# Documentatie Proiect Inteligenta Artificiala

### Optimizarea roi de particule - Implementrea functiei Ackley

Cuprins

1. Descrierea problemei considerate

2.Aspecte teoretice privind algoritmul

3.Modalitatea de rezolvare

4.Concluzii

5.Bibliografie

Studenti: Amariei Marinel

Blejusca Oana

Grupa: 1409A

1.Descrierea problemei considerate

Echipa noastra, formata din Amariei Marinel si Blejusca Oana (gr. 1409A), a ales sa prezinte o implementare in Java a algoritmului Particle Swarm Optimization. Implementarea va expune un API pentru a fi apelat de interfata din ReactJS. Am ales sa exemplificam functia Ackley deoarece reprezinta o provocare pentru orice algoritm de optimizare, prin prisma multiplelor minime locale ce pot fi un obstacol.

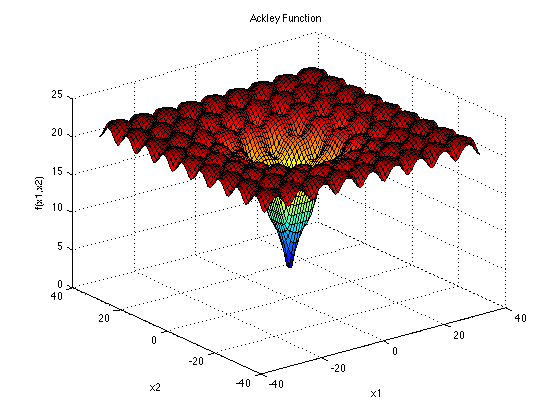


Fig.1.1 Reprezentare functie Ackley

Functia Ackley este folosita in general pentru testarea algoritmilor de optimizare. In forma bidimensionala din Figura 1.1, este caracterizata printr-o regiune exterioara aproape plata si o portiune de valori scazute in centru. Aceasta functie prezinta un risc pentru algoritmii de optimizare, deoarece acestia pot fi ramane blocati in una din multele minime locale.

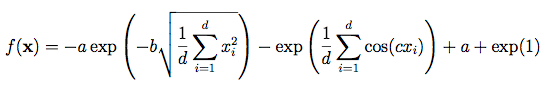


Fig.1.2 Formula functie Ackley

Parametrii functiei Ackley, a carei formula se gaseste in Figura 1.2, sunt:

* Dimensiune - d
* a - Valoare recomandata = 20
* b - Valoare recomandata = 0.2
* c - Valoare recomandata = 2 \* pi

Functia este evaluata pe hipercubul [-32.768, 32.768], pentru fiecare x[i], cu i = 1,d, insa poate fi restrictionata la domenii mai mici in jurul originii.

Minimul global se gaseste in originea hipercubului pentru fiecare argument x[i], cu i=1,d, nul.

2.Aspecte teoretice ale algoritmului

Algoritmul de optimizare roi de particule(eng. Particle Swarm Optimization - PSO) a fost dezvoltat in 1995 de catre James Kennedy si Russell Eberhart si se inspira din comportamentul organismelor sociale ce traiesc in grupuri. PSO emuleaza interactiunea intre membrii grupului pentru a impartasi informatii.

Parametrii initiali ai algoritmului:

* numarul de particule
* numarul maxim de iteratii
* valoarea maxima a vitezei
* inertia particulelor - notat w
* acceleratia cognitiva - notat c1
* acceleratia sociala - notat c2
* numere aleatorii in intervalul (0, 1) - r1, r2

Fiecare particula este influentata de pozitia cea mai buna atinsa de aceasta si de cea mai buna pozitie atinsa de intreg grupul.

Fiecare particula are mai multi parametri:

* pozitia curenta
* viteza curenta
* cea mai buna pozitie personala
* cea mai buna pozitie a vecinatatii

In faza intiala fiecare particula are valori aleatorii pentru pozitie si viteza. Pe parcursul unei iteratii se evalueaza functia obiectiv data de pozitia curenta a particulei, se actualizeaza optimul personal, daca este cazul unui cost mai bun, si se recalculeaza optimul vecinatatii.

Pentru fiecare dimensiune se actualizeaza viteza insumand urmatoarele componente:

* pozitia curenta inmultita cu inertia (w)
* diferenta dintre cea mai buna pozitie personala si pozitia curenta, care se inmulteste cu acceleratia cognitiva (c1)
* diferenta dintre cea mai buna pozitie a vecinatatii si pozitia curenta, care se inmulteste cu acceleratia sociala (c2)

Algoritmul ajunge la convergenta cand se atinge un numar prestabilit de iteratii sau diferenta de la o iteratie la urmatoarea nesemnificativa.

3.Modalitatea de rezolvare

Am ales sa implementam algoritmul intr-o aplicatie Java cu interfata in framework-ul ReactJS. In Java, algoritmul PSO a fost implementat prin urmatoarele clase:

1. Clasa Particle (particula) cu membrii:

* position (pozitie) - vector de numere in reprezentate in dubla precizie ce reprezinta valorile curente ale argumentelor functiei obiectiv pentru particula respectiva
* cost - numar in dubla precizie reprezentand valoarea functiei pentru particula curenta
* velocity (viteza) - viteza particulei la iteratia curenta
* bestPersonalPosition (optimul personal) - reprezinta pozitia avuta de particula curenta care a avut cel mai bun cost pana acum

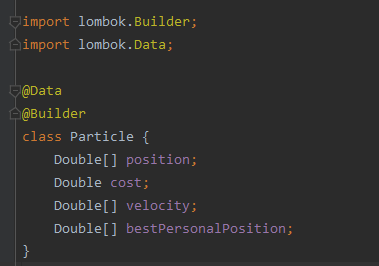


Fig. 3.1 - Cod clasa Particle

2. Clasa Parameters (parametri) cu urmatorii membri:

* particlesNo (numar particule) - numar intreg reprezentand dimensiunea populatiei de particule
* maxVelocity (viteza maxima) - numar reprezentat in dubla precizie ce simbolizeaza viteza maxima pe care o poate atinge o particula
* inertia - valoare reprezentata in dubla precizie ce reprezinta influenta pe care o are viteza curenta asupra vitezei de la urmatoarea iteratie
* c1, c2 - parametri ce reprezinta coeficientul de acceleratie cognitiva, respectiv coeficientul de acceleratie sociala
* maxIterations (numarul maxim de iteratii) - reprezinta criteriul de convergenta

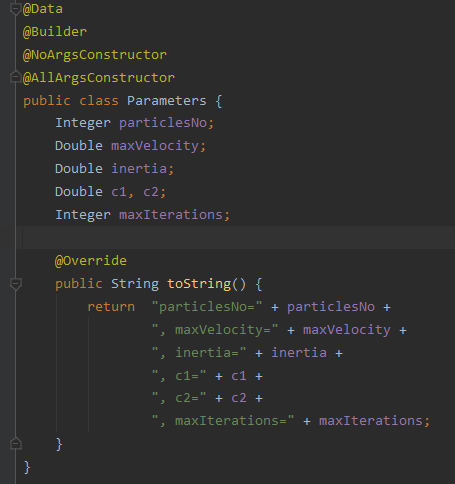


Fig. 3.2 - Cod clasa Parameters

3. Clasa Problem (problema) cu membrii:

* objectiveFunction (functia obiectiv) - instanta a functiei ce trebuie sa fie minimizata
* parametersDomain (domeniul parametrilor) - lista de valori reprezentate in dubla precizie care reprezinta minimul, respectiv maximul fiecarui argument al functiei

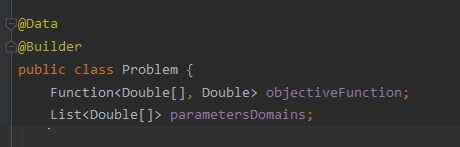


Fig. 3.3 - Cod clasa Problem

4. Clasa CurrentState (starea curenta) este clasa care se modifica de la o iteratie la urmatoarea, si are urmatorii membri:

* problem (problema) - instanta clasa Problem
* parameters (parametri) - instanta clasa Parameters
* particleList (lista de particule) - un vector de instante ale clasei Particle
* globalBestPosition (pozitia globala cea mai buna) - lista de valori reprezentate in dubla precizie ce ilustreaza cea mai buna pozitie atinsa pana la iteratia actuala de lista de particule
* globalBestCost (costul global cel mai bun) - valoare ce reprezinta cel mai bun cost atins pana acum de lista de particule
* currentIteration (iteratia curenta)

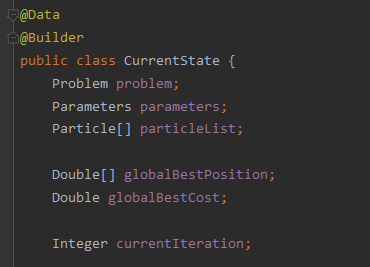


Fig. 3.4 - Cod clasa CurrentState

5. Clasa PSO (Particle Swarm Optimization) contine functii ce implementeaza propriu-zis algoritmul cerut:

* functia init (initializare) - primeste parametrii si problema ceruta si se ocupa cu initializarea valorilor necesare pentru fiecare particula si pentru starea intiala
* functia particleSwarmOptimize (optimizeaza prin roi de particule) - este functia care se ocupa cu ajustarile si avansarile fiecarei particula pe parcursul unei singure iteratii
* functia solve (rezolva) - este functia care apeleaza particleSwarmOptimize si se ocupa cu asigurarea criteriului de convergenta data de numarul maxim de iteratii prestabilit

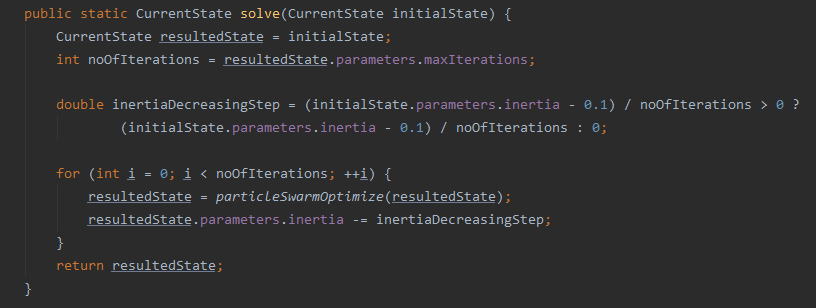


Fig. 3.5 - Cod functia solve

Cu ajutorul librariei Lombok, fiecare clasa dispune de metoda specifica sablonului de proiectare cu acelasi nume - Builder, ce ajuta la instantierea si popularea membrilor unei clase.

S-a folosit framework-ul Spring pentru a crea un serviciu sumar compus din clasele Service (serviciu) - pentru logica si Controller (controler) - pentru a primirea cererilor HTTP.

Rezultatele rularii algoritmului asupra unei probleme sunt prezentate prin intermediul clasei ProblemResults (rezultatele problemei) care este expusa printr-un api.

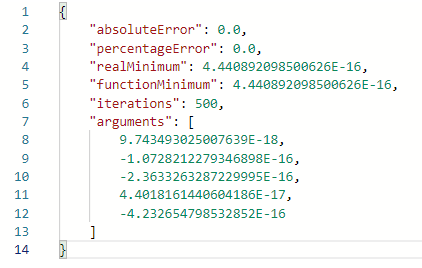


Fig. 3.6 - Exemplu de raspuns API

4. Concluzii

In urma experientelor avute pe parcursul implementarii algoritmului optimizare tip roi de particule am inteles mai bine cum functionarea acestuia. Prin intalnirea si depasirea obstacolelor aparute pe parcursul proiectului, am deprins cunostinte legate de functia Ackley, dar si cum prin prisma faptului ca prezinta o multitudine de minime locale, aceasta poate reprezenta o adevarata provocare pentru algoritmii de minimizare.

De asemenea, prin intermediul acestui proiect, am deprins o mai buna intelegere a influentei pe care o au parametrii din cadrul algoritmului. Spre exemplu, pentru o dimensiune a problemei relativ mare, 30, observam ca numarul prestabilit de 100 de particule nu este suficient, rezultatul obtinut fiind unul eronat. Cand marim numarul de particule la 300, observam ca rezultatul este mult mai apropiat de minimul real al functiei, dupa cum putem vedea in Figura 4.1.

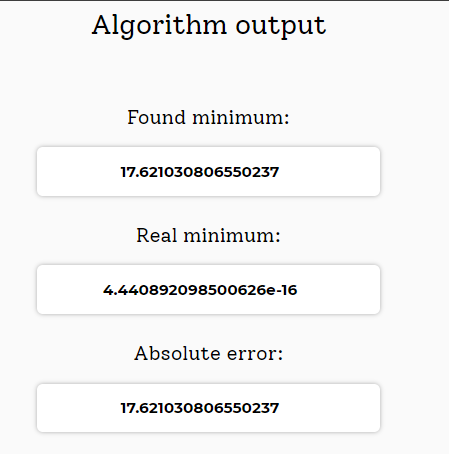
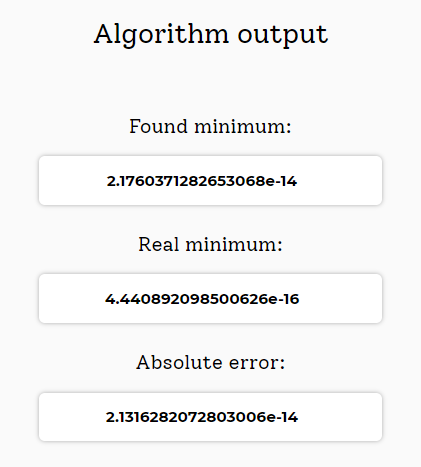


Fig. 4.1 - Rezultate algoritm cu nr particule = 300 (stanga), respectiv 100 (dreapta)

5. Bibliografie

5.1 Florin Leon, Cursuri Inteligenta Artificiala(2019)

5.2 Marcin Molga, Czesław Smutnicki - Test functions for optimization needs (2005)

5.3 Optimization Test Functions - [www.sfu.ca](http://www.sfu.ca)

6.Impartire sarcini intre membrii echipei:

Amariei Marinel:

* interfata frontend in ReactJS
* implementare functii pso optimize, solve si init
* depanarea problemelor intalnite pe parcursul implementarii algoritmului
* contribuit la documentatie

Oana Blejusca:

* arhitectura API in Spring
* structura clase proiect (Particle, Problem, CurrentState, etc.)
* depanarea problemelor intalnite pe parcursul implementarii algoritmului
* contribuit la documentatie