_m)$, zodat de kans dat een schema H onder mutatie behouden blijft gegeven wordt door,

$$(1-p_m)^{o(H)} \approx 1 - o(H) p_m, \tag{2.7}$$

indien $p_m << 1$, hetgeen in de praktijk steeds het geval is. Samen met uitdrukking (2.6) levert dit het volgende resultaat,

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H) \cdot p_m \right]. \tag{2.8}$$

De ongelijkheid in deze formule geeft aan dat schema's met een hoge fitness, een kleine definiëringslengte en een lage orde, naar verwachting in opeenvolgende generaties op zijn minst een stijgend aantal kopieën krijgen.
De schema's die aan deze eisen voldoen, worden *building blocks* genoemd. De building blocks zijn te beschouwen als partiële oplossingen van het op te lossen probleem en een genetisch algoritme bepaalt nieuwe oplossingen van het probleem door combinaties te maken van de deeloplossingen die de huidige populatie bevatten.

2.7.2 Impliciet parallellisme

Een populatie met n individuen bevat tussen de 2^l en $n2^l$ verschillende schema's. Dit komt omdat ieder individu in de populatie een element is van 2^l schema's, ieder bit heeft ofwel de werkelijke waarde of het metasymbool "*". In het vorige hoofdstuk is gesteld dat niet al deze schema's even veel waarde hebben voor het zoekproces. De schema's welke van belang zijn, zijn de schema's die van generatie op generatie in de populatie voorkomen en de schemastelling geeft aan dat dit de schema's zijn met een hoge fitness, een kleine definiëringslengte en een lage orde. De vraag is nu hoeveel van dergelijke schema's, building blocks, in de populatie aanwezig zijn.

Beschouw een populatie van n rijen binaire getallen met chromosoomlengte l. Er worden alleen schema's met een constante overlevingskans groter dan p_s bekeken.

Er worden alleen schema's toegestaan met een kans op vernietiging $\varepsilon < 1 - p_s$. Er wordt dus gekeken naar schema's met lengte $l_s < \varepsilon(l-1)+1$.
Bij een schema met een bepaalde lengte kan dus een ondergrens geschat worden voor het aantal unieke schema's in een aanvankelijk random populatie rijtjes. Eerst worden het aantal schema's ter lengte l_s of minder geteld. Dan wordt dit aantal vermenigvuldigd met een geschikte populatiegrootte, die zo gekozen wordt dat gemiddeld niet meer dan één van elk schema ter lengte $l_s/2$ (met l_s even) verwacht wordt.
Veronderstel dat het aantal schema's ter lengte van l_s geteld moeten worden in het volgenden rijtje met l=10; 1011100010. Om dat te doen wordt eerst het aantal schema's in de eerste cel van vijf bepaald,
|10111|00010, waarvoor geldt dat de laatste positie in het blok gefixeerd is, dus alle schema's in de vorm, %%%%1*****, waarbij de %-tekens aangeven dat op die plaats in het schema of de waarde van het chromosoom op die positie of het *-teken moet staan.
Het is duidelijk dat er 2^{l_s-1} van dergelijke schema's zijn, want l_s-1=4 kunnen of gefixeerd zijn of het *-teken bezitten. Om nu alle schema's met lengte l_s te tellen in het chromosoom 1011100010 moet het blok steeds een plaats opschuiven: 1|01110|0010, 10|11100|010 enz.
Dit moet dus (l–l_s+1) keer uitgevoerd worden om het totaal aantal schema's ter lengte l_s of kleiner te kunnen schatten, dit aantal kan nu geschat worden op $2^{l_s-1}\cdot(l-l_s+1)$.
Door deze schatting te vermenigvuldigen met n wordt een schatting gegeven voor de totale populatie.
Deze schatting, $n\cdot 2^{l_s-1}\cdot(l-l_s+1)$, is zeer waarschijnlijk te hoog omdat er zeker duplicaten van schema's met een lage orde in de populatie zullen voorkomen. Om deze schatting te verbeteren wordt een populatiegrootte van $n=2^{l_s/2}$. In dat geval wordt maximaal één van elk schema van orde $\frac{1}{2}l_s$ verwacht. Aangezien het aantal schema's binomiaal verdeeld is, kan verwacht worden dat de helft van de schema's een orde heeft die groter is dan $\frac{1}{2}l_s$ en de andere helft een orde die kleiner is dan $\frac{1}{2}l_s$. Door alleen de schema's met een hoge orde te tellen kan een benedengrens voor het aantal schemata bepaald worden,

$$n_s \geq n\cdot(l-l_s+1)\cdot 2^{l_s-2}. \qquad (2.9)$$

Dit verschilt van de eerdere (over)schatting met een factor ½. Bovendien resulteert de beperking van de populatiegrootte tot de waarde, $n=2^{l_s/2}$, in de expressie:

$$n_s = \frac{(l-l_s+1)\cdot n^3}{4}. \qquad (2.10)$$

In uitdrukking (2.10) is goed te zien dat het aantal schema's n_s van de orde n^3 is. Deze schatting geeft aan dat genetische algoritmen, ondanks de verstoring door crossover en mutatie van lange schema's met hoge orde, bewerkingen uitvoeren op een grote groep schema's, terwijl er slechts gewerkt wordt met een relatief kleine groep individuen.

De Schema Stelling en het impliciete parallellisme geven de verklaring voor de werking van genetische algoritmes.
Schema's met een hoge fitness, een kleine definiëringslengte en een lage orde worden geselecteerd, gecombineerd en opnieuw geselecteerd en vormen zo chromosomen met een potentieel hoge fitness.
Het genetische algoritme is eigenlijk steeds aan het proberen betere chromosomen te construeren met behulp van de goede eigenschappen van vorige chromosomen.

3. Item Respons Theorie

Toetsen worden geconstrueerd met als doel om daarmee de vaardigheid van een persoon te schatten.
Hoe kun je nu deze vaardigheid bepalen?
Bij de klassieke test theorie (KTT) gebeurt dit door de 'ware' score T van een persoon te bepalen. De ware score is de score die een persoon met een bepaalde vaardigheid naar verwachting zal halen. Er wordt gesteld dat de ware score T van een persoon in principe observeerbaar is door de scores van een groot aantal toetsafnames te middelen.
Bij de IRT staat, zoals de naam al doet vermoeden, het item en het antwoord daarop centraal. De IRT hanteert geen observeerbaar begrip, als ware score, maar een niet observeerbaar begrip namelijk vaardigheid. Om deze principiële onobserveerbaarheid aan te duiden gebruikt men de term latent.
De IRT is een geheel van uitspraken over de samenhang tussen de latente vaardigheid en het antwoordgedrag op een verzameling items.
De uitspraken in zo'n theorie zijn meestal niet heel specifiek: de voorspellingen over het gedrag hangen af van kenmerken van de items en van de personen. Deze kenmerken worden meestal gekwalificeerd als parameters, en de waarden van deze parameters zijn in de regel niet bekend, maar moet worden geschat.

3.1 Algemene Theorie

Zoals al is aangegeven staat het begrip latente variabele centraal in de IRT. Hoewel er verschillende opvattingen zijn over de status van deze variabele, zullen we ons hier beperken tot het geval dat het domein van de latente variabele de reële as is. Elk persoon in een populatie kan afgebeeld worden als een punt van de reële as, oftewel aan elke persoon kan een getal worden toegevoegd dat een uitdrukking is van de mate waarin die persoon over een vaardigheid beschikt. De latente variabele vaardigheid wordt aangegeven met het symbool θ, de getalswaarde die aan persoon v is toegekend duiden we aan als θ_v.
Om iets te kunnen zeggen over de θ-waarde van een persoon veronderstelt men dat de antwoorden op bepaalde items een indicatie geven over de vaardigheid.

Met X_i duiden we het antwoord aan op item i, X_i is dichotoom, met waarden toegekend volgens onderstaande regel:

$$X_i = \begin{cases} 1 \text{ indien het antwoord op item } i \text{ correct is,} \\ 0 \text{ indien het antwoord op item } i \text{ fout is,} \end{cases}$$

Nu noemen we de itemresponsfunctie $f_i(\theta)$, de kans dat item i juist beantwoord wordt. Dus,

$$f_i(\theta) = P(X_i = 1 \mid \theta). \tag{3.1}$$

Een realistische weergave van deze itemresponsfunctie moet aan een paar voorwaarden voldoen.

1. $0 < f_i(\theta) < 1$;
2. $f_i(\theta)$ moet continu zijn;
3. $f_i(\theta)$ moet overal differentieerbaar zijn
4. $f_i(\theta)$ moet strikt stijgend zijn.

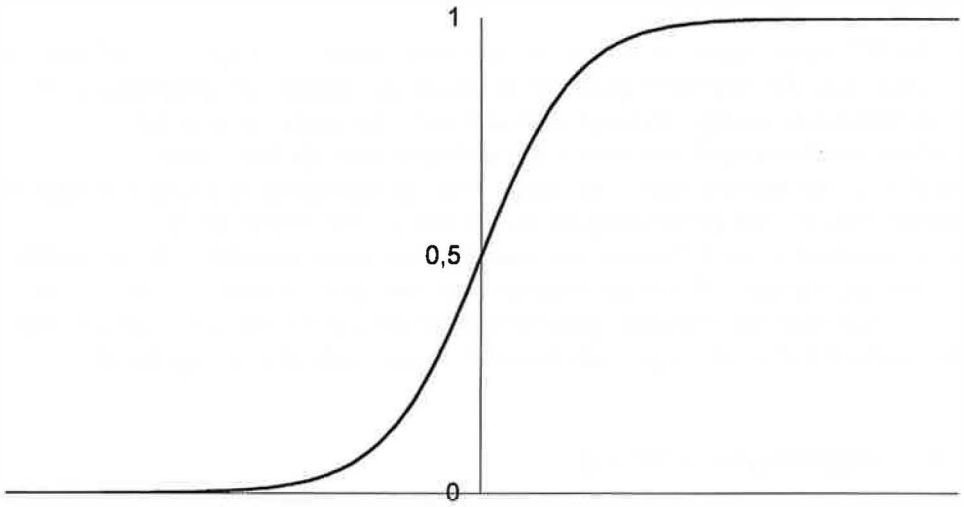
Een functie die aan al deze eisen voldoet en mathematisch goed hanteerbaar is, is gegeven in het Raschmodel.

3.2 Het Raschmodel

In het Raschmodel is de itemresponsfunctie de logistische functie,

$$f_i(\theta) = \frac{e^{(\theta-\beta_i)}}{1+e^{(\theta-\beta_i)}}. \tag{3.2}$$

Hierin is β_i een kengetal dat iets zegt over de moeilijkheidsgraad van item *i*. Het is eenvoudig in te zien dat in de situatie $\theta=\beta_i$, de functiewaarde ½ is. Dit wil dus zeggen dat de kans dat item *i*, met moeilijkheidsparameter β_i door een persoon met vaardigheid $\theta=\beta_i$, correct beantwoord wordt, ½ is. In figuur 1 is een grafiek getekend van een Raschmodel.



Figuur 3.1: Raschmodel

Uitdrukking (3.2) wordt vaak veralgemeniseerd tot de volgende formule,

$$f_i(\theta) = \frac{e^{a_i(\theta-\beta_i)}}{1+e^{a_i(\theta-\beta_i)}},\ a_i>0. \tag{3.3}$$

Hierin is een extra discriminatieparameter a_i toegevoegd. Als a_i groter wordt betekent dit dat een correct (of incorrect) antwoord meer informatie levert over de vaardigheid van een persoon, dit komt tot uiting in een steiler verloop van de itemresponsfunctie.

3.3 Informatiefunctie

Uit de itemresponsfunctie wordt informatie gehaald met betrekking tot het wel of niet correct beantwoorden van een item en daarmee tot het bezitten van genoeg vaardigheid.
Om voor een persoon een goede uitspraak te doen over diens intelligentie, wordt de iteminformatiefunctie gedefinieerd als,

$$I_i(\theta) = f_i(\theta)[1-f_i(\theta)]. \tag{3.4}$$

De iteminformatiefunctie bereikt haar maximum in $f_i(\theta)=0,5$, dit is dus het geval wanneer $\theta=\beta_i$. Hoe hoog het maximum ligt hangt af van de discriminatieparameter a_i. De functiewaarde wordt kleiner naarmate θ meer afwijkt van β_i.
Wanneer de vaardigheid in de buurt van de waarde β_i ligt, dan levert dit een hoge mate van informatie over het item. Voor personen waarvan de vaardigheid ver van de waarden β_i afligt, levert de toets weinig informatie.
Voor een toets met *k* items ziet de toetsinformatiefunctie er dan als volgt uit,

$$I(\theta) = \sum_{i=1}^{k} I_i(\theta) = \sum_{i=1}^{k} f_i(\theta)[1-f_i(\theta)]. \tag{3.5}$$

Zoals al vermeld is levert een item weinig informatie als de vaardigheid θ ver afwijkt van β_i. Dit geldt ook voor een toets, een toets kan dus voor een persoon zeer informatief zijn, en voor andere niet.

4. Het samenstellen van toetsen

Bij het samenstellen van toetsen kunnen we te maken krijgen met drie soorten eisen: psychometrische, inhoudelijke en praktische eisen. De psychometrische eisen zullen veelal betrekking hebben op de gewenste nauwkeurigheid van de samen te stellen toetsen. Met inhoudelijke eisen worden de vakinhoudelijke en onderwijskundige eisen bedoeld: de verdeling van de vragen over de leerstof, de gewenste moeilijkheidsgraad van de toets en dergelijke. Ook relaties op itemniveau kunnen een rol spelen bij het samenstellen van toetsen. Onder praktische eisen wordt bijvoorbeeld verstaan het maximaal aantal vragen dat in de toets mag voorkomen. In dit hoofdstuk laten we zien hoe met behulp van wiskundige modellen toetsen samengesteld kunnen worden die voldoen aan psychometrische, inhoudelijke en praktische specificaties. De modellen zijn ontleend aan een tak van wiskunde, aangeduid met operationele research, die als doel heeft het ontwikkelen van modellen ter ondersteuning van besluitvorming.

4.1 Het samenstellen van toetsen in de itemresponstheorie

Zoals al is beschreven hebben psychometrische eisen betrekking op de meetnauwkeurigheid van de toets. Binnen de IRT worden voor het specificeren van de meetnauwkeurigheid continue functies gebruikt, namelijk de iteminformatiefunctie en de toetsinformatiefunctie.
Voor het raschmodel is de iteminformatiefunctie gegeven door,

$$I_i(\theta) = \frac{e^{(\theta-\beta_i)}}{\left(1+e^{(\theta-\beta_i)}\right)^2}, \tag{4.1}$$

deze functie is maximaal als $\theta=\beta_i$. De toetsinformatiefunctie is nu,

$$I(\theta) = \sum_i I_i(\theta). \tag{4.2}$$

Belangrijk voor toetsconstructie is het feit dat deze functies lokale meetnauwkeurigheid aangeven, dat wil zeggen dat de informatie afhankelijk is van het vaardigheidsniveau. Items die niet te moeilijk en niet te gemakkelijk zijn geven een hogere meetnauwkeurigheid dan zeer gemakkelijke en zeer moeilijke items.

Een belangrijke overweging bij het construeren van toetsen is dat men over het algemeen slechts in een beperkt deel van de vaardigheidschaal geïnteresseerd is, bijvoorbeeld in het cesuurgebied bij zak-slaagbeslissingen. Men kan dan eisen stellen aan de meetnauwkeurigheid in het cesuurpunt en op twee punten daar net iets onder en boven. Men legt de toetsinformatiefunctie vast op een aantal punten maar blijft toch gebruik maken van het gegeven dat de toetsinformatie in ieder punt op de vaardigheidsschaal de som is van de iteminformaties. Op dit principe is het gebruik van operationele research bij toetsconstructie binnen de IRT gebaseerd. Al naar gelang de omstandigheden kan men eisen voor de toets met betrekking tot toetsinformatie formuleren als doel of als restrictie. Van beide gevallen zullen later voorbeelden gegeven worden.

4.2 Optimalisatiemodellen

In deze paragraaf zullen drie basismodellen besproken worden:

1. Maximinmodel (I-MAX)
2. Minimalisatiemodel (I-MIN)
3. Toetskarakteristieke curve model (TCC)

Niet alleen de werkwijze van de drie modellen verschillen onderling, maar ook de fitnessfuncties die zijn gedefinieerd verschillen onderling sterk. Er zijn per model zelfs nog verschillende varianten te bedenken. Maar alvorens de drie modellen besproken zullen worden, zal er kort ingegaan worden op het begrip fitnessfunctie.

4.2.1 Fitnessfunctie

Zoals al eerder besproken bestaat iedere iteratie van het genetische algoritme uit een drietal deelprocessen: het selecteren van de individuen, het generen van nageslacht en de strijd om te overleven. Omdat er bij deze processen gebruik gemaakt wordt van de fitness, moet dus voor alle individuen de fitness bepaald kunnen worden. De fitnessfunctie neemt binnen het genetische algoritme de rol over die de doelfunctie inneemt bij de optimalisatiemodellen. Er zal voor gezorgd moeten worden dat het genetische algoritme en het normale optimaliseringsmodel tot dezelfde optimale oplossing komen. Dit houdt in dat de fitnessfunctie op hetzelfde punt het maximum moet bereiken als de optimale oplossing van het optimalisatieprobleem.

Zoals eerder ter sprake is gekomen moet de fitnessfunctie voor ieder individu bepaald kunnen worden, zowel voor toegelaten oplossingen als voor niet toegelaten oplossingen. Dit leidt tot twee manieren om de fitnessfunctie te definiëren:

- Bij de eerste manier worden alleen toegelaten oplossingen (oplossingen die voldoen aan alle restricties) als nieuwe individuen toegestaan. Niet toegelaten oplossingen worden direct geëlimineerd.
  In dit geval is de fitnessfunctie direct afleidbaar van de doelfunctie van het optimalisatieprobleem.
- Bij de tweede manier worden alle oplossingen toegestaan.
  De fitnessfunctie zal worden afgeleid aan de hand van een relaxatie van het optimaliseringsmodel. Voor iedere gegenereerde oplossing is de fitnessfunctie gedefinieerd, waarbij boeteparameters moeten garanderen dat het optimum van de fitnessfunctie correspondeert met het optimum van de doelfunctie van het optimaliseringsmodel.

Opgemerkt moet worden dat het onmogelijk is om op eenvoudige wijze alleen toegelaten oplossingen in het nageslacht te genereren, zoals in het eerste geval wordt verondersteld. Het vinden van een toegelaten oplossing is even complex als het vinden van de optimale oplossing. Daarom wordt gekozen voor de tweede manier. Hierdoor ontstaat wel het probleem dat er een structuur van boetefuncties en boeteparameters moet worden opgezet, dat robuust genoeg is om alle mogelijke varianten van optimaliseringsmodellen op te lossen.

4.2.2 Maximinmodel (I-MAX)

Het maximinmodel is een optimalisatiemodel waarbij de toetsinformatie als doelfunctie gebruikt wordt en de gewenste vorm (target) van de toetsinformatiefunctie wordt gespecificeerd (4.1) onder kwantitatieve (4.2), classificatie (4.3) en interitem (4.4) restricties.

maximaliseer: y

onder voorwaarde dat: $y \leq \frac{\sum_i I_{ik} x_i}{T_k} \qquad \forall k \qquad (4.1)$

$$\sum_i q_{in} x_i \leq Q_n \qquad \forall n \qquad (4.2)$$

$$C_m^l \le \sum_i c_{im} x_i \le C_m^u \qquad \forall m \tag{4.3}$$

$$\sum_i l_{ip} x_i \le L_p \qquad \forall p \tag{4.4}$$

$$x_i \in \{0,1\} \qquad \forall i$$

Variabele x_i is de beslisvariabele die aangeeft of item i al dan niet geselecteerd is in de test. I_{ik} is de informatie in het punt θ_k en T_k is het informatie target. De coëfficiënten q_{in} zijn de kwantitatieve parameters, c_{im} de classificatie parameters, die de waarde 1 hebben als item i tot categorie m behoord en anders 0, en l_{ip} zijn de coëfficiënten van de interitem restricties. Q_n, C^l_m en C^u_m zijn de gewenste aantallen items in de test en de gewenste aantallen items per categorie. L_p zijn de interitem restricties.
Zoals al is aangegeven is een fitnessfunctie gebaseerd op een boetefunctie de beste oplossing. De fitnessfunctie voor het I-MAX model is geconstrueerd uit de doelfunctie en een boete voor het overschrijden van ieder restrictie:

$$f(x) = y - g(x) \tag{4.5}$$

$$\begin{aligned} g(x) = {} & \lambda \sum_n h\left(\sum_i q_{in} x_i - Q_n\right) \\ & + \mu \sum_m h\left(C_m^l - \sum_i c_{im} x_i\right) + \mu \sum_m h\left(\sum_i c_{im} x_i - C_m^u\right) \\ & + \varphi \sum_r 1 - p_r(x) \end{aligned} \tag{4.6}$$

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

De fitnessfunctie, zoals beschreven in vergelijking (4.5), kan negatieve waardes bevatten voor zekere individuen en dit kan problemen opleveren wanneer het genetische algoritme de selectiekans gaat bepalen, er bestaan immers geen negatieve kansen. Daarom is het verstandig de fitnessfunctie te herdefiniëren als,

$$f(x) = \frac{y}{1 + g(x)}. \tag{4.7}$$

4.2.3 Minimalisatiemodel (I-MIN)

Het minimalisatiemodel heeft als doel een minimale toets te construeren met betrekking tot een bepaald criterium, zoals toetslengte, gegeven een target en onder kwantitatieve, classificatie en interitem restricties.

$$\text{minimaliseer:} \quad \sum_i q_{i1} x_i \tag{4.8}$$

$$\begin{aligned} \text{onder voorwaarde dat:} \quad & \sum_i I_{ik} x_i \ge T_k && \forall k \\ & \sum_i q_{in} x_i \ge Q_n && \forall n \\ & C_m^l \le \sum_i c_{im} x_i \le C_m^u && \forall m \\ & \sum_i l_{ip} x_i \le L_p && \forall p \\ & x_i \in \{0,1\} && \forall i \end{aligned}$$

De fitnessfunctie voor dit model wordt dan als volgt gedefinieerd,

$$f(x)=\left(\frac{1}{1+\sum_i q_{i1}x_i}\right)\left(\frac{1}{1+g(x)}\right) \tag{4.9}$$

$$g(x)=\kappa\sum_k h\left(T_k-\sum_i I_{ik}x_i\right)+\lambda\sum_n h\left(Q_n-\sum_i q_{in}x_i\right)$$
$$+\mu\sum_m h\left(C_m^l-\sum_i c_{im}x_i\right)+\mu\sum_m h\left(\sum_i c_{im}x_i-C_m^u\right)$$
$$+\varphi\sum_r 1-p_r(x)$$

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

De fitnessfunctie in uitdrukking (4.9) is zo gedefinieerd dat er geen negatieve waarden voor kunnen komen.

4.2.3.1 Epistase

Het I-MIN model is zeer gevoelig voor epistase. Epistase treedt op wanneer er veel geschikte oplossingen met dezelfde fitness zijn. Dit gebeurt bijvoorbeeld bij het I-MIN model wanneer men het aantal items als doelfunctie stelt, er zijn dan verschillende individuen met dezelfde fitness mogelijk. Het genetische algoritme kan dan geen onderscheid maken tussen individuen die in potentie nageslacht kunnen produceren met een hogere fitness en andere individuen. Het zoekproces stagneert. Een oplossing voor dit probleem is er voor te zorgen dat er een klein verschil ontstaat in de fitness, op deze manier heeft ieder individu een unieke fitness. De verschillen in fitness moeten wel zo gekozen worden dat individuen met een hogere fitness in potentie beter nageslacht kunnen produceren dan individuen met een lagere fitness. Nu kan uitdrukking (4.9) vervangen worden door (4.10),

$$f(x)=\left(\frac{1}{1+\sum_i q_{i1}x_i}\right)\left(\frac{1+\gamma\sum_k h\left(\sum_i I_{ik}x_i-T_k\right)}{1+g(x)}\right), \tag{4.10}$$

hierin is γ de parameter die een bonus geeft voor het verschil van de informatiefunctie met de targetfunctie. Dus de fitness van een individu waarbij de informatiecurve ver boven het target ligt is groter dan de fitness van een individu waarbij dit verschil minder groot is, gegeven dat ze dezelfde doelfunctie hebben. Deze bonusparameter moet niet te groot gekozen worden, want anders komt het optimum van de fitnessfunctie niet meer overeen met het optimum van de doelfunctie. Bij het I-MIN model moet altijd gelden dat een toegelaten oplossing met een lagere doelfunctie altijd een hoger fitness heeft dan een toegelaten oplossing met een hogere doelfunctie.

4.2.4 Toetskarakteristieke Curvemodel (TCC)

Het Toetskarakteristieke Curvemodel (TCC) wordt gebruikt in situaties waar een hoge informatiefunctie niet de bepalende factor is in het optimalisatieproces. Het voordeel van het TCC model komt het best tot zij recht in praktische situaties. In het algemeen is het eenvoudiger om een toetskarakteristieke curvedoelfunctie te formuleren dan een toetsinformatiedoelfunctie. Het model minimaliseert de afstand tot een

toetskarakteristieke curvedoelfunctie onder kwantitatieve, classificatie, en interitem restricties.

$$\begin{aligned}
&\text{minimaliseer: } y && (4.11)\\
&\text{onder voorwaarde dat: } y \geq \left|\sum_i (S_{ik} - W_i T_k) x_i\right| && \forall k\\
&\sum_i q_{in} x_i \geq Q_n && \forall n\\
&C_m^l \leq \sum_i c_{im} x_i \leq C_m^u && \forall m\\
&\sum_i l_{ip} x_i \leq L_p && \forall p\\
&x_i \in \{0,1\} && \forall i
\end{aligned}$$

Hierin is variabele S_{ik} de verwachte score van item *i* bij het punt θ_k, W_i is de maximale gewogen score van item *i* en T_k is weer het target in het punt θ_k.
De fitnessfunctie voor dit model wordt ook weer zo gekozen dat het geen negatieve waarden kan bevatten namelijk,

$$\begin{aligned}
f(x) &= \left(\frac{1}{1+y}\right)\left(\frac{1}{1+g(x)}\right) && (4.12)\\
g(x) &= \lambda \sum_n h\left(\sum_i q_{in} x_i - Q_n\right)\\
&\quad + \mu \sum_m h\left(C_m^l - \sum_i c_{im} x_i\right) + \mu \sum_m h\left(\sum_i c_{im} x_i - C_m^u\right)\\
&\quad + \varphi \sum_r 1 - p_r(x)\\
h(u) &= \begin{cases} u, & u > 0 \\ 0, & u \leq 0 \end{cases}.
\end{aligned}$$

4.2.5 Parallelle toetsen

Het doel van parallelle toetsconstructie is om *J* toetsen te construeren met dezelfde eigenschappen maar zonder dat er dezelfde items in voor komen. Er zijn verschillende formuleringen mogelijk voor parallelle toetsconstructie, de standaard formulering is gebruik te maken van een de beslisvariabele x_{ij} die aangeeft of item *i* is geselecteerd in toets *j*.
Het I-MAX model wordt dan als volgt geformuleerd,

$$\begin{aligned}
&\text{maximaliseer: } y && (4.13)\\
&\text{onder voorwaarde dat: } y \leq \frac{\sum_i I_{ik} x_{ij}}{T_k} && \forall k, j\\
&\sum_i q_{in} x_{ij} \leq Q_n && \forall n, j\\
&C_m^l \leq \sum_i c_{im} x_{ij} \leq C_m^u && \forall m, j\\
&\sum_i l_{ip} x_{ij} \leq L_p && \forall p, j
\end{aligned}$$

$$\sum_j x_{ij} \le 1 \qquad \forall i$$

$$x_{ij} = \begin{cases} 1, \text{item } i \text{ in toets } j \\ 0, \text{anders} \end{cases} \qquad \forall i, j$$

Alle restricties zijn uitgebreid zodat ze gelden voor de beslisvariabele x_{ij}, ook is er een extra restrictie toegevoegd die er voor zorgt dat er geen overlap optreedt.
De fitnessfunctie wordt ook uitgebreid met een boeteparameter ξ voor het overschrijden van deze restrictie.

$$f(x) = \frac{y}{1+g(x)} \tag{4.14}$$

$$\begin{aligned} g(x) = {} & \lambda \sum_{n,j} h\left(\sum_i q_{in} x_{ij} - Q_n\right) \\ & + \mu \sum_{m,j} h\left(C_m^l - \sum_i c_{im} x_{ij}\right) + \mu \sum_{m,j} h\left(\sum_i c_{im} x_{ij} - C_m^u\right) \\ & + \varphi \sum_{r,j} 1 - p_r(x) + \xi \sum_i h\left(\sum_j x_{ij} - 1\right) \end{aligned}$$

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

Een andere formulering van de beslisvariabele is ook mogelijk, namelijk:

$$x_i = \begin{cases} 0, \text{als item } i \text{ niet geselecteerd is} \\ j, \text{als item } i \text{ geselecteerd is in toets } j \end{cases} \qquad j \in \{1,2,...,J\} \tag{4.15}$$

Het voordeel van deze notatie is dat er altijd voldaan wordt aan de overlap restrictie, ook is er voor ieder item nog maar één gen nodig, zodat de totale lengte van het chromosoom aanzienlijk verkort wordt.
Voordat nu het nieuwe I-MAX model geconstrueerd kan worden moet er eerst een nieuwe functie gedefinieerd worden,

$$F_j(x) = \begin{cases} 1, \text{als } x = j \\ 0, \text{anders} \end{cases} \tag{4.16}$$

Het I-MAX model wordt nu als volgt gedefinieerd,

maximaliseer: y (4.17)

onder voorwaarde dat:

$$y \le \frac{\sum_i I_{ik} F_j(x_i)}{T_k} \qquad \forall k, j$$

$$\sum_i q_{in} F_j(x_i) \le Q_n \qquad \forall n, j$$

$$C_m^l \le \sum_i c_{im} F_j(x_i) \le C_m^u \qquad \forall m, j$$

$$\sum_i l_{ip} F_j(x_i) \le L_p \qquad \forall p, j$$

$$x_i \in \{0,1,...,J\} \qquad \forall i$$

Door deze hercodering van de chromosomen treedt er een kleine verandering op in het mutatie proces. Bij mutatie kan nu iedere gen met een bepaalde kans een andere waarde aannemen van 0 tot *J*.
De boetefunctie komt er nu als volgt uit te zien:

$$\begin{aligned} g(x) = \lambda \sum_{n,j} h\left(\sum_i q_{in} F_j(x_i) - Q_n\right) \\ + \mu \sum_{m,j} h\left(C_m^l - \sum_i c_{im} F_j(x_i)\right) + \mu \sum_{m,j} h\left(\sum_i c_{im} F_j(x_i) - C_m^u\right) \\ + \varphi \sum_{r,j} 1 - p_r\left(F_j(x_i)\right) \end{aligned} \qquad (4.18)$$

5. Simulatiestudies

Zowel bij het genetische algoritme als bij de toetsconstructie modellen bestaan er nog onduidelijkheden: hoe groot moet de initiële populatie gekozen worden, na hoeveel iteraties moet er gestopt worden met het proces, hoe worden de boeteparameters gekozen enz.
Het overgrote deel van deze problemen zijn niet analytisch op te lossen, onderzoekers zijn dus aangewezen op simulatiestudies om hier een uitspraak over te doen.

In dit hoofdstuk zullen de simulaties die tijdens de stage zijn uitgevoerd besproken worden. Maar eerst zullen de stappen van de simulaties besproken worden.

5.1 Stappen van een simulatie

Bij de simulaties van het toetsconstructieproces wordt gebruik gemaakt van een realistische itembank met 500 items. Van deze items zijn de volgende gegevens bekend:

- ruwe score
- tot welke categorie(ën) het item behoort
- vaardigheidsniveau β_i met discriminatiefactor a_i

Met deze items worden de toetsen geconstrueerd.

5.1.1 Stap 1: keuze voor het model en de restricties

Zowel voor de enkelvoudige als de parallelle toetsconstructie zijn er binnen de IRT drie optimalisatiemodellen te kiezen.

- I-MAX model
- I-MIN model
- TCC model

Voor alle drie modellen geldt dat er voorwaarden gesteld kunnen worden aan de te construeren toetsen. Aan de modellen kunnen de volgende restricties toegevoegd worden: het aantal items in de toets (dit geldt niet voor het I-MIN model), de op te nemen categorieën per toets, het aantal items per categorie, de interitem relaties en de target informatiefunctie.

5.1.2 Stap 2: uitvoeren simulaties

Een simulatie bestaat uit een aantal runs, meestal 400. In iedere run worden er een aantal iteraties uitgevoerd en er worden aan het einde van iedere run en tussendoor rapporten geschreven waarin de fitness wordt genoteerd van de tot dan toe hoogste toegelaten oplossing. Ook wordt er genoteerd hoeveel iteraties er nodig waren om deze fitness te bereiken.

5.1.3 Stap 3: analyseren resultaten

Nadat de simulaties voltooid zijn worden de resultaten geanalyseerd. Dit gebeurt vaak aan de hand van een grafiek waarin de fitness tegen het aantal iteraties is uitgezet. Ook wordt van zowel de fitness als het aantal iteraties de spreiding bekeken.

5.2 *De Simulaties*

Binnen het onderzoek van toetsconstructie m.b.v. genetische algoritme hebben we ons tijdens de stage vooral gericht op parallelle toetsconstructie.

5.2.1 Simulatie 1

Er is de afgelopen jaren onderzoek gedaan naar hoe de boeteparameters gekozen moeten worden. De conclusies van deze onderzoeken wijzen erop dat het beter is om variabele boeteparameters in te stellen.
Dit gebeurt op de volgende manier:
Bekijk bij de laatste k iteraties de individuen met de hoogste fitness. Als voor al deze k individuen geldt dat aan de restricties voldaan is, dan wordt λ vermenigvuldigd met $1-\delta$. Als voor al deze k individuen er enkele restricties niet gelden, dan moet λ vermenigvuldigd worden met $1+\varepsilon$. In alle andere gevallen blijft λ gelijk. Doe dit ook voor de andere boeteparameters (μ, φ (en κ)).
Als startwaarde nemen we: $\lambda=\mu=\varphi=0,1$ (en $\kappa=0,2$).
Bij de eerste simulatie is er gekeken hoe de parameters k, δ en ε gekozen moeten worden bij de parallelle toetsconstructie, hiervoor hebben we vier modellen gebruikt:

Model 16: dit is een I-MAX model met relatief weinig restricties.
Model 17: dit is een I-MAX model met relatief veel restricties.
Model 18: dit is een I-MIN model met relatief weinig restricties.
Model 19: dit is een I-MIN model met relatief veel restricties.

We hebben 3 verschillende waarde voor k genomen namelijk: k=10 (F), k=20 (M) en k=40 (S).
En we hebben 2 verschillende combinaties van δ en ε gekozen namelijk: δ=0,11, ε=0,08 (H) en δ=0,03, ε=0,02 (L).
De verwachting is dat bij een model met relatief weinig restricties F/H sneller convergeert naar een optimale oplossing, maar bij meer complexe modellen S/L beter presteert.
Het algoritme stopt bij model 16 en 18 na 32000 iteraties en bij model 17 en 19 na 128000 iteraties.
Zoals al is verteld wordt zowel de beste oplossing gerapporteerd als de iteratie waarin deze tot stand gekomen is.

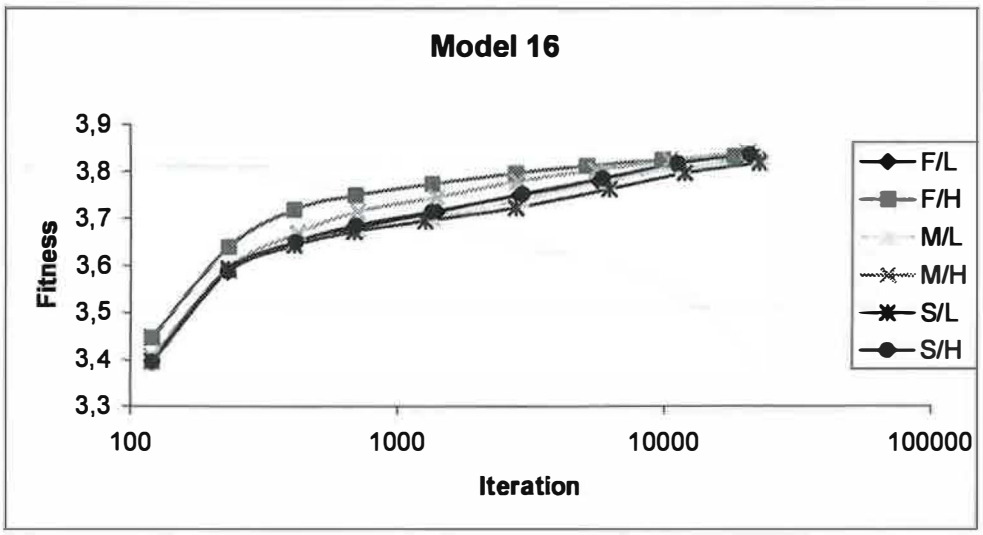
In tabel 5.1 zijn de resultaten te zien.

		L		H	
		fitness	iteraties	fitness	iteraties
16	F	3,829±0,034	22400±8300	3,832±0,030	18200±10100
	M	3,827±0,046	22500±8400	3,842±0,037	20300±9200
	S	3,818±0,054	22400±8000	3,835±0,046	20600±8100
17	F	3,392±0,112	97300±26900	3,345±0,121	102100±22300
	M	3,407±0,117	94200±30300	3,370±0,127	100200±26000
	S	3,420±0,129	89200±31700	3,400±0,118	96100±28900
18	F	1,274±0,005	19400±8400	1,273±0,004	16900±9400
	M	1,274±0,005	18600±1900	1,274±0,005	18400±9000
	S	1,273±0,004	18800±8700	1,274±0,005	18500±8600
19	F	1,266±0,004	71500±37400	1,263±0,004	64500±36000
	M	1,266±0,005	69500±37600	1,265±0,005	69500±37000
	S	1,266±0,005	74800±36400	1,266±0,005	69600±37600

Tabel 5.1: Variabele boeteparameters

5.2.1.1 Model 16

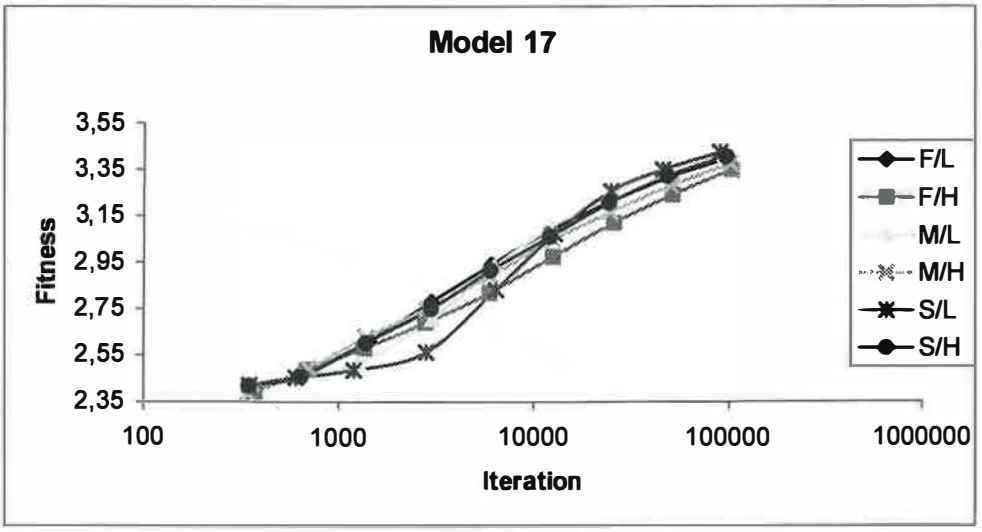
In figuur 5.1 is het resultaat te zien van model 16, hierin is de fitness uitgezet tegen het aantal iteraties (deze is uitgezet op een logaritmische schaal).



Figuur 5.1: Variabele boeteparameters voor model 16

Discussie:
Zoals in figuur 5.1 te zien is, is F/H vooral na weinig iteraties het beste. Maar naarmate er meer iteraties worden uitgevoerd wordt het verschil kleiner. Zoals in tabel 5.1 te zien is zijn de resultaten, na 32000 iteraties, statistisch niet significant.

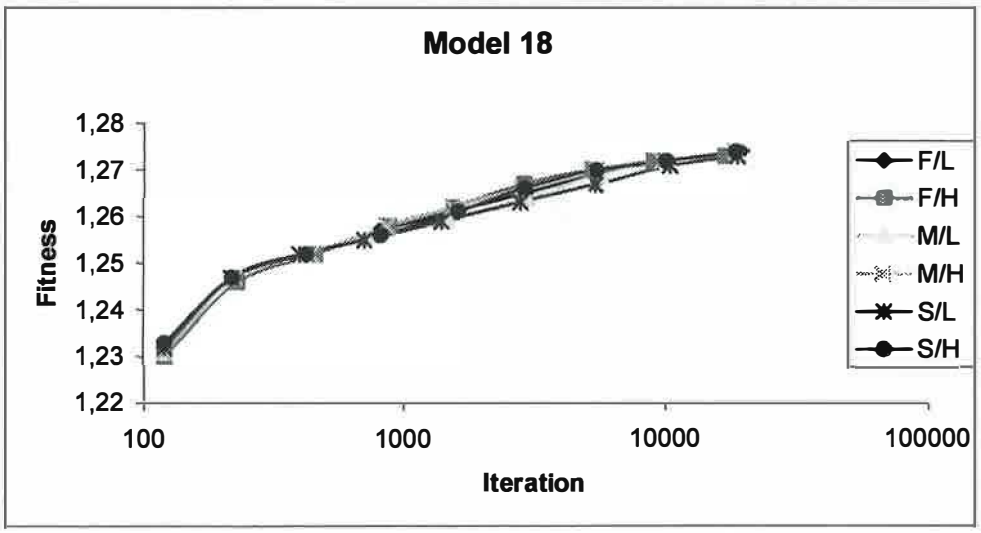
5.2.1.2 Model 17
In figuur 5.2 is het resultaat te zien van model 17.



Figuur 5.2: Variabele boeteparameters voor model 17

Discussie:
Zoals verwacht is de gemiddelde fitness in dit model een stuk lager, dit komt omdat er in dit model veel meer restricties verwerkt zijn.
Ook bij model 17 zijn de verschillen statistisch niet significant.
Als een proces naar een optimum convergeert moet er na een aantal iteraties een asymptoot in zicht komen. Daar dat hier niet echt het geval is kan men zich afvragen of het optimum al wel bereikt is.

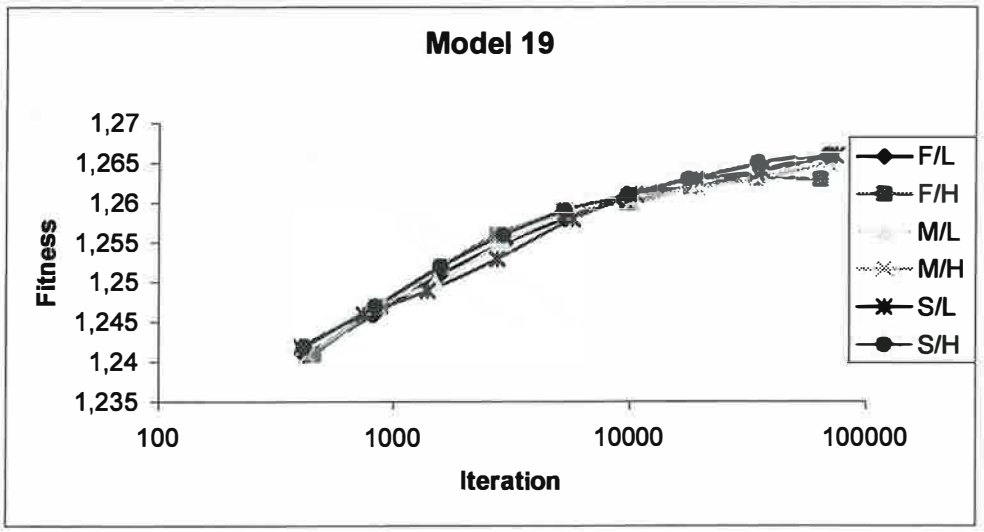
5.2.1.3 Model 18
In figuur 5.3 is het resultaat te zien van model 18.



Figuur 5.3: Variabele boeteparameters voor model 18

Discussie:
Het model 18 heeft een ander gedefinieerde fitness omdat het een I-MIN model is. Een vergelijking met voorgaande modellen is dus niet zinvol. Wat wel gezegd kan worden is dat het niet veel uitmaakt hoe k, δ en ε gekozen worden, de verschillen weer statistisch niet significant.

5.2.1.4 Model 19
In figuur 5.4 is het resultaat te zien van model 19.



Figuur 5.4: Variabele boeteparameters voor model 19

Discussie:
Ook voor dit model geldt weer dat er maar een klein verschil is tussen de verschillende instellingen voor k, δ en ε.

5.2.2 Simulatie 2

In de tweede simulatie zijn de verschillende manieren van notatie van de beslisvariabele onderzocht zoals beschreven is in paragraaf 4.2.5.
Een compacte notatie van de beslisvariabele:

$$x_i = \begin{cases} 0, \text{als item } i \text{ niet geselecteerd is} \\ j, \text{als item } i \text{ geselecteerd is in toets } j \end{cases}$$

Een uitgebreide notatie van de beslisvariabele:

$$x_{ij} = \begin{cases} 0, \text{als item } i \text{ niet geselecteerd is in toets } j \\ 1, \text{als item } i \text{ geselecteerd is in toets } j \end{cases}$$

De verwachting is dat de compacte notatie sneller convergeert dan de niet compacte notatie, dit komt omdat er bij de compacte versie al aan de overlap restrictie voldaan is, ook zijn de chromosomen *j* keer zo kort bij de compacte notatie en dit scheelt rekentijd.

Omdat de fitness bij parallelle toetsconstructie gebaseerd is op de toets met de laagste *y* waarde (zie paragraaf 4.2.5), is het waarschijnlijk dat epistase optreedt. Namelijk, alle toegelaten oplossingen waarvan de laagste *y* waarde hetzelfde is, hebben dezelfde fitness.
In deze simulatie worden twee oplossingen voor dit probleem getest, de bonus methode en de seizoenen methoden.

De bonus methode:
De bonus methode geeft, zoals de naam al doet vermoeden, een bonus aan iedere toets, dus ook de toets die niet de laagste *y* waarden heeft. Hierdoor ontstaan er dus kleine verschillen in fitness bij parallelle toetsen met dezelfde minimale toets.
Voor het I-MAX model komt de fitnessfunctie er dan als volgt uit te zien.

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

Hierin is ζ de bonusparameter
Nu moet vergelijking (4.17) vervangen worden door:

maximaliseer: y

onder voorwaarde dat: $y \le y_j \qquad \forall j$

$$y_j \le \frac{\sum_i I_{ik} F_j(x_i)}{T_k} \qquad \forall k, j$$

De seizoenen methode:
De seizoenen methode werkt met het principe dat als de verschillende fitnessfuncties in een korte cyclus van iteraties gebruikt worden, de individuen die het best aangepast zijn aan al deze fitnessfuncties de grootste kans hebben om te overleven. De analogie met de biologie is duidelijk, alleen individuen die in alle seizoenen een grote overlevingskans hebben, zullen nakomelingen produceren.
Voor de parallelle toetsconstructie zou dit bijvoorbeeld kunnen met een cyclus van twee iteraties. De eerste functie die geëvalueerd wordt is de functie beschreven in vergelijking (4.14). In de tweede iteratie zou een fitnessfunctie die niet gebaseerd is op de minimale toets gebruikt kunnen worden. Deze alternatieve functie zou er als volgt uit kunnen zien:

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

Voor deze simulatie gebruiken we weer model 16, model 17, model 18 en model 19. Bij ieder model wordt van zowel de compacte (C) notatie als de uitgebreide (U) notatie de bonus (B) en de seizoenen (S) methode getest. Bij model 16 en 17 is de

bonus methode getest met een ζ=0, ζ=0,001 en ζ=0,01. Bij model 18 en 19 is de bonus methode getest met een ζ=0 en ζ=0,001, maar omdat dit I-MIN modellen zijn treedt ook de epistase zoals beschreven in 4.2.3.1 op. Om dit te verhelpen zullen we zowel de seizoenen methode als de bonus methoden testen met γ=0 en γ=0,001.

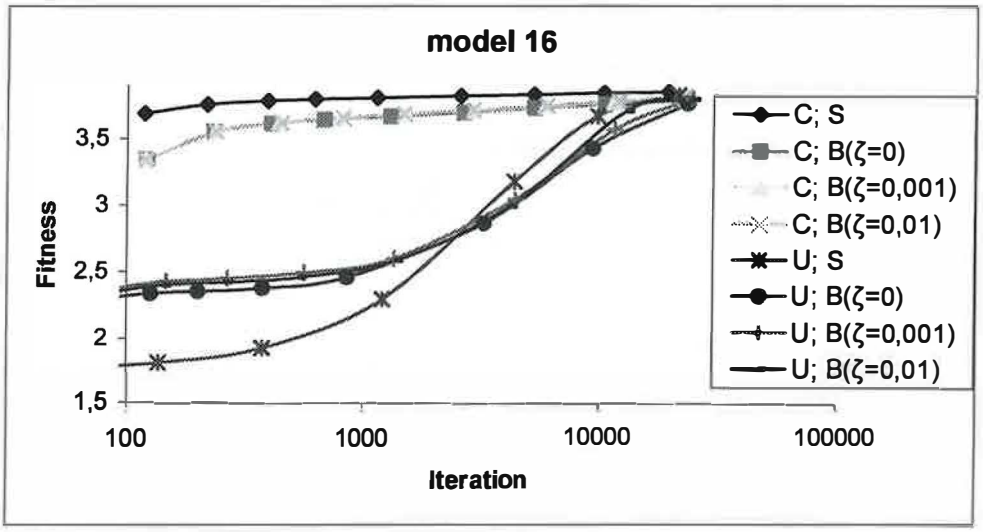
In tabel 5.2 zijn de resultaten te zien van model 16, 17, 18 en 19.

		C		U	
		fitness	iteraties	fitness	iteraties
16	S	3,8442±0,0180	19300±9400	3,8197±0,1567	21400±8300
	B(ζ=0)	3,7820±0,0562	22200±9100	3,7590±0,1871	23300±7300
	B(ζ=0,001)	3,8010±0,0525	22600±8800	3,7956±0,1391	24400±6700
	B(ζ=0,01)	3,8056±0,0510	22800±8500	3,7908±0,1589	24300±6500
17	S	3,4874±0,0821	120200±55800	3,4891±0,0643	120000±47800
	B(ζ=0)	3,4905±0,0853	125100±50800	3,4785±0,0789	124200±45500
	B(ζ=0,001)	3,4891±0,0841	125800±50100	3,5027±0,0585	137700±43600
	B(ζ=0,01)	3,4708±0,0984	141600±45000	3,4861±0,0860	127100±46800
		aantal items	iteraties	aantal items	iteraties
18	S; γ=0	30,025±0,753	4600±6900	29,421±0663	18200±10000
	B(ζ=0); γ=0	30,150±0,768	4500±7100	29,437±0,649	17300±9400
	B(ζ=0,001); γ=0,001	28,920±0,405	19200±8800	29,035±0,307	17200±8900
	S; γ=0,001	28,955±0,392	20600±8000	28,990±0,284	17900±8800
19	S; γ=0	30,390±0,700	35900±45300	29,710±0,517	52100±44400
	B(ζ=0); γ=0	30,460±0,813	33100±46500	29,690±0,485	54400±45600
	B(ζ=0,001); γ=0,001	29,340±0,475	101600±54200	29,335±0,473	87600±51200
	S; γ=0,001	29,375±0,496	103100±54000	29,340±0,475	88100±52200

Tabel 5.2: Seizoenen en bonus methode

5.2.2.1 Model 16

In figuur 5.5 is het resultaat te zien van model 16.

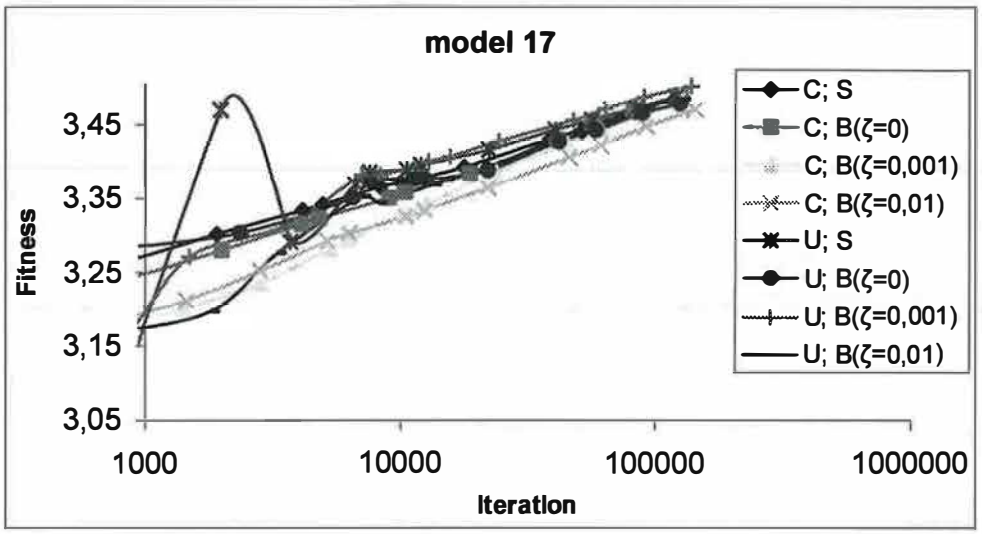


Figuur 5.5: Seizoenen en bonus methode voor model 16

Discussie:
Vooral na weinig iteraties is het verschil heel groot tussen de compacte notatie en de uitgebreide notatie. Ook presteert bij dit model de seizoenen methode iets beter dan de bonusmethode.

5.2.2.2 Model 17

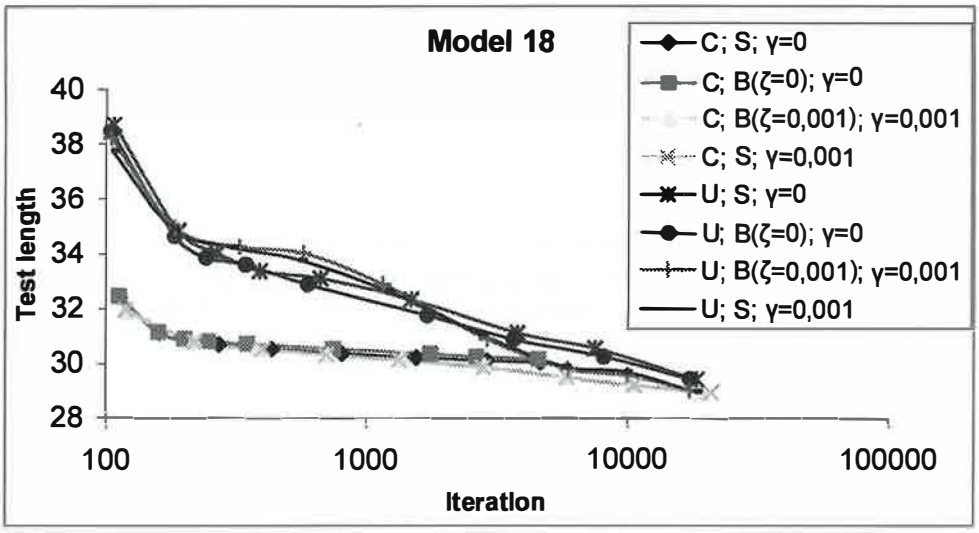
In figuur 5.6 is het resultaat te zien van model 17.



Figuur 5.6: Seizoenen en bonus methode voor model 17

Discussie:
Bij dit model presteert de uitgebreide notatie met de bonusmethode waarvan de bonusparameter ζ waarde 0,001 heeft het beste, maar de verschillen zijn erg klein. Wat opvalt is dat één van de modellen in het begin ver boven de andere uitschiet en daarna weer terugzakt. De oorzaak hiervan is dat de uitgebreide notatie meer moeite heeft met het vinden van een toegelaten oplossing dan de compacte notatie. Wat er gebeurt is dat er na ongeveer 2000 iteraties nog maar enkele toegelaten oplossingen gevonden zijn en als die 'toevallig' allemaal een hoge fitness hebben, zal de gemiddelde fitness ook hoog zijn.
Dat dit probleem zou optreden was te verwachten, immers de compacte notatie heeft al aan de overlaprestrictie voldaan. Dat de verschillen echter zo groot zijn is opvallend, de compacte notatie heeft al na gemiddeld 3000 iteraties een populatie met alleen maar toegelaten oplossingen, de uitgebreide notatie heeft daar 50000 iteraties voor nodig.

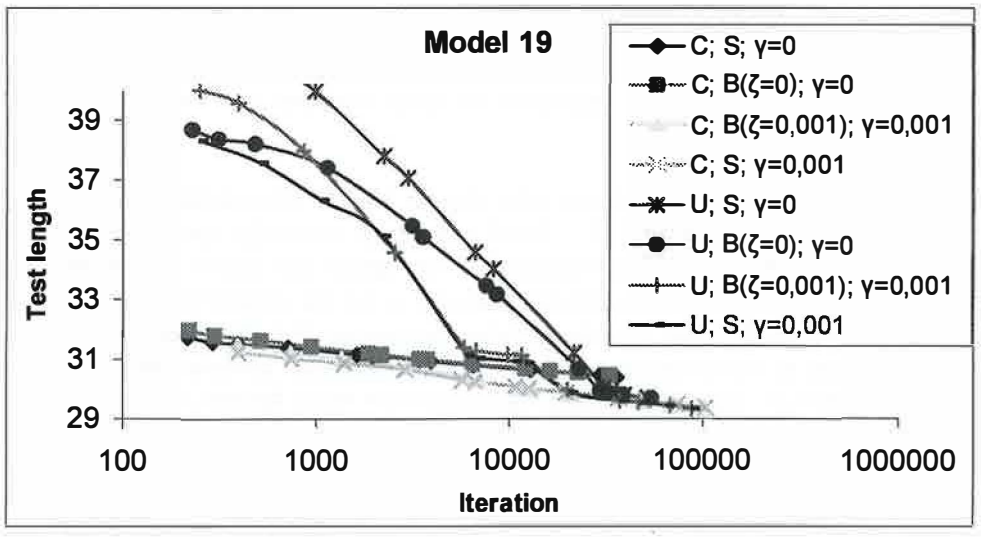
5.2.2.3 Model 18
In figuur 5.7 is het resultaat te zien van model 18. Omdat dit een I-MIN model is hebben we niet de fitness, maar de toetslengte op de y-as uitgezet. Immers de toetslengte is de doelfunctie.



Figuur 5.7: Seizoenen en bonus methode voor model 18

Discussie:
De compacte notatie is vooral na weinig iteraties veel beter dan de uitgebreide notatie. Naar mate er meer iteraties uitgevoerd worden wordt het verschil kleiner. Voor zowel de compacte als de uitgebreide notatie geldt dat de bonusmethode en de seizoenen methode betere oplossingen vinden wanneer voor de bonusparameter γ de waarde 0,001 gekozen word, de methode zoals beschreven in 4.2.3.1 is dus een goede manier om de epistase op te lossen. De bonus methode met ζ=0,001 verschilt niet veel met de seizoenen methode.

5.2.2.4 Model 19
In figuur 5.8 is het resultaat te zien van model 17.



Figuur 5.8: Seizoenen en bonus methode voor model 19

Discussie:
Ook voor model 19 geldt dat de compacte notatie vooral na weinig iteraties veel beter is dan de uitgebreide notatie. Ook voor dit model geldt dat de bonusmethode met ζ=0,001 en de seizoenen methoden het beste presteren mits voor de bonusparameter γ de waarde 0,001 gekozen word.

Het fenomeen beschreven bij model 17 treedt bij alle modellen op, de compacte notatie heeft na relatief weinig iteraties een populatie met alleen toegelaten oplossingen, de uitgebreide notatie heeft daar veel meer iteraties voor nodig.

5.2.3. Simulatie 3

In de derde simulatie zijn de verschillende stopcriteria getest.
We hebben vier verschillende criteria getest, die alle gerelateerd zijn aan het aantal restricties *I* van het model, het aantal parallelle toetsen *P* van het model en het aantal items *K* in de itembank.
De vier stopcriteria zijn:

1. Stoppen na *IK* iteraties
2. Stoppen na *IKP* iteraties
3. Stoppen na *IK*ln(*IK*) iteraties
4. Stoppen na *IKP*ln(*IKP*) iteraties

In tabel 5.3 zijn de resultaten te zien van model 17 en 19.

	Model 17		Model 19	
	fitness	iteraties	aantal items	iteraties
1	3,2938±0,1956	6300±2600	30,265±0,630	7100±2100
2	3,3417±0,1555	12700±5300	29,975±0,419	13000±4200
3	3,4425±0,1048	60700±22100	29,550±0,499	49100±24600
4	3,4891±0,0841	125800±50100	29,340±0,475	10200±54200

Tabel 5.3: stopcriteria

Om nu iets te zeggen over deze stopcriteria is het nuttig om het werkelijke optimum te weten. Daar we dit niet weten moeten we een schatting doen naar wat het werkelijke optimum zou zijn. Dit doe we door bij model 17 de groots waargenomen fitness als optimum te nemen en bij model 19 door de kleinst waargenomen toetslengten als optimum te nemen.

In tabel 5.4 is het optimum en de procentuele afwijking van de verschillende stopcriteria te zien.

Model	optimum	*IK*	*IKP*	*IK*ln(*IK*)	*IKP*ln(*IKP*)
17	3,5643	7,6%	6,2%	3,4%	2,1%
19	29	4,4%	3,4%	1,9%	1,2%

Tabel 5.4: stopcriteria en procentuele afwijking

Als de doelstelling bij het optimaliseren is dat je binnen 5% van de optimale oplossing moet zitten, dan is, voor model 17, het derde stopcriterium voldoende en voor model 19 de eerste.
Merk wel op dat tabel 5.4 gebaseerd is op de gemiddelde fitness/toetslengte, dus dit geldt niet altijd voor individuele gevallen.

5.2.4. Simulatie 4

Tijdens voorgaande simulaties viel op dat er een verschil in rekentijd is tussen de compacte en de uitgebreide notatie. En daar we op zoek zijn naar een goede oplossing in een korte tijd is dit iets om nader te onderzoeken.
Om andere factoren die de rekentijd kunnen beïnvloeden constant te houden is de simulatie op één computer uitgevoerd.
Uit de resultaten van deze simulatie blijkt dat de verschillen in rekentijd niet zo groot zijn als verwacht.
De rekentijd voor de uitgebreide notatie is ongeveer 1,25 keer zo langzaam als de rekentijd voor de compacte notatie.
Daar de verschillen redelijk klein zijn, zal de rekentijd geen doorslaggevende rol gaan spelen bij het besluit of we de compacte, dan wel de uitgebreide notatie kiezen.

5.2.5. Simulatie 5

In the laatste simulatie hebben we nog gekeken naar de verschillen tussen uniforme crossover en één-punt crossover. In het theoretische deel van dit verslag hebben we de één-punt crossover besproken.
De één-punt crossover ziet er schematisch als volgt uit:

$$\left.\begin{matrix}0010|1100\\0111|1001\end{matrix}\right\} \xrightarrow{\text{1-punt crossover}} \left\{\begin{matrix}0010|1001\\0111|1100\end{matrix}\right.$$

Er wordt random een positie gekozen waar een koppel chromosomen wordt doorgeknipt. De achterste delen van de chromosomen worden omgewisseld.

De uniforme crossover ziet er schematisch als volgt uit:

$$\left.\begin{matrix}00|1|01|1|0|0\\01|1|11|0|0|1\end{matrix}\right\} \xrightarrow{\text{uniforme crossover}} \left\{\begin{matrix}00|1|01|0|0|1\\01|1|11|1|0|0\end{matrix}\right.$$

Met een kans van ½ wisselt iedere gen van de ene chromosoom van waarden met hetzelfde gen van het andere chromosoom. In het schema zijn dit gen 3,6,7 en 8.

De keuze tussen één-punt crossover en uniforme crossover hangt af van de structuur van de chromosomen.
Als er een relatie bestaat tussen een gen en het naast gelegen gen, is het verstandig om de één-punt crossover operator te gebruiken.
Als er geen enkele relatie bestaat tussen een gen en het naast gelegen gen, is het verstandig om de uniforme crossover operator te gebruiken.

Omdat er bij de compacte notatie geen relatie bestaat tussen een gen en het naast gelegen gen, zal hier naar verwachting de uniforme crossover het snelste convergeren.
Bij de uitgebreide notatie is er wel een relatie tussen een gen en het naast gelegen gen, dus misschien zal hierbij de één-punt crossover beter presteren.

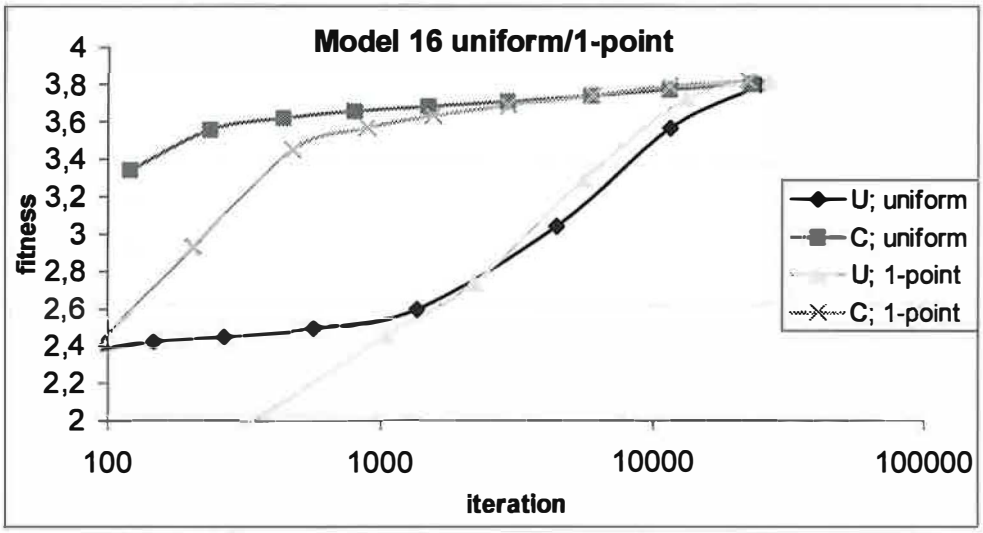
Bij deze simulatie zijn, voor model 16, 17, 18 en 19, de één-punt crossover en uniforme crossover getest op zowel de uitgebreide als de compacte notatie.

De resultaten zijn te zien in tabel 5.5.

		C		U	
		fitness	iteraties	fitness	iteraties
16	één-punt	3,8201±0,0723	22000±8900	3,8200±0,0344	26500±5300
	uniform	3,8010±0,0525	22600±8800	3,7956±0,1391	24400±6700
17	één-punt	3,5133±0,0716	123900±51400	3,4091±0,0853	152900±32400
	uniform	3,4891±0,0841	125800±50100	3,5027±0,0585	137700±43600
		aantal items	iteraties	aantal items	iteraties
18	één-punt	29,085±0,457	24100±6600	29,165±0,446	24500±6300
	uniforme	28,920±0,405	19200±8800	29,035±0,307	17200±8900
19	één-punt	29,380±0,487	122100±45800	29,385±0,498	103800±53900
	uniform	29,340±0,475	101600±54200	29,335±0,473	87600±51200

Tabel 5.5: één-punt en uniforme crossover

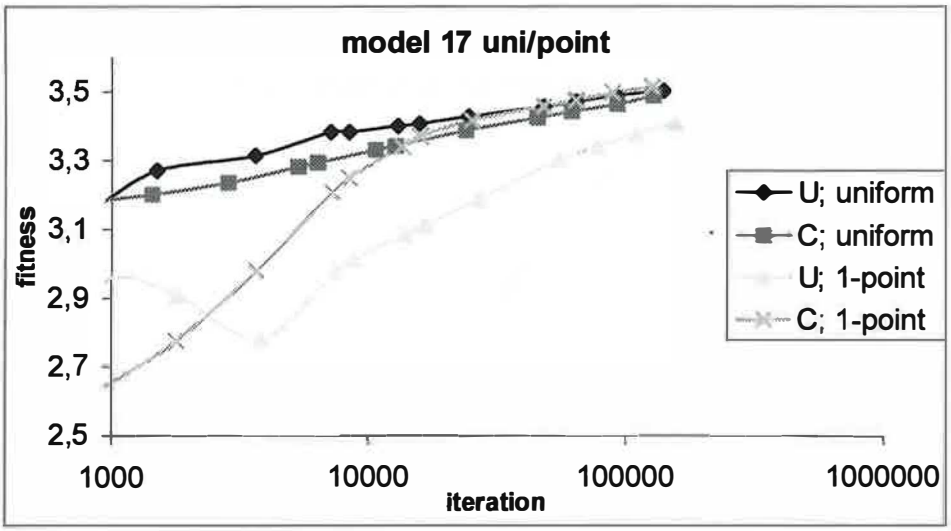
5.2.5.1 Model 16
In figuur 5.9 is het resultaat te zien van model 16.



Figuur 5.9: uniforme en 1-punt crossover voor model 16

Discussie:
De één-punt crossover is bij de uitgebreide notatie niet significant beter. Bij de compacte notatie is de uniforme crossover vooral na weinig iteraties beter.

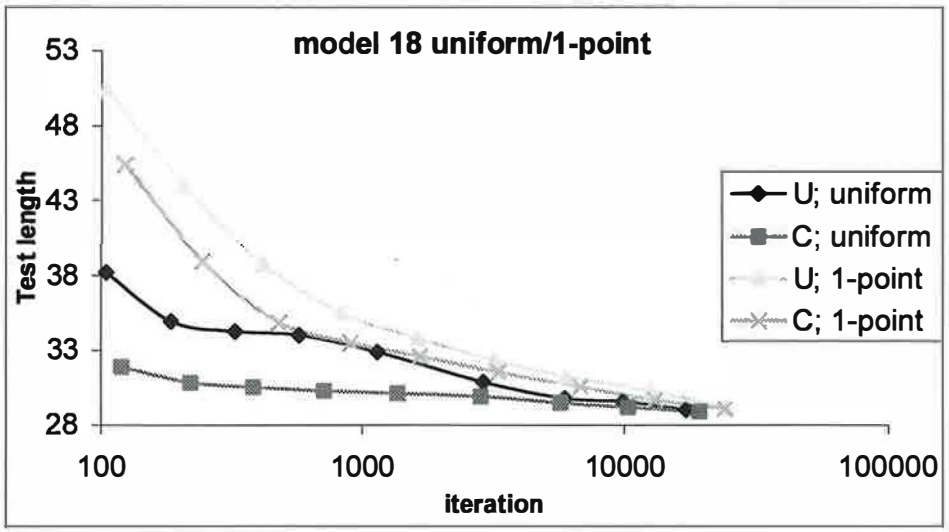
5.2.5.2 Model 17
In figuur 5.10 is het resultaat te zien van model 17.



Figuur 5.10: uniforme en 1-punt crossover voor model 17

Discussie:
De één-punt crossover is bij de uitgebreide notatie slechter dan de uniforme crossover. Bij de compacte notatie is het verschil tussen de één-punt crossover en de uniforme crossover, vooral na veel iteraties, erg klein. Dit is toch opmerkelijk want de verwachting was dat de uniforme notatie in alle gevallen beter zou zijn dan de één-punt.

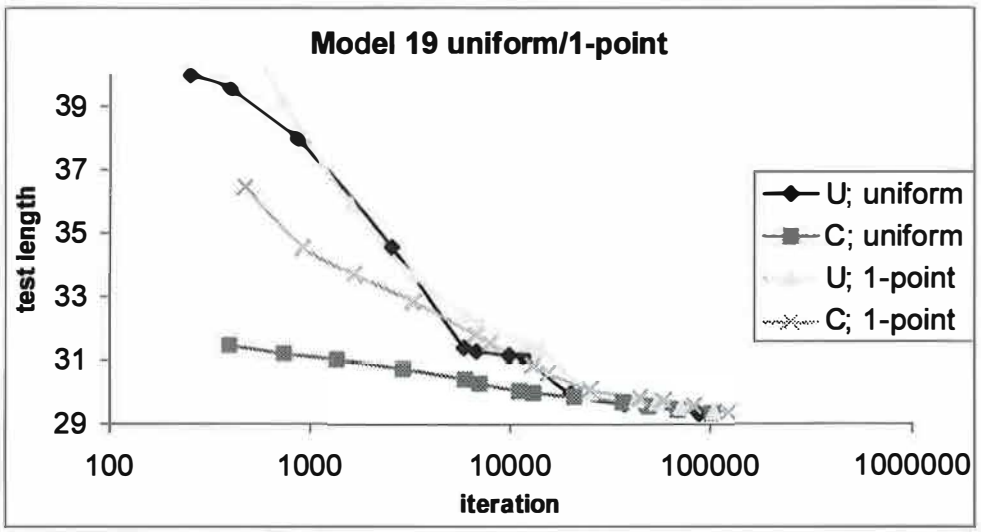
5.2.5.3 Model 18
In figuur 5.11 is het resultaat te zien van model 18.



Figuur 5.11: uniforme en 1-punt crossover voor model 18

Discussie:
De één-punt crossover is na weinig iteraties slechter dan de uniforme crossover. Na 32000 iteraties zijn de verschillen statistisch niet significant.

5.2.5.4 Model 19
In figuur 5.12 is het resultaat te zien van model 19.



Figuur 5.12: uniforme en 1-punt crossover voor model 19

Discussie:
Voor dit model geldt hetzelfde als het voorgaande model. De één-punt crossover is na weinig iteraties slechter dan de uniforme crossover. Hoe meer iteraties er uitgevoerd worden hoe kleiner het verschil.

6. Conclusies en Bevindingen

Dit zijn enkele kwalitatieve conclusies uit de simulaties:

- Bij een model met relatief weinig restricties kan het best gekozen worden voor variabele boeteparameters met een korte cyclus en grote sprongen (F/H). Bij een model met relatief veel restricties kan het best gekozen worden voor variabele boeteparameters met een lange cyclus en kleine sprongen (S/L). Maar de verschillen zijn gering.
- Bij een I-MAX model met relatief weinig restricties kan het beste gekozen worden voor de compacte notatie van de beslisvariabele, de beste oplossing voor de epistase van de parallelle toetsconstructie is de seizoenen methode. Bij een I-MAX model met relatief veel restricties is de keuze voor de notatie van de beslisvariabele afhankelijk van hoeveel iteraties je bereidt bent uit te voeren. De uitgebreide notatie van de beslisvariabele met de bonusmethode en bonusparameter ζ=0,001 geeft de beste resultaten, maar het nadeel van de uitgebreide notatie is dat deze minstens 50000 iteraties nodig heeft om met grote zekerheid een toegelaten oplossing te krijgen.
- Bij een I-MIN model kan het beste gekozen worden voor de compacte notatie van de beslisvariabele, de beste oplossing voor de epistase van de parallelle toetsconstructie is de bonus methode met ζ=0,001, maar het verschil met de seizoenenmethode is erg klein. Om de epistase van het I-MIN model op te lossen moeten we ook bonusparameter γ de waarde 0,001 geven.
- Er bestaat maar een gering verschil in rekentijd tussen de compacte en uitgebreide notatie van de beslisvariabele. De rekentijd zal geen doorslaggevende rol spelen bij het besluit om de compacte, dan wel de uitgebreide notatie te kiezen.
- De één-punt crossover presteert voor de I-MAX modellen in sommige gevallen beter dan de dan de uniforme crossover.
- De één-punt crossover presteert voor de I-MIN modellen slechter dan de uniforme crossover.

Door dit onderzoek zijn er nieuwe vragen naar boven gekomen waar een vervolgstudie helderheid in zou moeten verschaffen. Er zijn twee zaken die nog een nader bekeken dienen te worden:

i. Als een proces naar een optimum convergeert moet er na een aantal iteraties een asymptoot in zicht komen. Daar dit bij de meeste simulaties niet echt het geval is kan men zich afvragen of het optimum al wel bereikt is.
In een vervolgstudie zouden dezelfde simulaties nog eens uitgevoerd moeten worden, maar dan met veel meer iteraties. Dit om te kijken of er na veel meer iteraties wel een asymptoot komt.

ii. De simulaties beschreven in paragraaf 5.2.5 gaven onverwachte resultaten. Namelijk dat de één-punt crossover in sommige gevallen beter is dan de uniforme crossover. Omdat er, bij de simulaties voorafgaand aan die in paragraaf 5.2.5, gebruik is gemaakt van de uniforme crossover, moeten in een vervolgstudie dezelfde simulaties nogmaals uitgevoerd worden, maar dan gebruikmakend van de één-punt crossover.

Literatuurlijst:

- Aytug, H. en Koehler, G.J., 1995, *Stopping Criteria for Finite Length Genetic Algorithms*, Journal on Computing.
- Eggen, T.J.H.M. en Sanders P.F., 1993, *Psychometrie in de Praktijk*, Arnhem, CITO.
- Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company, Inc.
- Holland, J., 1975, *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press.
- Theunissen, T., 1985, *Binary Programming and Test Design*, Psychometrika.
- Verschoor, A., 2003, *Test Assembly Using Genetic Algorithms*.
- Whitley, L. D., 1993, *Foundations of Genetic Algorithms 2*, Morgan Kaufmann Publishers, Inc.