Transparanten bij het vak Inleiding Adaptieve Systemen: Evolutionary Computation. M. Wiering

---

Evolutionary Computation (EC)

Optimalisatie algoritmen geinspireerd door Darwin's evolutie theorie ("natural selection"), ontdekt in de jaren '60 en '70.

Goed in gedistribueerd zoeken in grote ruimtes.

Gebruiken genetische strings (meestal rij van nullen
en enen) als representatie

Evolutionaire algoritmen kunnen gebruikt worden voor:

- Controle taken
- Combinatorische optimalisatie
- Functie optimalisatie

Leerdoelen

- Begrijpen waar optimalisatie problemen om draaien
- Genetische algoritmen begrijpen
- Representatie kunnen bedenken voor bepaald probleem
- Mutatie en Recombinatie operatoren kunnen maken voor bepaalde representatie
- Verschillende selectie strategieën kennen
- Verschil genetische algoritmen en Evolutionaire strategieen begrijpen
- Genetisch programmeren begrijpen
- PIPE begrijpen

Optimalisatie problemen

In een optimalisatie probleem is er sprake van een representatie (oplossings) ruimte S en een fitness (evaluatie) functie.

Het doel is om de oplossing $s_{max} \in S$ te vinden met maximale fitness

$$f(s_{max}) \geq f(s) \; \forall \; s$$

Omdat de representatie ruimte vaak erg groot is, kunnen we deze niet helemaal doorzoeken.

Daarom moeten we gebruik maken van heuristieke zoekalgoritmen

Voorbeelden hiervan zijn: Genetische algoritmen, Tabu search, Local hill-climbing, Simulated Annealing, Ant Colony Systemen

---

Local hillclimbing

Local-hillclimbing werkt als volgt voor het vinden van een oplossing:

- Genereer een oplossing s_0, t = 0
- Herhaal tot stopconditie geldt:
  - s_{new} = verander(s_t)
  - Als $f(s_{new}) > f(s_t)$ then $s_{t+1} = s_{new}$
  - Anders: $s_{t+1} = s_t$
  - t = t +1

De belangrijkste functie is hierbij: verander(s)

Een nadeel van local-hillclimbing is dat deze vaak in een lokaal minimum terecht komt.

Om daar mee om te gaan, wordt het algoritme meestal vanaf verschillende beginpunten gestart.

Genetische algoritmen

Initialiseer: genereer populatie individuen (genetische strings / genotypes)

- Herhaal:

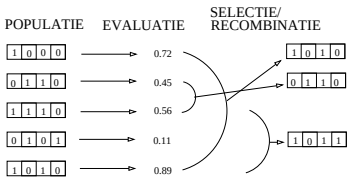
(1) Evalueer alle individuen (bereken fitness)
(2) Selecteer individuen aan de hand van hun fitness
(3) Recombineer geselecteerde individuen m.b.v. crossover en mutatie

Theorie: schemata worden overgedragen aan kinderen.

Stappen voor maken van EA

Hoe maken we een evolutionair algoritme?

Abstracte stappen:



- Ontwerp een representatie
- Initialiseer de populatie
- Ontwerp een manier om een genotype om te zetten naar een phenotype
- Ontwerp een evaluatie (fitness) functie om een individu te evalueren

Meer specifieke stappen:

- Ontwerp bepaalde mutatie operator(en)
- Ontwerp bepaalde recombinatie operator(en)
- Beslis hoe ouders worden geselecteerd
- Beslis hoe individuen worden gestopt in de nieuwe populatie
- Beslis wanneer het algoritme kan stoppen

Ontwerpen van een representatie

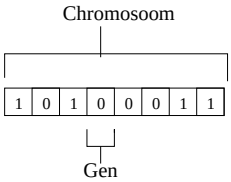
We moeten een manier verzinnen om een individu als een genotype te representeren

Er zijn hiervoor vele mogelijkheden. De manier die we kiezen moet relevant zijn voor het probleem dat we willen oplossen.

Voor het kiezen van een representatie moet er rekening gehouden worden met de evaluatie methode en de genetische operatoren

De representatie kan met discrete waarden (binair, integer etc)

Voorbeeld: binaire representatie :



Omzetten van genotype in phenotype

Als we een representatie voor de genotypes hebben, hebben we ook nog alle vrijheid om deze te vertalen in een phenotype

Het zoeken gebeurt in de genotype representatie ruimte

De phenotype wordt geevalueerd

8 bits Genotype

1	0	1	0	0	0	1	1

Phenotype:
* Integer
* Real number
* Schedule
* .....

Voorbeeld: Phenotype: natuurlijke getallen: uitkomst binaire representatie = 163

Voorbeeld: Phenotype: getal tussen 2.5 en 20.5:

$$x = 2.5 + \frac{163}{256}(20.5 - 2.5) = 13.9609$$

Representeren van reële getallen

Als we een phenotype van reële getallen willen verkrijgen, is een natuurlijke manier om de reële getallen direct te coderen in de genotype

Voorbeeld: een tupel met n reële getallen:
$X = (x_1, x_2, \ldots, x_n) \quad x_i \in \Re$

Een bekende applicatie hiervoor is parameter optimalisatie

De fitness functie f mapt de tupel op een enkel getal:

$$f : \Re^n \rightarrow \Re$$

Representatie voor orderingsprobleem. Voor bepaalde problemen zoals de TSP is het nodig een volgorde van steden te vinden. Dit coderen we als:

3	4	8	6	1	2	7	5

Initialisatie

Voor de intialisatie zijn er de volgende methoden:

- Uniform willekeurig over de gehele zoekruimte:
  1. Binaire strings: 0 en 1 met kans 0.5
  2. Reeële getallen: Uniform op een gegeven interval (werkt niet als interval niet gesloten is)
- Gebruik vorige resultaten om de populatie te initialiseren of gebruik heuristieken
  1. Nadeel: mogelijk verlies van genetische diversiteit
  2. Nadeel: soms onmogelijk om te ontsnappen aan initiële bias.

Evalueren van een individu

Dit kan soms een erg kostbaar proces zijn, zeker voor real-world problems

- Evalueer onveranderde individuen niet opnieuw

Het kan een subroutine zijn, een black-box simulator of een extern proces (bv robots)

Je kunt beginnen met een evaluatie functie welke de uitkomst van het proces benadert, maar dit kan niet te lang (bv. leer met neuraal netwerk hoe fitness landscape eruit ziet).

Omgaan met eisen: wat als phenotype niet aan een bepaalde eis voldoet?

- Gebruik een penalty term in de fitness functie
- Gebruik specifieke evolutionare algoritmen die omgaan met eisen

Probleem: omgaan met meerdere doelen in de fitness functie : gebruik compromissen tussen subdoelen

---

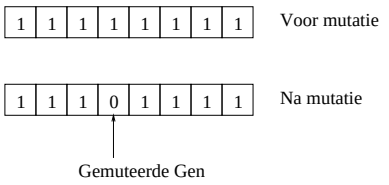
Mutatie operatoren

We kunnen 1 of meerdere mutatie operatoren maken voor een representatie

Belangrijk hierbij is:

- Tenminste 1 mutatie operator moet het mogelijk maken om de hele zoekruimte te kunnen doorzoeken
- De grootte van de mutatie stap moet controleerbaar zijn
- Mutatie moet geldige chromosomen opleveren

Voorbeeld mutatie:



Mutatie gebeurt gewoonlijk met kans P_m op elk gen

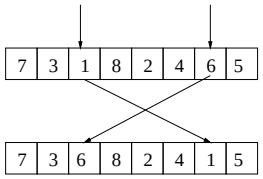
Speciale mutatie operatoren

Mutatie op reële getallen kan gedaan worden door een perturbatie aan te brengen met willekeurige ruis.

Vaak wordt een Gaussiaanse ruis distributie gebruikt (met gemiddelde 0):

$$x_i = x_i + N(0, \sigma)$$

Mutatie voor order specifieke representaties (swap): selecteer 2 genen en wissel ze om:



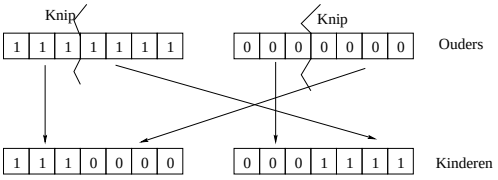
Recombinatie operatoren

Een recombinatie operator mapt gewoonlijk 2 ouders op 1 of 2 kinderen

We kunnen 1 of meer recombinatie operatoren hebben. Belangrijk is:

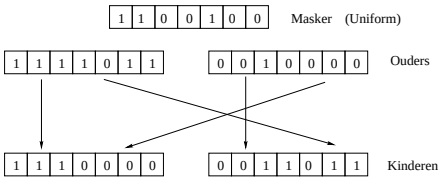
- Het kind moet iets van elke ouder overerven. Anders is het een mutatie operator
- De recombinatie operator moet samen met de representatie ontworpen worden zodat recombinatie niet vaak een catastrofe is.
- Recombinatie moet geldige chromosomen opleveren

Voorbeeld recombinatie (1-point crossover):



Recombinatie maskers

Recombinatie (crossover) kan one-point, two-point, of uniform zijn. We kunnen dit visualiseren als een crossover masker:



Crossover op reële getallen representaties kan gebeuren door getallen te middelen:

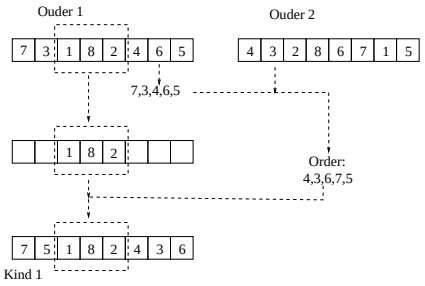
$$(x_1^c = \frac{x_1^a + x_1^b}{2}, \ldots, x_n^c = \frac{x_n^a + x_n^b}{2})$$

Crossover voor order afhankelijke representatie

Hier moeten we oppassen dat we wel geldige chromosomen krijgen

Kies een willekeurig deel van de eerste ouder en copieer dit naar het eerste kind

Copieer de overgebleven genen die niet in het gecopieerde deel zitten naar het eerste kind (gebruik de order van de genen van de 2e ouder)



Selectie Strategie

We willen een manier hebben zodat betere individuen een grotere kans hebben om ouders te zijn dan minder goede individuen

Dit geeft de selectie-druk welke de populatie in staat stelt zich te verbeteren

We moeten minder goede individuen een kleine kans geven zich ook voort te propageren, omdat ze bruikbaar genetisch materiaal kunnen bevatten.

Bv. Fitness proportionele selectie : ouder selectie met kans $f_i / \sum_j f_j$

Verwacht aantal keer dat individu i gekozen wordt als ouder $= f_i / \hat{f}$, met $\hat{f}$ als de gemiddelde populatie-fitness

---

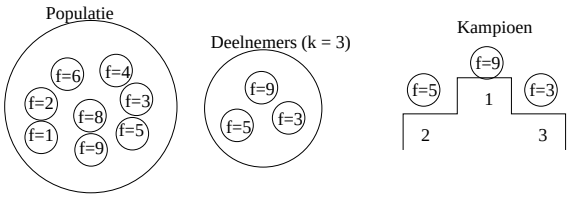
Nadelen fitness proportionele selectie:

- Gevaar van voorbarige convergentie, omdat zeer goed individuen hele populatie snel overnemen
- Lage selectie druk als fitness waarden dicht bij elkaar liggen
- Gedraagt zich verschillend voor translatieve veranderingen van de fitness functie

Oplossing: schalen van de fitness functie: fitness tussen 0 en 1 (beste), som van fitness waarden = 1.

Tournament selectie

Selecteer k random individuen zonder teruglegen



Neem de beste (k is de grootte van het toernooi)

Andere selectie strategie: rank-based selection

Rank-based selection ordent alle individuen op basis van hun fitness. De plaats in deze geordende lijst wordt de rank genoemd.

We gebruiken de rank om een individu te selecteren. Individuen met hoge rank (goede fitness) worden vaker gekozen.

Vervangings strategie

De selectie druk wordt ook beinvloed door de manier waarop individuen uit de vorige generatie gedood worden om plaats te maken voor nieuwe individuen

We kunnen steeds een nieuwe populatie genereren, of een deel van de oude populatie elimineren gebruik makende van hun fitness.

We kunnen bijvoorbeeld de elitist strategie gebruiken en besluiten om het beste individu nooit uit de populatie te verwijderen

In elk geval slaan we het beste individu op in een veilige plaats

---

Recombinatie vs. Mutatie

Recombinatie;

- Veranderingen hangen van hele populatie af
- Afnemend effect met convergentie van populatie
- Exploitatie operator

Mutatie:

- Verplicht om uit lokale minima te ontsnappen
- Exploratie operator

GA vs. ES

GA gebruikt crossover en mutatie

Evolutionaire strategieën (ES) gebruiken enkel mutatie

Keuze hangt af van:

- Is de fitness functie scheidbaar?
- Bestaan er building blocks?
- Is er een semantisch betekenisvolle recombinatie operator?

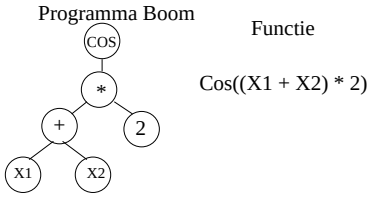
Als recombinatie betekenisvol is → gebruik het!

Genetisch Programmeren

Gebruikt functionele bomen als representatie i.p.v. binaire strings

Functies zoals: cos, sin, *, +, /, random constant

Voorbeeld programma:



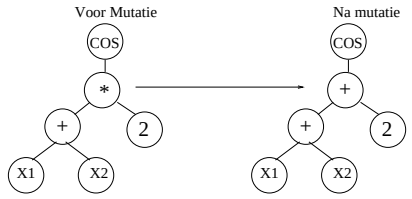
Gebruik:

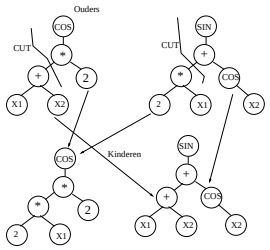
- Supervised leren op continue inputs/outputs
- Robot controle taken
- Beeld classificatie
- Flexibel: loops, geheugen registers, speciale random getallen etc.

Mutatie en Recombinatie in GP

De mutatie operator kan dan een knoop van de boom aanpassen:

Ook de recombinatie operator werkt op programma's:

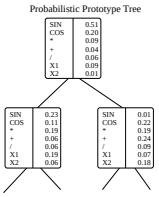




PIPE

Probabilistic Incremental Program Evolution (PIPE) gebruikt kansen om functies op een knoop in programma boom te genereren

Het slaat één probabilistic prototype tree (PPT) op i.p.v. alle individuen:



Met behulp van de PPT worden individuen gemaakt:

- Begin bij wortel en kies hiervoor functie volgens kansverdeling
- Ga naar de subbomen van de PPT om de benodigde argumenten voor de eerder gegenereerde functies te genereren
- Totdat programma helemaal af is

Leerproces in PIPE

Herhaal tot stopconditie geldig:

(1) Genereer op bovenstaande manier een populatie van M individuen

(2) Evalueer alle M individuen

(3) Selecteer de beste en schuif de kansen op zodat de kans dat de gevonden individu opnieuw gegenereerd wordt toeneemt

(4) Muteer de PPT zodat de kansenverdeling licht gemuteerd wordt

PIPE werd vergeleken met GP en bleek in staat voor sommige problemen betere oplossingen te vinden

---

Voorbeeld van taak voor GA

Het Bin Packing probleem is een NP-moeilijk combinatorisch optimalisatie probleem:

Gegeven een lijst van objecten met hun gewicht en een bin met een bepaalde maximale capaciteit.

Doel: deel de objecten zo in de bins dat er een minimum aantal bins nodig is voor het verpakken van alle objecten.

Representatie: lijst van bin-nummer waarin elk object zit:

Voorbeelden: $[1, 2, 1, 3, 1, 2, 1, 3, 1, 2, 1, 3]$
en $[2, 2, 4, 3, 1, 3, 1, 2, 1, 2, 1, 3]$

Fitness functie: hoeveel bins worden gebruikt? Hoe vol zijn gebruikte bins?

Vraag: hoeveel kinderen kunnen er gegenereerd worden door uniform crossover te gebruiken op 2 strings van lengte n?

Voor- en Nadelen EC

Voordelen:

- Evolutionaire algoritmen kunnen vaak direct ingezet worden
- Ze zijn goed in doorzoeken grote zoekruimtes
- Kunnen gebruikt worden voor vele soorten problemen
- Makkelijk te parallelliseren

Nadelen:

- Soms last van voorbarige convergentie: verlies aan genetische diversiteit

- Veel individuen worden geevalueerd en dan niet meer gebruikt
- Kan traag leerproces opleveren
- Als meeste individuen gelijke, slechte fitness hebben is er weinig selectie druk (bv. taak: goed antwoord/ fout antwoord).