De Creatieve Computer

J.I. van Hemert*
jvhemert@cs.leidenuniv.nl

1 Introductie

Als we de evolutie van computers vluchtig bekijken dan zien we dat de taken die door computers worden uitgevoerd steeds ingewikkelder worden. Met dit uitgangspunt is het heel aardig dat juist evolutie nu een belangrijk concept is geworden bij de vervulling van hedendaagse complexe taken. Hierbij denken we aan het inroosteren van lessen van een gehele universiteit, het analyseren van enorme hoeveelheden financiële data en zelfs het aansturen van een kernreactor.

Er zijn meerdere variaties bedacht van deze evolutionaire probleem oplossers. Stuk voor stuk komen ze voort uit een specifiek probleem domein. Tegenwoordig zijn deze domeinen wat minder van belang. Zodoende is er besloten alles onder één naam te presenteren: *evolutionary computation*. Hier zullen we ons beperken tot de modernste variant, welke veel populariteit heeft gewonnen in de jaren '90. Deze variant, genaamd *genetisch programmeren* is door John Koza bedacht, een student van John Holland, ook wel gezien als de vader van het genetisch algoritme.

Om een gevoel te krijgen wat genetisch programmeren is zullen we eerst een eenvoudig voorbeeld geven. Dit beschrijven we op dezelfde manier als we gewend zijn bij koken, namelijk met een recept. Daarna laten we zien hoe dit recept aangepast kan worden om andere toepassingen te krijgen.

2 Evolutionaire algoritmen

Voordat we een evolutionair algoritme creëren moeten we ons eerst afvragen wat het nut van zo'n algoritme zal zijn. Hierbij moeten we ons realiseren dat een evolutionair algoritme niets meer is dan een stochastische optimalisatie techniek. Dat wil zeggen, gegeven een of andere functie die afhangt van een representatie van het probleem, kunnen we een evolutionair algoritme inzetten om deze functie te minimaliseren of te maximaliseren. Bijvoorbeeld, gegeven het handelsreizigers probleem en de functie die berekend hoe lang een route is, kunnen we een evolutionair algoritme gebruiken om deze route zo kort mogelijk te maken.

Wat heeft een evolutionair algoritme nu met evolutie te maken? Deze vraag blijkt lastig te beantwoorden gezien de vele discussies over dit onderwerp. Er is dan ook niet precies vastgelegd wat minimaal nodig is om te kwalificeren als een evolutionair algoritme. Overeengekomen is in ieder geval dat het een populatie bevat, dus meer dan één kandidaat oplossing. We praten over een kandidaat oplossing zolang we nog niet weten of het daadwerkelijk een oplossing is. Op de populatie worden twee krachten uitgeoefend, enerzijds genetische operatoren om de variatie te verhogen en anderzijds selectie om te convergeren naar een optimum.

Belangrijk zijn ook de concepten fenotype en genotype. Het eerste concept wordt in de biologie gerepresenteerd door de buitenwereld. Aangezien wij een probleem willen oplossen zal ons fenotype bestaan uit een van te voren gedefinieerd probleem. Het tweede concept is in de biologie het genetisch materiaal van een individu. Voor ons is het de representatie van kandidaat oplossingen. De brug tussen de twee wordt gevormd door de fitness. Deze is in de fysieke wereld

onmogelijk exact te berekenen, maar in een evolutionair algoritme is dat meestal geen probleem. Hoewel we ook voorbeelden kunnen bedenken waar dat niet geldt, met name als we een evolutionair algoritme willen inzetten dat een probleem "realtime" in de fysieke wereld moet oplossen. Dan zullen we ons vaak moeten redden met benaderingen.

3 Genetisch programmeren

De belangrijkste eigenschap van genetisch programmeren is de manier waarop het omgaat met kandidaat oplossingen. Deze zijn namelijk *executeerbaar*. Dat wil zeggen dat we een kandidaat oplossing moeten uitvoeren met een zekere invoer alvorens we een daadwerkelijke oplossing krijgen voor deze invoer. Dit is een wezenlijk verschil met andere evolutionaire algoritmen waarbij elke kandidaat oplossing afgestemd is op precies één instantie van een probleem. Een kandidaat oplossing bij genetisch programmeren kan dus voor meerdere instanties van een probleem ingezet worden. Hierbij willen we nog de kanttekening maken dat individuen in de fysieke wereld vaak de mogelijkheid hebben tot leren. Ondanks dat er onderzoeken lopen op dit gebied is het gebruikelijk dat een kandidaat oplossing, eenmaal gecreëerd niet meer veranderd.

Het resultaat wanneer we een genetisch programma hebben gedraaid is een verzameling "executables". Tevens hebben we waarschijnlijk de beste kandidaat oplossing over alle generaties bewaard. Bedenk dat we zo'n oplossing later onafhankelijk weer kunnen (her)gebruiken. Stel bijvoorbeeld dat we met genetisch programmeren een formule voor de oppervlakte van een cirkel hebben gevonden, dan kunnen we die formule gebruiken om de oppervlakte van cirkels te berekenen die we tot nu nog niet hebben gezien.

Natuurlijk is het niet zo eenvoudig om executeerbare kandidaat oplossingen te genereren. De eerste poging hiertoe is dan ook redelijk simpel. Deze bestaat uit een verzameling expressies gerepresenteerd in de computer als bomen. Bomen hebben het voordeel dat ze eenvoudig te evalueren zijn. Er kunnen bijvoorbeeld geen complicaties als oneindige lussen voorkomen. Recente onderzoeken hebben zich gericht op meer gecompliceerdere structuren.

Een hoge drempel voor het succes van genetisch programmeren is de hoeveelheid benodigde ruimte en tijd. Naast de grote hoeveelheid kandidaat oplossingen welke stuk voor stuk een flinke omvang kunnen hebben, moeten we voor elk nieuw individu ook nog eens de fitness berekenen. We hebben hier met executeerbare oplossingen te maken zodat we elk individu op een groot aantal test cases los moeten laten. Deze techniek vereist dan ook grote hoeveelheden computer geheugen, tijd en geduld.

4 Het recept

We zullen hier een korte beschrijving geven van een doorsnee genetisch programma. Belangrijk hierbij is te beseffen dat er nauwelijks een standaard bestaat. Vele varianten en technische trucs zijn geprobeerd. Sommige om een bepaald probleem op te lossen, andere om het algemene karakter van een evolutionair algoritme te verbeteren. Daarom beschrijven we hier een evolutionair algoritme als een recept. Een recept bevat een aantal ingrediënten en een beschrijving die verteld hoe deze gecombineerd moeten worden om tot het eindproduct te komen. Maar een recept hoeft niet precies te zijn, allerlei variaties zijn mogelijk, al was het alleen maar een verandering in de volgorde van het gebruik van de ingrediënten. Wat we hiermee willen zeggen is dat variaties wel eens een beter algoritme zullen opleveren. Experimenteren op dit gebied is zeker aan te raden.

De volgorde in het recept zoals die hier gepresenteerd wordt is bepaald door de voorkeur van de auteur. Het is deze volgorde die we gebruiken bij de daadwerkelijke implementatie. Volkomen anders is de manier waarop een recept wordt ingevuld wanneer we een nieuw probleem aanpakken. Vaak beginnen we met het bedenken van een representatie en een fitness en volgt daarna de rest vanzelf met behulp van literatuur.

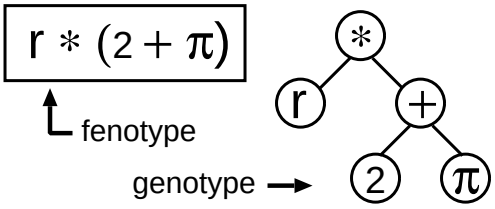
Nu volgt het recept uitgesplitst in ingrediënten en beschrijving. Om het geheel wat levendiger te maken zullen we een recept geven voor een simpel probleem bekend als symbolische regressie. Bij symbolische regressie krijgen we een set punten in een vlak en de opdracht een kromme hier doorheen te trekken. Als voorbeeld gebruiken we de oppervlakte van een cirkel.

De punten bestaan uit paren van de straal en bijbehorende opgemeten oppervlakte. De opdracht is een formule te vinden die voor een willekeurige straal de oppervlakte zo goed mogelijk berekend.

4.1 Ingrediënten

Alvorens we de ingrediënten langs lopen vermelden we dat normaal gesproken de hoeveelheid die we per ingrediënt nodig hebben staat aangegeven. Hier noemen we een hoeveelheid een parameter van het ingrediënt. In de loop der tijd zijn er al veel verschillende parameters gepubliceerd. Vrijwel allemaal zijn ze door mensen bedacht en gekozen uit een kleine verzameling van probeersels. We nemen hier de parameters zoals die tegenwoordig vaak gebruikt worden. Voor de eerste "standaard" keuzes verwijzen we naar het eerste boek van John Koza genaamd "Genetic Programming".

Representatie bestaat uit een boom welke equivalent staat aan een formule (zie Figuur 1). In deze boom zijn wiskundige bewerkingen gerepresenteerd door knopen. De constanten $(2, 3, \pi, 0.42, \ldots)$ en de variabelen (r = straal van cirkel) vormen de bladeren. De maximale diepte van een boom tijdens het programma is meestal 17. Voor de initiële diepte wordt 6 gehanteerd.

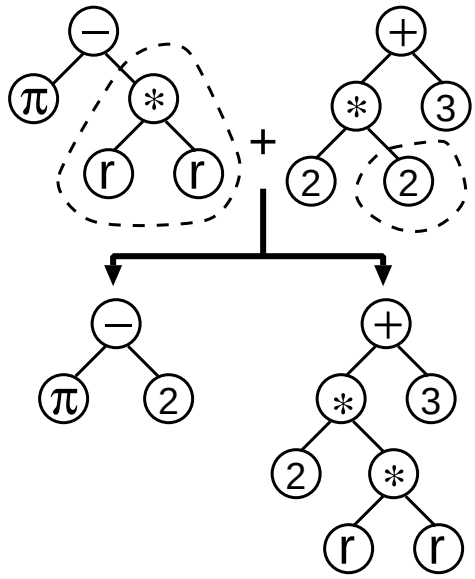


Figuur 1: De formule $r * (2 + \pi)$ gerepresenteerd door een boom ("parse-tree")

Initialisatie van de populatie gebeurd op twee manieren, de helft van de begin populatie wordt gevormd door volle bomen (maximale initiële diepte) en de andere helft door bomen die tot een willekeurige (lagere) diepte groeien. Merk op dat alle knopen en bladeren willekeurig worden gekozen. De populatie grootte van een genetisch programma is aan de hoge kant, meestal 500 of 1000.

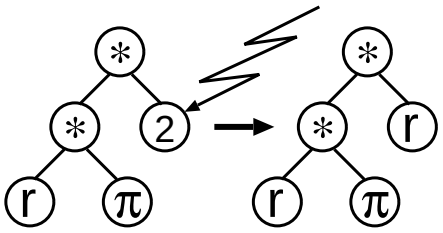
Genetische operatoren

Crossover van twee ouders wordt gedaan door willekeurig een sub-boom in elke ouder aan te wijzen en deze daarna om te wisselen (zie Figuur 2). Meestal is de kans 0.9 dat een interne knoop als wortel van een sub-boom wordt gekozen.



Figuur 2: Een voorbeeld van crossover, twee ouders (boven) genereren twee nieuwe kinderen (onder) door het omwisselen van sub-bomen

Mutatie bestaat in het simpelste geval uit het willekeurig kiezen van een knoop of blad in de tak, waarna deze door een andere knoop dan wel tak wordt vervangen (zie Figuur 3). Andere mutaties worden ook gebruikt, bijvoorbeeld het vervangen van een sub-boom met een nieuwe sub-boom die dan op willekeurige wijze wordt gegenereerd. De mutatie operator werd in het begin vrijwel niet gebruikt.



Figuur 3: Een voorbeeld van mutatie, een knoop uit de boom wordt vervangen door een andere

Fitness is bij genetisch programmeren een tijdrovende zaak. Elke keer dat de fitness van één individu berekend moet worden testen we het individu op een set met metingen. We hebben immers bij genetisch programmeren te maken met executeerbare individuen, zodat we deze dus eerst meerdere malen moeten testen met verschillende invoer waarden. Uit de fouten gemaakt op iedere invoer vormen we een fitness, bijvoorbeeld door de som van de absolute fouten zoals in Tabel 1.

straal	gemeten	individu	fout
1.0000	3.1415	5.1415	2.0000
1.5000	7.0684	7.7123	0.6439
2.0000	12.5660	10.2830	2.2830
2.5000	19.6344	12.8538	6.7806
3.0000	28.2735	15.4245	12.8490
3.5000	38.4834	17.9953	20.4881

Tabel 1: Een test set voor het cirkel probleem met gemeten fouten bij de formule uit Figuur 1, de totale fout is hier 45.0446

Selectie

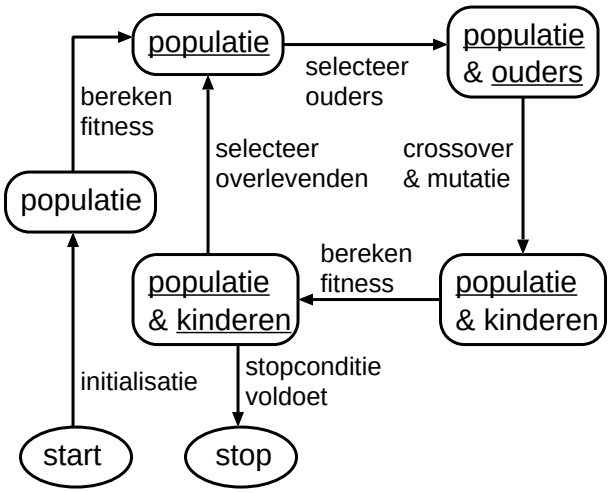
Ouders selectie is nodig voordat we crossover toe kunnen passen. We zullen tenslotte eerst moeten bepalen wie ouders worden. Bij genetisch programmeren wordt vaak "tournament" selectie gebruik. Hierbij selecteren we willekeurig een aantal individuen uit de populatie en laten deze een "wedstrijd" aangaan. Daarna mag het individu met de beste fitness waarde zich ouder noemen. We herhalen deze procedure totdat we genoeg ouders hebben. Er bestaan vele alternatieven, zoals fitness-proportional selectie welke in de begin jaren veel gebruikt werd.

Overlevenden selectie doen we nadat we met behulp van de genetisch operatoren kinderen hebben gemaakt. Dit bepaald hoe de populatie er de volgende generatie uit zal zien. Bij genetisch programmeren wordt er vaak uitgegaan van een zogenaamd "generational model". Hierbij worden alle ouders vervangen door kinderen.

Stopconditie bestaat vaak uit meerdere criteria. De belangrijkste is dat we stoppen als we een oplossing hebben gevonden. In ons voorbeeld betekend dat een perfect model. Dat is lastig te bepalen, want we weten niet van te voren wat het perfecte model is. We zullen stoppen als de fitness nul is, want dan is de fout nul. Helaas kunnen we dan nog modellen hebben die wij als mens niet als perfect aanvaarden. Neem bijvoorbeeld deze formule met een aantal nodeloze bewerkingen: $\pi * r^2 + r * \frac{\pi}{\pi} - r$. Natuurlijk weten we niet zeker of we een optimum zullen vinden dus stoppen we ook na een vooraf bepaalde hoeveelheid generaties, meestal 51.

4.2 Beschrijving

Nu alle ingrediënten klaar staan moeten we ze nog samenvoegen tot een genetisch programma. Dat lijkt in grote lijnen op de meeste evolutionair algoritmen en is het gemakkelijkst te zien in een plaatje. Figuur 4 geeft aan hoe het geheel werkt. Allereerst wordt er een populatie gemaakt met behulp van de initialisatie en van elk individu in deze populatie wordt onmiddellijk de fitness bepaald. Dan begint de evolutionaire kringloop door eerst een lijst met ouders te selecteren en deze lijst vervolgens te onderwerpen aan crossover en mutatie om zo een verzameling kinderen te krijgen. Van deze kinderen moeten we eerst weer de fitness berekenen alvorens we de overlevenden selectie gebruiken om te bepalen wie er door mag gaan naar de volgende generatie. Deze kringloop herhaald zich totdat we tijdens het bepalen van de fitness van de kinderen merken dat we ons doel, de stop conditie, hebben bereikt.



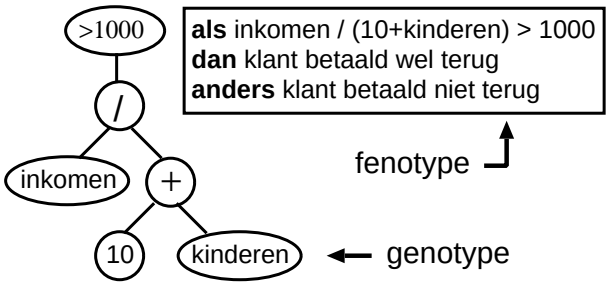
Figuur 4: De evolutionaire kringloop waarbij onderstreepte termen staan voor onderdelen waarvan de fitness bekend is

5 Toepassingen

Nu we een recept hebben voor het beschrijven van een genetisch programma kunnen we heel eenvoudig een aantal toepassingsgebieden de revue laten passeren. Immers, we kunnen volstaan met de wijzigingen in het eerste recept.

5.1 Data classificatie

Deze toepassing komt uit het gebied "data mining". Een veelbelovend gebied nu er veel bedrijven zijn met enorme hoeveelheden data waaruit zij graag informatie willen halen. Bij data classificatie is het de bedoeling een model te creëren dat gegeven een record uit de data kan voorspellen tot welke klasse dit record hoort. Ter verduidelijking nemen we als voorbeeld een bank. Stel dat deze een lijst van klanten heeft met bijbehorende informatie zoals inkomen, schulden, aantal kinderen, gehuwd en dergelijke. Als een klant uit deze lijst om een lening vraagt dan zou het reuze handig zijn om een model te bezitten dat, aan de hand van deze gegevens, zou kunnen voorspellen of deze klant daadwerkelijk de lening gaat terugbetalen. Met andere woorden valt deze klant in de klasse goede klant of in de klasse slechte klant.



Figuur 5: Een voorbeeld hoe een model (rechts) voor data classificatie kan worden gerepresenteerd als een boom (links)

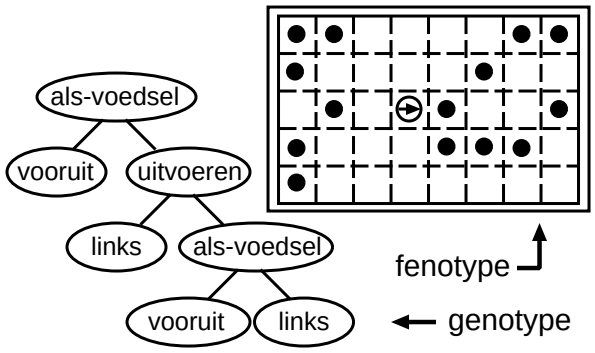
We kunnen genetisch programmeren inzetten om een model voor data classificatie te ontwerpen. Dit doen we net zoals in het symbolische regressie voorbeeld. We nemen bestaande informatie, dat wil zeggen voorbeelden van wel en niet terugbetalen, uit het verleden van de bank en creëren kandidaat oplossingen die er uit zien zoals in Figuur 5. Een kandidaat oplossing is hier een regel die probeert te voorspellen of een klant wel of niet de lening terugbetaald. We hoeven vrijwel geen verdere veranderingen te maken aan het eerste recept, behalve de fitness functie. Deze bestaat hier uit de som van verkeerd geclassificeerde records.

5.2 Besturing van een agent

Bij een autonome agent kunnen we aan veel toepassingen denken. Van stofzuigen tot het besturen van een auto. We zullen hier wat eenvoudiger beginnen met een kunstmatige mier. Deze mier heeft als doel het opeten van zoveel mogelijk voedsel in een kunstmatige ruimte. De ruimte, te zien in Figuur 6, is verdeeld in kaders en elk kader kan één item voedsel bevatten. Zo'n item is herkenbaar als een •.

Om de mier fatsoenlijk te kunnen besturen hebben we informatie nodig over de omgeving. We houden dit simpel door de mier alleen de mogelijkheid te geven te kijken of er voedsel ligt in het vakje recht vooruit. Verder kan de mier een vakje naar voren bewegen en naar links of naar rechts draaien. Zodra de mier in een vakje met voedsel beland wordt het voedsel direct genuttigd. Figuur 6 laat een voorbeeld

zien van een mogelijke besturing van de mier. De knoop `uitvoeren` voert de onderliggende instructies van links naar rechts uit. De knoop `als-voedsel` kijkt of er voedsel aanwezig is in het volgende vakje, zo ja dan wordt de linker sub-boom uitgevoerd en anders de rechter sub-boom.



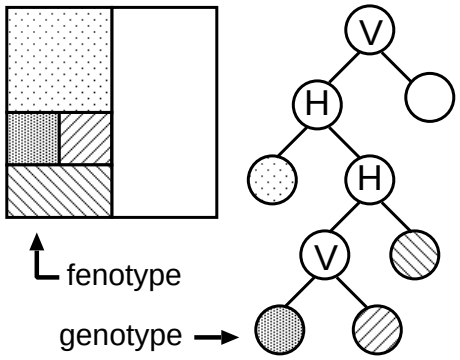
Figuur 6: Een voorbeeld hoe de besturing van een kunstmatige mier in een omgeving met voedsel (rechts) als boom gerepresenteerd kan worden (links)

Nu we een representatie voor de besturing van een mier hebben kunnen we de rest van het eerste recept kopiëren behalve de fitness functie en de stop conditie. De fitness functie definieren we als de hoeveelheid voedsel gegeten door de mier verminderd met de hoeveelheid instructies uitgevoerd. De stop conditie bestaat uit twee delen: we stoppen als al het voedsel op is of na een vaste hoeveelheid uitgevoerde instructies.

Het genetisch programma bestaat dus uit een populatie mieren die om de beurt getest worden in de kunstmatige wereld. Uiteindelijk hopen we een mier te genereren die efficiënt voedsel kan vinden. Hoe simplistisch dit voorbeeld ook lijkt, het is vrij eenvoudig de fitness functie verder uit te breiden om complexere taken te vervullen. Stel we breiden de kunstmatige wereld uit met obstakels, dan kunnen we in de fitness functie opnemen dat we willen vermijden dat onze mier tegen obstakels aanbotst door dit negatief te laten meetellen.

5.3 Mondriaan

Weer iets heel anders, wat meer aan creativiteit doet denken, is het genereren van kunst door de computer. Hier laten we zien hoe we kunstwerken kunnen creëren met een genetisch programma zodat deze lijken op kunst gemaakt door Pieter Cornelis Mondriaan. We beginnen met een beschrijving van de representatie en de vertaling van genotype naar fenotype. Figuur 7 laat zien hoe een kunstwerk in de computer als boom wordt gerepresenteerd. Een `V` staat voor het verticaal opsplitsen van het canvas en een `H` voor het horizontaal opsplitsen. De patronen in de bladeren corresponderen met vlakken in het kunstwerk.

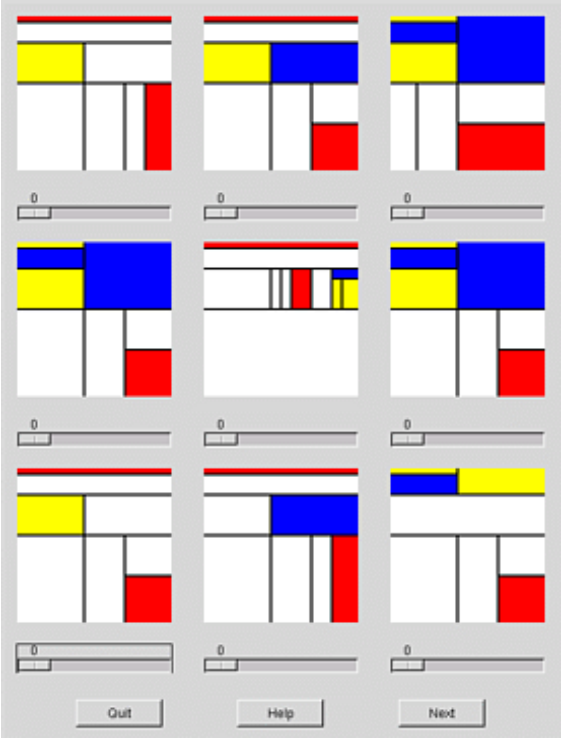


Figuur 7: Een voorbeeld van een imitatie Mondriaan (links) gerepresenteerd als een boom in een computer (rechts)

In principe kunnen we weer het eerste recept kopiëren, met één groot verschil. De fitness functie kunnen we hier niet uitrekenen. Wat is überhaupt de waarde van een schilderij? Dit willen we eigenlijk niet zelf berekenen, maar door een gebruiker laten bepalen. De oplossing is om na elke generatie alle individuen, kunstwerken dus, te presenteren op een computer scherm (zie Figuur 8). Een gebruiker geeft een cijfer aan elk kunstwerk en drukt op "volgende generatie" als hij of zij klaar is. De cijfers kunnen als fitness gebruikt worden bij de selectie methodes uit het eerste recept. Op deze manier vangen we de subjectieve manier waarop mensen naar kunst kijken.

De nauwlettende lezer heeft zijn bedenkingen moeten hebben gehad bij de uiteenzetting van de evoluti-

onaire Mondriaan generator. Bij een strenge inspectie zullen we er al snel achter komen dat we hier niet met een genetisch programma te maken hebben. Immers elk individu correspondeert met één kunstwerk. Vals spel dus, maar waarom? Zoals wij al eerder noemden is voor het meten van de fitness van een individu een serie cases nodig. Als we dat hier ook hadden ingevoerd dan zouden we met elk individu bijvoorbeeld tien kunstwerken kunnen voortbrengen. Stel dat we negen individuen hebben. Dat zou betekenen dat we de gebruiker 90 kunstwerken moeten voorschotelen, dat is nog meer dan een op zondagmiddag in een galerie. Daarbij komt nog eens dat al deze kunstwerken elke generatie opnieuw bekeken moeten worden. Een ondoenlijke zaak dus. Gelukkig maar, want anders zouden we misschien in staat zijn de ultieme Mondriaan "formule" te genereren.



Figuur 8: Een verkleind voorbeeld van het evolutionaire Mondriaan programma

6 Discussie

We hebben laten zien hoe genetisch programmeren ingezet kan worden op een aantal uiteenlopende applicatie gebieden. Hoewel er nog vele andere gebieden bestaan en wellicht bij zullen komen, hopen we dat deze voorbeelden genoeg zijn om aan te geven hoe veelzijdig "evolutionary computation", en in het bijzonder genetisch programmeren, is. Met name het principe van executeerbare oplossingen maakt genetisch programmeren een intrigerende techniek. Wie weet wat er ooit door een genetisch programma nog gecreëerd wordt? Wat dat dan ook moge zijn, het blijft voorlopig de menselijke creativiteit die op zoek gaat naar toepassingen voor deze technieken.

Maar hoe zit het dan met creatieve computers? Een interessante vraag, gezien er nu steeds meer wetenschappers zich op dit onderwerp richten. Wellicht is de vraag wat eenvoudiger te beantwoorden als we deze anders stellen: wat hebben we nodig voor creativiteit? Een aardige antwoord lijkt te zijn: *intelligentie*. Maar net zo goed is creativiteit onmisbaar om van intelligentie te kunnen spreken. De technieken zoals wij die hier hebben besproken zijn nog verre van intelligent, misschien wel omdat zij creativiteit missen. Hoe mooi of goed we een kunstwerk of respectievelijk een oplossing vinden, het is nog steeds een programma dat gedreven door een stochastisch proces en gestuurd door een fitness functie dit kunstwerk of deze oplossing voort heeft gebracht. Zolang deze fitness functie nog steeds door mensen wordt bepaald kunnen we niet spreken van een op zich zelf staand intelligent programma.

Dat computers voorlopig nog geen tekenen van creativiteit afgeven wil niet meteen impliceren dat zij niet in staat zijn taken te doen die voorheen uitsluitend door mensen werden uitgevoerd en waarbij wij een zekere mate van creativiteit verwachten om deze taken uit te kunnen voeren. We moeten hier vooral rekening houden met het feit dat de programmatuur voor het uitvoeren van creatieve taken heel wat creativiteit vereist bij de constructie ervan. Zelfs het gebruik van een programma vereist vaak nog heel wat inzicht en ervaring. Zoals bijvoorbeeld de Mondriaan generator, deze zal alleen met een hoge mate van succes Mondriaan kunst voortbrengen als we het door

iemand laten gebruiken die ook daadwerkelijk kan beslissen hoeveel een kunstwerk op een Mondriaan lijkt.

Natuurlijk zouden we dolgraag creatieve computers aan de gang zien, maar net zoals bij intelligentie zullen we ook hier stuiten op een groot probleem. Hoe weet je wanneer een computer creatief is? We kunnen wellicht een soort Turing test bedenken waarbij een mens de opdracht krijgt te beoordelen of iets door een mens dan wel een creatieve computer is gemaakt. Het blijft de vraag of deze één-persoons subjectieve test de doorslag kan geven. Dit gezegd willen we in ieder geval nog de nadruk leggen op het creatief gebruik maken van deze nieuwe technieken door mensen. Dit is, zeker op de korte termijn, een realistisch doel.

7 Verder leesmateriaal

In deze sectie staan een aantal bronnen voor de lezer die zich verder wil verdiepen in het vakgebied. Zelf raad ik aan eerst even op het Internet te kijken aangezien de meeste wetenschappers tegenwoordig vrijwel al hun werk daar beschikbaar stellen.

7.1 Op papier

An Introduction to Genetic Programming dit boek met drie auteurs, waarvan Wolfgang Banzhaf de bekendste, is een goede inleiding tot het onderwerp. Voorop staan alle verschillende varianten en geteste toepassingen uit de afgelopen tien jaar.

Genetic Programming I,II & III John Koza wordt gezien als de vader van genetisch programmeren. Ondertussen heeft hij drie (dikke) boeken vol geschreven over dit onderwerp. De eerste bevat voornamelijk speelgoed problemen. De tweede een nieuwe techniek genaamd "automatically defined functions" en de derde beschrijft hoe genetisch programmeren kan worden toegepast bij het fabriceren van onder meer elektrische circuits.

Advances in Genetic Programming 1,2 & 3 voorzichtigheid geboden voor de leken, deze serie boeken is bedoeld om verder verdiept te raken in specifieke onderdelen binnen genetisch programmeren, zoals bijvoorbeeld parallellisme en "code bloat".

Genetic Programming and Evolvable Hardware (journal) op het moment van schrijven zijn deel 1 en 2 van het eerste volume gepubliceerd. Deze journal serie is voor diegene die het beste van het nieuwste uit het vakgebied willen volgen.

Evolutionary Design een boek van Peter Bently over design, optimalisatie, creativiteit en kunstmatig leven, dit alles met een sterke geur naar evolutionaire algoritmen. Hoofdstuk 1 van het boek kan van zijn home page gratis gedownload worden.

7.2 Elektronisch

EvoNet's Flying Circus met informatie, demo's en presentaties over het hele gebied van evolutionary computation:
`http://www.liacs.nl/~gusz/Flying_Circus/`

The home page of Genetic Programming Inc. met vrijwel alles over wat er gebeurd op dit gebied van onderzoek:
`http://www.genetic-programming.com/`

Mondriaan Art by Evolution met informatie over het onderwerp, maar ook met de software:
`http://www.liacs.nl/~jvhemert/mondriaan/`

The European Network of Excellence on Evolutionary Computation (EvoNet) biedt veel informatie aan over het vakgebied via verschillende kanalen, zeker aan te bevelen is het blad *Synergy* wat periodiek toegestuurd wordt nadat een gratis individueel lidmaatschap is aangegaan:
`http://evonet.dcs.napier.ac.uk/`

The Genetic Programming Notebook is een groeiende bron van goed gestructureerde informatie over genetisch programmeren, genetische algoritmen en kunstmatige intelligentie.
`http://www.geneticprogramming.com/`