Algoritmul de evoluție diferențială

Studenți: Dorneanu Răzvan-Sebastian

Enache Maria

Moisoiu Andreea-Nicola

Link GitHub: https://github.com/Mariaa023/PROIECT_IA.git

1. Descrierea problemei considerate

În acest proiect sunt abordate două probleme reale de optimizare cu reprezentare reală, pentru care nu există o formulă analitică simplă de rezolvare. Ambele probleme sunt tratate independent, fiecare având o funcție de fitness numerică proprie.

1.1 Problema programului de somn

Prima problemă urmărește determinarea unui program de somn optim, definit prin ora de culcare și ora de trezire. Un program de somn echilibrat este esențial pentru sănătate și performanță, iar stabilirea acestuia reprezintă o problemă de optimizare continuă.

Obiectivele principale sunt obținerea unei durate de aproximativ 8 ore de somn, evitarea culcării după miezul nopții și apropierea de o oră rezonabilă de trezire.

1.2 Problema meniului zilnic

A doua problemă constă în optimizarea unui meniu zilnic din punct de vedere nutrițional. Soluția este definită prin cantitățile de proteine, carbohidrați și grăsimi consumate într-o zi.

Se urmărește determinarea unui meniu zilnic echilibrat din punct de vedere nutrițional, cu un aport caloric apropiat de 2000 kcal.

2. Aspecte teoretice privind algoritmul de evoluție diferențială

Evoluția diferențială (differential evolution, DE) este o euristică de optimizare asemănătoare cu un algoritm evolutiv classic. Caracteristică este operația de generare a noilor cromozomi, care implică adăugarea diferenței dintre doi cromozomi la al treilea și compararea cu al patrulea. De asemenea, are mutație, încrucișare și selecție.

Inițializarea populației

În prima etapă, se generează o populație inițială formată dintr-un număr fix de indivizi. Fiecare componentă a unui individ este generată aleator și trebuie să acopere cât mai uniform spațiul parametrilor. Astfel, se asigură diversitatea inițială a soluțiilor.

Mutația diferențială (rand/1)

Pentru fiecare individ din populație, se selectează aleator trei indivizi diferiți între ei și diferiți de individul curent. Pe baza acestora se construiește un vector mutant, folosind relația:

vi= xr1+F*(xr2-xr3)

unde:

- xr1, xr2, xr3 sunt indivizi aleși aleator;
- F este factorul de mutație, care controlează amplificarea variației diferențelor.

Încrucișarea binomială

Vectorul mutant este combinat cu individul curent printr-o încrucișare binomială. Pentru fiecare componentă a vectorului, se decide cu o probabilitate CR (rata de încrucișare) dacă valoarea provine din vectorul mutant sau din individul original. Pentru a evita obținerea unui individ identic cu cel inițial, se impune selectarea obligatorie a cel puțin unei componente din vectorul mutant.

Rezultatul acestei etape este candidat pentru a înlocui individul curent.

Selecția

În etapa de selecție, individul este evaluat folosind funcția de fitness. Dacă fitness-ul individului este mai bun (mai mic, în cazul unei probleme de minimizare) decât fitness-ul individului curent, acesta îl înlocuiește în populația următoare. În caz contrar, individul curent este păstrat.

Această strategie de selecție este una elitistă, asigurând faptul că populația nu se degradează de la o generație la alta.

Observații finale

Algoritmul de evoluție diferențială are o implementare simplă și o bună eficiență pentru optimizarea problemelor reale. Separarea clară între algoritm și definirea problemei permite reutilizarea implementării pentru mai multe aplicații, așa cum este demonstrat în cadrul acestui proiect.

3. Modalitatea de rezolvare

Algoritmul este implementat într-o formă generală, astfel încât să poată fi reutilizat pentru orice problemă de optimizare cu reprezentare reală. Acesta primește ca parametri funcția de fitness și limitele variabilelor.

Aceeași implementare a algoritmului de evoluție diferențială este utilizată atât pentru problema programului de somn, cât și pentru problema meniului zilnic.

Pentru fiecare problemă sunt definite separat funcția de fitness și limitele variabilelor, iar algoritmul caută soluția care minimizează valoarea fitness-ului.

4. Listarea părților semnificative din codul sursă

4.1 Inițializarea populației

```
for i in range(nr_cromozomi):
    individ = []
    for j in range(dim):
        val = random.uniform(limite[j][0], limite[j][1])
        individ.append(val)
    populatie.append(individ)
```

Populația inițială este generată aleator în limitele specificate pentru fiecare variabilă. Fiecare individ reprezintă o soluție posibilă a problemei.

4.2 Mutația diferențială

```
val = x1[j] + factor_f * (x2[j] - x3[j])
```

Mutația diferențială este realizată prin adăugarea unei diferențe scalate dintre doi indivizi la un al treilea individ ales aleator.

4.3 Încrucișarea și selecția

```
if random.random() < cr or j == poz_oblig:
    individ_nou.append(vector_mutant[j])
else:
    individ_nou.append(populatie[i][j])
```

Încrucișarea binomială combină vectorul mutant cu individul curent, iar selecția păstrează soluția cu fitness mai bun.

4.4 Funcția de fitness pentru programul de somn

```
if ora_culcare > ora_trezire:
    durata_somn = ora_trezire + 24 - ora_culcare
```

else:
    durata_somn = ora_trezire - ora_culcare

Durata somnului este calculată ținând cont de posibilitatea trecerii peste miezul nopții, asigurând astfel o evaluare corectă a programului de somn.

penalizare_durata = abs(durata_somn - 8)
penalizare_trezire = abs(ora_trezire - 8)

Aceste penalizări calculează abaterea față de durata ideală de somn și față de ora ideală de trezire.

4.5 Funcția de fitness pentru meniul zilnic

calorii = proteine * 4 + carbohidrati * 4 + grasimi * 9

Aportul caloric este calculat pe baza valorilor nutriționale standard, utilizate în mod curent în domeniul nutriției.

penalizare_calorii = abs(calorii - 2000) / 100

penalizare_proteine = abs(proteine - 150) / 50

penalizare_carbohidrati = abs(carbohidrati - 250) / 50

penalizare_grasimi = abs(grasimi - 70) / 20

Aceste penalizări calculează abaterile față de aportul caloric dorit și față de valorile de referință ale componentelor meniului, contribuind la evaluarea calității soluției obținute.

4.6 Rularea algoritmului și testarea manuală

solutie, fitness = evolutie_diferentiala(fitness_somn, limite_somn)

Algoritmul de evoluție diferențială este apelat din fișierul principal pentru a obține soluția optimă a problemei.

5. Rezultatele obținute și analiza acestora

Rularea programului pentru problema somnului conduce la soluții cu fitness foarte mic, indicând un program de somn aproape optim.

```
Problema somn:
Ora culcare: 23:59
Ora trezire: 08:00
Fitness: 1.4475302734240358e-08
```

Testarea manuală, realizată prin introducerea unor valori diferite de cele optime, produce valori mai mari ale fitness-ului, ceea ce confirmă corectitudinea funcției de penalizare.

```
Test manual program somn
Introdu ora de culcare:22
Introdu ora de trezire:7
Ora culcare: 22:00
Ora trezire: 07:00
Fitness (penalizare): 2.0
```

Pentru problema meniului zilnic, algoritmul identifică o soluție de compromis, cu un fitness relativ mic.

```
PROBLEMA MENIU
Proteine: 120
Carbohidrati: 223
Grasimi: 70
Fitness: 1.150007783865194
```

În testele manuale, valorile introduse conduc la un aport caloric diferit de cel dorit, iar fitness-ul obținut este semnificativ mai mare, demonstrând faptul că soluțiile neechilibrate sunt penalizate corespunzător.

```
TEST MANUAL MENIU ZILNIC
Introdu proteine(g):22
Introdu carbohidrati(g):333
Introdu grasimi(g):123
Proteine: 22
Carbohidrati: 333
Grasimi: 123
Calorii totale: 2527
Fitness(penalizare): 12.14
```

8. Contribuția membrilor echipei

- **Maria**
  - implementarea algoritmului de evoluție diferențială în varianta standard rand/1;
  - realizarea structurii generale a algoritmului (inițializare, mutație diferențială, încrucișare binomială și selecție);
  - definirea și explicarea parametrilor algoritmului;
  - redactarea părții teoretice din documentație (descrierea algoritmului, principiul de funcționare și pașii algoritmului);
  - contribuție la redactarea concluziilor generale ale proiectului.
- **Nicola**
  - modelarea problemei programului de somn;
  - definirea reprezentării soluției și a limitelor variabilelor pentru problema somnului;
  - implementarea funcției de fitness pentru programul de somn;
  - realizarea funcțiilor de testare manuală și afișare a rezultatelor pentru somn;
  - redactarea secțiunii din documentație aferente problemei programului de somn;
  - descrierea testelor efectuate și interpretarea rezultatelor pentru această problemă.
- **Sebi**
  - modelarea problemei meniului zilnic;
  - definirea funcției de fitness și a limitelor variabilelor pentru problema meniului;
  - implementarea testării manuale pentru problema meniului;
  - analiza și interpretarea rezultatelor obținute pentru meniul zilnic;
  - redactarea secțiunii din documentație aferente problemei meniului zilnic;
  - contribuție la completarea bibliografiei.

6. Concluzii

În cadrul acestui proiect a fost implementat algoritmul de evoluție diferențială în varianta standard, conform materialelor prezentate la curs, și aplicat pe două probleme reale de optimizare cu reprezentare continuă. Rezultatele obținute demonstrează că algoritmul este capabil să identifice soluții cu fitness redus, apropiate de valorile optime definite prin funcțiile de fitness.

Pentru problema programului de somn, algoritmul a convergent rapid către o soluție aproape ideală, cu fitness foarte mic, ceea ce indică o bună definire a funcției de fitness și

eficiența metodei de optimizare. În cazul problemei meniului zilnic, unde există mai multe obiective simultane, algoritmul a furnizat soluții de compromis echilibrate.

Implementarea realizată este generală și modulară, separând clar algoritmul de definiția problemelor. Această abordare permite reutilizarea algoritmului pentru alte probleme reale de optimizare continuă, prin simpla definire a unei noi funcții de fitness și a limitelor corespunzătoare. Astfel, algoritmul de evoluție diferențială se dovedește a fi o metodă eficientă, flexibilă și ușor de adaptat.

7. Bibliografie

1. Metode de optimizare

2. Documentația oficială Python – https://docs.python.org

3. Inteligenta artificiala - Cursuri