Inleiding Computationele Intelligentie - Opdracht 1; 8-koninginnenprobleem

Albert-Jan Yzelman 0216313
anyzelma@students.math.uu.nl

Deze opdracht bestaat uit het bedenken van een evolutionair algoritme om het 8-koninginnen-probleem op te lossen. Het probleem is om 8 koninginnen op een schaakbord te plaatsen zonder dat deze elkaar kunnen slaan. Een algoritme om dit op te lossen zal ik nu gaan bespreken aan de hand van 4 onderdelen; representatie, operatoren, fitnessbepaling en de competitie.

Representatie
Koninginnen kunnen naast diagonaal slaan ook over rijen slaan. Bij voorbaat is het dus uitgesloten dat er koninginnen op dezelfde rij of kolom mogen staan. Met dit in het oog heb ik ervoor gekozen om er een permutatie probleem van te maken, waar ik 8 *verschillende* getallen van 1 t/m 8 achterelkaar opschrijf, bijvoorbeeld 63275148. Dit betekent dan dat ik op de eerste rij een koningin plaats op de zesde kolom, op de 2e rij een op de derde kolom, en op de 3e rij eentje op de tweede kolom, enz;

1	2	3	4	5	6	7	8
					X		
		X					
	X						
						X	
				X			
X							
			X				
							X

<==> 63275148

Operatoren
Bovenstaande representatie maakt het makkelijk om operatoren te maken; een is al snel bedacht: er worden 2 random gekozen getallen omgewisseld, bijvoorbeeld 15423678 > 15823674. Dit is een mutatie. Ik heb gekozen voor precies 1 transpositie, zodat de oplossing niet te sterk muteert; 2 kolommen omwisselen kan in dit probleem heel wat uitmaken.
Een 2e operator is een ‘kruisbestuiving’; ik dacht eerst hiervoor een cyclus-recombinatie te gebruiken, aangezien absolute posities ook hier belangrijk waren. Echter, in mijn nieuwsgierigheid had ik dit algoritme met java geïmplementeerd, en merkte ik dat bijna elke recombinatie volgens deze methode een slechtere fitness had.
Ik heb later dus ook een andere methode gebruikt; een variant op positionele recombinatie. Eerst worden er random (50% kans) getallen uit een 1e reeks geselecteerd, deze worden op dezelfde plaats gekopieerd naar de nieuwe oplossing:

opl 1: 15**24387****6**
opl 2: 51427386
new: --2-38-6

Vervolgens wordt alles uit de 2e oplossing naar de lege plekken in de nieuwe oplossing gekopieerd, zolang het kopiëren maar geen dubbele cijfers tot gevolg had (dat mocht niet):

opl 1: 15243876
opl 2: **51***4273***8**6
new: 512-38-6

In bovenstaand geval mochten de 2 en de 8 uit oplossing 2 niet meegekopieerd worden, aangezien die al op een andere positie in de ‘kruising’ zaten. Deze overgebleven plekken worden gewoon opgevuld van links naar rechts met de ontbrekende getallen:

opl 1: 15243876
opl 2: 51427386
new: 512**4**38**7**6

Op deze manier heeft de nieuwe oplossing heel wat weg van zijn 2 ouders. Deze recombinatie ging naar mijn gevoel wel beter, alhoewel de transposities veel vaker tot verbeteringen leidden. Dit waarschijnlijk toch omdat bij het kruisen oplossing veel resoluter verandert dan de mutatie.

Fitness-functie:
Een fitness functie is op zich niet zo heel moeilijk te vinden; ik heb de volgende gekozen: de functie kijkt ‘van boven naar beneden’ voor elke koningin hoeveel meer koninginnen er zijn op haar diagonalen. Zijn dit er twee, dan wordt er 2 bij opgeteld voor de totale fitness. Hoe meer koninginnen op een diagonaal, hoe hoger de functiewaarde dus. Het algoritme moet dus proberen deze functie te minimaliseren (bij 0 heb je een oplossing). Deze functie is wat simpeler te programmeren dan een fitnessfunctie die voor elke koningin kijkt hoeveel anderen die kan slaan. Daar komt nog bij dat zo’n functie eigenlijk alles dubbel telt, in tegenstelling tot deze.

Competitie:
Ik heb gekozen voor een poule van tien oplossingen, waarvan de vier besten (met de laagste fitness dus) doorgaan naar de volgende ronde. Deze vier worden allen 1 keer gemuteerd. Dit levert dus een nieuwe populatie van acht oplossingen. De overige twee worden geconstrueerd door middel van kruisbestuiving. Het kruisen vindt plaats tussen twee oplossingen uit de vier besten van de vorige populatie. Welke twee er precies met elkaar kruisen wordt random bepaald.

Opmerkingen:
Bij het veelvuldig laten lopen van het programmaatje valt op dat er blijkbaar soms wordt geconvergeerd naar een lokaal minimum, maar helaas is dat er eentje dat nooit 0 wordt; en dus blijft het programma hangen... Dit impliceert dat er weinig variatie is aan oplossingen naarmate er meerdere rondes worden gespeeld. Dit zou waarschijnlijk kunnen worden opgelost door een grotere populatie te nemen, alhoewel het hierdoor wel heel snel makkelijker wordt voor het algoritme een goede oplossing te vinden. Dankzij de gekozen representatie is de oplossingsruimte namelijk erg verkleind geraakt, van 64 mogelijke plaatsen tot alleen maar acht. Het is dus ook niet vreemd om te zien dat zelfs bij een populatie van tien soms nog wel eens in ronde 0 een oplossing wordt gevonden.

Verbeteringen:
Bovenstaande in acht genomen had ik de populatie verdubbeld naar twintig (en dus nu ook 8 winnaars die doorgaan, 8 permutaties en 8 kruisingen). Na een reeks testjes bleek dat het programma inderdaad veel minder vaak bleef hangen in een niet-oplosbare oplossing. Het aantal rondes gemiddeld nodig voordat er een oplossing gevonden is ongeveer 20 (gemiddelde van duizend runs waarbij er niet verder werd gerekend dan honderd rondes (wat waarschijnlijk een ‘hangende’ oplossing was). Wegens de moeilijkheidsgraad van het programmeren was het echter niet meer random bepaald welke beste oplossing met welke andere ging kruisen; dit gebeurde nu gewoon ‘van boven naar beneden’. Al met al lijkt dit wel een bevredigend algoritme.