2024

Evolutionary Algorithms

IMPLEMENTATIE VAN RIDDERS VAN DE RONDE TAFEL

AHMET SERDAR ÇANAK EN ROEL STIERUM

1. Inleiding

Dit verslag beschrijft de implementatie van het evolutionaire algoritme wat het ridders van de ronde tafel probleem op probeert te lossen. De opdracht is uitgevoerd door Ahmet Serdar Çanak en Roel Stierum in samenwerking. Het bestand is genoemd als "Programming NNs_AhmetSerdarÇ_RoelS.pdf".

2. Voorbeschouwing

A) *Bepaal het fenotype voor dit vraagstuk. Beargumenteer je keuze*

In dit geval is het fenotype de volgorde van stoelen rond de tafel, waar elke stoel overeenkomt met een persoon (Arthur en zijn elf ridders).

B) *Bepaal een geschikt genotype voor dit vraagstuk. Beargumenteer je keuze.*

Voor dit specifieke vraagstuk, waarbij de volgorde van stoelen rond de tafel moet worden bepaald, kan een geschikt genotype een lijst van genen zijn, waarbij elk gen een verwijzing is naar een individu (Arthur of een van zijn ridders).

Een mogelijke representatie van het genotype zou een lijst van numerieke indices kunnen zijn, waar elk nummer overeenkomt met de positie van een persoon (Arthur of een ridder) in de stoelvolgorde.

Bijvoorbeeld: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Hierbij staat 0 voor Arthur, 1 voor de eerste ridder, 2 voor de tweede ridder, enzovoort. Deze lijst kan de genetische code zijn die wordt gebruikt voor het construeren van een individu (stoelvolgorde) in het genetisch algoritme.

C) *Bepaal een geschikte fitness functie. Beargumenteer je keuze.*

(Het antwoord was al gegeven op canvas)

Gebruik de volgende fitness functie: elke plek tussen twee opeenvolgende ridders aan de tafel, zeg tussen ridder A en ridder B, heeft een waarde die gelijk is aan (affiniteit van A naar B) * (affiniteit van B naar A).

De fitness functie is de som van de waarden voor al deze "tussenplekken". Des te hoger die som is, des te beter de tafelzetting is.

D) Bedenkt geschikte crossover operator(en). Beargumenteer je keuze(s).

Order-based crossover is een geschikte keuze voor dit vraagstuk waarbij de volgorde van stoelen rond de tafel moet worden bepaald. Order-based crossover is een genetische operator die elementen behoudt en tegelijkertijd voor diversificatie zorgt. Hier is hoe het kan worden geïmplementeerd:

Ouder 1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] | Ouder 2: [4, 2, 8, 3, 7, 5, 0, 1, 6, 9, 10, 11, 12]
Geselecteerde segment van Ouder 1: [2, 3, 4, 5, 6, 7]
Kind (na eerste crossover): [_, _, 2, 3, 4, 5, 6, 7, _, _, _, _, _]
Vul aan met genen van Ouder 2 (zonder duplicaten): [8, 9, 10, 11, 12, 1, 0]
Kind (na tweede crossover): [8, 9, 2, 3, 4, 5, 6, 7, 10, 11, 12, 1, 0]

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

Swap Mutation is een geschikte keuze voor dit vraagstuk, omdat het past bij de aard van het probleem en bijdraagt aan het behouden van diversiteit en het verkennen van verschillende oplossingen.

3. Uitleg van de code

3.1. Call-Tree

readCSV()
__main__
……..evolutionaryAlgorithm.__init__()
…….individual.__init__()
…….rankPopulation()
……individual.fitness()
……..perf_counter()
……..evolutionaryAlgorithm.train()
…….evolve()
……crossover()
……..mutate()
……..individual.__init__()
…….rankPopulation()
……individual.fitness()
……..perf_counter()
……..winningOrderPrint()
……..plt.show()

3.2. Totaal-uitleg

De '**readCSV()**' functie wordt aangeroepen om de gegeven dataset met affiniteiten en namen van de ridders in te laden en te splitsen in twee lijsten een met namen en een met affiniteiten.

Er wordt een '**evolutionaryAlgorithm**' object gemaakt. Dat object is een populatie opgebouwd uit '**individual**' objecten. Tijdens het initialiseren van de populatie worden de '**individual**' objecten gemaakt deze objecten worden geïnitialiseerd met een willekeurig genotype om mee te beginnen en nadat alle individuen geïnitialiseerd zijn wordt de populatie gerankt (**rankPopulation**) gebaseerd op hun fitness (**individual.fitness**).

Nadat de populatie gecreëerd is hebben we al de beste kandidaat van de huidige generatie maar omdat die nog niet goed genoeg is kunnen we de populatie ook trainen (**evolutionaryAlgorithm.train**) voor x aantal generaties. Het trainingsproces ontwikkelt (**evolve**) de huidige populatie naar een nieuwe populatie door een bepaald aantal van de beste individuen en een bepaald aantal willekeurige individuen te selecteren uit de huidige populatie als ouders voor de nieuwe generatie. Vervolgens worden twee recombinatie-operatoren toegepast: 'order-based crossover' (**crossover**) en mogelijk de 'swap-mutatie' (**mutate**) als aan bepaalde voorwaarden is voldaan, dit proces gaat door totdat de gewenste aantal kinderen gecreëerd is. Vervolgens wordt de populatie weer gerankt en daarna wordt de nieuwe beste kandidaat van de huidige generatie uitgekozen, dit proces gaat door totdat het gewenste aantal generaties is gecreëerd.

Daarna worden de beste kandidaten per generatie gelogd en grafisch weergegeven. Uiteindelijk wordt ook de beste kandidaat over alle generaties gedetailleerd gelogd.

3.3. Reproduceerbare Resultaten

1. **Trainen van het algoritme**

   - Het algoritme is getraind voor 10 generaties (epochs).

   - Het algoritme is getraind met een populatie van 100 individuals.

   - Het algoritme moest 20% van de beste individuen bewaren als parents.

   - Het algoritme moest willekeurig 5% van de individuen bewaren als parents.

   - De gecreëerde kinderen hadden 2% kans op een mutatie (swap mutatie).

2. **Evaluatie van resultaat**

   - Het algoritme voldoet na 3 generaties aan het minimale vereiste van een fitness van >= 6.0 voor 70% van de punten.

   - Het algoritme voldoet na 9 generaties aan het minimale vereiste van een fitness van >= 6.8 voor 100% van de punten.

   - Het algoritme plateaued na 10 generaties op een fitness van ~7.06.

3.4. Conclusie

De implementatie van het evolutionaire algoritme is succesvol uitgevoerd. Het algoritme creëert een individu met een bevredigende fitness na 10 generaties bij een population size van 100 en een mutatie kans van 2% en dit allemaal in 0.03 seconden.

4. Nabeschouwing

In het geval van een sorteerprobleem, waarbij je ridders aan een tafel sorteert op basis van affiniteit, is gradient descent niet de meest geschikte methode, omdat het geen gradueel dalende fitness functie heeft met een absoluut maximum. De individuele oplossingen hebben namelijk grillige waardes.