Inhoud

Samenvatting

Boers en Kuiper hebben in 1992 een methode ontwikkeld om de architectuur van modulaire artificiële neurale netwerken te bepalen. Deze methode is gebaseerd op twee op in de biologie voorkomende mechanismen. Genetische algoritmen worden gebruikt als zoekmethode naar een optimale architectuur en met behulp van L-systemen worden de netwerken opgebouwd. De behaalde resultaten indiceren dat modulaire netwerken gevonden kunnen worden die de standaard oplossingen overtreffen. Een nadeel is echter de enorme rekencapaciteit die nodig is om tot een optimale oplossing te komen.

Het doel van dit onderzoek was om de hierboven beschreven methode te optimaliseren. Hierbij is de nadruk gelegd op het optimaliseren van de parameters die het genetisch algoritme sturen en de parameters die het L-systeem optimaal laten werken. Het blijkt dat aanpassing van deze parameters een aanzienlijke snelheidswinst oplevert en dat de benodigde rekentijd een normale verdeling benadert.

1. Inleiding

Zowel in de biologie en de psychologie als in de informatica is er grote belangstelling naar de werking van de hersenen. Bij alle drie de vakgebieden wordt gebruik gemaakt van modellen die de werking van de hersenen op kleine schaal nabootsen, de zogenaamde artificiële neurale netwerken. In de biologie en psychologie hoopt men door het ontwerpen en trainen van deze netwerken meer inzicht te verkrijgen in de manier waarop de hersenen kunnen leren en gegevens kunnen opslaan. In de informatica is men op zoek naar systemen die uiteindelijk in staat zijn om te leren en in kunnen spelen op onverwachte situaties. Met kleine eenvoudige problemen zijn al goede resultaten bereikt. Er zijn bijvoorbeeld artificiële neurale netwerken ontworpen die in staat zijn om backgammon te leren spelen [Tesauro and Sejnowski, 1989] of die een voertuig kunnen besturen [Pomerleau, 1989].

Artificiële neurale netwerken zijn opgebouwd uit units die onderling verbonden kunnen zijn. Dit is vergelijkbaar met de bouw van de hersenen. Deze bestaan uit een groot aantal zenuwcellen die onderling verbonden kunnen zijn. Grofweg zijn de units in drie verschillende taakgebieden te verdelen:

- *invoerunits* ontvangen signalen van buitenaf en zijn vergelijkbaar met de zintuigen
- *uitvoerunits* geven signalen aan de buitenwereld
- *verborgen units* verwerken de signalen van de invoerunits en geven het resultaat door aan de uitvoerunits.

Het ontwerpen van artificiële neurale netwerken is al jaren een groot probleem. Hoeveel units moet het netwerk minimaal bevatten om een taak te kunnen leren? Welke verbindingen zijn er nodig tussen de verschillende units? Is het gevonden netwerk het optimale netwerk? Deze vragen zijn doorgaans wel te beantwoorden voor kleine netwerken die slechts één simpele taak uitvoeren. Zodra netwerken één of meer ingewikkelde problemen moeten kunnen uitvoeren, zijn bovenstaande vragen niet of nauwelijks meer te beantwoorden.

Het is dan ook een logische stap om het ontwerpen van artificiële neurale netwerken te automatiseren. Boers en Kuiper [1992] hebben hiervoor een methode ontwikkeld die gebruik maakt van twee systemen die gebaseerd zijn op in de natuur voorkomende mechanismen. Deze systemen zijn de genetische algoritmen en de L-systemen.

Genetisch algoritme
Het genetische algoritme is een op de evolutie gebaseerde zoekmethode. Het maakt gebruik van een populatie bestaande uit individuen die door één of meerdere chromosomen worden gerepresenteerd. Deze chromosomen bevatten gecodeerd de parameters die een mogelijke oplossing voor het probleem vormen. De kwaliteit van deze mogelijk oplossingen wordt omgezet in een getal, de *fitnesswaarde*. Deze fitnesswaarde wordt aan elk individu gekoppeld, elk individu bezit zodoende een bepaalde overlevingskans. Individuen worden met elkaar gekruist, waarbij in de natuur voorkomende mechanismen, zoals crossing-over en mutatie kunnen optreden. Van de nakomelingen wordt de fitness bepaald. Individuen met een hoge fitness, en dus een grote overlevingskans, mogen zich vaker voortplanten dan individuen met

een lage fitness. Individuen met een lage fitness verdwijnen uit de populatie. Door deze selectie wordt de gemiddelde fitness van de populatie steeds hoger en de oplossingen benaderen steeds beter de optimale oplossing (het principe van survival of the fittest van Darwin [1859]).

L-systemen

L-systemen zijn in 1968 ontwikkeld door de bioloog Aristid Lindenmayer om de groei van planten te modelleren. Er wordt gebruik gemaakt van *produktieregels*: regels die het mogelijk maken om een *string* (een rijtje karakters) te herschrijven in een andere string. Later is aan dit L-systeem een interpretatie toegevoegd zodat het oorspronkelijk wiskundige model omgezet kon worden in een grafisch model. Door aan elk karakter een bepaalde betekenis te geven kan de string worden omgezet in een figuur. Wanneer bijvoorbeeld de karakters *F* en + het volgende betekenen:

F: Teken een rechte lijn van 10 centimeter van je af

+: Draai het papier over 90^o met de klok mee.

dan ontstaat er uit de string F+F+F+F een vierkant van 10x10 centimeter.

Door met verschillende strings te beginnen (*axioma*), door andere produktieregels te gebruiken, door het aantal stappen te variëren waarin de ene string in de ander wordt herschreven of door de interpretatie van de karakters te veranderen, zijn er vele verschillende structuren te ontwerpen.

De methode van Boers en Kuiper

Bij de methode die Boers en Kuiper [1992] hebben ontwikkeld, wordt een genetisch algoritme gebruikt om produktieregels te vinden voor het L-systeem. Door een specifieke interpretatie te geven aan de verschillende karakters wordt de structuur van een artificieel neuraal netwerk bepaald. Deze methode is grofweg in drie stappen in te delen:

stap 1. Het genetisch algoritme levert een chromosoom die produktieregels bevat aan het L-systeem.

stap 2. Het speciale L-systeem vormt uit de verkregen produktieregels, door middel van een speciale interpretatie, een netwerk.

stap 3. Van dit netwerk wordt bepaald in hoeverre het geschikt is om een vastgestelde taak te leren. De mate van geschiktheid wordt omgezet in een fitnesswaarde. Deze fitnesswaarde wordt teruggekoppeld naar het chromosoom dat de produktieregels heeft geleverd.

Het genetisch algoritme zal vervolgens bepalen of de fitness van het individu hoog genoeg is om in de populatie te worden opgenomen. Indien dit het geval is dan wordt het individu met de laagste fitness vervangen door het nieuwe individu.

In hoofdstuk 2 wordt het genetisch algoritme, het L-systeem en de door Boers en Kuiper [1992] ontwikkelde methode uitgebreid beschreven.

Resultaten

Met bovenstaand algoritme hebben Boers en Kuiper veelbelovende resultaten behaald. Bij de problemen die zij onderzochten werd altijd een geschikte architectuur voor het netwerk gevonden, waardoor het netwerk in staat was het probleem te leren. Een van de onderzochte problemen is een patroonherkenningsprobleem, het TC-probleem. Hierbij moet het netwerk

leren of een T of een C wordt aangeboden. Beide letters bestaan uit 3x3 punten en kunnen geroteerd zijn over een hoek van 0, 90, 180 of 270°. De letters staan op een willekeurige plaats in een 4x4 raster. Het netwerk dat na 7500 genetische recombinaties (fitnessbepalingen) gevonden werd, leerde sneller en beter welk patroon er werd aangeboden dan een aantal met de hand geconstrueerde netwerken.

Ondanks de kracht van de combinatie van L-systemen met genetische algoritmen is het bepalen van de architectuur van een neuraal netwerk een tijdrovende klus. Voor het hierboven genoemde TC-probleem waren 7500 fitnessbepalingen nodig. Voor het ingewikkelder mapping-probleem, waarbij het netwerk een figuur uit een kaart van 10x10 figuren correct moet indelen in één van vier mogelijke klassen, waren maar liefst 35.000 fitnessbepalingen nodig om een netwerk te vinden dat het probleem kon leren (3 dagen op 11 SUN Sparc4 workstations). Een verbetering van dit netwerk werd pas na 85.000 fitnessbepalingen gevonden.

Doel van dit onderzoek

Het moge duidelijk zijn dat het bepalen van de architectuur van een neuraal netwerk een enorme rekencapaciteit vergt. *De doelstelling van dit onderzoek is om het door Boers en Kuiper ontworpen algoritme te optimaliseren zodat het sneller tot resultaten komt*. Hierbij is de nadruk gelegd op het optimaliseren van het genetisch algoritme en de produktieregels van het L-systeem. Omdat het trainen van een neuraal netwerk in dit onderzoek niet noodzakelijk is, is dit onderdeel vervangen. Hiertoe is gebruik gemaakt van een alternatieve fitnessbepaling die aanzienlijk sneller de fitness van een netwerk bepaalt dan de conventionele bepaling.
Hoofdstuk 2 beschrijft een aantal theoretische aspecten van de gebruikte technieken. Daarnaast wordt de methode die Boers en Kuiper hebben ontwikkeld uitgebreid behandeld. *Hoofdstuk 3* bevat een beschrijving van de voor dit onderzoek gebruikte software.

Alternatieve fitnessbepaling

Voor het bepalen van de fitness van een neuraal netwerk wordt het getraind met behulp van een leeralgoritme. Dit trainen is zeer tijdrovend en niet noodzakelijk wanneer alleen het genetisch algoritme en het L-systeem wordt geoptimaliseerd. Bij dit onderzoek zijn voor het uitvoeren van experimenten tienduizenden fitnessbepalingen nodig, hetgeen extreem veel rekencapaciteit zou kosten. Daarom is het noodzakelijk dat de fitness van een netwerk op een andere snellere wijze wordt bepaald. Als eerste zijn daarom een aantal criteria opgesteld waaraan een netwerk zou moeten voldoen om een probleem goed te kunnen leren. Door het combineren van deze criteria in een fitnessfunctie kan bepaald worden of het netwerkprogramma geschikte netwerk architecturen produceert. De onderzochte criteria en de daaruit resulterende alternatieve fitnessfunctie worden beschreven in *hoofdstuk 4*. Met deze alternatieve fitnessbepaling zijn uiteindelijk alle experimenten uitgevoerd.

Optimaliseren van de parameters van het genetisch algoritme

Het genetisch algoritme bevat een aantal verschillende parameters. Wanneer twee individuen gekruist worden, wordt er gebruik gemaakt van een aantal genetische operatoren, zoals mutatie (het willekeurig veranderen van genetische informatie) en crossing-over (uitwisselen van delen van chromosomen). Er worden parameters gebruikt die de kans bepalen dat

crossing-over of mutatie plaatsvindt. De kans dat een individu uit de populatie wordt gekozen wordt bepaald door een selectie factor, de pressure. Deze bepaalt in hoeverre individuen met een hoge fitness vaker worden geselecteerd dan individuen met een lage fitness. Daarnaast worden er parameters gebruikt die de grootte van de populatie en de lengte van de chromosomen waaruit de individuen bestaan vastleggen.

De invloed van de waarde van een aantal parameters op de snelheid waarmee een "optimale" oplossing gevonden wordt, is groot. Bijvoorbeeld bij een te grote mutatiefactor ontaardt het zoeken naar een optimaal netwerk in random zoeken. Wanneer de pressurewaarde waarmee de selectie wordt uitgevoerd te hoog is, worden alleen de beste individuen uit de populatie gehaald om verder te kruisen. De variatie in de populatie wordt dan erg klein, hetgeen kan resulteren in een populatie die uit identieke individuen bestaat (vroegtijdige convergentie).

Om het bestaande genetische algoritme te optimaliseren is het van belang de waarde van deze parameters zo goed mogelijk vast te stellen. In *hoofdstuk 5* wordt beschreven hoe het optimaliseren van deze parameters met behulp van een tweede genetisch algoritme in zijn werk is gegaan.

Optimaliseren van de selectiemethode van het genetisch algoritme
Bij een genetisch algoritme ontstaan na kruising van twee individuen twee nakomelingen. Bij de methode die door Boers en Kuiper [1992] is gebruikt, wordt er van deze twee nakomelingen willekeurig één gekozen waarvan vervolgens de fitness wordt bepaald. De ander wordt direct verworpen. Deze methode wordt aangehouden omdat het bepalen van de fitness van een individu een tijdrovende aangelegenheid is. Een nadeel van deze werkwijze is dat daardoor niet de beste nakomeling wordt geselecteerd en bewaard. In *hoofdstuk 6* worden experimenten beschreven die tot doel hebben de beste nakomeling te selecteren zonder dat de fitness van beide nakomelingen bepaald hoeft te worden.

Optimaliseren van de conversietabel van het L-systeem
In een chromosoom staan de produktieregels die nodig zijn voor het L-systeem gecodeerd opgeslagen. Voor het decoderen van het chromosoom wordt gebruik gemaakt van een *conversietabel*. Om deze tabel te optimaliseren is een aantal experimenten uitgevoerd die vooral gericht zijn om het aantal bruikbare produktieregels te vergroten. In *hoofdstuk 7* worden deze experimenten met een aantal verschillende conversietabellen beschreven.

Gelijktijdig optimaliseren van de parameterset en de conversietabel.
Het gelijktijdig optimaliseren van zowel de parameters van het genetische algoritme als de conversietabel van het L-systeem is eveneens met behulp van een tweede genetisch algoritme uitgevoerd. Dit experiment wordt beschreven in *hoofdstuk 8*.

Hoofdstuk 9 bevat de conclusies en aanbevelingen voor verder onderzoek.

2. Genetische algoritmen, L-systemen en Neurale netwerken

In dit onderzoek zijn drie verschillende technieken gecombineerd: genetische algoritmen, L-systemen en neurale netwerken. Deze gebruikte technieken hebben een biologische oorsprong. Genetische algoritmen zijn gebaseerd op de genetica en de evolutie-theorie, L-systemen zijn oorspronkelijk ontworpen om de groei van planten te modelleren en neurale netwerken zijn gebaseerd op de werking van de hersenen.

2.1. Genetische algoritmen

2.1.1. Inleiding

Er bestaan tal van problemen waarvoor geen eenvoudig rekensommetje bestaat dat het optimale antwoord geeft. Met behulp van een computer is het antwoord op dit soort problemen vaak wel simpel te bepalen, maar dit kost maanden rekentijd. Een voorbeeld is de kortste weg tussen 3038 soldeerpunten van een printplaat. Het antwoord op dit probleem is onlangs gevonden door David Applegate van het Amerikaanse AT&T Bell laboratorium. Een computer heeft er anderhalf jaar over gedaan om de optimale route te berekenen [Vuik, 1992]. Behalve door simpelweg alle mogelijkheden te bepalen of door random te zoeken naar het antwoord, zijn deze optimalisatieproblemen ook anders op te lossen: bijvoorbeeld door de natuur te imiteren.

De principes die hiervoor gebruikt worden zijn afkomstig uit de evolutieleer en zijn voornamelijk gebaseerd op het Neo-Darwinisme. Darwins theorieën over de evolutie zijn gebaseerd op twee opvattingen: 'struggle for life' (strijd om te leven), en 'survival of the fittest' (het overleven van de geschiktste) door natuurlijke selectie. Daarnaast wordt er gebruik gemaakt van een aantal uit de genetica bekende verschijnselen zoals crossing-over en mutatie.

Genetische algoritmen, oorspronkelijk ontwikkeld door John Holland [1975], blijken uitstekend te voldoen bij optimalisatieproblemen en kunnen eenvoudig geïmplementeerd worden. Bij genetische algoritmen worden de te maximaliseren parameters gecodeerd opgeslagen in een string. Deze string is analoog aan een chromosoom, de drager van het erfelijke materiaal in de genetica. Iedere parameter, opgeslagen in de string, kan gezien worden als een gen. Een aantal strings vormen een populatie en met behulp van genetische operatoren kunnen de strings zich voortplanten en worden nieuwe strings gevormd. Door middel van een evaluatiefunctie wordt een *fitnesswaarde* aan een string gegeven. Hiervoor geldt dat hoe hoger de fitness van een string, hoe beter de gerepresenteerde oplossing is. Het doel van de voortplanting is om via het creëren van individuen met een zo hoog mogelijke fitness de oplossing van het optimalisatie-probleem zo goed mogelijk te benaderen.

Bij de voortplanting van de strings spelen een aantal operatoren een belangrijke rol. Op grond van een selectiecriterium worden strings uit de populatie gekozen. Vervolgens worden door middel van een aantal recombinatie-operatoren nieuwe nakomelingen gegenereerd. Per stringpaar levert dit twee nakomelingen op die in de populatie kunnen worden teruggezet.

2.1.2. Genetische operatoren

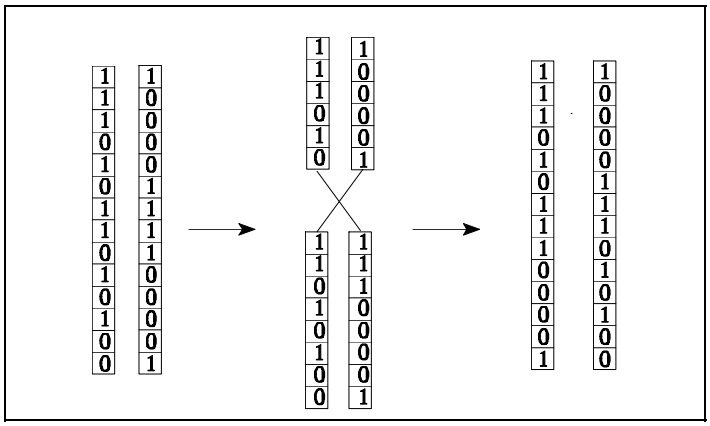
Selectie
Selectie wordt gebruikt om twee strings uit een populatie te halen, waarna deze strings zich kunnen voortplanten. Van belang is dat strings met een hoge fitness meer kans krijgen om zich voort te planten dan strings met een lage fitness, omdat de kans groot is dat strings met een hoge fitness weer strings met een hoge fitness opleveren. Het is niet zo dat alleen strings met hoge fitness geselecteerd mogen worden. In dat geval zou de populatie uiteindelijk alleen maar identieke strings bevatten, zonder dat een optimale oplossing gevonden is.

De methode die gebruikt is in dit onderzoek is gebaseerd op rank based selectie zoals beschreven door Whitley [1989, 1993]. Hierbij worden alle strings in de populatie gesorteerd op hun fitnesswaarde. De kans dat een string wordt geselecteerd, wordt bepaald door een factor, de *pressure waarde*, en de *positie* van deze string in de populatie. In principe wordt de selectiekans dus niet direct door de fitness bepaald.
De pressure waarde hoeft niet hoog te zijn: een pressure waarde van 2 zorgt er al voor dat het individu met de hoogste fitness minstens twee maal zo vaak wordt gekozen als een individu met een positie halverwege de populatie. Een individu met een positie op 75% niveau heeft ongeveer een anderhalf keer zo grote kans gekozen te worden dan één met een positie halverwege de populatie. Deze selectiemethode heeft als voordeel dat de fitnessfunctie niet geschaald hoeft te worden om te voorkomen dat strings met een relatief hoge fitnesswaarde, ervoor zorgen dat het algoritme vroegtijdig convergeert.

Voor het terugplaatsen van nieuwe individuen in de populatie wordt vaak "one-at-a-time selection and replacement" gebruikt. Eén paar strings wordt uit de populatie gehaald. Deze strings leveren twee nakomelingen, waarvan er maximaal één wordt teruggezet in de populatie. Deze nakomeling vervangt het individu met de laagste fitness in de populatie. In dit onderzoek is een variatie op deze methode gebruikt: er worden meerdere paren strings uit de populatie gehaald, die vervolgens per paar één nakomeling opleveren. Deze nakomelingen vervangen vervolgens de strings met de laagste fitnesswaarden in de populatie. Het aantal paren dat in één keer uit de populatie wordt gehaald is variabel. Het is één van de parameters die in dit onderzoek is onderzocht.

Crossing-over
Bij crossing-over worden twee strings uit de populatie op dezelfde plaats in tweeën gesplitst. De twee delen worden vervolgens uitgewisseld en er ontstaan twee nakomelingen die gedeeltelijk identiek zijn aan beide ouderstrings (zie figuur 1). Het punt waarop de strings gesplitst worden is het *cross-over punt*. Het aantal cross-over punten kan variëren. Bij meer dan één cross-over punt worden de delen van de string om en om uitgewisseld. De kans dat crossing-over (p_c) plaatsvindt kan variëren van 0 tot 1.

Crossing-over vergroot de genetische variatie. Dankzij dit mechanisme worden eigenschappen die zich op één enkel chromosoom bevinden niet altijd samen overgeërft, maar kan de ene eigenschap afkomstig zijn van de ene ouder en de andere eigenschap van de andere ouder.

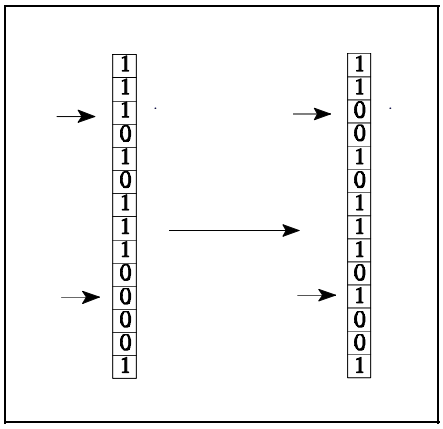


Figuur 2.1. *Crossing-over: twee ouders (links) die na crossing-over (midden) twee nakomelingen produceren.*

Crossing-over komt voor in de natuur, bij de aanmaak van geslachtscellen. In tegenstelling tot de hierboven beschreven methode komt crossing-over in de natuur niet tussen twee verschillende individuen voor.

Mutatie

Bij mutatie wordt een bit in een string met een mutatiekans p_m ($0 \leq p \leq 0.5$) veranderd van een 0 in een 1 of van een 1 in een 0 (zie figuur 2). De kans p_m geldt dus voor ieder bit in de string en moet in het algemeen erg klein zijn om te voorkomen dat de methode gaat lijken op een random zoekmethode.
Mutatie is essentieel om een populatie te kunnen voorzien van nieuwe informatie. Alleen crossing-over en selectie zorgen er niet voor dat een bit die in alle individuen op nul staat in de nakomelingen op één kan staan. Mutatie zorgt ervoor dat dit wel kan voorkomen.



Figuur 2.2. *String (links) muteert op twee plaatsen en vormt string (rechts)*

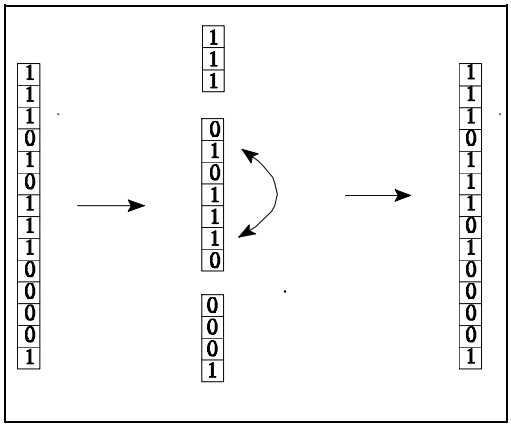
Een voorbeeld van een succesvolle mutatie in de natuur is de verandering van de kleur van een in Engeland voorkomende nachtvlinder. Oorspronkelijk was de kleur van deze vlinder wit, maar door een mutatie ontstonden er ook zwarte vlinders. Deze bleken een veel grotere overlevingskans in de geïndustrialiseerde gebieden te hebben, omdat zij onopvallend waren op de door roet zwart gekleurde boomstammen. Zij werden minder snel opgemerkt door vlindervangende vogels. In combinatie met een zeer sterke selectie heeft deze soort zich binnen enkele tientallen jaren kunnen ontwikkelen [Crow, 1979].

Inversie

Met een kans p_i wordt er bepaald of inversie plaatsvindt. Wanneer dit het geval is, worden er twee punten bepaald in de string waartussen alle bits omgedraaid worden (zie figuur 3). Wanneer inversie plaatsvindt tussen de punten a en b ($a<b$), dan wordt het bit op punt a omgewisseld met dat van punt b, het bit op punt a+1 met b-1 enz.. Inversie is een mechanisme om genen te verwisselen. Bij het voorgaande voorbeeld komt een gen overeen

met een enkel bit. Inversie werkt alleen als de bits op het chromosoom bij het uitlezen niet plaatsgebonden zijn, anders komt inversie overeen met een groot aantal mutaties.

Inversie in levende organismen komt bijvoorbeeld voor bij bepaalde salmonella's (bacteriën die diarree veroorzaken). Zij maken van dit mechanisme gebruik om van het ene eiwit over te schakelen op het andere. Als de gastheer immuun wordt voor het ene eiwit, geeft overschakeling op het andere eiwit een kans op overleving van het organisme. Inversie bepaalt in dit geval welk van twee verschillende genen aan of uit wordt geschakeld [Duve, 1987].



Figuur 2.3. *String (links) vormt na inversie (midden) nieuwe string (rechts).*

Voorbeeld:
Een simpel voorbeeld illustreert de werking van het genetisch algoritme: bepaal het maximum van de functie $f(x) = x^2$ voor waarden van x tussen 0 en 31.
Allereerst moet er een codering bepaald worden voor de parameter x. Met behulp van een stringlengte van 5 bits, kunnen de x-waarden als binair getal worden gecodeerd. (zie tabel 1). De evaluatiefunctie is in dit geval ook erg eenvoudig: kwadrateer de waarde die de string weergeeft.
Vervolgens wordt een initiële populatie aangemaakt. Deze bevat strings met een lengte van 5 bits die random met enen en nullen zijn gevuld.

string	fitness
01000	64
10010	324
00001	1

Tabel 2.1. *De fitness van een aantal strings bij het bepalen van het maximum van* $f(x) = x^2$

Met behulp van selectie worden een aantal individuen uit de populatie gehaald. Met crossing-over en mutatie kunnen daaruit nieuwe strings voortkomen, die in de populatie kunnen worden teruggezet.

Een klein experiment met een dobbelsteen leverde het resultaat dat te zien is in tabel 2. De initiële populatie links in de tabel, is willekeurig samengesteld. Daarna is met behulp van twee dobbelstenen selectie toegepast. De laagste waarde van beide dobbelstenen bepaalde welke string geselecteerd werd. Wanneer beide dobbelstenen dezelfde waarde gaven, werd er opnieuw gegooid. Vervolgens is met behulp van één dobbelsteen een cross-over punt bepaald en de crossing-over uitgevoerd. Met één dobbelsteen is vervolgens mutatie gesimuleerd. Per bit werd met de dobbelsteen gegooid. Bij een waarde 1 veranderde dat bit van een 0 in een 1 of van een 1 in een 0. Een waarde 2..6 gaf geen mutatie. De gebruikte mutatiefactor p_m is dus 1/6. Daarna is van alle nakomelingen de fitness bepaald. Deze is links in de tabel te zien.

string	fitness
11100	784
10101	441
10001	289
01101	169
00101	25
00001	1

selectie	cross	mutat
10001	1.0101	10101
10101	1.0001	100**1**1
11100	111.01	111**1**1
10101	101.00	1010**1**
10001	100.01	1000**0**
10101	101.01	1010**0**

string	**fitnes**
11111	961
10101	441
10101	441
10100	400
10011	361
10000	256

***Tabel 2.2**. De overgang van de huidige generatie in een nieuwe generatie, d.m.v. selectie, crossing-over en mutatie.*

In dit voorbeeld is de totale populatie vervangen door de nieuwe populatie. Dit is in tegenstelling tot de eerder beschreven methode waarbij één individu werd teruggezet in de populatie.

De kracht van de verschillende operatoren komt in dit experimentje duidelijk naar voren. Door selectie worden voornamelijk die strings uitgekozen die al een hoge fitness bezitten.
Crossing-over combineert de beide strings tot twee nieuwe, waardoor de fitness van strings ook hoger kan worden.
Mutatie zorgt voor nieuwe informatie. In de oorspronkelijke populatie was het vierde bit altijd nul, in de nieuwe populatie is er een chromosoom met dit bit gelijk aan 1 door mutatie in de populatie gekomen. Zonder mutatie was er nooit een optimale oplossing gevonden.

2.1.3. Werking van het genetisch algoritme

De vraag blijft, hoe het komt dat een genetisch algoritme de optimale waarde kan vinden, uit een aantal willekeurige strings.
Om strings met elkaar te kunnen vergelijken is door John Holland [1975] het begrip schema ingevoerd. Een schema is een string over het zelfde alfabet als de oorspronkelijke string, met daarnaast een uitbreiding met een speciaal "don't care" karakter, de *. Een schema correspondeert met een string als op iedere plaats waar in het schema een nul of een 1 staat, deze ook in de string staat. Op de plaats waar in het schema een * staat, maakt het niet uit wat er in de string staat. Het schema **111 bijvoorbeeld correspondeert met vier verschillende strings: {00111, 01111, 10111, 11111}.
Van de strings die corresponderen met een bepaald schema, is in dit voorbeeld aan te geven wat de maximale en minimale fitness zal zijn. Het schema **111 zal bij een fitnessfunctie $f(x)=x^2$ corresponderen met een minimale fitness van (binair) $(00111)^2$ of $7^2= 49$ en een maximale fitness van $(11111)^2$ of $31^2= 961$.

De gemiddelde fitness van de strings in een populatie die corresponderen met een schema geven de fitness van dit schema, deze is dus afhankelijk van de populatie. Het blijkt dat het zoeken naar schema's met een hoge fitness door het genetisch algoritme bevorderd wordt. In het probleem dat hierboven als voorbeeld is gegeven zal het schema 1**** uiteindelijk de overhand krijgen omdat deze een fitness van minimaal 256 heeft. Een schema als ****1 zal veel minder invloed hebben. Dit is ook te zien in tabel 2. In de originele populatie correspondeert 50% van de strings met het schema 1****, in de populatie die daaruit ontstaat is dat 100% geworden. Het schema ****1 correspondeert in de originele populatie met 5 van de 6 strings, in de nieuwe populatie is dat 4 van de 6 strings geworden, dit schema is minder van invloed. Een schema met een nul erin zal in dit geval ook weinig invloed uitoefenen.

Om het effect van de verschillende genetische operatoren op de fitness van schema's en dus de hele populatie te bekijken moeten eerst wat definities gegeven worden.
Bekijk een populatie met n binaire strings van lengte l. Omdat ieder van de l string posities kan bestaan uit een 0, 1 of * is het totaal aantal verschillende schema's 3^l.
Eén string correspondeert met 2^l schema's omdat iedere positie in het schema de waarde van de string kan hebben of een *. Het totaal aantal schema's in een populatie ligt dus tussen de 2^l en $n*2^l$ afhankelijk van de verschillende strings in de populatie. Dit geeft een enorme hoeveelheid aan informatie over de overeenkomsten en verschillen in de verschillende strings.

Het effect van selectie op het aantal schema's is duidelijk, omdat strings met een hogere fitness een hogere kans hebben om geselecteerd te worden, worden de schema's met de hoogste fitness gemiddeld beter vertegenwoordigd.
Laat f_s de fitness van schema A zijn, en $m(t)$ het aantal keren dat schema A in de populatie vertegenwoordigd wordt op een tijdstip t. Laat f_{avg} de gemiddelde (Eng. average) fitness van de hele populatie zijn.
Het verwachte aantal strings op tijdstip t+1 dat met schema A correspondeert zal

$$m(t+1) = m(t)\,\frac{f_s}{f_{avg}} \qquad (1)$$

zijn. Het aantal schema's met een hogere fitness dan de gemiddelde fitness van de hele populatie zal toenemen, terwijl schema's waarbij dit niet geldt in aantal zullen afnemen en uiteindelijk zullen verdwijnen.

Het effect van crossing-over hangt af van de definiërende lengte δ van een schema. Dit is de afstand tussen de eerste 1 of 0 en de laatste 1 of 0 in het schema. Het schema 1***0 met een definiërende lengte van 4 heeft een gotere kans om door crossing-over verbroken te worden dan het schema 11*** met een definiërende lengte van 1 (respectievelijk 1 en 1/4). Een reeds aanwezig schema blijft bestaan als de crossing-over links van de eerste 0 of 1 of rechts van de laatste 0 of 1 plaatsvindt. Zodoende is er een ondergrens te geven in de overlevingskans p_s van een specifiek schema:

$$p_s \geq 1 - p_c \cdot \frac{\delta}{l - 1} \qquad (2)$$

Hierbij geldt dat p_c de kans is dat crossing-over plaatsvindt en l de lengte is van een string.

Het effect van mutatie op een schema is over het algemeen erg klein, en hangt af van de mutatiekans p_m . Er geldt dat iedere positie in een schema waarin een nul of een 1 staat met een kans van p_m, veranderd kan worden. Mutatie op een positie waar een * staat verandert het schema niet. Noem het aantal enen en nullen in een schema de orde o van het schema. De kans dat een schema de mutatie overleeft is dan $(1-p_m)^o$. Voor hele kleine waarden van p_m wordt de overlevingskans benaderd door:

$$m_s \approx 1 - o \cdot p_m \qquad (3)$$

Een combinatie van 1, 2 en 3 levert het totale aantal strings in de populatie dat correspondeert met een specifiek schema.

$$m(t+1) \geq m(t) \cdot \frac{f_s}{f_{avg}} \left[1 - p_c \cdot \frac{\delta}{t - 1} - o \cdot p_m \right] \qquad (4)$$

Schema's met een korte definierende lengte, een lage orde en een fitness die boven het gemiddelde zit hebben in volgende generaties exponentieel meer invloed, deze schema's worden *building blocks* genoemd. Dit wordt de "Schema Theorem" of schema stelling genoemd [Goldberg, 1989].

Het aantal verschillende toepassingen dat gebruik maakt van genetische algoritmen neemt dankzij het krachtige zoek- en optimalisatiemechanisme enorm toe. Het bepalen van de codering van een probleem in een string bestaande uit enen en nullen en het koppelen van een waarderingsfunctie aan een string zijn de enige delen van het algoritme die voor problemen kunnen zorgen. Het optimaliseren kan vervolgens voor ieder willekeurig probleem op dezelfde wijze gebeuren.
Het effect van mutatie op een schema is over het algemeen erg klein en hangt af van de mutatiekans p_m. Er geldt dat iedere positie in een schema waarin een nul of een 1 staat met een kans van p_m, veranderd kan worden. Mutatie op een positie waar een * staat verandert het schema niet. Noem het aantal enen en nullen in een schema de orde o van het schema. De kans dat een schema de mutatie overleeft is dan $(1-p_m)^o$. Voor hele kleine waarden van p_m wordt de overlevingskans benaderd door:

$$m_s \approx 1 - o \cdot p_m \qquad (3)$$

Een combinatie van 1, 2 en 3 levert het totale aantal strings in de populatie dat correspondeert met een specifiek schema.
Schema's met een korte definiërende lengte, een lage orde en een fitness die boven het gemiddelde zit hebben in volgende generaties exponentieel meer invloed, deze schema's worden *building blocks* genoemd. Dit wordt de "Schema Theorem" of schema stelling

$$m(t+1) \geq m(t) \cdot \frac{f_s}{f_{avg}} \left[1 - p_c \cdot \frac{\delta}{t-1} - o \cdot p_m \right] \quad (4)$$

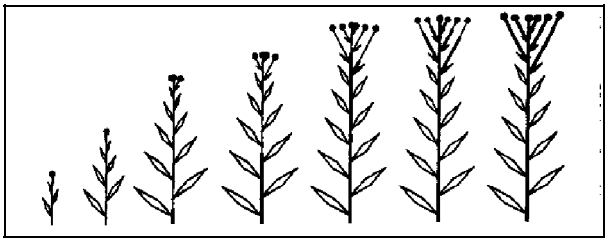
genoemd [Goldberg, 1989].

Het aantal verschillende toepassingen dat gebruik maakt van genetische algoritmen neemt dankzij het krachtige zoek- en optimalisatiemechanisme enorm toe. Het bepalen van de codering van een probleem in een string bestaande uit enen en nullen en het koppelen van een waarderingsfunctie aan een string zijn de enige delen van het algoritme die voor problemen kunnen zorgen. Het optimaliseren kan vervolgens voor ieder willekeurig probleem op dezelfde wijze gebeuren.

2.2. L-systemen

2.2.1. Inleiding

Al jaren proberen onderzoekers met behulp van computers de groei en ontwikkeling van levende wezens te simuleren. Vooral met fractals [Mandelbrot, 1982], gecompliceerde meetkundige figuren, worden goede resultaten behaald. Met L-systemen, die gezien kunnen worden als een speciale klasse van fractals, wordt de ontwikkeling van lineaire en vertakte structuren gesimuleerd. Ze zijn geïntroduceerd in 1968 door Aristid Lindenmayer [1968], die er de groei van planten mee modelleerde. Sindsdien zijn er talloze variaties op het oorspronkelijke systeem ontwikkeld om zo goed mogelijk de groei van planten en bomen in de natuur te simuleren. De oorspronkelijke L-systemen geven de groei van een plant alleen weer op vaste tijdsintervallen. De stadia tussen deze intervallen zijn niet gedefinieerd. Prusinkiewicz [1993] beschrijft echter een variant van deze L-systemen die het groeiproces continu weergeeft (zie figuur 2.4).



Figuur 2.4. *Modellering van een continu groeiproces m.b.v. L-systemen volgens Prusinkiewicz [1993]*

Daarnaast worden variaties van L-systemen toegepast bij het opbouwen van grafen. Map L-systemen bijvoorbeeld [Prusinkiewicz 1994] zijn een uitbreiding op de vertakte structuur van de traditionele L-systemen, naar grafen met cykels, de zogenaamde *maps*. Met deze Map L-systemen kunnen cellulaire lagen worden gemodelleerd.
Een nieuwe variant van L-systemen is in dit onderzoek toegepast voor het opbouwen van modulaire neurale netwerken.

2.2.2. Werking van L-systemen

Het L-systeem formalisme is een mechanisme om parallel strings te herschrijven met behulp van een grammatica. Het proces begint met een initiële string of *axioma* bestaande uit karakters over een *alfabet*. Door het toepassen van *produktieregels* of 'herschrijf-strings' worden bij iedere stap alle karakters van de string vervangen door de karakters die een produktieregel voorschrijft. Zo wordt een nieuwe string gevormd. Produktieregels bestaan uit een linkerdeel, de '*voorganger*' en een rechterdeel, de '*opvolger'*. Het toepassen van een produktieregel op een string kan alleen als de voorganger van deze regel voorkomt in de string. Dat deel van de string wordt dan herschreven in de opvolger van deze produktieregel. Door aan ieder karakter in de string een interpretatie toe te kennen is het mogelijk allerlei verschillende modellen te creëren.

Om produktieregels aan bepaalde condities te laten voldoen, worden ze context afhankelijk gemaakt. Een herschrijfregel heeft dan de volgende vorm:

$$L < P > R \rightarrow S$$

P en S zijn de voorganger (Eng. "predecessor") en de opvolger (Eng. "successor"), L en R zijn respectievelijk de linker (Eng. "left") en rechter (Eng. "right") context. Een produktieregel kan nu alleen maar toegepast worden op een string als deze de karakters P bevat die voorafgegaan worden door L en gevolgd worden door R.
L-systemen zonder context zijn 0L-systemen. L-systemen met één context zijn 1L systemen en 2L-systemen hebben aan beide zijden een context. Als er twee of meer produktieregels toegepast kunnen worden op één karakter, dan wordt de regel met de langste context gebruikt. Wanneer beide contexten even lang zijn, wordt de eerst voorkomende regel toegepast.

Voorbeeld:
Gegeven is het axioma A en produktieregels

$C < C > A$	$\rightarrow BC$	(1)	
A	$\rightarrow CAB$	(2)	
$A < B$	$\rightarrow AC$		(3)

stap 1: toepassen van regel 2 op het axioma levert de string	CAB
stap 2: Alleen regel 2 en 3 kunnen toegepast worden:	CCABAC
stap 3: Alle drie de regels kunnen toegepast worden:	CBCCABACCABC

enz.

Dit proces kan eindigen doordat er geen produktieregels meer toegepast kunnen worden, of door een eindig aantal herschrijfstappen toe te laten.

Grafische interpretatie van een L-systeem

Met behulp van een op LOGO gelijkende interpretatie [Szilard en Quinton, 1979] van de verschillende karakters, kunnen strings zichtbaar worden gemaakt. Het uitlezen van een string gebeurt karakter voor karakter. Elk karakter zorgt ervoor dat de LOGO-*turtle* een actie

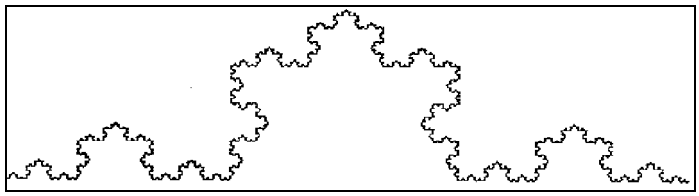
onderneemt en een figuur gaat tekenen. De functie van de LOGO-turtle is het aangeven van de huidige positie en richting.

Een veel gebruikte interpretatie wordt bijvoorbeeld gegeven met de volgende regels:

F: Teken een lijn in de richting die de turtle aangeeft.
+: Roteer α^o naar links
-: Roteer α^o naar rechts

De hoek α en de lengte van de te tekenen lijn kunnen als parameters worden meegegeven, of globaal worden gedefinieerd.

Figuur 2.5 is verkregen in 5 herschrijfstappen uit axioma F en produktieregel F $\rightarrow$ F - F + + F - F. De hoek α is 60^o [Boers en Kuiper, 1992].



Figuur 2.5. *Koch-graph gemaakt met axioma F en produktieregels F $\rightarrow$ F - F + + F - F, $\alpha = 60^o$*

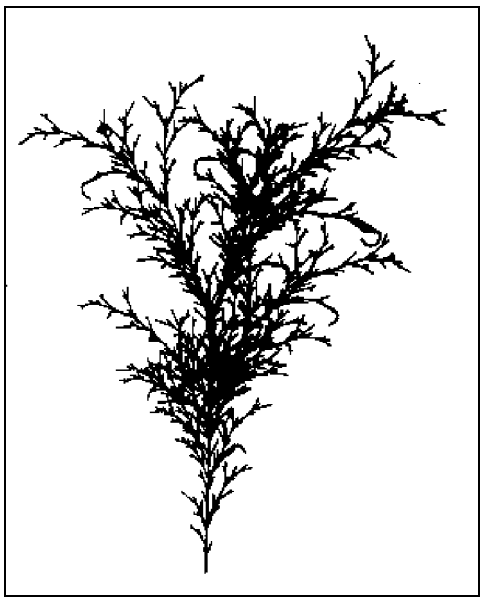
Om te zorgen dat er vertakkingen in de figuur kunnen ontstaan, kunnen er twee karakters worden toegevoegd met de volgende interpretatie:

[: Onthoud laatste turtle positie en richting
]: Zet turtle op laatst onthouden positie en richting.

De context gevoelige 1L en 2L-systemen zorgen ervoor dat iedere ontwikkeling in een figuur gebaseerd is op de voorgaande ontwikkeling. Dit levert meer natuurgetrouwe planten op.

Figuur 2.6 [Prusinkiewicz and Hanan, 1989] is gemaakt met een 2L systeem met axioma F1F1F1 en de produktieregels:

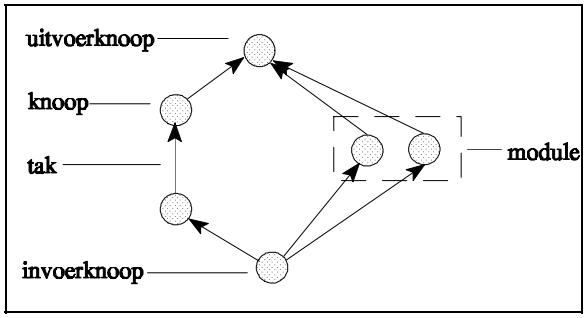
0 < 0 > 0 $\rightarrow$ 0
0 < 0 > 1 $\rightarrow$ 1[-F1F1]
0 < 1 > 0 $\rightarrow$ 1
0 < 1 > 1 $\rightarrow$ 1
1 < 0 > 0 $\rightarrow$ 0
1 < 0 > 1 $\rightarrow$ 1F1F
1 < 1 > 0 $\rightarrow$ 1
1 < 1 > 1 $\rightarrow$ 0
\+ $\rightarrow$ -
\- $\rightarrow$ +



Figuur 2.6. *Gebruik van een 2-L-systeem levert een natuurgetrouwe plant*

2.2.3. Grafen construeren met behulp van L-systemen, het G2L-systeem

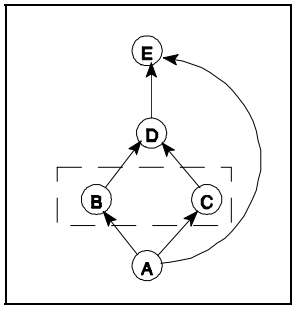
In dit onderzoek zijn L-systemen gebruikt om modulaire neurale netwerken op te bouwen. Een modulair neuraal netwerk wordt gerepresenteerd door een gerichte graaf (zie figuur 2.7). Hiervoor is een speciale interpretatie ontwikkeld waardoor uit produktieregels een graaf wordt opgebouwd, het G2L-systeem [Boers et al., 1993]. Het alfabet waaruit de strings opgebouwd worden, bevat de karakters { A-Z, 1-9, [,] }. De interpretatie van een string gaat door het van links naar rechts aflopen van de karakters van de string.



Figuur 2.7. *De verschillende onderdelen van een graaf*

Een knoop in de graaf wordt weergegeven met een letter. De knopen tussen vierkante haken vormen een *subgraaf*. De verbindingen tussen de verschillende *eenheden* worden aangegeven door cijfers. Een eenheid is een knoop of een subgraaf. Een cijfer *j* volgend op een eenheid geeft aan dat er een verbinding bestaat **van** deze eenheid **naar** de eenheid die *j* eenheden verder in de string zit. Een module is een deel van de graaf waarvoor geldt dat alle knopen binnen die module verbonden zijn met dezelfde knopen buiten de module. De knopen binnen een module zijn onderling niet verbonden. De verbinding tussen twee modules is niet volledig. Alle *uitvoerknopen* van de eerste module zijn verbonden met de *invoerknopen* van de tweede module. De invoerknopen van een module ontvangen geen invoer van knopen van dezelfde module. De uitvoerknopen van een module zijn die knopen die geen uitvoer geven naar andere knopen binnen de module (zie figuur 2.7).

Figuur 2.8 geeft de graaf weer die verkregen is uit de string A13[BC]1D1E. De string is opgebouwd uit vier eenheden, de knopen A, D en E en subgraaf BC. Knoop A is verbonden met subgraaf BC en met E, wat de derde eenheid is vanaf knoop A. De subgraaf BC is vervolgens verbonden met knoop D. Knoop D is verbonden met knoop E. (voorbeeld volgens Boers en Kuiper [1992]). In deze graaf vormen de knopen B en C een complete module: ze zijn met dezelfde knopen verbonden, en hebben onderling geen verbinding.



Figuur 2.8. *Graaf verkregen uit de string A13[BC]1D1E*

Voor de vier delen waaruit een produktieregel bestaat gelden een aantal beperkingen. Niet alle strings over het alfabet {A-Z, 1-9, [,] } zijn dus mogelijk. De voorganger en de opvolger mogen alleen maar complete subgrafen en knopen bevatten. Het aantal linker en rechter haken moet daarom aan elkaar gelijk zijn en in de goede volgorde staan. Zo is de string A]BC[D niet correct omdat de haken niet in de goede volgorde staan, en dus geen complete subgraaf vormen. Daarnaast

mogen er geen lege subgrafen ([ ]) voorkomen. In tegenstelling tot de opvolger moet de voorganger minstens één knoop bevatten. Een lege opvolger zorgt ervoor dat de voorganger verwijderd wordt uit de string als de produktieregel wordt gebruikt.

De rechter- en linkercontext van een produktieregel mogen leeg zijn. Indien er een context aanwezig is, dient deze aan dezelfde eisen te voldoen als de voorganger en de opvolger. Bovendien geldt een restrictie: elk cijfer moet volgen op een knoop of subgraaf. De string 1A[B] is niet toegestaan vanwege de eerste 1.

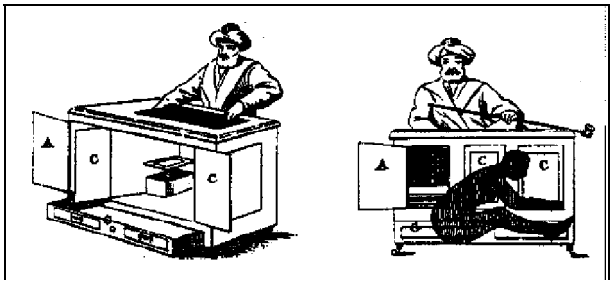
De definitie van context is bij dit L-systeem anders dan bij traditionele L-systemen. Bij normale L-systemen wordt er naar het voorgaande of volgende karakter gekeken in de string. Hier wordt de context pas bekeken nadat de string geïnterpreteerd is.
Een produktieregel met een rechter- en/of linker context kan toegepast worden op een string als:
1. De *linkercontext* uit een set knopen of modules bestaat die een verbinding hebben *naar* de knopen en modules van de voorganger.
2. De *rechtercontext* uit een set knopen of modules bestaat die een verbinding hebben *van* de knopen en modules van de voorganger.
Een tweede verschil met normale L-systemen is dat ook meerdere karakters tegelijkertijd vervangen kunnen worden. Dit geeft de mogelijkheid om complete modules te vervangen door andere knopen of modules.

In figuur 2.9 worden de verschillende stadia van de ontwikkeling van een netwerk afgebeeld. Hierbij is gebruik gemaakt van axioma A en de volgende produktieregels:

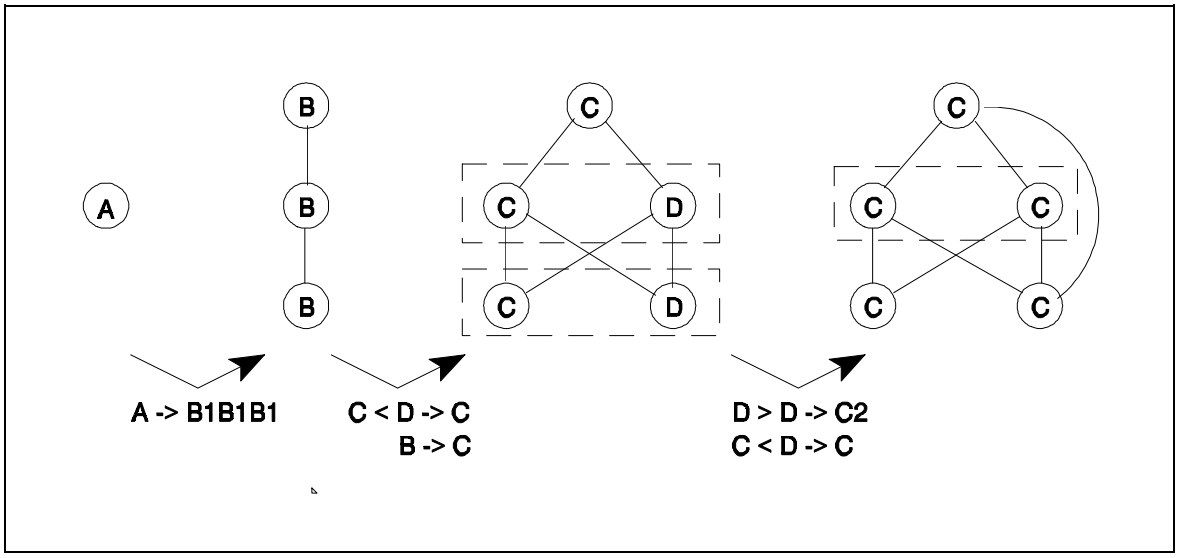
1 A → B1B1B
2. B > B → [CD]
3. B → C
4. C < D → C
5. D > D → C2

2.3. Neurale netwerken

2.3.1. Inleiding

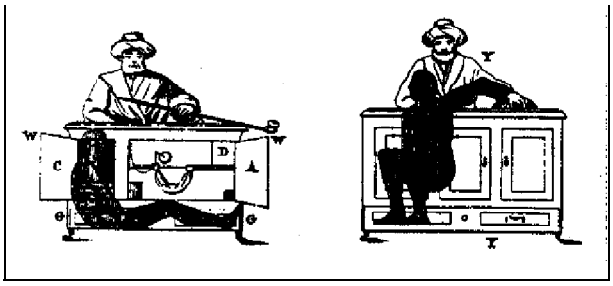


Al vele eeuwen lang zijn mensen gefascineerd door intelligente machines. In 1796 werd bijvoorbeeld de automatische schaakmachine uitgevonden [Van den Herik, 1983]. Pas tientallen jaren later kwam men erachter dat hier geen sprake was van een machine met intelligentie, maar van een kleine menselijke schaker die in de machine verborgen zat. Deze deed een zet door z'n arm in de arm van de turk te stoppen en hij verplaatste zich snel van het ene compartiment naar het andere zodra de machinerie werd getoond (zie figuur 2.10).



Figuur 2.9. *De verschillende stadia van de ontwikkeling van een netwerk.*

Op vele verschillende manieren is gezocht naar een wijze om machines intelligent te maken. Een centrale vraag op dit gebied is: wanneer kan een machine intelligent worden genoemd?
In 1950 stelde de Britse wiskundige Alan Turing [1963] de volgende methode voor om uit te zoeken of een machine intelligent is. Zet een persoon en de machine die onderzocht moeten worden in twee aparte ruimtes. Een tweede persoon, de ondervrager, moet vervolgens uitzoeken in welke van de twee ruimtes de machine zich bevindt. De communicatie tussen de ondervrager en de machine of de andere persoon verloopt uitsluitend via getypte vragen en antwoorden. Het doel van de machine is om de ondervrager om de tuin te leiden en te doen alsof hij de mens is. Als bijvoorbeeld de vraag gesteld wordt "hoeveel is 12,324 maal 2435" dan kan de machine even stil blijven en vervolgens het foute antwoord geven. Tot nu toe is er nog geen enkele computer geslaagd voor deze Turing test. Wel zijn er computers (programma's) die op een heel specialistisch gebied experts kunnen verslaan. Zij doorstaan met succes de Turing test op een begrensd domein.

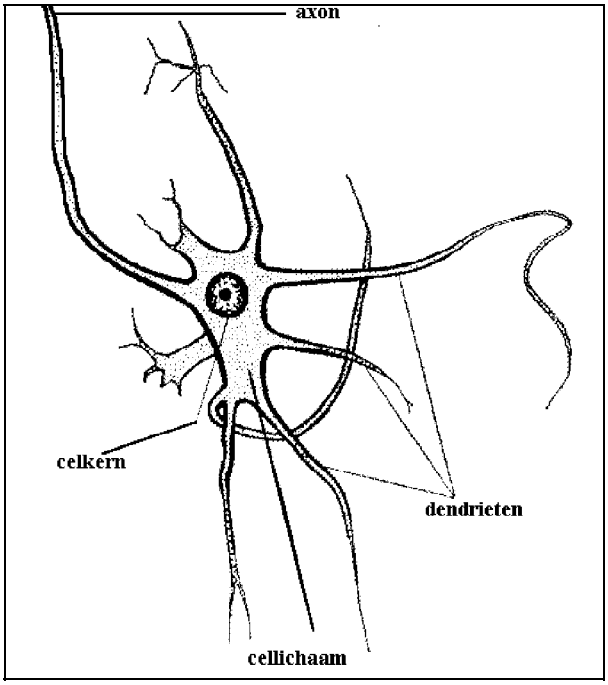


Figuur 2.10. *"De Turk", de eerste "schaakmachine"*

Een logische ontwikkeling in de zoektocht naar intelligente machines is het nabootsen van de werking van de hersenen met een computer. Artificiële neuronen (neuronen zijn cellen waaruit het zenuwstelsel is opgebouwd) worden dan ook al zeer lang onderzocht. De eerste resultaten op dit gebied werden al in 1943 bereikt door McCulloch en Pitts [1943]. Daarna is het onderzoek met wisselende belangstelling voortgezet.

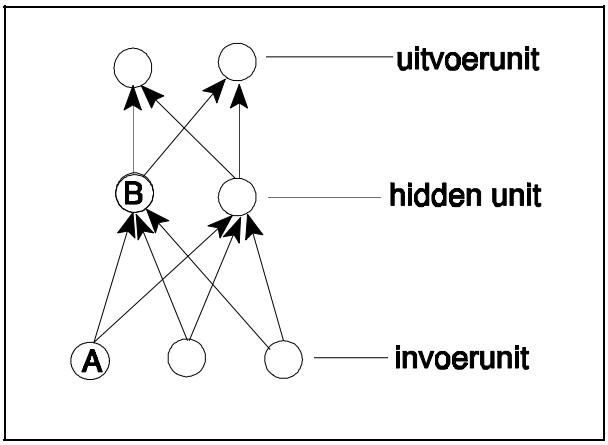
2.3.2. Artificiële neurale netwerken en de hersenen

Artificiële neurale netwerken bestaan uit onderling verbonden eenheden, de units. Hun werking is gebaseerd op de werking van de zenuwcellen (neuronen) in het centrale zenuwstelsel van de mens. Een neuron bestaat uit een cellichaam met daarin de kern (figuur 2.11). Vanuit dit cellichaam ontspringen een aantal uitlopers. Eén van deze uitlopers is het axon. De anderen zijn dendrieten. De functie van een zenuwcel is het ontvangen en het eventueel doorgeven van prikkels van en naar omliggende zenuwcellen. Dit doorgeven van prikkels is een chemisch proces en gebeurt altijd in één richting. Als de dendrieten geprikkeld worden, wordt een elektrisch potentiaal opgebouwd in de cellen. Als deze boven een bepaalde drempelwaarde (actie-potentiaal) komt gaat het axon vuren, ofwel de naburige zenuwcellen worden op hun beurt geprikkeld. Naast deze exciterende processen kunnen ook inhiberende processen optreden. In dit laatste geval worden de prikkels naar de naburige zenuwcellen verzwakt. Dit is bijvoorbeeld duidelijk te constateren wanneer we ons concentreren op het lezen van een boek. Geluiden uit de omgeving worden op dat moment niet bewust meer waargenomen.



Figuur 2.11. *Neuron [Freeman and Bracegirdle, 1966]*

De werking van een artificieel neuraal netwerk is gebaseerd op de werking van het zenuwstelsel. Een eenheid (unit) in een neuraal netwerk is verbonden met verschillende andere units. In een feedforward netwerk lopen de verbindingen tussen de verschillende units in één richting (zie figuur 2.12). Bij een verbinding van A naar B, dient de verbinding als invoer voor unit B, terwijl deze de uitvoerverbinding voor unit A is. Een unit kan meerdere invoer- of uitvoerverbindingen hebben. Een unit zonder invoerverbindingen is een invoerunit, zo'n unit krijgt z'n invoer van de "buitenwereld". Een unit zonder uitvoerverbindingen is een uitvoerunit. Units met zowel in- als uitvoerverbindingen zijn de verborgen units.



Figuur 2.12. *Artificieel neuraal netwerk*

Analoog aan de zenuwcellen, worden signalen over de verbindingen gestuurd. Een signaal bestaat uit een getal dat zowel positief als negatief kan zijn. Aan de hand van het gewicht van de verbinding en de waarde die binnen komt over deze verbinding, wordt een nieuwe

waarde berekend en in de unit opgeslagen. Als de waarde in een unit boven een bepaalde drempelwaarde komt, dan geeft deze een waarde door aan zijn aangrenzende units. Een negatief getal werkt inhiberend, een positief getal werkt exciterend. Door aan het netwerk een signaal aan te bieden, worden de invoerunits van dat netwerk geactiveerd. Het signaal plant zich verder voort over de verborgen units naar de uitvoerunits. De respons van het netwerk op het signaal is af te lezen uit de waarden van de uitvoerunits.

Er zijn vele manieren waarop de units en de verbindingen daartussen georganiseerd kunnen worden. Vaak is een netwerk opgebouwd uit meerdere lagen units, waarbij alle units van een laag volledig verbonden zijn met de units van een aangrenzende laag. Van netwerken bestaande uit drie lagen, een invoer-, uitvoer- en verborgen laag, is bewezen dat ze alle wiskundige bewerkingen kunnen uitvoeren.

De grote kracht van neurale netwerken is dat ze bepaalde functies kunnen *leren*. Door het herhaaldelijk aanbieden van een invoerpatroon en het bijbehorende uitvoerpatroon, kunnen feedforward netwerken automatisch de gewichten van de verbindingen en de drempelwaarden bijstellen. Dit bijstellen van deze waarden hangt af van de fout die gemaakt wordt in het uitvoerpatroon. Het netwerk wordt als het ware *getraind* om een bepaalde functie te leren. Een algoritme dat op deze manier werkt is back-propagation. Een uitgebreide beschrijving van dit algoritme kan gevonden worden in Rumelhart et al. [1986].

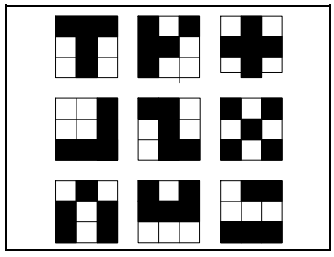
2.3.3. Modulaire artificiële neurale netwerken

De hersenen van een volwassen mens bestaan naar schatting uit 10^{11} zenuwcellen. Deze neuronen zijn onderling niet allemaal met elkaar verbonden. Zo bestaat er tussen de rechter hersenhelft en de linker hersenhelft maar één verbinding: het corpus callosum (weliswaar bestaande uit vele axonen). Daarnaast blijkt dat verschillende delen van de hersenen verschillende taken hebben. Het verschil tussen de linker en de rechter hersenhelft is bijvoorbeeld dat de linkerhelft vooral in staat blijkt te zijn tot verbale en analytische functies, terwijl de rechterhelft ruimtelijke en synthetische vermogens heeft. Ook op kleinere schaal vindt deze specialisatie plaats. Voor de drie belangrijke zintuigsystemen bijvoorbeeld, de tastzin, het gezicht en het gehoor, zijn (meerdere) centra aan te wijzen die deze specifieke signalen verwerken. Voor het gezichtsvermogen blijken er zelfs meer dan twintig verschillende gebieden aan te wijzen, elk met z'n eigen taak [Kaas 1990].

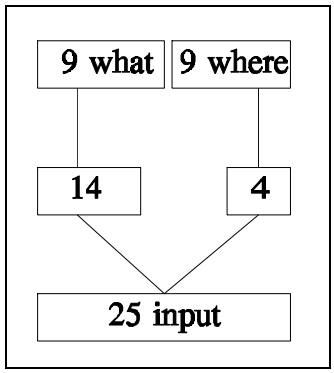
De opbouw van de hersenen is dus modulair. Ieder gebied heeft zo een eigen specifieke taak en door onderlinge samenwerking wordt alles wat wij ervaren tot één geheel versmolten. De vraag is of modulariteit ook in artificiële neurale netwerken tot een beter resultaat leidt.

Het blijkt dat de traditionele netwerken erg veel moeite hebben met het leren van twee verschillende taken. Rueckl, Cave en Kosslyn [1989] onderzochten het "what, where"-probleem. Hierbij moeten negen patronen herkend worden die op een groter raster zijn geplaatst. Ieder patroon bestaat uit 3x3 pixels (figuur 2.13), het raster is 5x5. Behalve de vorm van het patroon (what) moet het netwerk eveneens de positie (where) in het raster

bepalen. Ieder patroon kan op negen verschillende posities geplaatst zijn. Ze trainden een aantal verschillende netwerken met 25 invoer- en 18 uitvoerunits en een hidden laag met 18 units. Het bleek dat wanneer de hidden laag in twee delen gesplitst was, die ieder met de uitvoerknopen voor één van de taken verbonden werden, het netwerk sneller leerde en minder fouten maakte dan de niet gesplitste versie. In figuur 2.14 is het door Rueckl et al. [1989] gevonden netwerk weergegeven. Het bleek dat de hidden laag in het ongesplitste model inconsistente trainingsinformatie kreeg aangeboden omdat deze units zowel verbonden waren met de uitvoerunits voor de "what-" en de "where-" taak. Er trad interferentie op tussen de verschillende taken.



***Figuur 2.13**. De 9 patronen*



***Figuur 2.14**. Het netwerk [Rueckl et al., 1989]*

Jacobs, Jordan en Barto [1991] stellen dat modulaire netwerken sneller leren, beter generaliseren, een begrijpelijker architectuur hebben en beter kunnen voldoen als er beperkingen van de hardware zijn. Om een modulair netwerk 2 verschillende taken te laten leren gebruiken zij feitelijk drie complete niet modulaire netwerken. Twee netwerken leren ieder een taak, het derde netwerk coördineert de samenwerking tussen beide netwerken. Alle drie de netwerken worden tegelijk getraind. In feite leert het modulaire netwerk het probleem op te delen in twee afzonderlijke taken. Ook zij hebben met het "what, where"- probleem goede resultaten bereikt.

Ook studies van anderen [Murre 1992, Happel & Murre 1994] wijzen uit dat modulaire netwerken even goed of beter in staat zijn om taken te leren dan de traditionele artificiële netwerken.

2.4. Combinatie van genetische algoritmen, L-systemen en neurale netwerken

2.4.1. Inleiding

Het zoeken naar een geschikte architectuur voor een neuraal netwerk dat een bepaald probleem kan leren, is zoeken naar een speld in een hooiberg. Het aantal units ligt niet vast en het aantal verbindingen evenmin. Daarnaast kan een netwerk, afhankelijk van de initiële waarden, een taak de ene keer wel en de andere keer niet leren. Tot nu toe zijn er geen regels gevonden waaraan de architectuur van een netwerk moet voldoen om een taak te kunnen leren. Het is een logische stap om het ontwerpen van een netwerk te automatiseren. De genetische algoritmen zijn bij uitstek geschikt om deze taak aan te kunnen, gezien de voordelen in dit soort situaties boven de traditionele zoekmethodes.

Al enige jaren worden genetische algoritmen ingezet bij de speurtocht naar netwerken die een bepaalde taak (of taken) kunnen leren. De toepasbaarheid van het genetisch algoritme kan op twee verschillende gebieden plaats vinden:

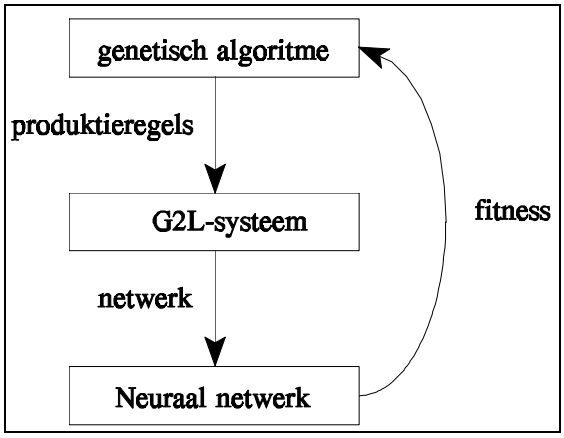
•. Bij het ontwerpen van een netwerk, [Harp et al., 1989, Kitano, 1990] waarbij het gevonden netwerk met back propagation getraind wordt.
• Bij het bepalen van een optimale set gewichten van een bestaand netwerk [Whitley en Hanson, 1989].

Combinaties van beide gebieden worden eveneens onderzocht. Dasgupta en McGregor [1992] definiëren zowel het netwerk als de gewichten in één chromosoom en optimaliseren deze gelijktijdig.

In dit onderzoek is van genetische algoritmen gebruik gemaakt om geschikte netwerken te ontwerpen. In tegenstelling tot de meeste onderzoeken, waarbij netwerken direct uit de chromosomen worden opgebouwd, wordt er een aparte techniek, de L-systemen, gebruikt om de netwerken te ontwerpen. Het trainen van een netwerk gebeurde in eerste instantie met backpropagation, nu worden ook andere methoden onderzocht.

2.4.2. Werking

Een combinatie van genetische algoritmen, een variant van L-systemen, de G2L-systemen en neurale netwerken is gebruikt om netwerken te ontwerpen. In figuur 2.15 is de samenhang weergegeven tussen de verschillende technieken.



***Figuur 2.15**. Samenwerking van de drie systemen*

Het genetisch algoritme gebruikt een populatie bestaande uit strings. Deze strings bevatten gecodeerd de produktieregels die nodig zijn voor het G2L-systeem. Het G2L-systeem zet de produktieregels afkomstig uit één string om in een netwerk. Dit netwerk wordt vervolgens getraind met bijvoorbeeld backpropagation. Afhankelijk van de mate waarin het netwerk in staat is het opgegeven probleem te leren, wordt er een fitnesswaarde bepaald. Deze fitnesswaarde wordt teruggekoppeld naar de string (uit het genetisch algoritme) die de produktieregels leverde voor het netwerk. Een netwerk dat z'n taak goed kan leren levert een hogere fitness dan een netwerk dat daartoe niet in staat blijkt te zijn.

Het genetisch algoritme levert een chromosoom aan het G2L-systeem. Hoe de codering van de produktieregels in het chromosoom is opgeslagen wordt in de volgende paragraaf beschreven.

2.4.3. Codering produktieregels

Een produktieregel, L < P > R $\rightarrow$ S, bestaat uit vier delen, waarvan er drie leeg kunnen zijn namelijk L, R en S. Ieder deel van deze produktieregel is een string over het alfabet { A-H, 1-6, [,]}. Het totaal aantal karakters in het alfabet is 16. Om een scheiding aan te

kunnen geven tussen de verschillende delen van de produktieregel is een asterisk ingevoerd. Het aantal karakters dat gecodeerd moet worden, komt hiermee op 17.

Alhoewel een binaire string met lengte 5 in principe voldoende is om 17 verschillende karakters te coderen, werd ervoor gekozen om een karakter te coderen door een binaire string van lengte 6. Op deze manier wordt er een grotere overeenkomst bereikt met de biologisch genetische code (zie bijlage A). Een binaire string van lengte 6 levert een totaal van $2^6 = 64$ mogelijke coderingen. Voor de conversie van deze codering naar één enkel karakter uit het alfabet wordt een conversietabel gebruikt met 64 ingangen (tabel 2.3). De 17 karakters komen in verschillende aantallen voor en de verdeling is gebaseerd op de genetische code (zie tabel 1 in bijlage A).

	00	01	10	11	
00	4 4 * *	[[[[	D D 3 3	]] 3 6	00 01 10 11
01	* * * *	2 2 2 2	E E F F	]]]]	00 01 10 11
10	3 3 3 5	A A A A	G G H H	[[ ]]	00 01 10 11
11	1 1 1 1	B B B B	* * [[	C C C C	00 01 10 11

Tabel 2.3. *De gebruikte conversietabel*

Het karakter dat correspondeert met een bitstring wordt als volgt bepaald: de eerste twee bits van de string moeten overeenkomen met de linker kolom van de tabel. Dit levert 4 regels en 4 kolommen waarop het karakter kan voorkomen. De juiste kolom wordt bepaald door het tweede bitpaar uit de bitstring te vergelijken met de topregel van de tabel. Het laatste bitpaar dient overeen te komen met de rechterkolom en het karakter kan worden uitgelezen. Het karakter dat, bijvoorbeeld, overeenkomt met de bitstring 011000 is de eerste E die in de tabel voorkomt.

Andersom kan een bitstring van een karakter bepaald worden door de bitstrings uit respectievelijk de linkerkolom, de bovenste regel en de rechterkolom aan elkaar te plakken. Een bitstring voor karakter 5 is 100011

Een produktieregel is van de vorm $L < P > R \rightarrow S$ en kan als volgt uit een chromosoom worden gehaald: (STRING is het huidige deel van de produktieregel dat gelezen wordt).

1. Bepaal een willekeurig startpunt in het chromosoom en zoek tot de eerste *. STRING bepaald in eerste instantie de linkercontext van de produktieregel en is nu nog leeg.
2. Lees de eerste 6 bits van het chromosoom in, en zoek in de conversietabel op met welk karakter deze string correspondeert.
3. Als het karakter geen asterisk is, wordt deze aan STRING toegevoegd..
4. Is het karakter een asterisk dan wordt STRING toegekend aan het huidige deel van de produktieregel (L, P, R of S). Laat STRING vervolgens corresponderen met het volgende deel van de produktieregel. Als het huidige deel een S is, wordt met de

volgende stap een nieuwe produktieregel gestart. STRING correspondeert dan met de linkercontext van de nieuwe produktieregel.

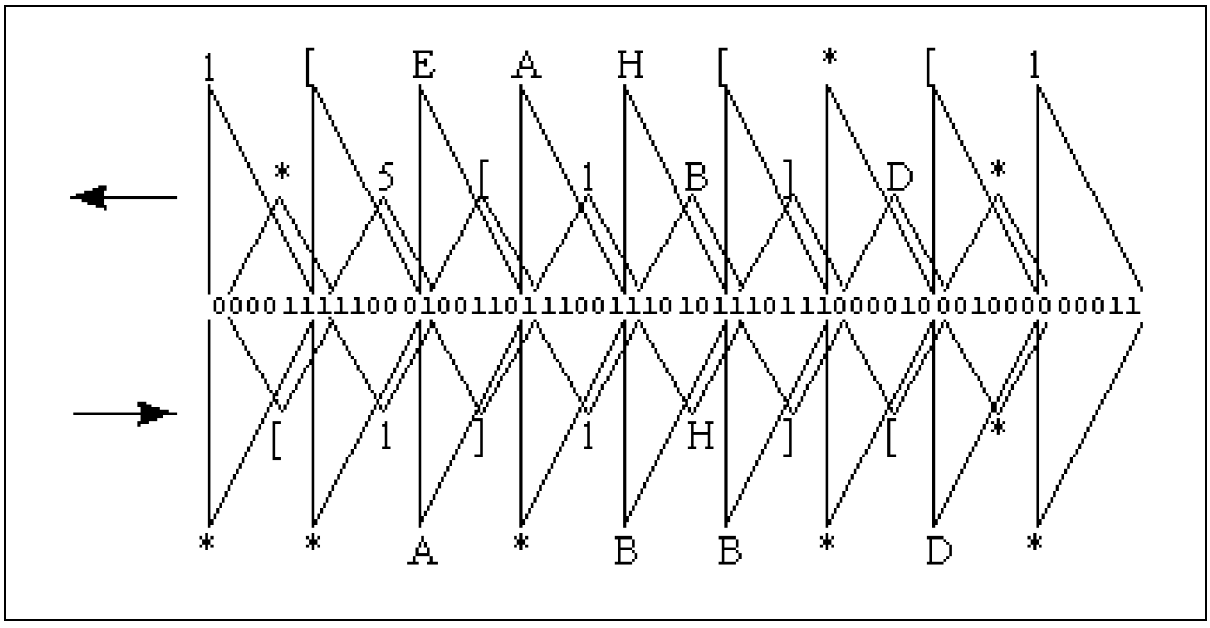
5. Herhaal stap 2 - 4 totdat het hele chromosoom is gelezen. Als de laatst gelezen produktieregel niet uit de vier delen bestaat dan wordt deze verwijderd.

Daar er in een chromosoom op een willekeurige positie begonnen kan worden, kan een chromosoom op 6 verschillende manieren gelezen worden. Starten op positie 1,2,3,4,5 of 6 levert iedere keer compleet andere produktieregels. Bovendien kan een chromosoom ook nog van achter naar voren worden gelezen. Het totaal aantal malen dat een chromosoom gelezen kan worden komt zo op 12.

Een voorbeeld volgens Boers en Kuiper [1992] (figuur 2.16) laat zien hoe een chromosoom gedecodeerd wordt. Voor de duidelijkheid zijn maar vier van de twaalf verschillende mogelijkheden om het chromosoom te decoderen, weergegeven.
De vier decoderingen leveren de volgende vier regels op:
**A*BB*D*
[1]1H][*
D]B1[5
1[*[HAE[1
Alleen de eerste van deze vier is een complete produktieregel die aan alle eisen zoals genoemd in paragraaf 2.2.3 voldoet. Herschreven in de gebruikelijke notatie is dit de volgende produktieregel:

$A > BB \rightarrow D$

Omdat produktieregels aan een aantal eisen moeten voldoen voordat het bruikbare regels zijn en er een kleine kans is dat aan deze regels voldaan is, worden er een aantal reparatie-

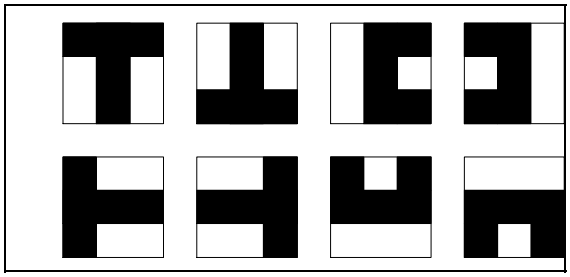


Figuur 2.16. *Produktieregel halen uit chromosoom*

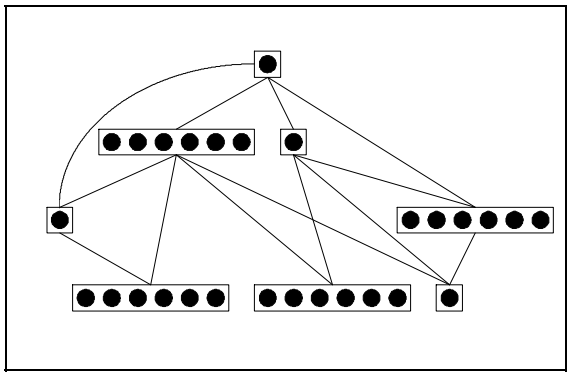
mechanismen toegepast op de gelezen regels. Als [en] niet gepaard voorkomen, dan wordt een teveel aan [of] uit de regel verwijderd. Overbodige cijfers worden op deze manier ook uit de regel verwijderd en het aantal 'goede' produktieregels neemt fors toe. Zulke reparatiemechanismen zijn ook aanwezig in levende cellen om fouten in DNA-replicatie te voorkomen (zie bijlage A).

2.4.4. Resultaten:

De resultaten die Boers en Kuiper tot nu toe bereikt hebben met de combinatie van genetische algoritmen en L-systemen om neurale netwerken te ontwerpen zijn veelbelovend. Eén van de problemen die onderzocht werd is het TC-probleem. Het netwerk moet leren of een T of een C wordt aangeboden. Beide letters, bestaande uit 3x3 pixels kunnen geroteerd zijn over 0, 90, 180 of 270^{o} en kunnen overal op een 4x4 raster staan (zie figuur 2.17). In figuur 2.18 is het gevonden netwerk afgebeeld. Opmerkelijk is dat het netwerk niet 16 maar 13 invoer units heeft; de laatste drie posities van het raster waren niet relevant. De werking van dit gevonden netwerk is vervolgens vergeleken met een aantal simpele netwerken met 3 tot 8 knopen in de hidden laag. Het trainen van deze netwerken met backpropagation bestond uit het 250 maal aanbieden van de 32 verschillende patronen. Ieder netwerk is 50 keer getest. Het gevonden netwerk bleek in 45 van de 50 gevallen alle 32 patronen te herkennen, terwijl de beste van de simpele netwerken dit maar in 30 van de 50 gevallen kon.



Figuur 2.17. *De 8 letters voor het TC-probleem*



Figuur 2.18. *Netwerk voor TC-probleem*

Het "what, where"-probleem, eerder genoemd in 2.3.3, leverde een netwerk op zonder hidden laag. Het probleem blijkt hiermee uitstekend opgelost te kunnen worden. Interferentie tussen de verschillende units treedt namelijk alleen maar op als er een hidden laag gebruikt wordt.

3. Implementatie

3.1. Inleiding

De oorspronkelijke software om netwerken te genereren bestaat uit een hoofdprogramma en drie subprogramma's

- Het hoofdprogramma, *genalg*, manipuleert de populatie van produktieregels en bevat het eigenlijke genetische algoritme;
- *chr2gram* vertaalt een individu van de populatie in een set produktieregels;
- *lsystem* herschrijft deze produktieregels hetgeen resulteert in een adjacency matrix van het netwerk;
- *backprop* traint het netwerk.

Van deze vier programma's zijn in dit onderzoek alleen de eerste drie gebruikt. De laatste is vervangen door een module die aan de hand van een aantal gestelde eisen snel kan bepalen of een gevonden netwerk voldoet. Een nieuw programma, *parmtest*, is aan de bestaande software toegevoegd om verschillende parameters te kunnen optimaliseren. Parmtest maakt gebruik van hetzelfde genetisch algoritme waar het hoofdprogramma genalg gebruik van maakt.

Dit hoofdstuk beschrijft in het kort de bestaande software. De delen die voor dit onderzoek zijn herschreven komen gedetailleerd aan bod. Bij de beschrijving van de experimenten in de hoofdstukken 4, 5, 6, 7 en 8 wordt, indien noodzakelijk, de software beschreven die voor die experimenten gebruikt is.

3.2. Genetisch algoritme

Het hoofdprogramma genalg maakt evenals parmtest, het programma om parameters te optimaliseren, gebruik van een library, Extended GenLib.

Extended GenLib

Deze library verzorgt de afhandeling van het genetisch algoritme. Allereerst wordt de oorspronkelijke versie beschreven, daarna de aanpassingen die gemaakt zijn om het algoritme met twee chromosomen te laten werken.

Datastructuren

Een chromosoom (*member*) wordt beschreven door het volgende structure:

```
typedef struct          /* contains information for each member          */
{
    float fitness;      /* fitness of the member                         */
    unsigned *genPos;   /* pointer to array of genpositions              */
    BYTE **genValue;    /* pointer to array of chromosomes               */
} MEMBER;
```

Behalve de fitness en pointers naar de chromosomen van een individu bevat deze structure ook een pointer naar een array van genposities. Dit maakt het mogelijk om genen van plaats te laten wisselen door middel van inversie. Bij dit onderzoek is van deze mogelijkheid echter geen gebruik gemaakt.

De complete populatie wordt beschreven door:

```
typedef struct              /* contains population info                    */
{
  unsigned popSize,         /* number of members in this population        */
           nrGenes,         /* number of genes in each member              */
           genSize;         /* size of gene in bits (must be multiple of 8)*/
  MEMBER  *member;          /* array of member structures                  */
} POPULATION;
```

nrGenes is oorspronkelijk het aantal genen in een chromosoom. Hier wordt het gebruikt als het aantal chromosomen van een individu. Om de source zo simpel mogelijk te houden, moet de lengte van een chromosoom een veelvoud van 8 zijn.

Als de populatie naar disk geschreven wordt, wordt er eerst een header weggeschreven:

```
typedef struct                  /* the header of a populationfile on disk  */
{
    unsigned long generation;   /* generation number                       */
    unsigned popSize,
             nrGenes,
             genSize;
} POPFILEHEADER;
```

Deze header bevat de specifieke informatie over de populatie en het nummer van de generatie waartoe de populatie behoort. Op deze manier is op ieder willekeurig tijdstip van een 'run' te bepalen hoe lang het genetisch algoritme al draait. Volgend op de header wordt voor elk individu de fitness, het genpositie array en de chromoso(o)m(en) weggeschreven.

De library bestaat uit een aantal modules. In elke module zijn functies ondergebracht met een specifieke taak. De functies die nodig zijn om de opbouw te begrijpen en de functies die zijn veranderd, worden hierna besproken.

De initialisatiemodule bevat functies om geheugen te reserveren en vrij te geven voor een complete populatie of één enkel individu. De boolean "initialize" wordt gebruikt om het geheugen te initialiseren, zodat er een (random) startpopulatie aangemaakt kan worden.

```
POPULATION *DefinePopulation(unsigned popSize, unsigned nrGenes,
                             unsigned genSize, BOOLEAN initialize);
void FreePopulation(POPULATION *p);
MEMBER *DefineMember(unsigned nrGenes, unsigned genSize);
void FreeMember(MEMBER *m, unsigned nrGenes);
```

Bewaren en lezen van disk
Zoals ook bij de voorgaande functies is het bewaren en lezen van disk mogelijk voor een hele populatie en voor een enkel individu.

```
int SaveMember(MEMBER *m, unsigned nrMember, FILE *fp,
                    unsigned nrGenes, unsigned genSize);
int LoadMember(MEMBER *m, unsigned nrMember, FILE *fp,
                    unsigned nrGenes, unsigned genSize);
```

Hierbij wordt de parameter nrMember gebruikt om een individu uit de populatie aan te duiden.

```
int SavePopulation( POPULATION *p, FILE *fp, unsigned long generation);
int LoadPopulation( POPULATION *p, FILE *fp, unsigned long *generation);
int SavePopName(POPULATION *p, char *filename, unsigned long generation);
int LoadPopName(POPULATION *p, char *filename, unsigned long *generation);
```

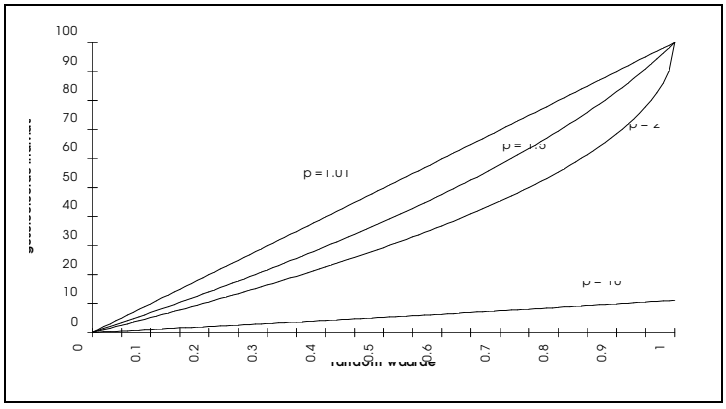
Het opslaan en lezen kan op twee manieren plaatsvinden, als er een filepointer bestaat naar de file of als de naam van de file wordt meegegeven. De generation parameter die opgeslagen wordt in de file kan beschouwd worden als een voortgangsmeter. Het geeft aan welke generatie er in de populatie is opgeslagen.

Genetische operatoren
De genetische operatoren zoals beschreven in hoofdstuk 2.1 worden uitgevoerd met de hieronder beschreven functies

Selectie van een individu vindt plaats met de functie:

```
unsigned Rankselect(POPULATION *p, double pressure);
```

Deze functie is gebaseerd op rank based selectie zoals beschreven in hoofdstuk 2.1.
Wanneer de pressurewaarde tussen de 1.0 en 2.0 ligt, kan de functie alle waarden aannemen tussen 0 en de grootte van de populatie. Met een pressurewaarde van 2.0 is de kans dat het individu met de hoogste fitnesswaarde wordt geselecteerd 2 maal zo groot als de kans dat het individu halverwege de populatie wordt geselecteerd. In figuur 1 is te zien dat wanneer de pressure groter dan 2 wordt niet alle individuen meer



Figuur 3.19*. Invloed pressure op selectiekans.*

geselecteerd worden. In het geval van een populatiegrootte van 100 en een pressurewaarde van 10.0 wordt er een selectie gemaakt uit de 10 individuen met de hoogste fitness.

Om rank-based selectie te kunnen uitvoeren, is het noodzakelijk dat de populatie gesorteerd is op de fitnesswaarde. Dit kan met

```
void SortPopulation(POPULATION *p);
```

Het terugzetten van een individu in de populatie kan met "one at a time replacement", waarbij *member* een individu is uit een nieuw gevormde populatie *np*, die wordt teruggeplaatst in de oorspronkelijke populatie *p*.

```
unsigned RankReplace(POPULATION *p, POPULATION *np, unsigned member);
```

Om crossing-over uit te kunnen voeren is de functie :

```
BitCrossover(POPULATION *oldpop, unsigned parent1, unsigned parent2,
             POPULATION *newpop, unsigned child, double pCross, unsigned Sites);
```

gebruikt. *Oldpop* bevat de beide ouders parent1 en parent2. Na crossing-over ontstaat er één nakomeling. Deze wordt in newpop gezet.

```
void Mutate(POPULATION *p, unsigned member, int variance, double *pMut);
void BitInvert(POPULATION *p, unsigned member,  double pInv);
```

zorgen voor mutaties en inversie in een individu. Mutatie verandert ieder bit met een kans pMut van een 0 in een 1 of een 1 in een 0. Inversie is, zoals in hoofdstuk 2 beschreven, een operator die genen verwisseld. Aangezien er hier sprake is van maar één gen, werkt inversie op een andere manier. Tussen twee punten a en b in een chromosoom worden alle bits verwisseld. Inversie vindt plaats met een kans pInv. Zowel Mutate als Bitinvert veranderen het individu. Om te voorkomen dat de originele ouders worden veranderd, is er een functie die een kopie maakt van een individu. Hierbij wordt individu *member* uit *p* gekopieerd naar *newmember* uit *np*.

```
void CopyMember(POPULATION *p, unsigned member, POPULATION *np,
                unsigned newmember);
```

De basis van het hele programma is hieronder weergegeven. De functie Fitness voert de complete bepaling uit van de fitness van een individu.

```
#define POPSIZE     512     /* grootte van de populatie                */
#define CHROMSIZE   1024    /* lengte van een chromosoom               */
#define PRESSURE    2.1     /* selectiedruk                            */
#define PCROSS      0.8     /* crossing-over kans                      */
#define SITES       2       /* Aantal crossing-over punten             */
#define PINV        0.6     /* inversie kans                           */
#define PMUT        0.003   /* mutatie kans                            */
```

```
void main(void)
{   POPULATION *pop;        /* Bevat de populatie                          */
    POPULATION *localpop;   /* Bevat de nakomelingen van een generatie     */
    POPULATION *newparent   /* Nodig als tussenresultaat van cross-over    */
    unsigned p1, p2;        /* De gekozen ouderstrings                     */
    unsigned g;             /* Buitenste loopteller, geeft generatie weer  */
    unsigned i;             /* loopteller voor aflopen nieuwe individuen   */

    pop       = DefinePopulation(POPSIZE,   1, CHROMSIZE, TRUE);
    localPop  = DefinePopulation(nrMembers, 1, CHROMSIZE, FALSE);
    newparent = DefinePopulation(2, 1, CHROMSIZE, FALSE);

    /*  In de volgende loop worden nieuwe strings aangemaakt en in de populatie
        geplaatst. Als de maximale fitness in de populatie voorkomt wordt de loop
        beeindigd      */
    for (g=0; pop->member[0].fitness < MINFITNESS; g++)
    {
        /* Creeer eerst nrMembers nieuwe individuen per generatie          */
        for(i=0; i < nrMembers; i++)
        {
            do
            {   p1 = RankSelect(pop, PRESSURE);
                p2 = RankSelect(pop, PRESSURE);
            } while(p1 == p2);

            CopyMember(pop, p1, newparent, 0);
            CopyMember(pop, p2, newparent, 1);
            BitInvert(newparent, 0, PINV);
            BitInvert(newparent, 1, PINV);
            BitCrossover(newparent, 0, 1, localPop, i, PCROSS, SITES);
            Mutate(localPop, i, 10, PMUT);
        }
        /* Bepaal fitness van de nieuwe individuen                         */
        for(i=0; i < nrMembers; i++)
            localPop->member[i].fitness=Fitness(localPop, i);

        /* Sorteer oorspronkelijke populatie                               */
        SortPopulation(pop);

        /* Alle individuen worden één voor één in de populatie gezet */
        for(i=0; i<nrMembers; i++)
            RankReplace(pop, localPop, i);
    }
    /* Zet populatie op disk, en maak gereserveerd geheugen vrij           */
    SavePopName(pop, "test.pop", g);
    FreePopulation(newparents);
    FreePopulation(pop);
    FreePopulation(localPop);
}
```

3.3. van bitstring naar netwerk, het G2L-systeem

De ontwikkeling van bitstring naar netwerk loopt via een tweetal modules:
chr2gram, en lsystem.
In chr2gram (chromosoom to grammar) worden de produktieregels uit het chromosoom gehaald. In lsystem wordt uit deze produktieregels samen met het opgegeven axioma een netwerk gegenereerd.

chr2gram
De functie

```
int getProduction(byte chromosome[], unsigned size, unsigned pos, char dir, char buffer[]);
```

leest vanaf positie pos het chromosoom uit, waarbij iedere 6 bits met behulp van de in hoofdstuk 2 beschreven conversietabel worden vertaald in een karakter.
Voor elke bit in het chromosoom wordt deze functie tweemaal aangeroepen. Eenmaal met de leesrichting mee, *dir* = FORWARD, en eenmaal tegen de leesrichting in, dir = BACKWARD. In de variabele *buffer* wordt de eventuele produktieregel teruggegeven.
Een verschil met de oorspronkelijke versie is dat naast een asterisk de punt als scheidingsteken tussen de verschillende delen van een produktieregel wordt gebruikt (zie experimenten in hoofdstuk 7). In de opvolger doet de punt niets en wordt uit de string gefilterd. In linkercontext, rechtercontext en voorganger werkt de punt als een scheidingsteken tussen de verschillende delen.
Dit heeft enkele kleine wijzigingen in deze functie tot gevolg:
1. Het eerste karakter dat gelezen wordt mag zowel een '*' als een '.' zijn.
2. Zowel linkercontext, rechtercontext als voorganger worden gescheiden door een punt of asterisk. Wanneer het aantal gelezen asterisken kleiner is dan 3 wordt de punt vervangen door een asterisk. Zodra het aantal gelezen asterisken 3 is geldt dit niet meer; dan wordt het 'opvolger' deel van het chromosoom gelezen en wordt de punt genegeerd.

voorbeeld:
chromosoom: ABC*C.ABC*A.AC.BC*CBBA
Op de oude manier geïnterpreteerd: L = ABC, P = CABC, R = AACBC, S = CBBA
Op de nieuwe manier geïnterpreteerd: L = ABC, P = C, R = ABC, S = AACBC

Nadat de vier verschillende delen van de produktieregels in vier verschillende strings zijn gezet worden ze gecontroleerd en van overbodige haakjes ontdaan. Voor elk deel van de produktieregel bestaat een verschillende functie omdat de eisen voor elk deel anders zijn. (zie hoofdstuk 2).

```
static int checkC(char *s);                 /* controleer de context                    */

static int checkP(char *s);                 /* controleer de voorganger (Predecessor) */
```

```
static int checkS(char *s);                    /* controleer de opvolger (Successor)    */
```

Deze functies hebben een return waarde van TRUE als de string correct is, en van FALSE als het aantal openhaakjes in de string groter is dan het aantal sluithaakjes. Een kleine wijziging in deze functies resulteert in meer produktieregels. Een string wordt maar éénmaal van links naar rechts gescand op gepaarde haakjes. Een sluithaakje teveel wordt direct verwijderd. Een string is niet correct als er meer openhaakjes dan sluithaakjes in voorkomen. Door nogmaals de string van achter naar voren te scannen en het teveel aan openhaakjes te verwijderen zijn produktieregels altijd correct.

De gevonden produktieregels worden in een bestand opgeslagen en kunnen door de volgende module worden ingelezen.

lsystem
Deze module zet in een aantal stappen de produktieregels om in een netwerk, dat wordt gerepresenteerd door een adjacency matrix.
Deze module bestaat uit drie delen. Het eerste deel leest de produktieregels en het axioma, en zet deze om in een string. Het tweede deel zet deze string om in een adjacencymatrix. Het laatste deel reorganiseert de matrix om de verbindingen te verwijderen die overbodig zijn, knopen met één ingaande verbinding en één uitgaande verbinding kunnen bijvoorbeeld verwijderd worden. In deze module zijn geen significante wijzigingen aangebracht. Voor een gedetailleerde beschrijving wordt verwezen naar Boers en Kuiper [1992].

3.4. Aanpassingen aan de bestaande software en toegevoegde programma's

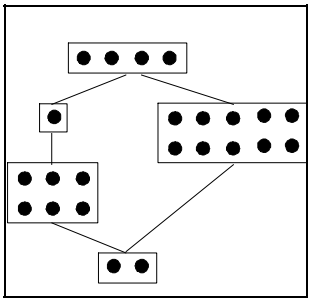
Parmtest, het parametertest-programma
De parameters die nodig zijn voor het genetisch algoritme zijn geoptimaliseerd met behulp van een genetisch algoritme. Het programma waarmee de parameters bepaald zijn, wordt verder het parametertest programma genoemd. De hierboven beschreven software om netwerken te genereren wordt hierna aangeduid als netwerk-programma.
Het genetisch algoritme van het parametertest-programma wordt gevormd door de code die beschreven is in 3.2.
Een chromosoom bevat gecodeerd de parameters die voor het netwerkprogramma nodig zijn. Deze code wordt in de module chr2ipop omgezet in een parameterfile die vervolgens ingelezen wordt door het netwerk-programma. Het netwerk-programma tracht vervolgens, gebruik makende van deze parameterset, binnen een vast aantal stappen een netwerk te leveren dat voldoet aan een aantal gestelde eisen. Als dit lukt dan levert dat een hoge fitness op voor het individu dat deze parameterset codeerde. Als dit niet lukt dan levert dat een lage fitness voor dat individu.

4. Bepalen van een alternatieve fitnessfunktie

4.1. Inleiding

Het optimaliseren van het door Boers en Kuiper [1992] ontwikkelde programma is alleen mogelijk wanneer een groot aantal experimenten uitgevoerd kan worden. Het bepalen van de fitness van een neuraal netwerk met behulp van backpropagation zou teveel rekentijd kosten. Derhalve is het noodzakelijk de fitness op een andere, snellere wijze te bepalen. Essentieel is dat de netwerken die het genetisch algoritme uiteindelijk produceert dezelfde vorm hebben als netwerken waarbij de fitness met backpropagation is bepaald.

Om de juiste vorm van het geproduceerde netwerk vast te stellen, is uitgegaan van een aantal specifieke kenmerken van de netwerken die Boers en Kuiper [1992] gegenereerd hebben. Het door hun gevonden netwerk voor het TC-probleem, besproken in hoofdstuk 2, is afgebeeld in figuur 2.18. Daarnaast is gebruik gemaakt van de twee netwerken die gevonden zijn voor het mapping-probleem. Bij dit classificatieprobleem moet het netwerk een figuur uit een kaart van 10x10 figuren correct indelen in een van de vier mogelijke klassen. Eén van de twee netwerken is afgebeeld in figuur 4.1. Overigens is een belangrijk kenmerk dat de geproduceerde netwerken modulair dienen te zijn.



Figuur 4.20*. Netwerk voor mapping probleem*

Om na te gaan of bepaalde kenmerken een geschikt netwerk opleveren wordt een fitness-functie samengesteld. Deze vervangt de oorspronkelijke module waarbij het netwerk getraind wordt met backpropagation (Backprop). Vervolgens wordt het programma gedraaid. Op het moment dat een netwerk gevonden wordt dat de maximale fitness bezit, wordt het programma afgebroken. Door dit netwerk vervolgens te tekenen, kan worden vastgesteld of het netwerk inderdaad lijkt op de netwerken die door Boers en Kuiper [1992] gevonden zijn.

4.2. Onderzochte kenmerken

Een netwerk bestaat uit een verzameling eenheden die onderling verbonden kunnen zijn. Een verbinding loopt bij feedforward netwerken in één richting en dient zowel als uitvoer voor de ene eenheid en als invoer voor de andere eenheid. Daarnaast is er een onderscheid te maken tussen de verschillende eenheden. De invoer-eenheden ontvangen geen invoer van andere eenheden, maar alleen van buiten. De uitvoereenheden sturen de uitvoer niet naar andere eenheden, maar naar buiten. De eenheden kunnen vervolgens opgedeeld worden in modules. Dit zijn groepen eenheden die dezelfde verbindingen hebben naar andere eenheden.
Kenmerken van een netwerk zijn bijvoorbeeld:

- aantal eenheden in het netwerk
- aantal verbindingen in het netwerk
- aantal modules in het netwerk

- gemiddeld aantal eenheden per module
- gemiddeld aantal verbindingen tussen modules
- aantal invoereenheden
- aantal uitvoereenheden

In totaal zijn er 11 verschillende kenmerken en combinaties van kenmerken onderzocht. De meest relevante worden in de volgende paragrafen behandeld.

4.2.1. Gemiddeld aantal aangrenzende eenheden per module

Per module is nagegaan met hoeveel eenheden uit andere modules er een verbinding is. Wanneer de modules van het in figuur 4.1 getekende netwerk van het mapping probleem van **onder** naar **boven** en van **links** naar **rechts** worden genummerd wordt de berekening van dit criterium als volgt uitgevoerd (tabel 4.1):

aantal aangrenzende eenheden per module

module	1	2	3	4	5
1	-	6	-	10	-
2	2	-	1	-	-
3	-	6	-	-	4
4	2	-	-	-	4
5	-	-	1	10	-

***Tabel 4.4**. Aantal aangrenzende eenheden per module in het mapping-probleem*

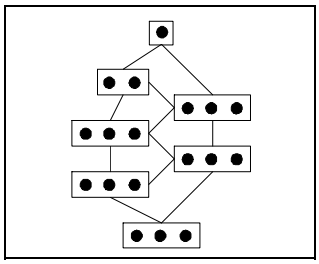
De 5 modules hebben in totaal 46 aangrenzende eenheden, hetgeen een gemiddelde van 9.2 aangrenzende eenheden per module geeft. Voor het netwerk van het TC probleem ligt dit aantal hoger, namelijk gemiddeld 10.6 aangrenzende eenheden per module. Het aantal aangrenzende eenheden per module is vastgesteld op de ondergrens 9.

Gebruikte fitnessfunktie:
AverageInOutNodes is het gemiddeld aantal aangrenzende eenheden per module voor het netwerk waarvan de fitness bepaald wordt.

$$\text{fitness} = \begin{cases} 9^2 - (9 - \text{AverageInOutNodes})^2 & \text{als} \quad \text{AverageInOutNodes} \leq 18 \\ 0 & \text{als} \quad \text{AverageInOutNodes} > 18 \end{cases}$$

Met deze fitnessfunktie werden twee netwerken gegenereerd die voldeden aan de bovenstaande specificaties. Het eerste netwerk bevatte 18 eenheden en 7 modules (figuur 4.2). Het tweede netwerk (niet afgebeeld) bevatte 16 eenheden en 10 modules. In beide gevallen is het aantal modules in verhouding tot het aantal eenheden relatief hoog. Daarom is gezocht naar een specificatie die dit voorkomt.

4.2.2. Gemiddeld aantal eenheden per module



Figuur 4.21. *Netwerk dat voldeed aan criterium*

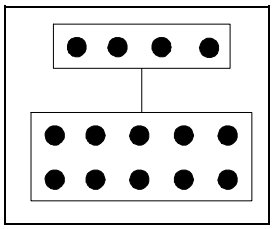
Om het zoeken naar grotere modules te bevorderen werd als specificatie het gemiddeld aantal eenheden per module ingesteld. Voor het netwerk van figuur 4.1 geldt:

module 1: 2 eenheden
module 2: 6 eenheden
module 3: 1 eenheden
module 4: 10 eenheden
module 5: 4 eenheden
totaal: 23 eenheden

Het gemiddeld aantal eenheden per module is dus 4.6. Het netwerk van het TC-probleem (figuur 2.18) toont een gemiddelde van 28/8 = 3.5 eenheden per module. Het tweede netwerk voor het mapping-probleem (niet afgebeeld) heeft gemiddeld 24/4 = 6.0 eenheden per module. Het gemiddeld aantal eenheden per module voor de nieuwe netwerken is daarom op 5 gesteld.

Gebruikte fitnessfunktie:
AverageNodes is het gemiddeld aantal eenheden per module.

$$\text{fitness} = \begin{cases} 5^2 - (5 - \text{AverageNodes})^2 & \text{als} \quad \text{AverageNodes} \leq 10 \\ 0 & \text{als} \quad \text{AverageNodes} > 10 \end{cases}$$



Figuur 4.22. *Gemiddeld 5 eenheden per module*

De netwerken die door het gebruik van deze fitnessfunktie werden gevonden bleken uit 2 modules en 10 eenheden te bestaan (figuur 4.3). Het aantal modules is tamelijk klein en niet vergelijkbaar met de oorspronkelijke netwerken van Boers en Kuiper. Daarom werd bij het volgende experiment een combinatie van criteria gebruikt

4.2.3. Gemiddeld 5 eenheden per module en precies 4 modules

Om het zoeken naar meer modules te bevorderen werd naast gemiddeld 5 eenheden per module het criterium toegevoegd dat een netwerk uit 4 modules moet bestaan. Voor de netwerken van het mapping probleem geldt dat het ene netwerk (figuur 4.1) uit 5 modules bestaat en het andere (niet afgebeeld) uit 4. Om de netwerken niet te groot te laten worden is het aantal modules vastgesteld op 4.

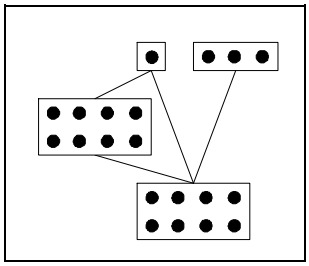
Gebruikte fitnessfunktie:
AverageNodes is het gemiddeld aantal eenheden per module, *Modules* is het aantal modules in een netwerk

$$A = \begin{cases} 5^2 - (5 - \text{AverageNodes})^2 & \text{als} \quad \text{AverageNodes} \leq 10 \\ 0 & \text{als} \quad \text{AverageNodes} > 10 \end{cases}$$

$$B = \begin{cases} 4^2 - (4 - \text{Modules})^2 & \text{als} \quad \text{Modules} \leq 8 \\ 0 & \text{als} \quad \text{Modules} > 8 \end{cases}$$

$$\text{fitness} = A + B$$

Dit bleek geschikte netwerken op te leveren (figuur 4.4). De netwerken bleken echter te veel op elkaar te lijken en nogal klein te zijn. De criteria werden daarom enigszins verruimd.



Figuur 4.23. *Netwerk dat voldeed aan criteria*

4.2.3. Minimaal 4 en maximaal 10 eenheden per module, minimaal 4 modules en minimaal gemiddeld 1.5 verbinding tussen modules

Door de specificatie, dat netwerken precies uit vier modules dienen te bestaan, te verruimen naar tenminste vier modules, wordt het aantal verschillende netwerken dat gegenereerd wordt aanzienlijk vergroot. Een nieuwe toegevoegde specificatie, minimaal gemiddeld 1.5 verbindingen tussen modules, zorgt ervoor dat de verschillende modules niet geheel los van elkaar staan. Het houdt in dat een module gemiddeld met minimaal 1.5 andere module verbonden is. Het netwerk afgebeeld in figuur 4.1 heeft gemiddeld 2 verbindingen tussen de modules. Het tweede netwerk dat gevonden werd bij het mapping-probleem heeft eveneens gemiddeld 2 verbindingen per module.
Omdat netwerken met minimaal 5 eenheden per module erg groot werden, is dit aantal teruggebracht tot minimaal 4 eenheden en maximaal 10 eenheden per module.

Gebruikte fitnessfunktie:
AverageNodes is het gemiddeld aantal eenheden per module, *Modules* is het aantal modules in een netwerk, *Connections* is het gemiddeld aantal verbindingen tussen modules.

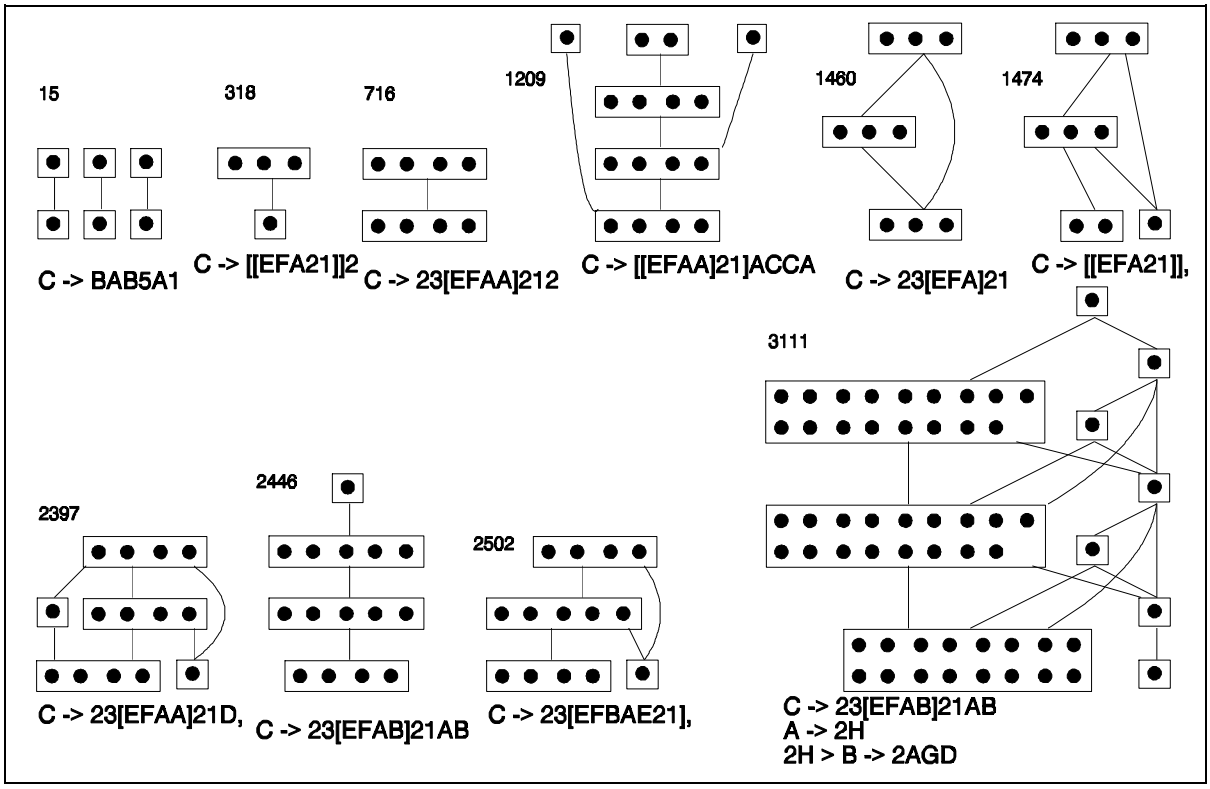
$$A = \begin{cases} 4^2 - (4 - \text{AverageNodes})^2 & \text{als} \quad \text{AverageNodes} < 4 \\ 4^2 & \text{als} \quad 4 < \text{AverageNodes} < 10 \\ 4^2 - (\text{AverageNodes} - 10)^2 & \text{als} \quad 10 < \text{AverageNodes} < 14 \\ 0 & \text{als} \quad \text{AverageNodes} > 14 \end{cases}$$

$$B = \begin{cases} 4^2 - (4 - \text{Modules})^2 & \text{als} \quad \text{Modules} \leq 4 \\ 4^2 & \text{als} \quad \text{Modules} > 4 \end{cases}$$

$$C = \begin{cases} 1.5^2 - (1.5 - \text{Connections})^2 & \text{als} \quad \text{Connections} \leq 1.5 \\ 1.5^2 & \text{als} \quad \text{Connections} > 1.5 \end{cases}$$

$$\text{fitness} = A + B + C + 10.0$$

De netwerken die nu gegenereerd werden kwamen goed overeen met de netwerken waarbij de fitness werd bepaald met backpropagation. Voor één van de gevonden netwerken is het volledige verloop van het genereren van een netwerk afgebeeld in figuur 4.5. Iedere stap resulteert in een verbetering van de fitness voor het netwerk, totdat het netwerk met de maximale fitness ontstaat. Van elke gevonden tussenstap zijn ook de produktieregels gegeven die tot het resultaat hebben geleid. Het gebruikte axioma is CCC.



Figuur 4.24*. Verloop van het bepalen van modulaire netwerken met behulp van een eenvoudige fitnessfunktie. Van linksboven naar rechtsonder verbetert de fitness.*

De specificaties die in 4.2.3 zijn bepaald zijn voor alle experimenten gebruikt.

5. Optimaliseren parameterset

5.1. Inleiding

Het gebruik van een genetisch algoritme brengt met zich mee dat er parameters nodig zijn die het algoritme gebruikt. Dit zijn onder andere de kans op mutaties, inversie en crossing-over en het aantal cross-over punten. Het algoritme zoals in dit onderzoek gebruikt, is gebaseerd op het Genitor-algoritme dat beschreven is door Whitley [1989] (zie ook hoofdstuk 2.1). Dit maakt gebruik van rank-based-selectie en het één voor één vervangen van individuen in de populatie in plaats van de hele populatie opnieuw te creëren. Voor deze rank-based-selectie is eveneens een parameter nodig, de pressure, die de kans bepaalt dat een individu gekozen wordt uit de basispopulatie. Daarnaast gebruikt het L-systeem ook nog een tweetal parameters: het axioma en het aantal herschrijfstappen.
De invloed van de waarden van deze parameters op de snelheid waarmee een geschikt netwerk wordt gevonden, is erg hoog. Zo kan het verkeerd vaststellen van de parameters die specifiek nodig zijn voor het genetisch algoritme, resulteren in vroegtijdige convergentie van het algoritme.

Het netwerkprogramma maakt gebruik van een file die de waarde van alle parameters bevat. Aan het begin van een 'run' worden de parameters ingelezen. Een voorbeeld van een parameterfile is in tabel 5.1 weergegeven.
#size geeft het aantal individuen in de populatie. #nakomelingen geeft het aantal nieuwe individuen dat gegenereerd wordt, voordat ze in de populatie worden teruggeplaatst. #pmut, #pcross, #sites, #pinv en #pressure beïnvloeden het genetisch algoritme. #steps en #axiom worden door L-systeem gebruikt. #chromsize geeft de lengte in bits van ieder individu (= chromosoom).

#size	500
#nakomelingen	20
#pmut	0.001
#pcross	0.7
#sites	4
#pinv	0.3
#pressure	2.1
#steps	6
#axiom	ABC
#chromsize	1024

Tabel 5.1. *Voorbeeld parameter-file die als invoer dient voor het genetisch algoritme*

5.2 Chromosoomlengte

5.2.1. Werking

Behalve de lengte van een chromosoom en de grootte van de populatie, zijn alle parameters tegelijkertijd bepaald met behulp van een genetisch algoritme. Wanneer de chromosoomlengte of het aantal individuen in de populatie eveneens kan veranderen werkt het oorspronkelijke algoritme niet meer op één en dezelfde startpopulatie. Daarom zijn deze parameters apart onderzocht. De verschillende parametersets die bepaald worden in alle volgende experimenten kunnen daardoor op dezelfde startpopulatie worden uitgetest, zodat er uiteindelijk een goede vergelijking is te maken.

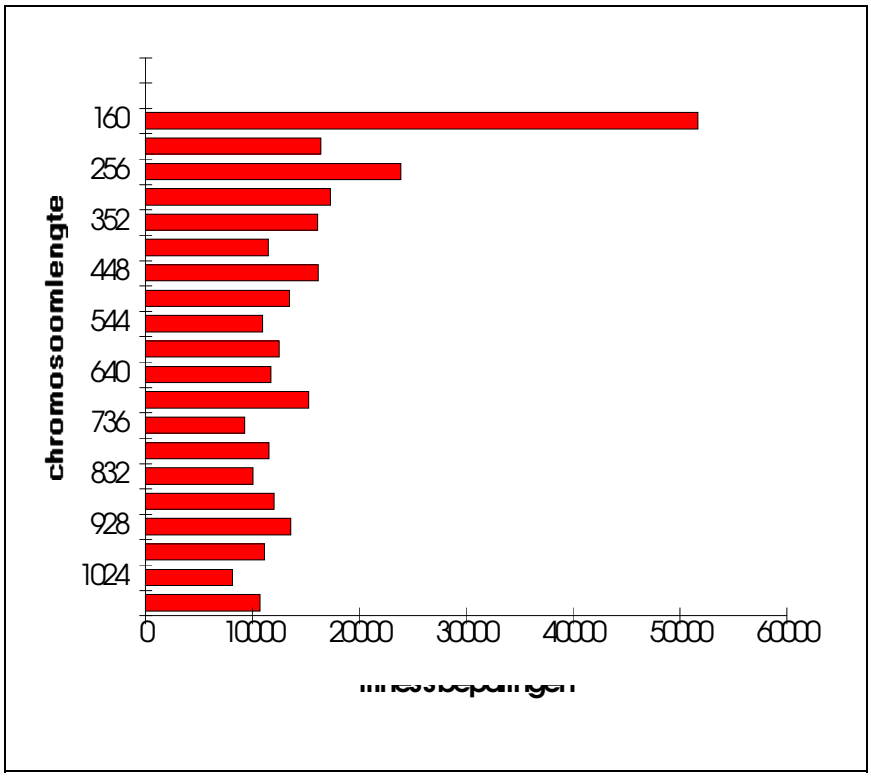
Om de invloed van de lengte van het chromosoom op de snelheid waarmee een optimaal netwerk gevonden wordt, te onderzoeken, werd het volgende experiment uitgevoerd. Bij 20 verschillende chromosoomlengten werden elk 10 startpopulaties aangemaakt. De chromosoomlengten varieerden van 1024 bits tot 112 bits, met stappen van 48 bits. Als maat voor de performance werd het aantal fitnessbepalingen genomen dat nodig is om een netwerk te genereren met een maximale fitness. De chromosoomlengte waarbij het minste aantal fitnessbepalingen nodig is, is de optimale lengte. De overige parameters werden voor elke test gelijk gehouden (zie tabel 5.2).

#size	500
#pmut	0.01
#pcross	1
#sites	chromosoomlengte/100
#pinv	0.7
#steps	6
#axiom	A
#pressure	2.1

***Tabel 5.2**. Ingestelde parameters bij een variabele chromosoomlengte*

5.2.2. Resultaten & discussie

Het gemiddelde aantal fitnessbepalingen, over de 10 startpopulaties, dat nodig was per chromosoomlengte is afgebeeld in figuur 5.1.



***Figuur 5.25**. Invloed chromosoomlengte op aantal fitnessbepalingen dat nodig is om een optimaal netwerk te verkrijgen*

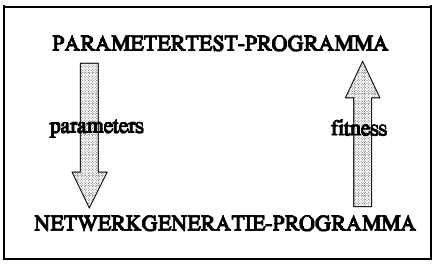
Het aantal bits in een chromosoom vertoont in het gebied van 500 tot 1024 bits, geen grote invloed op de snelheid waarmee een optimaal netwerk wordt gevonden. Pas bij een

lengte van minder dan 500 bits blijkt dat meer fitnessbepalingen nodig zijn. Deze "verslechtering" is significant bij een chromsoomlengte van 448 bits ten opzichte van een lengte van 976 bits. Ten opzichte van een lengte van 1024 bits is een lengte van 304 bits significant ongunstiger. Hierbij is gebruik gemaakt van de Student-T-test met betrouwbaarheid van 95% ($\alpha = 0.5$) [Rijken van Olst, 1974].

5.3. Overige parameters bepalen

5.3.1. Werking

Voor het bepalen van de waarden van de overige parameters is een genetisch algoritme gebruikt. Dit parametertest-programma levert parameter-files voor het netwerk-programma. Het netwerk-programma bepaalt de fitnesswaarde van de parameters en geeft deze waarde weer terug aan het parametertest-programma (zie figuur 2). Voor het genetisch algoritme van het parametertest-programma is gebruik gemaakt van het algoritme van het netwerk-genereer programma. Hiervoor wordt verwezen naar hoofdstuk 3. De populatie van het parametertest-programma bestaat uit strings met een lengte van 48 bits. De waarde van elke parameter kan variëren binnen een vastgesteld minimum en maximum. Op grond van deze waarden is de codering van de parameters in de string als volgt vastgelegd.



***Figuur 5.26**. Wisselwerking tussen beide programma's*

	min	max	stap	codering
#nakomeling	1	512	1	9 bits Gray code
#pressure	1.1	13.9	0.1	7 bits Gray code
#pcross	0.0625	1.0	0.0625	4 bits Gray code
#sites	1	16	1	4 bits Gray code
#pinv	0.0625	1.0	0.0625	4 bits Gray code
#steps	1	16	1	4 bits Gray code
#axiom	(A*B*C*)*			8 bits
#pmut	0.001	0.128	0.001	7 bits Gray code

Deze codering levert een string op bestaande uit 47 bits. Voor de eenvoud is gebruik gemaakt van een string bestaande uit 48 bits; het laatste bit werd niet gebruikt.
Bij de codering van de parameters is met een aantal zaken rekening gehouden. De parameters die mogelijk invloed op elkaar uitoefenen zijn achter elkaar opgeslagen. Dit is zeker het geval bij #pcross en #sites en mogelijk het geval bij #steps en #axiom of #nakomeling en #pressure. Volgens Goldberg [1989] levert dit betere resultaten op omdat

parameters bij cross-over minder vaak gescheiden worden en zo beter op elkaar afgestemd blijven. Daarnaast is er gebruik gemaakt van Gray-coding. Dit heeft als voordeel dat een mutatie van 1 bit maar een kleine verandering van de waarde tot gevolg heeft [Goldberg, 1989].

De fitness (parameterfitness) van een chromosoom van het parametertest-programma wordt bepaald door de fitness (verder netwerkfitness genoemd) van het netwerkprogramma. Deze netwerkfitness wordt bepaald door de criteria waaraan een netwerk moet voldoen die opgesteld zijn in hoofdstuk 4. Wanneer het netwerk aan alle criteria voldoet wordt de maximale netwerkfitness gehaald. Het aantal netwerkfitnessbepalingen waarbinnen het optimale netwerk gevonden moet worden is vastgesteld op 4000. Wordt het optimale netwerk niet binnen 4000 netwerkfitness bepalingen gevonden, dan wordt de netwerkfitness de maximale netwerkfitness die tot dan toe is gehaald. Deze netwerkfitness wordt teruggegeven aan het parametertest-programma.
Om te voorkomen dat parametersets die bij toeval een goed resultaat opleveren net zo gewaardeerd worden als parametersets die herhaaldelijk goede resultaten opleveren, wordt de parameterfitness aangepast. Zolang de netwerkfitness maximaal is, wordt met dezelfde parameterset opnieuw een netwerkbepaling gedaan. Wanneer de maximale netwerkfitness niet (meer) wordt gehaald, dan wordt de definitieve parameterfitness bepaald en de volgende parameterset wordt getest. Als drie maal na elkaar de maximale netwerkfitness gehaald is, dan wordt de parameterset apart opgeslagen. De definitieve parameterfitness wordt als volgt bepaald:

maximum niet gehaald: 3 * netwerkfitness
maximum 1 maal gehaald: 2 * netwerkfitness + 1 * maximum netwerkfitness
maximum 2 maal gehaald: 1 * netwerkfitness + 2 * maximum netwerkfitness
maximum 3 maal gehaald: 3 * maximum netwerkfitness

Het parametertest-programma maakt eveneens gebruik van een parameterset, deze staat afgebeeld in tabel 5.3. Omdat inversie bij het parametertest-programma alleen zorgt voor een enorm aantal mutaties is de parameter Pinv op 0 gesteld.

#size	512
#nakomeling	20
#pmut	0.01
#pcross	0.25
#sites	1
#pinv	0
#pressure	2.1
#chromsize	48

Tabel 5.3. *Parameterset voor het parametertest-programma*

5.3.2. Resultaten & discussie
Na lang draaien bleek slechts één parameterset (tabel 5.4) drie maal achtereen de maximale netwerkfitness te hebben gehaald. Deze parameterset is vervolgens uitgetest met het netwerk-generatie-programma op 30 verschillende startpopulaties. Daarnaast is de oorspronkelijke

parameterset (tabel 5.5) uitgetest op dezelfde 30 startpopulaties.

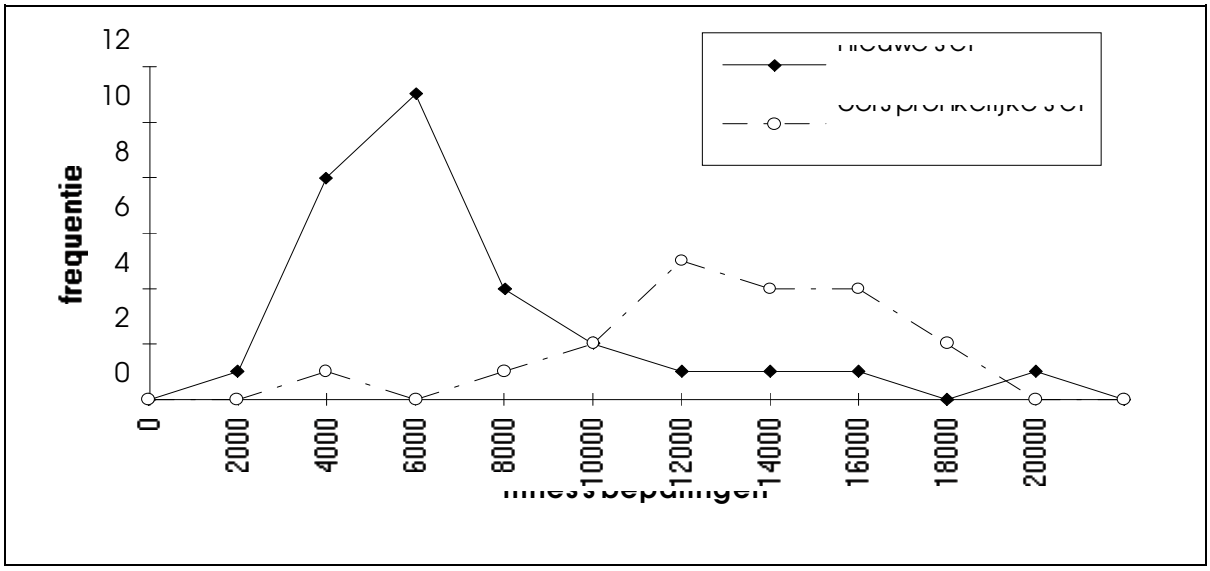
#size	512
#nakomeling	346
#pmut	0.007
#pcross	0.625
#sites	4
#pinv	1.0
#steps	16
#axiom	CCC
#pressure	10.4
#chromsize	1024

Tabel 5.4. *Gevonden parameterset*

#size	512
#nakomeling	1
#pmut	0.01
#pcross	0.5
#sites	10
#pinv	0.7
#steps	6
#axiom	A
#pressure	2.1
#chromsize	1024

Tabel 5.5*. Oorspronkelijke parameterset*

Het aantal fitnessbepalingen dat nodig was om het optimale netwerk te vinden is voor beide tests uitgezet in figuur 5.3. Het blijkt dat de gevonden parameterset significant beter is dan degene die oorspronkelijk gebruikt is. Hierbij is gebruik gemaakt van de symmetrietoets van Wilcoxon en is significant op het 5% niveau [Rijken van Olst, 1974].
Omdat in de parameterfitness geen gebruik is gemaakt van het aantal fitnessbepalingen dat nodig is om een optimaal netwerk te creëren en omdat het programma mogelijk te kort gedraaid heeft, is het niet uit te sluiten dat er parametersets zijn die (nog) betere resultaten geven dan de parameterset die nu bepaald is. Er is later nogmaals een dergelijk experiment uitgevoerd waarbij tevens de conversietabel voor het L-systeem door het genetisch algoritme wordt geoptimaliseerd. De resultaten van deze test worden beschreven in hoofdstuk 8. In bijlage C is een experiment opgenomen waarin het aantal cross-over punten apart bepaald wordt.



Figuur 5.27*. Invloed nieuwe en oorspronkelijke parameterset op de snelheid van netwerk generatie*

6. Selectie van beste nakomeling

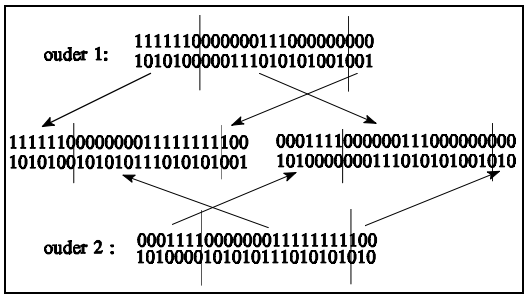
6.1. Inleiding

Het genetisch algoritme zoals tot nu toe gebruikt, heeft als nadeel dat van slechts één nakomeling de fitness bepaald wordt. De andere nakomeling wordt direct verworpen. Dit houdt in dat in 50% van de gevallen de verkeerde nakomeling gekozen wordt. In het ideale geval wordt de nakomeling met de hoogste fitness bepaald, voordat één van beide nakomelingen gekozen wordt. Informatie over beide nakomelingen kan verkregen worden van de ouders van de nakomelingen. Zo is het wenselijk om te weten welke delen van het chromosoom van een ouder goede produktieregels hebben opgeleverd, omdat dan precies te voorspellen is welke nakomeling deze goede produktieregels erft.

6.2. Werking

Om de goede produktieregels van de ouders te kunnen bewaren is een tweede chromosoom van dezelfde lengte als het oorspronkelijke chromosoom geïmplementeerd. Een individu bestaat nu uit twee chromosomen, het oorspronkelijke produktiechromosoom en een nieuw markeerchromosoom. Het markeerchromosoom is initieel geheel gevuld met nullen. Tijdens het scannen van het produktiechromosoom wordt bijgehouden op welke bits de produktieregel gevonden wordt. Tijdens het opbouwen van een netwerk door het L-systeem wordt bekend welke produktieregels gebruikt worden. Voor elke gebruikte produktieregel wordt het markeerchromsoom van startbit tot eindbit gevuld met enen. Bij de paring van twee individuen kan zo bepaald worden welke nakomeling de meeste kans heeft op een hoge fitness.

Met behulp van het tweede chromosoom kan op verschillende manieren invloed worden uitgeoefend op de paring van twee individuen. Zo is het mogelijk de kans op crossing-over te verkleinen op die plaatsen in het chromosoom waar bij één of beide ouders bits zijn gezet in het markeerchromosoom. Ook de kans op mutatie kan op die plaatsen afwezig of lager worden ingesteld. Dit verkleint de kans dat goede produktieregels door crossing-over of mutatie verloren gaan. Daarnaast is het mogelijk om na te gaan welke van de twee nakomelingen de meeste bits in het markeerchromosoom gezet heeft. Crossing-over dient dan op het markeerchromosoom op dezelfde wijze uitgevoerd te worden als op het produktiechromosoom. Dat wil zeggen dat dezelfde cross-over punten gekozen moeten worden (zie figuur 6.1). De nakomeling die de meeste bits gezet heeft zou de beste nakomeling moeten zijn, want deze bevat het hoogste aantal bruikbare produktieregels.



***Figuur 6.28**. Crossing-over met twee chromosomen*

De experimenten die gedaan zijn met betrekking tot het gebruik van het tweede chromosoom hebben zich beperkt tot het selecteren van de beste nakomeling. Bij het paren

van twee individuen werden zowel de bits op de produktiechromosomen als de bits op de markeerchromosomen uitgewisseld. Ook inversie vond op beide chromosomen plaats; mutaties alleen op het al geselecteerde chromosoom.

6.3. Implementatie

De implementatie van een tweede chromosoom was eenvoudig omdat de bestaande structuur is opgezet om gebruik te kunnen maken van meerdere genen en omdat een gen in de implementatie van het netwerk-programma gelijk is aan een chromosoom. Aangezien de functie van beide chromosomen verschillend is, mogen ze niet van plaats wisselen. Dit was in het oorspronkelijke algoritme wel toegestaan, omdat in dat geval de genen door inversie van plaats konden wisselen.
Het tweede chromosoom codeert de posities van de produktieregels in chromosoom 1. Op die plaatsen wordt in chromosoom 2 een 1 gezet. Het is te verwachten dat een individu met veel enen op het tweede chromosoom beter is dan een chromosoom met weinig enen omdat die de meeste bruikbare produktieregels zal bevatten.

Zodra de produktieregels uit het eerste chromosoom worden gehaald, worden de posities van deze produktieregels opgeslagen in een array.
Met de functies:

```
void ResetDominanceBits(POPULATION *p,     /* populatie                          */
                        unsigned m);       /* chromosoom dat gereset moet worden */
void setDominanceBits(POPULATION *p,       /* populatie                          */
                        unsigned m,        /* chromosoom dat gevuld moet worden*/
                        unsigned startbit[], /* startposities van produktieregels */
                        unsigned endbit[]);  /* eindposities van produktieregels  */
```

worden respectievelijk de bits in het tweede chromosoom op nul gezet of gevuld met enen volgens de posities van de produktieregels in het eerste chromosoom.

Daarnaast geldt dat inversie en crossing-over op beide chromosomen op dezelfde plaatsen moet gebeuren. Een kleine aanpassing in de functie **BitInvert** zorgt ervoor dat in beide chromosomen inversie plaatsvindt. Voor **BitCrossover** was dit ingewikkelder omdat één van de oorspronkelijke nakomelingen door BitCrossover wordt veranderd. BitCrossover zorgt er voor dat een nakomeling direct wordt geselecteerd. Nadat de cross-over punten zijn bepaald en voordat de crossing-over wordt uitgevoerd wordt al de 'beste' nakomeling bepaald.

Het bepalen van de 'beste' nakomeling gaat met de functie

```
int count_bits(POPULATION *pop, unsigned p1, unsigned p2, int *pos, int points)
{
   int p, bits[2], bit, bpos, b1, b2;
   unsigned char *a, *b;
```

```
  bits[0] = 0; bits[1] = 0;
  a = &pop->member[p1].genValue[1][0];
  b = &pop->member[p2].genValue[1][0];
  p = 0; bit = 0; bpos = 0;
  while (p<=points && bit<pop->genSize)
  {
     if (bit==pos[p])  p++;
     b1 = ExtractBit(*a, bit);
     b2 = ExtractBit(*b, bit);
     if (p&0x01) {
          bits[1]+=b1; bits[0]+=b2;
     }
     else    {
          bits[0]+=b1; bits[1]+=b2;
     }
     bit++;
     if ((bit%8)==0)
        { a++; b++; }
  }
  return (bits[0]>bits[1]);
}
```

Aan de hand van de cross-over punten die bepaald zijn in BitCrossover wordt geteld hoeveel enen van chromosoom 2 in nakomeling 1 (bits[0]) en nakomeling 2 (bits[1]) terecht gaan komen. Door het aantal enen te tellen in het markeerchromosoom kan er een keuze worden gemaakt voor de nakomeling. Om de beste nakomeling te selecteren zijn er verschillende methoden onderzocht:

A. Selectie op meeste enen

De nakomeling die na het hele kruisingsproces het grootste aantal enen bezat in het markeerchromosoom, werd geselecteerd als zijnde de beste nakomeling. Deze zou het hoogste aantal goede produktieregels moeten bevatten

B. Selectie minste enen

De nakomeling die het kleinste aantal enen in het markeerchromosoom overhield werd geselecteerd.

C. Selectie op gemiddeld 40 bits

Na een eerste onderzoek bleek dat de lengte van goede produktieregels ongeveer 40 bits is. Door na het volledige kruisings-proces het aantal rijtjes enen dat aaneengesloten voorkwam en het totaal aantal enen te tellen, was het gemiddelde aantal enen per rijtje vast te stellen. Er wordt zo indirect geselecteerd op individuen waarvan de gemiddelde lengte van de produktieregels uit 40 bits bestaat.

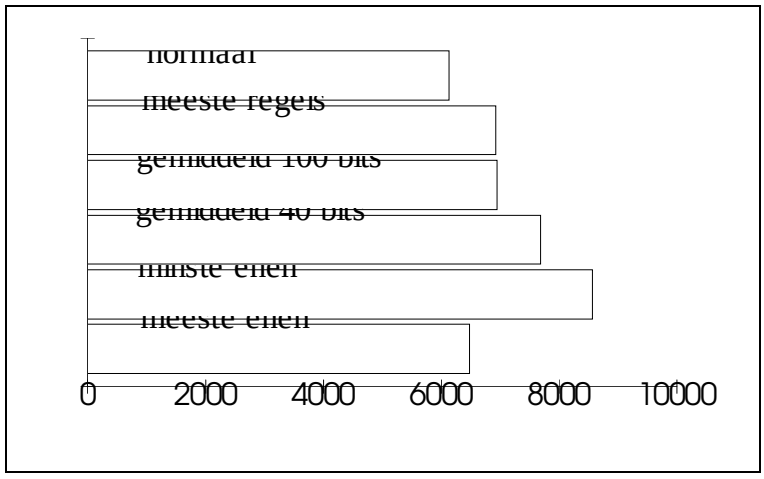
D. Selectie op gemiddeld 100 bits
Na een nader onderzoek bleek het aantal bits dat een goede produktieregel opleverde niet 40 maar 100 te zijn. Van de beste nakomeling moesten de rijtjes enen in het markeerchromosoom een gemiddelde lengte van 100 hebben.

E. Selectie op meeste regels
De nakomeling die in het markeer chromosoom het grootste aantal rijtjes met enen bevatte, werd als beste nakomeling gekozen. Deze zou het hoogste aantal goede produktieregels moeten bevatten.

Om deze verschillende selectiemethodes te onderzoeken werden 30 startpopulaties aangemaakt, elk bestaande uit 512 individuen. Voor de onderzochte selectiemethodes werd vervolgens bepaald hoeveel fitnessbepalingen er per startpopulatie nodig waren om een optimaal netwerk te vinden. Daarnaast werd het oorspronkelijke algoritme gedraaid met deze dertig startpopulaties. De gebruikte parameterset staat afgebeeld in tabel 5.4.

6.4. Resultaten & discussie

De resultaten van bovenge noemde tests zijn weergegeven in figuur 6.2. Voor de verschillende experimenten is het gemiddeld aantal fitnessbepalingen afgebeeld dat nodig is om een optimaal netwerk te vinden. Geen enkele selectiemethode bleek significant beter te zijn dan het selecteren van een willekeurige nakomeling ('= normaal'). Bij alle experimenten lag het gemiddeld aantal fitnessbepalingen rond de 62000. Alleen het experiment waarbij geselecteerd werd op het minste aantal enen bleek significant slechtere resultaten op te leveren dan de oorspronkelijke instelling.



***Figuur 6.29**. Invloed van verschillende selectiemethodes op de snelheid waarmee het optimale netwerk gevonden wordt.*

Ook het experiment waarbij gezocht werd naar produktieregels bestaande uit gemiddeld 40 bits, bleek slechter te werken dan de oorspronkelijke methode. Duidelijk is bij deze methode dat een aantal van 40 bits per produktieregel veel te laag is, het experiment waarbij geselecteerd werd op produktieregels bestaande uit 100 bits leverde aanzienlijk betere resultaten op.

Het beïnvloeden van de selectiemethode heeft geen betere resultaten opgeleverd. Bij selectie op het grootste aantal enen, op gemiddeld 100 enen en op het grootste aantal regels lijkt vroegtijdige convergentie de oorzaak van de tegenvallende resultaten. Samen

met een hoge pressure bij het selecteren van individuen uit de hele populatie wordt de selectie op goede nakomelingen te hoog. Dit levert uiteindelijk een populatie waarin erg weinig variatie zit.

De selectiemethode op het aantal bits is eveneens een tamelijk grove methode om goede produktieregels te bewaren. Er wordt bijvoorbeeld geen onderscheid gemaakt tussen produktieregels die door crossing-over zijn veranderd of verdwenen en regels die wel intact zijn gebleven. Daarnaast blijkt vaak dat de produktieregels die voor het opbouwen van een netwerk gebruikt worden op elkaar zijn afgestemd. Het missen van één enkele schakel bij deze produktieregels kan veroorzaken dat geen netwerk meer gevonden wordt.

7. Optimaliseren van de conversietabel

7.1. Inleiding

Bij het genereren van optimale netwerken wordt gebruik gemaakt van een tweetal algoritmen. Van deze twee is het genetisch algoritme al uitgebreid besproken en onderzocht in de voorgaande twee hoofdstukken. In dit hoofdstuk komt het optimaliseren van de tweede gebruikte techniek aan de orde, het L-systeem.

Het genetisch algoritme levert een individu aan het L-systeem. In het L-systeem worden produktieregels uit het chromosoom gehaald en met behulp van een axioma en een aantal herschrijfstappen omgezet in een netwerk. De optimalisatie van dit deel van het programma is voornamelijk gericht op de grammatica, met name de conversietabel die gegevens uit het chromosoom omzet in produktieregels.

7.2. Werking

Zoals in hoofdstuk 2.4.3 beschreven is wordt voor de codering van het binaire chromosoom naar de produktieregels een conversietabel gebruikt. In deze tabel wordt elk karakter van het gebruikte alfabet { A-H, 1-6, [,] } gerepresenteerd door een string met een lengte van 6 bits. Iedere permutatie van deze 6 bits levert een karakter uit het alfabet. Dit geeft een conversietabel bestaande uit $2^6 = 64$ karakters. Naast de 16 karakters van het alfabet wordt de asterisk gebruikt als scheidingsteken tussen de verschillende delen van de produktieregel. Deze in totaal 17 karakters komen in verschillende aantallen voor en zijn willekeurig verdeeld over de conversietabel.

Een chromosoom bestaande uit 1024 bits, levert gemiddeld 40 produktieregels. Het blijkt echter dat vaak geen van deze 40 regels toepasbaar is op het gegeven axioma. Bij het gebruik van een axioma van 1 letter, bijvoorbeeld A, moet de voorganger van de produktieregel precies A zijn en zowel de rechter- als de linkercontext moeten leeg zijn. Het blijkt dat de kans hierop minimaal is. In een behoorlijk aantal gevallen is er geen enkele produktieregel die voldoet, doordat de voorganger, de linker context of de rechter context te lang is.

Om het aantal toepasbare produktieregels te vergroten is de conversietabel aangepast. Iedere aanpassing is getest op dertig verschillende startpopulaties elk bestaande uit 512 individuen. Van elke conversietabel werd vervolgens bepaald hoeveel fitnessbepalingen er nodig waren om een optimaal netwerk te vinden. Het gevonden netwerk moet voldoen aan de in hoofdstuk 4 bepaalde specificaties. De gebruikte parameterset is afgebeeld in tabel 5.4.

A. Oorspronkelijke conversietabel:

Allereerst is de oorspronkelijke conversietabel (zie ook tabel 2.3) getest met de dertig startpopulaties:

4	4	*	*	[	[	[	[	D	D	3	3	]	]	3	6
*	*	*	*	2	2	2	2	E	E	F	F	]	]	]	]
3	3	3	5	A	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	B	B	B	B	*	*	[	[	C	C	C	C

B. Vervang letters door asterisken:
Door in de conversietabel meer asterisken te plaatsen wordt er een groter aantal produktieregels aangemaakt. De lengte van de produktieregels zal daarnaast kleiner worden. Bij dit experiment zijn één A, één B en één C vervangen door asterisken.

gebruikte conversietabel:

4	4	*	*	[	[	[	[	D	D	3	3	]	]	3	6
*	*	*	*	2	2	2	2	E	E	F	F	]	]	]	]
3	3	3	5	*	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	*	B	B	B	*	*	[	[	*	C	C	C

C. Vervang cijfers door asterisken:
In de produktieregels die gevonden werden met de oorspronkelijke tabel, bleken vaak meerdere cijfers achter elkaar te staan. Omdat dit vaak dezelfde cijfers betrof en dit geen nut heeft, zijn bij dit experiment een aantal cijfers vervangen door asterisken. In totaal zijn twee drieën en één twee vervangen door asterisken.

gebruikte conversietabel:

4	4	*	*	[	[	[	[	D	D	3	3	]	]	*	6
*	*	*	*	*	2	2	2	E	E	F	F	]	]	]	]
*	3	3	5	A	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	B	B	B	B	*	*	[	[	C	C	C	C

D. Voeg nieuw karakter toe:
Omdat bij de voorgaande experimenten ook de opvolger kleiner werd is een nieuw karakter ingevoerd, de punt. In de opvolger doet de punt niets en wordt uit de string gefilterd. In het linker deel van de produktieregel fungeert de punt als asterisk. Zo is de kans op matchende produktieregels groot en wordt de opvolger niet te klein. Bij dit experiment zijn drie drieën en één twee vervangen door punten.

gebruikte conversietabel

4	4	*	*	[	[	[	[	D	D	3	3	]	]	.	6
*	*	*	*	.	2	2	2	E	E	F	F	]	]	]	]
.	.	3	5	A	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	B	B	B	B	*	*	[	[	C	C	C	C

E. Vervang nieuw karakter door een asterisk:
Om na te gaan of het invoeren van een nieuw karakter effect heeft op het verloop van de netwerkbepaling, is de punt uit het voorgaande experiment vervangen door een asterisk.

gebruikte conversietabel

4	4	*	*	[	[	[	[	D	D	3	3	]	]	*	6
*	*	*	*	*	2	2	2	E	E	F	F	]	]	]	]
*	*	3	5	A	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	B	B	B	B	*	*	[	[	C	C	C	C

F. Nog meer asterisken en punten:
In dit experiment werden zowel cijfers als letters vervangen door asterisken of punten. Er werd één A, één B, één 2 en één asterisk vervangen door een punt en twee drieën door een asterisk. Bij dit experiment werd C niet vervangen omdat deze letter in het gebruikte axioma voorkomt.

gebruikte conversietabel:

4	4	*	*	[	[	[	[	D	D	3	3	]	]	*	6
.	*	*	*	.	2	2	2	E	E	F	F	]	]	]	]
*	3	3	5	.	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	.	B	B	B	*	*	[	[	C	C	C	C

G. Veel asterisken
Om de invloed van de punt te bepalen is opnieuw dezelfde conversietabel als hierboven gebruikt, alleen zijn de punten vervangen door asterisken.

gebruikte conversietabel:

4	4	*	*	[	[	[	[	D	D	3	3	]	]	*	6
*	*	*	*	*	2	2	2	E	E	F	F	]	]	]	]
*	3	3	5	*	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	*	B	B	B	*	*	[	[	C	C	C	C

7.3. Resultaten & discussie

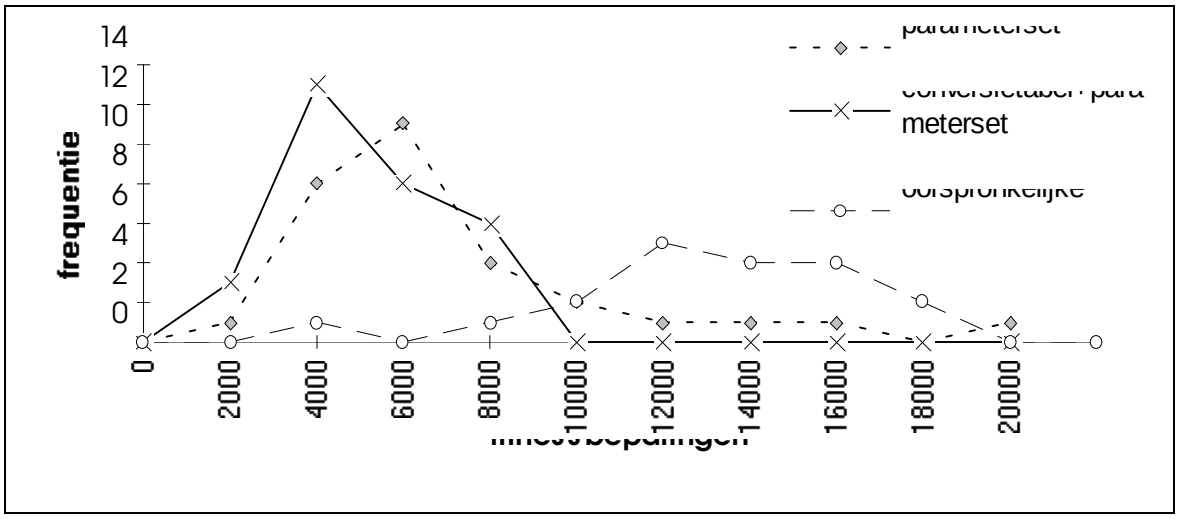
Per conversietabel is het gemiddeld aantal fitnessbepalingen over 30 startpopulaties en de standaardafwijking weergegeven in tabel 7.1. Daarnaast is met behulp van de symmetrie-toets van Wilcoxon bepaald of de tabellen B, C, D, E en F significant betere resultaten opleverden op het 5% niveau [Rijken van Olst, 1974].

	gemiddeld aantal fitnessbepalingen	standaard afwijking	significante verbetering t.o.v. tabel A
tabel A	6143	3827	NVT
tabel B	4828	3034	NEE
tabel C	5517	2993	NEE
tabel D	4542	3104	JA
tabel E	4585	2529	JA
tabel F	4230	1832	JA
tabel G	4585	2529	JA

Tabel 7.1. *Resultaten bij verschillende conversietabellen*

Bij gebruik van conversietabel F blijkt het benodigd aantal fitnessbepalingen behoorlijk gereduceerd te worden. Tevens blijkt dat de gegevens nu nagenoeg normaal verdeeld zijn (zie figuur 7.1). Opvallend is dat het maximaal aantal fitnessbepalingen niet meer boven de 8000 uitkomt en het minimum niet onder de 1700 (zie bijlage C). Bij de andere experimenten lag het minimum lager en het maximum hoger. Dit is ook terug te vinden in de standaard afwijking die duidelijk kleiner is geworden.

De snelheid waarmee het optimale netwerk wordt gevonden, is opnieuw significant sneller geworden. In figuur 7.1 is het aantal fitnessbepalingen dat nodig was om het optimale netwerk te vinden uitgezet voor drie verschillende experimenten. De oorspronkelijke parameterset en conversietabel (oorspronkelijke). De beste parameterset en oorspronkelijke conversietabel (parameterset) en de beste parameterset met de beste conversietabel (tabel F) (conversietabel F + parameterset).



***Figuur 7.30**. Verbeteringen van het aantal benodigde fitnessbepalingen ten opzichte van de oorspronkelijke conversietabel*

8. Gelijktijdig optimaliseren parameterset & conversietabel

8.1. Inleiding

Uit de experimenten die verricht zijn om de conversietabel te optimaliseren is gebleken dat de conversietabel een grote invloed heeft op de snelheid waarmee het netwerk-programma een netwerk vindt dat aan de gestelde eisen voldoet. Daarnaast is uit hoofdstuk 5 gebleken dat dit eveneens geldt voor de waarden van de verschillende parameters. Om de parameterset en de conversietabel op elkaar af te stemmen, is het noodzakelijk om deze gelijktijdig te optimaliseren. Hiertoe is het genetisch algoritme van het parametertest-programma (hoofdstuk 5) uitgebreid met de mogelijkheid de conversietabel eveneens te optimaliseren.

8.2. Werking

De populatie van het genetisch algoritme van het parametertest-programma bestaat nu uit strings met een lengte van 368 bits. De codering van de eerste 48 bits is ongewijzigd en bevat de waarden van de parameters. De overige 320 bits coderen de conversietabel. De conversietabel bevat 18 verschillende karakters die in willekeurige aantallen voorkomen in de in totaal 64 karakters tellende conversietabel. De karakters bestaan uit het al eerder gebruikte alfabet { A-H, 1-6, [,] } met daaraan de asterisk en de punt (zie hoofdstuk 7) toegevoegd als scheidingsteken tussen de verschillende delen van een produktieregel. Voor de codering van een enkel karakter zijn 5 bits nodig hetgeen in totaal 32 verschillende karakters kan opleveren. Dit heeft als gevolg dat er een tabel nodig is om deze karakters te coderen. Hiervoor is gebruik gemaakt van:

```
static const char symbols[] = "**..11[]22334456AABCCDDEEFFGGHH";
```

Het uitlezen van de conversietabel uit een chromosoom is als volgt geïmplementeerd:

```
void chr2table(char *chromosome, char *convtable)
{       unsigned mask = 0x80, bytepos = 0, c, b, i;
        for (c=0; c<64; c++) {
                for (i=0, b = 0; b<BITLENGTH; b++)
                {       i=i<<1;
                        if (chromosome[bytepos]&mask) i++;
                        mask=mask>>1;
                        if (!mask)
                        {       mask = 0x80;
                                bytepos++;
                        }
                }
                convtable[c] = symbols[i];
        }
}
```

In tegenstelling tot de fitnessfunctie gebruikt bij het parametertest-programma uit hoofdstuk 5 is nu wel het aantal fitnessbepalingen, dat nodig is om een optimaal netwerk te vinden, opgenomen in de fitnessfunctie van het nieuwe parametertest-programma. De fitnessfunctie is als volgt geïmplementeerd:

```
localPop->member[i].fitness = 18000;
rr = 0;
for (test = 0; test<3; test++)
{
      netwerk_programma( &matrixfitness, &fitnesscounts);
      localPop->member[i].fitness += matrixfitness;
      makeresult = 0;
      if (matrixfitness<maxfitness)
      {
            if (test==0) localPop->member[i].fitness -= 18000; /* netwerk niet gevonden      */
            if (test==1) localPop->member[i].fitness -= 12000; /* netwerk 1 keer gevonden   */
            if (test==2)                                       /* netwerk 2 keer gevonden   */
                  makeresult = 1; localPop->member[i].fitness -= 6000;
            test = 5;                                          /* beëindig test             */
      }
      else localPop->member[i].fitness -= fitnesscounts;
}
if (test == 3) makeresult(localPop->member[i].fitness);
```

De functie *netwerk_programma* test de parameterset en de conversietabel die in een file op disk staan. *Matrixfitness* is de fitness die door het netwerk-programma bepaald is voor het gevonden netwerk. *Fitnesscounts* is het aantal fitnessbepalingen dat nodig is om het optimale netwerk te vinden. Het maximaal aantal fitnessbepalingen waarbinnen het optimale netwerk gevonden moet zijn is 4000.

De functie *makeresult* slaat de gevonden parameters en conversietabel op in een apart bestand met de behaalde fitness. Dit bestand bevat uiteindelijk alle parametersets en conversietabellen die drie maal achtereen binnen 4000 fitnessbepalingen een optimaal netwerk opleverden.

De parameterfitnessfunctie wijkt alleen af van de functie die in hoofdstuk 5 wordt gebruikt doordat het aantal netwerkfitnessbepalingen is opgenomen.

8.3 resultaten & discussie

Het parametertest-programma heeft ruim anderhalve maand gedraaid op een HP. In totaal zijn er 6640 parametersets getest (overigens zijn deze niet allemaal verschillend). Het programma is afgebroken omdat er al lange tijd geen verbetering kwam in de maximale fitness van het parametertest-programma.
De beste parameterset (tabel 8.1) en conversietabel (tabel 8.2) die gevonden werden door het parametertest-programma had in totaal maar 3442 netwerkfitnessbepalingen nodig om drie maal achtereen het optimale netwerk te vinden. Ook variaties op deze set blijken goede resultaten te behalen (zie bijlage D).

Deze door het parametertest-programma gevonden combinatie is vervolgens uitgetest met het netwerk-programma op dertig verschillende startpopulaties.

#size	*512*
#nakomeling	*83*
#pmut	*0.0040*
#pcross	*0.25*
#sites	*10*
#pinv	*0.0625*
#steps	*15*
#axiom	*CAC*
#pressure	*11.4*
#chromsize	*1024*

Tabel 8.1. *De door het genetisch algoritme gevonden parameterset*

E	*	3	.	[	.	A	H	D	C	F	2	F	C	5	F
]	]	D	.	A	2	G	5	.	1	E	*	H	F	[	5
.	]	1	H	]	3	2	B	A	1	.	B	G	*	H	2
.	.	1	*	3	*	*	*	B	H	*	2	*	[	6	D

Tabel 8.2. *Conversietabel die door het genetisch algoritme gevonden is*

Daarnaast is de beste combinatie van parameterset (tabel 8.3 en tabel 5.4) en conversietabel (tabel 8.4) die tot nu toe gevonden was eveneens uitgetest op deze dertig startpopulaties. Het aantal fitnessbepalingen dat nodig was om een optimaal netwerk te vinden, is voor beide sets uitgezet in figuur 8.1. De nu gevonden parameterset en conversietabel blijken significant slechter te zijn dan de beste die tot nu toe gevonden is (symmetrietoets van Wilcoxon [Rijken van Olst, 1974].

#size	*512*
#nakomeling	*346*
#pmut	*0.007*
#pcross	*0.625*
#sites	*4*
#pinv	*1.0*
#steps	*16*
#axiom	*CCC*
#pressure	*10.4*
#chromsize	*1024*

Tabel 8.3. *De tot nu toe beste parameterset*

4	4	*	*	[	[	[	[	D	D	3	3	]	]	*	6
.	*	*	*	.	2	2	2	E	E	F	F	]	]	]	]
*	3	3	5	.	A	A	A	G	G	H	H	[	[	]	]
1	1	1	1	.	B	B	B	*	*	[	[	C	C	C	C

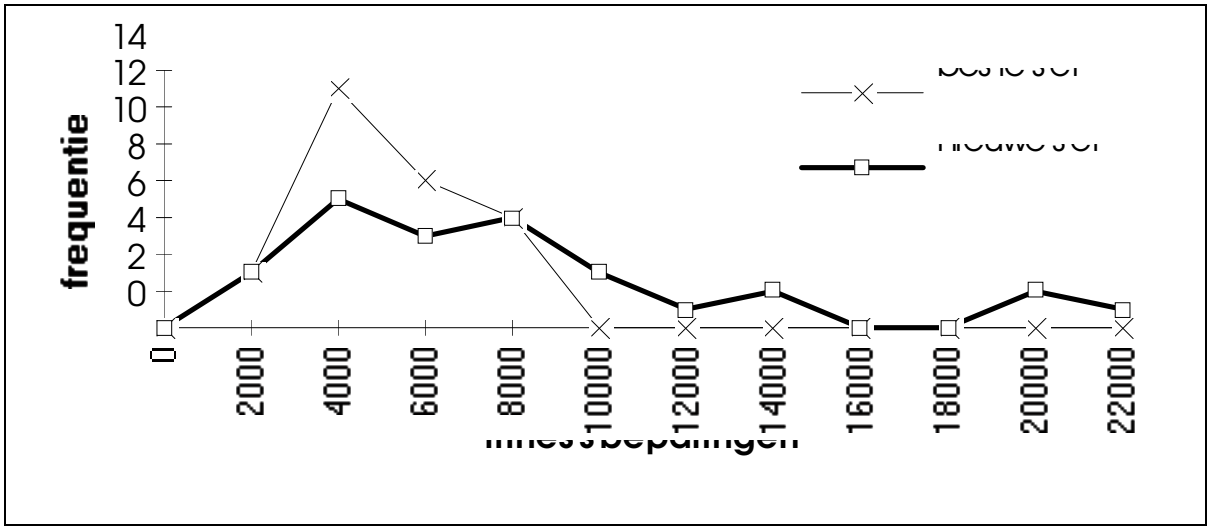
Tabel 8.4. *Tot nu toe beste conversietabel*

Om de oude conversietabel en de nieuwe conversietabel te vergelijken is het aantal verschillende symbolen op een rijtje gezet (tabel 8.5). Opvallend is dat het aantal open en sluithaken afgenomen is. Een gevolg hiervan is dat het aantal subnetwerken zal afnemen.

nieuw	oud
16 cijfers	15 cijfers
3 openhaken	8 openhaken
4 sluithaken	8 sluithaken
24 letters	20 letters
9 asterisken	9 asterisken
8 punten	4 punten

Tabel 8.5. *Vergelijking van het aantal symbolen in de nieuwe en oude tabel*

Het gemiddeld aantal fitness bepalingen dat nodig was voor de dertig testpopulaties bedroeg 7072. De beste set had hier 4230 bepalingen voor nodig. Dat een parameterset driemaal achtereen binnen 4000 fitnessbepalingen een optimaal netwerk vindt is kennelijk geen garantie dat de parameterset dat voor alle mogelijke populaties kan.



Figuur 8.31*. Gedrag van de nieuw gevonden parameterset & conversietabel t.o.v. de beste die tot nu toe gevonden is.*

9. Conclusies & aanbevelingen

9.1. Conclusies

Uit dit onderzoek blijkt dat de parameterset en de conversietabel kunnen worden verbeterd, hetgeen leidt tot een aanzienlijk snellere bepaling van een optimale architectuur voor een neuraal netwerk. Uit de resultaten beschreven in hoofdstuk 7 is gebleken dat niet alleen de snelheid toeneemt, maar dat het benodigde aantal fitnessbepalingen tevens een normale verdeling benadert. Dit heeft als voordeel dat de tijd die nodig is om tot een optimale oplossing te komen beter te voorspellen is. Extreem veel fitnessbepalingen komen immers nauwelijks meer voor.

Opgemerkt moet worden dat er gedurende het hele onderzoek gebruik is gemaakt van een alternatieve fitnessfunktie. Verondersteld is dat de gevonden parameterset en conversietabel tevens een optimaal resultaat zullen geven wanneer de originele fitnessbepaling wordt gebruikt. Dit dient nog nader onderzocht te worden.

9.2. Verder onderzoek

Parameterset
Pressure: Gedurende het onderzoek is gebleken dat de pressure-waarde van grote invloed is op de snelheid waarmee het algoritme convergeert. Het wekt verbazing dat bij het bepalen van de parameterset met behulp van een genetisch algoritme de pressure waarde extreem hoog wordt. In verband hiermee kan het volgende onderzocht worden:

- De pressure zou apart bepaald moeten worden op meerdere startpopulaties. Hierdoor is het beter vast te stellen of er niet teveel vroegtijdige convergenties optreden.
- Een individu wordt uit de populatie geselecteerd met behulp van de functie RankSelect. Deze functie is echter specifiek geschreven voor pressure-waarden tussen 1 en 2. Wanneer de pressure een waarde groter dan 2 heeft, wordt er alleen geselecteerd uit de beste individuen. Het gebruik van de functie BitRankSelect, zoals beschreven door Boers en Kuiper [1992], zorgt dat bij iedere willekeurige pressurewaarde alle individuen uit de populatie een kans hebben om geselecteerd te worden.
- Wanneer bij het bepalen van de parameterset met behulp van een genetisch algoritme de pressure voor het netwerk-generatie-programma constant wordt gehouden is de invloed van de overige parameters beter vast te stellen.
- De pressure hangt samen met het aantal geselecteerden per keer, en dus met het aantal nakomelingen dat per keer geproduceerd wordt. Deze samenhang zou nauwkeuriger onderzocht kunnen worden.

Chromosoomlengte en populatiegrootte: De chromosoomlengte en de populatiegrootte lijken van weinig invloed te zijn op het aantal benodigde fitnessbepalingen. Niettemin is

het voordelig om een zo compact mogelijke representatie te hebben, om de zoekruimte te verkleinen. Het volgende zou hiervoor onderzocht kunnen worden:

- Bepaal de chromosoomlengte en de populatiegrootte eveneens met behulp van het genetisch algoritme.

L-systeem

De conversietabel is van grote invloed op het genereren van een netwerk (zie hoofdstuk 7). Het komt voor dat er voldoende verschillende produktieregels worden gegenereerd, maar dat geen van deze regels toepasbaar is op het gebruikte axioma.

- Om het axioma beter af te stemmen op de produktieregels, zou deze parameter kunnen worden opgenomen in het chromosoom in plaats van het gebruik van een van te voren vastgesteld axioma.
- Laat in de conversietabel maar één letter voorkomen in plaats van de 8 verschillende letters die nu voorkomen. Waarschijnlijk is het dan wel noodzakelijk om de chromosoomlengte te verkleinen of een goed selectiecriterium voor de te kiezen produktieregel te bepalen, omdat het aantal toepasbare produktieregels enorm zal toenemen.

Genetisch algoritme

Selectie: In hoofdstuk 6 zijn een aantal experimenten beschreven om van te voren de beste nakomeling te bepalen. Vroegtijdige convergentie van het genetische algoritme is waarschijnlijk de oorzaak van de tegenvallende resultaten. Nog onderzocht zou kunnen worden:

- Herhaal de experimenten met een lagere pressure-waarde. Op deze manier wordt vroegtijdige convergentie door een te hoge pressure waarde voorkomen.
- Gebruik een nauwkeuriger indicatie van de positie van een produktieregel op het chromosoom. Doordat het chromosoom zes maal gelezen wordt, kunnen produktieregels overlappend op het chromosoom voorkomen. Het is met behulp van het markeringssysteem zoals in dit onderzoek is gebruikt niet uit te maken waar de produktieregels zich precies bevinden, hoeveel produktieregels er in een chromosoom aanwezig zijn en hoe lang de produktieregels zijn. Wanneer dit wel bekend is kunnen de selectiecriteria nauwkeuriger worden opgesteld.
- Gebruik de markeringen op het tweede chromosoom om produktieregels niet door crossing-over, inversie of mutatie te veranderen. Of maak de kans dat dit op de gemarkeerde plaatsen gebeurt kleiner.

Parametertest-programma: Het optimaliseren van de parameters en de conversietabel met behulp van een genetisch algoritme (hoofdstuk 8) was niet succesvol. Er zijn een aantal verbeteringen aan te brengen waardoor dit algoritme mogelijk wel positieve resultaten oplevert:

- Gebruik aging ("veroudering") om individuen die eenmalig een redelijk resultaat opleveren langzaam uit de populatie te verwijderen zonder dat de nakomelingen gaan overheersen in de populatie. Aging kan geïmplementeerd worden door na elke fitnessbepaling de fitness van elk individu te vermenigvuldigen met een getal kleiner dan 1. Om ervoor te zorgen dat de goede individuen niet te vroeg uit de populatie verdwijnen mag deze factor niet te klein zijn. Het tot nu toe

beste individu dient wel apart opgeslagen te worden, om te voorkomen dat deze niet meer te achterhalen is.

- Bepaal regelmatig van alle individuen in de populatie opnieuw de fitness. Individuen die in de populatie zijn gaan overheersen kunnen zo onderdrukt worden wanneer ze toch niet goed blijken te zijn.
- Zet de verschillende parameters die bepaald moeten worden op verschillende genen en laat de volgorde van deze genen bepalen door het genetisch algoritme met behulp van inversie. In de beschreven experimenten is de volgorde van de verschillende parameters in het chromosoom vastgesteld door afhankelijkheid te veronderstellen. Dit is minder nauwkeurig.

Overige

Database:

Boers en Kuiper [1992] hebben een suggestie gedaan om een database samen te stellen waarin verschillende produktieregels zijn opgeslagen. Hiermee is een begin gemaakt door de produktieregels die gebruikt werden om een netwerk te genereren op te slaan in een apart bestand. Doordat het axioma, dat gebruikt werd tijdens het genereren van deze database, bestond uit drie C's, bleek de hele database gevuld met een relatief groot aantal produktieregels bestaande uit C's. Om meer variatie in deze regels te krijgen is het zinnig om het axioma te variëren tijdens het genereren van de database.

BIJLAGE A

Het DNA

Alle informatie voor de ontwikkeling van een organisme bevindt zich in een ketenvormig molekuul, het *DNA*. De bouwstenen van het DNA zijn de *nucleotiden* Er zijn vier verschillende nucleotiden, adenosine (A), guanosine (G), cytidine (C) en thymidine (T). Iedere nucleotide is opgebouwd uit een base die gekoppeld is aan een suiker. Deze zijn twee aan twee complementair; A bindt altijd met T en G met C, dit zijn *baseparen*. Het DNA bestaat feitelijk uit twee strengen waarbij de ene streng het complement van de ander is. Deze twee strengen vormen één geheel door de sterke binding tussen de baseparen. De volgorde van de nucleotiden in het DNA bevat in code de genetische informatie. Deze code wordt omgezet in aminozuren, de bouwstenen van eiwitten.

Voor de aanmaak van eiwitten wordt het DNA niet zelf gebruikt. De eiwit coderende gedeelten van het DNA, de genen, worden eerst gekopieerd en omgezet in *messenger-RNA of mRNA* (*transcriptie*). mRNA molekulen zijn dus kleiner dan DNA maar de bouwstenen zijn nagenoeg gelijk. Alleen het suikerdeel is vervangen door een ander suiker en de base thymine van de nucleotide thymidine is vervangen door uracil (U). Het is de vertaling van de nucleotidenvolgorde (*translatie*) van dit mRNA dat uiteindelijk een eiwit oplevert.

De genetische code

De sleutel voor deze vertaling wordt geleverd door de genetische code. Deze code wijst door het uitlezen van drie opeenvolgende nucleotiden, een *codon*, uit het mRNA één van de twintig verschillende aminozuren aan waaruit een organisme kan kiezen voor zijn eiwit opbouw. Deze code geldt, op enkele kleine afwijkingen na, voor ieder organisme van bacteriën en planten tot mensen. Aangezien er 64 verschillende codons uit vier nucleotiden kunnen worden gevormd, zijn de meeste aminozuren gespecificeerd door meer dan één codon. De volgorde waarin de aminozuren worden uitgelezen, bepaalt het soort eiwitmolekuul dat gevormd wordt. Deze eiwitten vervullen vervolgens allemaal hun eigen specifieke taken, bijvoorbeeld als hormoon of als enzym. Om alle levensfunkties in een organisme goed te laten werken moeten de eiwitten zeer nauwkeurig worden gemaakt, één verkeerd aminozuur in een eiwit kan al zorgen voor meer of minder ernstige

	U	C	A	G	
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	**	**	A
	Leu	Ser	**	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gin	Arg	A
	Leu	Pro	Gin	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val*	Ala	Glu	Gly	G

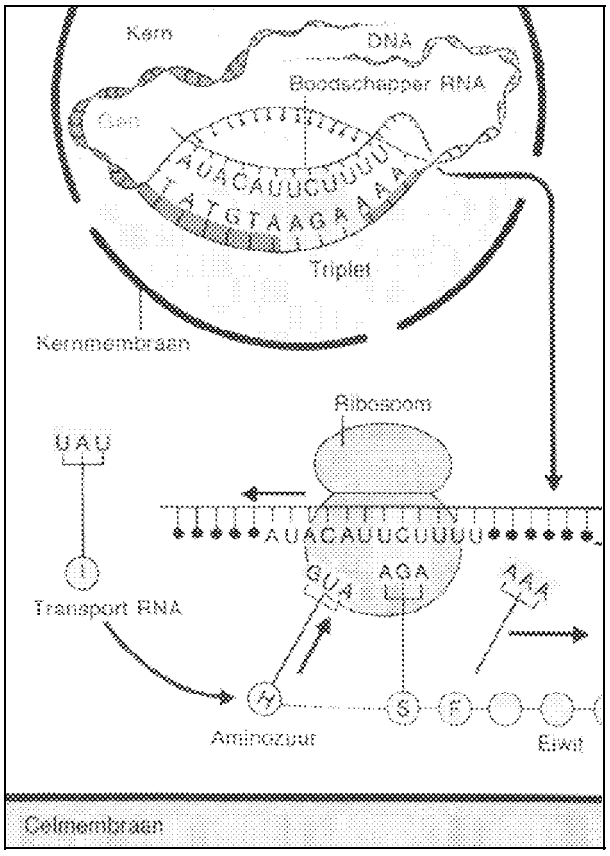
Tabel A.1. *De genetische code*
** ook initiator van de translatie*
*** Stop tekens*

stofwisselingsproblemen. (In de meeste gevallen heeft een fout bij de eiwitsynthese echter nauwelijks gevolgen. Aangezien de vergissingen min of meer willekeurig voorkomen, is energieverlies het enige kwaad dat aangericht wordt).

De eiwitsynthese

De aminozuren lezen niet zelf hun eigen codons. Elk aminozuur is bevestigd aan een speciaal type RNA, *transfer RNA* of tRNA genaamd. Dit tRNA bevat een groepje van drie nucleotiden die complementair zijn aan het codon dat codeert voor het aminozuur dat aan het tRNA is bevestigd, het zogenaamde *anticodon*. De aminozuren worden tijdens de eiwitsynthese één voor één op volgorde langs het mRNA gebracht door de tRNA's. De juiste manier van koppelen wordt volledig bepaald door de specifieke interacties tussen codon en anticodon. Het kritieke punt bij deze synthese zit in het koppelen van de aminozuren aan het tRNA. Het blijkt dat wanneer er kunstmatig een verkeerd aminozuur aan het tRNA wordt gekoppeld dit terecht komt op de plaats waar ook het goede aminozuur terecht zou zijn gekomen. De volgorde van de aminozuren blijkt dus bepaald te worden door de combinatie codon en anticodon.

Het koppelen van tRNA aan aminozuren gebeurt door enzymen de *aminoacyl-tRNA-synthetasen* of vertaalenzymen. Het anticodon van het tRNA moet vertaald worden om het juiste aminozuur aan het tRNA te koppelen. Voor elk aminozuur is een apart synthetase verantwoordelijk. Een enzym werkt gewoonlijk volgens het sleutel-slot principe. De twee vormen die aan elkaar gekoppeld moeten worden, passen precies in elkaar. Wanneer de synthetasen alleen op deze manier zouden werken, zouden er enorm veel fouten gemaakt worden. Onderzoekers hebben uitgerekend dat de fout voor isoleucine en valine, twee sterk op elkaar lijkende aminozuren, gemiddeld twintig procent zou bedragen. Dat wil zeggen dat in de aminozuurketen van een eiwit gemiddeld iedere vijfde isoleucine met valine zou zijn omgewisseld en omgekeerd. Het blijkt echter dat het enzym dat isoleucine moet herkennen slechts een foutenwaarde heeft van één op dertigduizend. Nadat tRNA en een aminozuur aan elkaar zijn gekoppeld, wordt er door het enzym nog tweemaal gecontroleerd of het juiste aminozuur wel aan het tRNA gekoppeld is. Als dit niet het geval is wordt de binding weer losgemaakt en kan er een andere koppeling plaatsvinden.



***Figuur A.1**. Schematische voorstelling van de eiwitsynthese*

RNA editing

Het blijkt niet altijd zo te zijn dat het DNA het RNA rechtstreeks codeert. Bij een eencellige, Trypanosoma brucei, is de nucleotiden volgorde van het RNA behorende bij een bepaald gen niet geheel gelijk aan dat gen. In het RNA bleken vier extra uridylzuur nucleotiden aanwezig te zijn, waardoor het RNA de juiste code bevatte voor een eiwit. Dit proces wordt *RNA editing* genoemd. Ook blijkt dat het verwijderen van nucleotiden een mogelijke verandering van het RNA is. Ook andere organismen blijken RNA te editen. In de meeste gevallen wordt er een nucleotide veranderd in een ander. In de hersenen van de mens bijvoorbeeld wordt de base adenine, in het mRNA, veranderd in guanine voor de glutamaatreceptor. Het tussenvoegen of verwijderen van een nucleotide levert een compleet andere codering op van de codons die achter deze wijziging in het RNA zitten. AUGAUAGAC bijvoorbeeld codeert voor de aminozuren Met, Ile en Asp. Terwijl AU**U**GAUAGAC codeert voor de aminozuren Ile, Asp en Arg. Het *leesraam* is een positie naar links geschoven. In een aantal gevallen codeert hetzelfde stuk DNA meerdere verschillende eiwitten. Het aantal genen kan op deze manier kleiner zijn.

Voor deze appendix is gebruik gemaakt van de volgende literatuur: [Benne en van der Spek, 1993], [Frese en Cramer, 1991] en [Duve, 1987].

BIJLAGE B

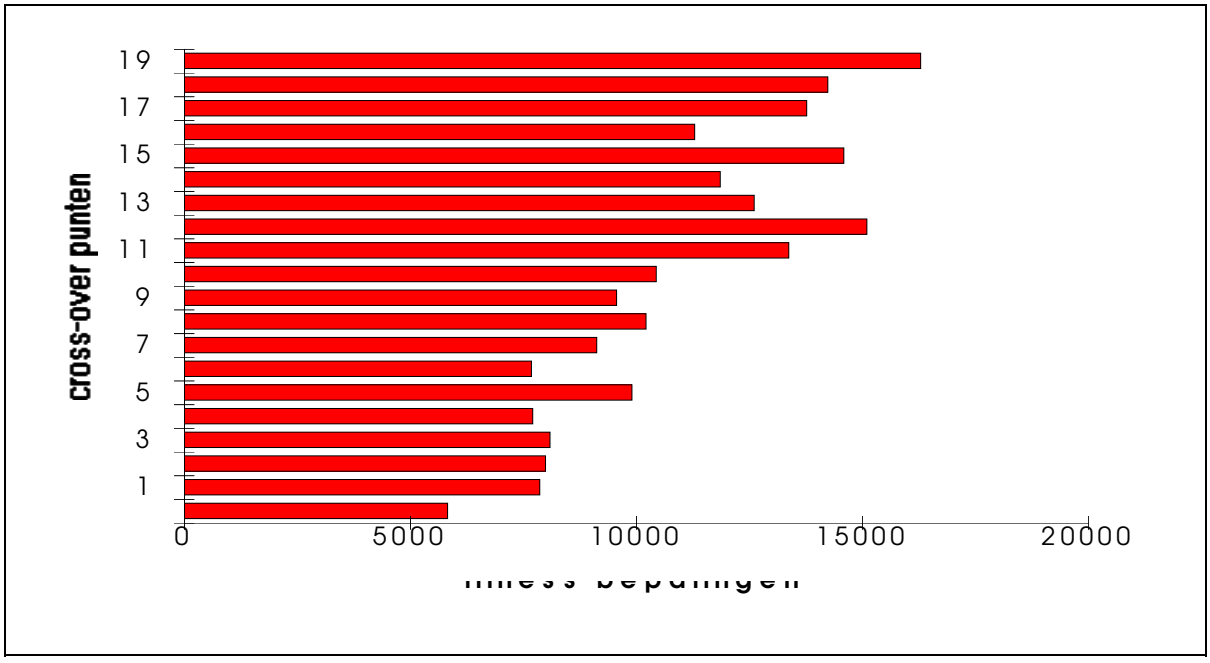
aantal cross-over punten

De invloed van het aantal cross-over punten op het aantal fitnessbepalingen dat nodig is om een optimaal netwerk te vinden is apart onderzocht. Op grond van de resultaten die bereikt waren bij het bepalen van de chromosoomlengte rees het vermoeden dat deze invloed erg groot moest zijn.

Met 10 verschillende startpopulaties zijn netwerken gegenereerd met het aantal cross-over punten oplopend van 1 tot 20. De volgende parameterset afgebeeld in tabel 1 is hiervoor gebruikt.

#nakomeling	20
#size	500
#pmut	0.01
#pcross	1
#sites	variabel
#pinv	0.7
#steps	6
#axiom	A
#pressure	2.1

Tabel B.1. *Ingestelde parameters*



Figuur B.1*. Invloed aantal cross-overpunten op snelheid netwerkgeneratie*

Het blijkt dat een lager aantal cross-over punten in een lager aantal fitnessbepalingen resulteert dan een hoog aantal.

BIJLAGE C

Resultaten behorende bij hoofdstuk 7.

De snelheid waarmee een netwerk gevonden werd is met 7 verschillende conversietabellen uitgetest. Deze conversietabellen A tot en met G staan afgebeeld in hoofdstuk 7. Iedere conversietabel is uitgetest op dertig verschillende startpopulaties. De resultaten van elke test staan afgebeeld in onderstaande tabel.

BIJLAGE D

Resultaten behorende bij hoofdstuk 8.
Resultaten van het gelijktijdig optimaliseren van de conversietabel en de parameters door middel van een genetisch algoritme. Iedere set vond drie maal achtereen het optimale netwerk.

BIJLAGE D