Evolutionary Algorithms

Geschreven door: Matthijs Koelewijn

Uitleg code

Call tree

main

……Population.__init__

……Individual.__init__

……evolutionary_algorithm

……Population.fitness

……Individual.evaluate_fitness

……Population.get_population

……truncation_selection

……sorted

……reproduce

……swap_mutate

……Individual.copy

……swap_mutate

……Individual.copy

……partially_mapped_crossover

……generate_child

……generate_child

……Individual.evaluate_fitness

Totaal-uitleg

In het hoofdgedeelte van het programma, de `**main**`-functie, begint het evolutionaire algoritme. Het initialiseert de populatie met een opgegeven grootte en affiniteiten. De populatie bestaat uit individuen die elk een set genen hebben, die in dit geval een lijst van 12 elementen is.

Vervolgens wordt het evolutionaire algoritme uitgevoerd in de functie `**evolutionary_algorithm**`. Het begint met het bepalen van de fitnesswaarden van de individuen in de populatie door de methode `**Population.fitness**` aan te roepen. Voor elk individu wordt de fitnesswaarde berekend door de methode `**Individual.evaluate_fitness**` op te roepen en de opgegeven affiniteiten te gebruiken. De individuen worden gesorteerd op basis van hun fitnesswaarden in aflopende volgorde.

Daarna wordt de selectie uitgevoerd met behulp van de functie `**truncation_selection**`. Hier worden de individuen verdeeld in twee groepen: de geselecteerde individuen die de beste fitnesswaarden hebben (een percentage van de totale populatie) en de niet-geselecteerde individuen. Dit wordt gedaan door een bepaald percentage van de individuen te behouden op basis van hun fitnesswaarden.

Vervolgens wordt de reproductie uitgevoerd in de functie `**reproduce**`. De geselecteerde individuen ondergaan mutatie en kruising om nieuwe nakomelingen te genereren. Voor elk paar geselecteerde individuen worden willekeurige genen uitgewisseld met behulp van de `**Individual.swap_mutate**`-methode. Dit introduceert variatie in de populatie en helpt bij het verkennen van verschillende genetische combinaties.

De functie `**partially_mapped_crossover**` wordt gebruikt om de gedeeltelijk gemapte kruising (PMX) uit te voeren tussen de twee ouders. Dit resulteert in twee kinderen met genen die zijn afgeleid van beide ouders. Het genereren van kinderen gebeurt in de functie `**generate_child**`, waarbij een subset van genen van ouder één wordt genomen en ontbrekende genen worden aangevuld met genen van ouder twee.

Deze stappen van selectie, mutatie en kruising worden herhaald voor meerdere paren geselecteerde individuen totdat een nieuwe populatie van nakomelingen is gegenereerd.

De nieuwe populatie bestaat uit de gegenereerde nakomelingen en de geselecteerde individuen (winnaars). Deze nieuwe populatie wordt teruggegeven aan de `**evolutionary_algorithm**`-functie voor verdere iteraties.

Het evolutionaire algoritme wordt herhaald voor een bepaald aantal generaties. Tijdens elke iteratie wordt de beste fitnesswaarde per epoch en de beste fitnesswaarde over alle epochs bijgehouden. Dit wordt gebruikt om de voortgang van het algoritme te controleren en te laten zien welke individuen de beste resultaten behalen.

Aan het einde van de uitvoering worden de beste individuen geïdentificeerd en hun genen, fitnesswaarden en gewichtsproduct

en worden afgedrukt. Daarnaast worden de resultaten opgeslagen in een JSON-bestand met behulp van de `**json**`-module.

Resultaten

Antwoorden van de opdrachten uit de PDF “RiddersVanDeRondeTafel”

Opdracht 1:

a. Bepaal het phenotype voor dit vraagstuk. Beargumenteer je keuze.

Het phenotype is de volgorde waarin de ridders rond de tafel moeten zitten. Het is de concrete oplossing van het probleem, aangezien de gewenste volgorde van ridders de “ruzie” minimaliseert.

b. Bepaal een geschikt genotype voor dit vraagstuk. Beargumenteer je keuze.

Het genotype is in dit geval een reeks van getallen. Bijvoorbeeld als we 5 ridders hebben, kunnen we het genotype als [1, 2, 3, 4, 5] definiëren. Dit betekent dat als ridder 1 op de eerste positie zit, ridder 2 op de tweede positie…

Dit genotype heb ik gekozen omdat het eenvoudig is, en omdat hiermee genetische operatoren zoals mutatie en kruising kunnen worden toegepast. Tenslotte kunnen we met de reeks van getallen de eigenschappen zoals de positie gemakkelijk worden overgeërfd. Dit maakt het mogelijk om de beste eigenschappen te behouden en te optimaliseren.

c. Bepaal een geschikte fitness functie. Beargumenteer je keuze.

Een geschikte fitnessfunctie voor dit vraagstuk zou de fitnesswaarde berekenen op basis van de affiniteiten tussen de ridders in de individuele volgorde. Door de individuele volgorde en zijn positie in de functie mee te geven, kunnen we de affiniteit van het individu met zijn naburige ridders bepalen. De resulterende fitnesswaarde is een getal tussen 0 en 1, waarbij een waarde dichter bij 1 aangeeft dat het individu fitter is.

d. Bedenk geschikte crossover operator(en). Beargumenteer je keuze(s).

Ik heb gekozen voor een uniforme crossover. Deze manier verbreed de zoekruimte, doordat elk geen een gelijke kans heeft om afkomstig te zijn van een van de ouders. Hierdoor wordt genetische diversiteit behouden en kunnen veel verschillende combinaties van genen worden gegenereerd.

e. Bedenk geschikte mutatie operator(en). Beargumenteer je keuze(s).

Een geschikte mutatieoperator is de swap-mutatie. Deze operator selecteert willekeurig twee genen (posities) in de individuele volgorde en wisselt ze onderling om. De swap-mutatie maakt subtiele wijzigingen binnen de lokale zoekruimte mogelijk door het “swappen” van genen. Hierdoor kunnen naburige oplossingen worden verkend en kan de volgorde geleidelijk worden verbeterd. Een belangrijk aspect van de swap-mutatie is dat er geen duplicaten ontstaan doordat de genen onderling worden uitgewisseld. Doordat er geen duplicaten kunnen voorkomen wordt de consistentie van de set van ridders behouden.

Geëvalueerde strategiëen.

Fitness proportional selection / Roulette-wheel selection:

Deze strategie is geschikt wanneer je een goede balans wilt tussen het behouden van de beste individuen en het verkennen van genetische diversiteit. Het geeft individuen met hogere fitnessniveaus een hogere kans om geselecteerd te worden, maar behoudt ook een zekere mate van diversiteit door zwakkere individuen een kans te geven.

Functie roullette_wheel_selection()

Truncation selection / Rank-based selection:

Deze strategie kan nuttig zijn als je snel wilt convergeren naar de beste oplossing. Door alleen de topindividuen te selecteren, elimineer je zwakkere individuen en versnelt dit het convergentieproces. Het kan echter leiden tot een gebrek aan genetische diversiteit en het risico op convergentie naar een lokaal optimum. Het kan nodig zijn om lagere presterende individuen op een later tijdstip opnieuw in te voegen om diversiteit te behouden.

Functie truncation_selection()

Tournament selection:

Deze strategie kan nuttig zijn als je snel wilt convergeren naar de beste oplossing. Door alleen de topindividuen te selecteren, elimineer je zwakkere individuen en versnelt dit het convergentieproces. Het kan echter leiden tot een gebrek aan genetische diversiteit en het risico op convergentie naar een lokaal optimum. Het kan nodig zijn om lagere presterende individuen op een later tijdstip opnieuw in te voegen om diversiteit te behouden.

Functie tournament_selection()

Etilist strategy:

De elitistische strategie wordt meestal gebruikt in combinatie met andere selectiestrategieën. Het kan nuttig zijn wanneer je absoluut wilt voorkomen dat het beste individu verloren gaat. Door een beperkt aantal individuen met de beste fitnesswaarden te selecteren, kun je ervoor zorgen dat de beste oplossingen behouden blijven totdat andere individuen vergelijkbare prestaties bereiken. Het is belangrijk om het aantal elite-individuen te beperken om degeneratie van de populatie te voorkomen.

Functie elitist_strategy()

Truncation selection / Rank-based selection gaf het beste resultaat.

Opdracht 2:

a. Schrijf een genetisch algoritme dat binnen 3 minuten een goede tafelvolgorde oplevert. Start het programma met het op 0 zetten van de random seed

Draai main.py met truncation_selection, partially_mapped_crossover en swap mutation

Resultaat binnen 40 seconden:

```
Progress: [=========>]  97.02%
Progress: [==========]  101.07%
Result run: Best Individual: [9, 11, 7, 10, 8, 3, 2, 1, 5, 6, 0, 4] Fitness: 0.9152166666666667
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project>
```

Single-point crossover, swap mutation en truncation selection

```
Progress: [==========]  100.45%
Result run: Best Individual: [0, 10, 7, 11, 3, 8, 9, 4, 2, 1, 5, 6] Fitness: 0.9022666666666667
```

Partially mapped crossover (PMX), swap mutation en truncation selection

```
Progress: [==========]  100.13%
Result run: Best Individual: [6, 9, 5, 1, 2, 3, 8, 10, 7, 11, 4, 0] Fitness: 0.923925
```

Partially mapped crossover (PMX), swap mutation en tournament selection

```
Progress: [==========]  100.51%
Result run: Best Individual: [10, 0, 8, 3, 2, 4, 11, 7, 6, 9, 5, 1] Fitness: 0.903925
```

Single-point crossover, swap mutation en roulette wheel selection

```
Progress: [=========>]  99.3%
Progress: [==========]  100.55%
Result run: Best Individual: [0, 6, 7, 9, 5, 1, 2, 4, 11, 3, 8, 10] Fitness: 0.9029
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project>
```

Partially mapped crossover (PMX), swap mutation en roulette wheel selection

```
Progress: [=========>]  95.6%
Progress: [=========>]  97.91%
Progress: [==========]  100.24%
Result run: Best Individual: [11, 8, 3, 2, 1, 5, 9, 6, 4, 0, 10, 7] Fitness: 0.9156916666666667
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project>
```

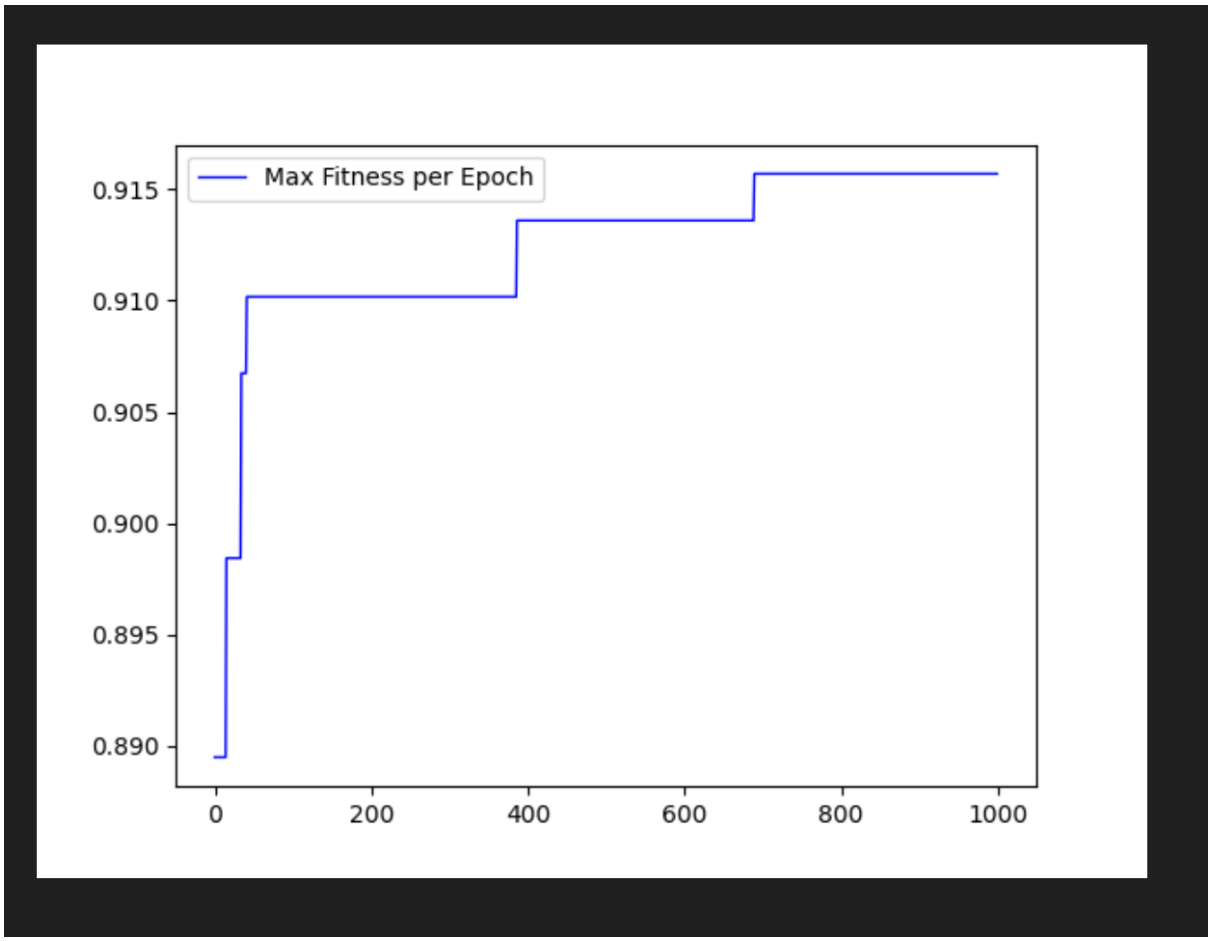
Single-point crossover, swap mutation en elitist selection

```
Progress: [=========>]  96.33%
Progress: [=========>]  98.58%
Progress: [==========]  100.84%
Result run: Best Individual: [0, 10, 7, 11, 3, 8, 9, 4, 2, 1, 5, 6] Fitness: 0.9022666666666667
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project>
```

Partially mapped crossover (PMX), swap mutation en elitist selection

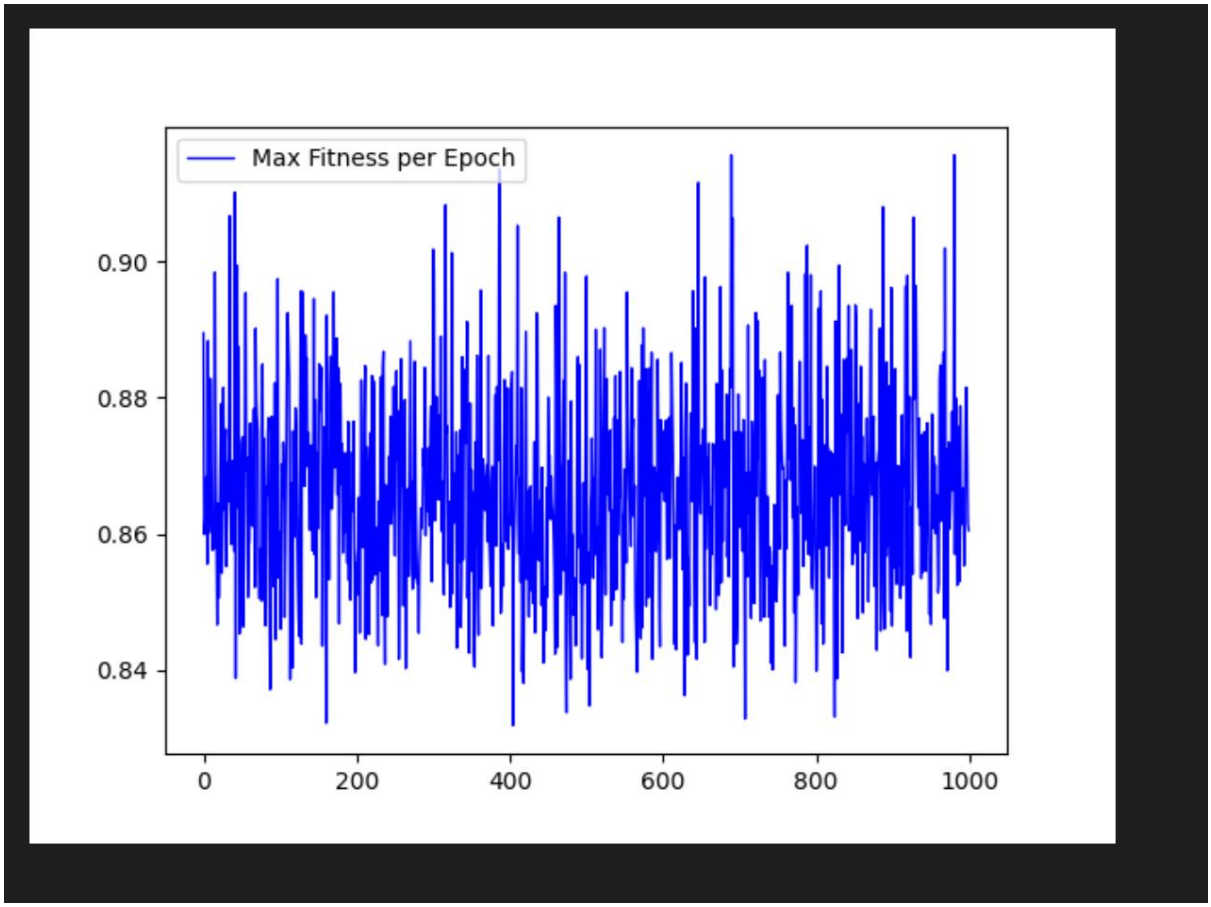
```
Progress: [==========]  101.64%
Result run: Best Individual: [3, 8, 11, 4, 0, 10, 7, 6, 9, 5, 1, 2] Fitness: 0.918875
```

b. Visualiseer na elke epoch de waarde van de beste fitness in de populatie tot dan toe.



Dit resultaat krijgen we bij het doen van truncation selection, PMX en swap mutation

c. Toon achteraf ook een plot, met genoemde waarde uitgezet tegen het epoch-volgnummer.



Dit resultaat krijgen we bij het doen van truncation selection, PMX en swap mutation

d. Toon vervolgens de lijst met de winnende tafelvolgorde van alle tafelgenoten en de bijbehorende gewichtsproducten ertussen. Begin met Arthur. Voorbeeld: "Arthur (0.84x0.5) Lancelot (0.62x0.87) Percival .. etc .. Lamorak (0.6x0.4)"

Zie output.json en onderstaand

```
Individual genes: [8, 0, 10, 7, 6, 11, 4, 9, 5, 1, 2, 3],
Fitness: 0.9067333333333334
Knight: Bedivere, Knight 2: Arthur, weight_product 0.27
Knight: Arthur, Knight 2: Galahad, weight_product 0.087
Knight: Galahad, Knight 2: Kay Sir Gareth, weight_product 0.024300000000000002
Knight: Kay Sir Gareth, Knight 2: Lamorak, weight_product 0.2379
Knight: Lamorak, Knight 2: Tristan, weight_product 0.1368
Knight: Tristan, Knight 2: Percival, weight_product 0.0102
Knight: Percival, Knight 2: Gaheris, weight_product 0.0584
Knight: Gaheris, Knight 2: Bors the Younger, weight_product 0.072
Knight: Bors the Younger, Knight 2: Lancelot, weight_product 0.0275
Knight: Arthur, Knight 2: Lamorak, weight_product 0.24
Knight: Lamorak, Knight 2: Kay Sir Gareth, weight_product 0.2379
Knight: Kay Sir Gareth, Knight 2: Galahad, weight_product 0.024300000000000002
Knight: Galahad, Knight 2: Bedivere, weight_product 0.0504
Knight: Bedivere, Knight 2: Geraint, weight_product 0.0
Knight: Geraint, Knight 2: Tristan, weight_product 0.0096
Knight: Tristan, Knight 2: Gaheris, weight_product 0.08000000000000002
Knight: Gaheris, Knight 2: Bors the Younger, weight_product 0.072
Knight: Bors the Younger, Knight 2: Lancelot, weight_product 0.0275
Knight: Lancelot, Knight 2: Gawain, weight_product 0.19040000000000004
Knight: Gawain, Knight 2: Percival, weight_product 0.019000000000000003
Individual genes: [9, 6, 4, 0, 10, 7, 11, 8, 3, 2, 1, 5],
Fitness: 0.9156916666666667
Knight: Gaheris, Knight 2: Lamorak, weight_product 0.1056
Knight: Lamorak, Knight 2: Percival, weight_product 0.1843
Knight: Percival, Knight 2: Arthur, weight_product 0.0858
Knight: Arthur, Knight 2: Galahad, weight_product 0.087
Knight: Galahad, Knight 2: Kay Sir Gareth, weight_product 0.024300000000000002
Knight: Kay Sir Gareth, Knight 2: Tristan, weight_product 0.102
Knight: Tristan, Knight 2: Bedivere, weight_product 0.1281
Knight: Bedivere, Knight 2: Geraint, weight_product 0.0
Knight: Geraint, Knight 2: Gawain, weight_product 0.0047
Knight: Gawain, Knight 2: Lancelot, weight_product 0.19040000000000004
Knight: Lancelot, Knight 2: Bors the Younger, weight_product 0.0275
Knight: Bors the Younger, Knight 2: Gaheris, weight_product 0.072
['Arthur', 'Lancelot', 'Gawain', 'Geraint', 'Percival', 'Bors the Younger'
```

Dit resultaat krijgen we bij het doen van truncation selection, PMX en swap mutation

Opdracht 3

a. ***Zou gradient descent ook een geschikte methode zijn om dit vraagstuk op te lossen? Waarom wel/niet ?***

Nee, gradient descent is geen geschikte methode voor dit vraagstuk. Gradient descent is een optimalisatie-algoritme dat wordt gebruikt om een lokaal minimum (of maximum) van een functie te vinden door iteratief te zoeken naar de richting van de sterkste afname (of toename) van de functie.

In het geval van het ordenen van ridders rond de tafel met affiniteiten, hebben we te maken met een discrete optimalisatieprobleem waarbij de volgorde van de ridders een permutatie is. Gradient descent is echter ontworpen voor continue optimalisatieproblemen en werkt niet goed met discrete oplossingsruimtes zoals permutaties. Het is moeilijk om gradient descent toe te passen op de volgorde van ridders, omdat kleine incrementele veranderingen in de volgorde niet noodzakelijk leiden tot kleine veranderingen in de fitnesswaarde.

Voor discrete optimalisatieproblemen zoals het ordenen van ridders rond de tafel, zijn evolutionaire algoritmen zoals genetische algoritmen beter geschikt. Ze kunnen de oplossingsruimte efficiënt verkennen door middel van genetische operatoren zoals crossover en mutatie, en ze kunnen oplossingen vinden die voldoen aan de vereisten van het probleem.