Tema4

Horeanga Bogdan gr.332

18 Aprilie 2021

Algoritm evolutiv

```python
def fitness(x):
    value=0;
    for i in range(len(x)):
        value=value+5*(i+1)*(x[i] ** 2)
    return value

def generateRandom(n):
    x=np.random.uniform(-5.12, 5.12, n)
    return x
```

(a)

```python
def incrConvSimpla(parent1,parent2,alpha):
    point=random.randint(0,len(parent1))
    kid1=parent1.copy()
    kid2=parent2.copy()
    for i in range(point,len(parent1)):
        kid1[i]=alpha*parent2[i]+(1-alpha)*parent1[i]
        kid2[i]=alpha*parent2[i]+(1-alpha)*parent1[i]
    return kid1,kid2

def incrCont(parent1,parent2,pb):
    kid1 = parent1.copy()
    kid2 = parent2.copy()
    for i in range(len(parent1)):
        if(random.uniform(0,1)<pb):
            kid1[i]=(parent1[i]+parent2[2])/2
            kid2[i]=(parent1[i]+parent2[2])/2
    return kid1,kid2
```

(b)

```python
def uniformMutation(kid):
    point = random.randint(0, len(kid)-1)
    kid[point]=np.random.uniform(-5.12, 5.12)
    return kid

def mutationKids(kids):
    mutated=kids.copy()
    n = len(mutated)
    i = 0
    while i < n:
        copiemutated = mutated[i].copy()
        m = uniformMutation(copiemutated)
        mutated[i] = m.copy()
        i += 1
    return mutated
```

(c)

```python
def rankSelection(population):
    listPb=[]
    bestParents=[]
    population.sort(key=lambda x: fitness(x), reverse=True)
    prevPb=0.0
    k=1.5
    for i in population:
        pb=(1+k)/(len(population))-2*k*(population.index(i)+(k-1))/(len(population)*(len(population)-1))
        pb = ((2 - k) - 2 * (population.index(i) + (k - 1)) / ((len(population) - 1)))/(len(population))

        listPb.append((prevPb,pb+prevPb))
        prevPb=pb+prevPb

    while len(bestParents)<len(population):
        pb=random.uniform(0,1)
        for j in listPb:
            if pb>=j[0] and pb<j[1]:
                bestParents.append(population[listPb.index(j)])
                break
    return bestParents
```

(d)

```python
def turnirSelection(population,k):
    i=0
    bestParents=list()
    while i <len(population):
        poz = np.random.default_rng().choice(len(population), size=k, replace=False)
        bestParent = population[poz[0]]
        for j in poz:
            if fitness(population[j]) < fitness(bestParent):
                bestParent=population[j].copy()
        bestParents.append(bestParent)
        i += 1
    return bestParents
```

(a)

```python
def sortAllPeople(allPeople):
    allPeople.sort(key=lambda x: fitness(allPeople), reverse=False)
    return allPeople

def selectSurvivors(allPeople,populationNR): # din lista tuturor persoanelor din generatie care sunt sortati
    bestOfAll=list()
    n=int(0.1*len(allPeople))
    for i in range(n):
        bestOfAll.append(allPeople[i]) #se salveaza 10% din cei mai buni si se elimina dupa
    allPeople=allPeople[n:]
    selectiaTurnir=turnirSelection(allPeople,3)
    for i in range(len(bestOfAll),populationNR):
        bestOfAll.append(selectiaTurnir[i])
    return bestOfAll
```

(b)

```python
def evolutionaryAlg(populationNR, n,nrGeneratii,nrRulari,m,alpha):
    f = open("REAL.txt", "w")
    f.write("")
    f.close()
    i=0
    toateRularile=list()
    while i<nrRulari:
        populatie=generatePopulation(populationNR,n)
        listaCuValAVG=list()
        bestOne,avg=bestAvgPopulation(populatie)
        listaCuValAVG.append((float(fitness(bestOne)),avg))
        t=1
        while t < nrGeneratii:
            parents=turnirSelection(populatie,k) #se selecteaza parintii
            kids=parentsCrossoverSimple(parents,alpha) #se genereaza copii
            kidsM=mutationKids(kids)# se fac mutatiile la copii
            allPeople=parents+kids+kidsM    #se formeaza o noua populatie
            allPeople=sortAllPeople(allPeople) # se sorteaza dupa fitness
            populatie=selectSurvivors(allPeople,populationNR) #se face selectia populatiei
            bestOne, avg = bestAvgPopulation(populatie) #se determina cel mai bun individ din populatie si avg
            listaCuValAVG.append(( (float(fitness(bestOne)), avg))    #se adauga in vectorul specific pe care il scriu in fisier
            t+=1#Creste numarul de generatii
            #print(listaCuValAVG)
        scrieVectorTuple("REAL.txt",listaCuValAVG)#scriem intr-un fisier vectorul de best/avg pentru fiecare generatie pentru plot
        #adaugam in toateRularile bestul si Avg ultimei populatii pentru a putea determina bestul din n rulari
        toateRularile.append(listaCuValAVG[len(listaCuValAVG)-1])
        i+=1
    return toateRularile
```

(c)

```python
def evolutionaryAlg2(populationNR, n,nrGeneratii,nrRulari,k,pb):
    f = open("REAL.txt", "w")
    f.write("")
    f.close()
    i=0
    toateRularile=list()
    while i<nrRulari:
        populatie=generatePopulation(populationNR,n)
        listaCuValAVG=list()
        bestOne,avg=bestAvgPopulation(populatie)
        listaCuValAVG.append((float(fitness(bestOne)),avg))
        t=1
        while t < nrGeneratii:
            parents=turnirSelection(populatie,k) #se selecteaza parintii
            kids=parentsCrossoverCont(parents,pb) #se genereaza copii
            kidsM=mutationKids(kids)# se fac mutatiile la copii
            allPeople=parents+kids+kidsM    #se formeaza o noua populatie
            allPeople=sortAllPeople(allPeople)# se sorteaza dupa fitness
            populatie=selectSurvivors(allPeople,populationNR) #se face selectia populatiei
            bestOne, avg = bestAvgPopulation(populatie) #se determina cel mai bun individ din populatie si avg
            listaCuValAVG.append(( (float(fitness(bestOne)), avg))    #se adauga in vectorul specific pe care il scriu in fisier
            t+=1#Creste numarul de generatii
            #print(listaCuValAVG)
        scrieVectorTuple("REAL.txt",listaCuValAVG)#scriem intr-un fisier vectorul de best/avg pentru fiecare generatie pentru plot
        #adaugam in toateRularile bestul si Avg ultimei populatii pentru a putea determina bestul din n rulari
        toateRularile.append(listaCuValAVG[len(listaCuValAVG)-1])
        i+=1
    return toateRularile
```

(d)

PSO

```python
class Particula():
    def __init__(self,n):
        self.position=generateRandom(n)
        self.persBest=self.position
        self.viteza=[0]*n
        self.vecini=list()
    def __str__(self):
        return f"My position now {self.position}, my best Position {self.persBest}"
    def fitness(self):
        return fitness(self.position)
    def valBest(self):
        return fitness(self.persBest)

    def setPersBest(self):
        if fitness(self.position) < fitness(self.persBest):
            self.persBest=self.position
    def modificarePozitie(self):
        for i in range(len(self.position)):
            self.position[i]=self.position[i]+self.viteza[i]
    def getViteza(self,i):
        return self.viteza[i]
```

(a)

```python
class Space():
    def __init__(self,popSize,n):
        self.popSize=popSize
        self.population=[Particula(n) for i in range(popSize)]
        self.gBest=self.population[0].position
    def valBest(self):
        return self.gBest.fitness()
    def printSpace(self):
        for i in self.population:
            print(i)
    def setGBest(self):
        self.population.sort(key=lambda x: x.fitness(), reverse=False)
        self.gBest=self.population[0].position
    def getGlobalBest(self):
        return self.gBest

    def modificareViteza(self,c1,c2,w):

        for particle in self.population:
            for j in range(len(particle.viteza)):
                rand = random.random()
                vel=(w*particle.viteza[j])+ (c1*rand)*(particle.persBest[j]-particle.position[j])+(c2*rand)*(self.gBest[j]-particle.position[j])
                if -5.12<vel<5.12:
                    particle.viteza[j]=vel
```

(b)

```python
    def updatePopulatie(self):
        for particle in self.population:
            particle.setPersBest()
    def modificarePozitii(self):
        for i in self.population:
            i.modificarePozitie()
```

(c)

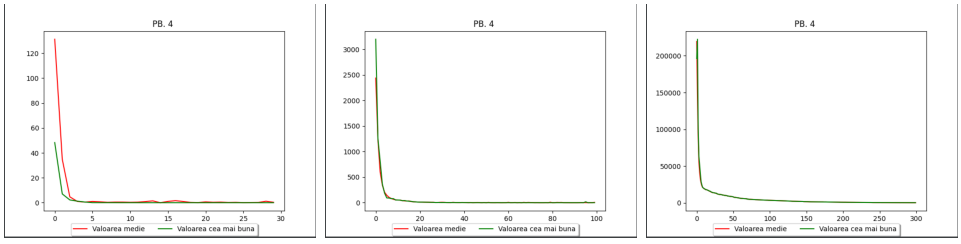
```python
def PSO(c1,c2,w,popSize,n,maxIteration,nrRulari,racire):
    f = open("PSO.txt", "w")
    f.write("")
    f.close()
    j=0
    topRulari=list()
    while j< nrRulari:
        s=Space(popSize,n)
        i=0
        listCuValAvg=list()
        best=s.getGlobalBest()
        avg=s.mediaPopulatiei()
        listCuValAvg.append((float(fitness(best)),avg))
        wcopy=w
        while i < maxIteration:
            s.updatePopulatie()
            s.setGBest()
            s.modificareViteza(c1,c2,wcopy)
            s.modificarePozitii()
            avg = s.mediaPopulatiei()
            best=s.getGlobalBest()
            listCuValAvg.append((float(fitness(best)), avg))
            i+=1
            wcopy=wcopy*racire
        scrieVectorTuple("PSO.txt", listCuValAvg)
        j+=1

        topRulari.append(listCuValAvg[len(listCuValAvg)-1])
    return topRulari
```

(d)

In tabelul de mai jos este algoritmul evolutiv in care se foloseste incrucisarea simpla. Adica def incrConvSimpla(parent1,parent2,alpha)

Valori AE cu incrucisare simpla si turnir selection								
Dimensiunea populatiei	Dim.	K = nr. pentru turnir	NR. Rulari	NR. Generati	Alpha	Best	AVG	Timp (secunde)
10	2	3	10	10	0.3	0.004427	0.0027	0.288
10	2	3	10	1000	0.3	7.131e-08	7.131e-08	30.375
100	2	3	10	20	0.5	2.764e-12	0.2087	6.7031
100	2	3	10	20	0.3	3.755e-13	1.26877	6.2912
100	2	3	10	30	0.4	1.059e-12	0.58368722	9.738
1000	2	10	10	40	0.3	3.5101e-37	0.9887	126.949
10	10	3	10	10	0.3	136.459	223.6036	0.3822
10	10	3	10	100	0.3	0.9201	0.9201	3.8703
100	10	3	10	100	0.3	0.0134	2.124	40.1723
1000	10	3	10	100	0.3	0.000125	1.672	401.8754
10	100	5	10	50	0.6	32109.4149	32491.7349	8.5958
10	100	5	10	200	0.6	6239.197	6268.796	30.8136
100	100	5	10	300	0.3	157.463	160.6443	516.365
100	100	3	10	300	0.2	115.1144	124.288	507.0749

Figurile de sub tabel sunt reprezentarile grafice pentru linile din tabel colorate cu rosu.



(a) (b) (c)

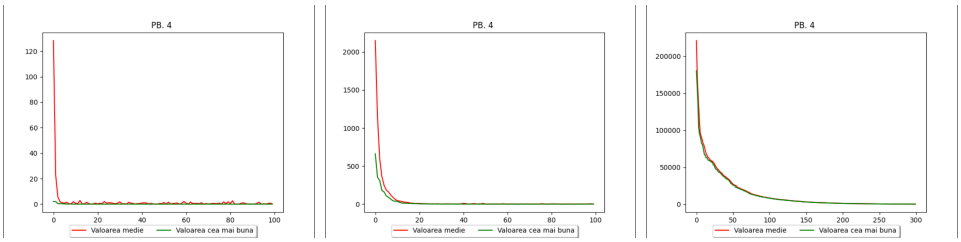
Din laboratorul trecut am observat ca atunci cand k este mai mic valorile sunt mai bune, de aceea l-am pastrat asa mic si nu am schimbat foarte mult k-ul.Dar in cazul de fata se observa ca atunci cand alpha-ul este mai mic, solutia devine mai buna, cu taote ca AVG nu se imbunatateste la fel de mult. Numarul de generatii joaca un rol important in cazul de fata si nu se atinge un plafon chiar asa rapid ca in laboratorul trecut.
Timpul de executie creste odata cu numarul e generatii si dimensiunea populatiei, fiind mai strans legata de nr de indivizi din populatie.

Se mai observa ca atunci cand creste dimensiunea spatiului de cautare, se apropie din ce in ce mai greu de solutia optima(0). Si este necesar de un numar mai mare de rulari, un numar mai mare de indivizi in populatie, iar acest lucru creste foarte mult timpul de executie.

In tabelul de mai jos este algoritmul evolutiv in care se foloseste incrucisarea continua. Adica def incrCont(parent1,parent2,pb))

Valori AE cu incrucisare continua si turnir selection								
Dimensiunea populatiei	Dim.	K = nr. pentru turnir	NR. Rulari	NR. Generati	Pb.	Best	AVG	Timp (secunde)
10	2	3	10	10	0.3	0.005588	1.094	0.2977
100	2	3	10	20	0.4	5.87033e-06	0.7414	6.2077
100	2	3	10	100	0.4	6.00346e-07	6.049673e-07	30.6547
100	2	3	10	200	0.3	3.4888e-10	1.04851	68.6228
1000	2	10	10	200	0.6	3.8281e-11	0.74686	710.865
10	10	3	10	10	0.3	82.519	96.4845	0.4186
10	10	3	10	100	0.3	0.2211	0.2218	4.337
100	10	3	10	100	0.3	0.0037	1.3032	39.874
10	100	10	10	50	0.4	48369.800	48369.800	12.226
10	100	5	10	200	0.3	8275.7919	8332.224	35.852
100	100	5	10	300	0.3	326.0160	333.7897	504.0154

Figurile de sub tabel sunt reprezentarile grafice pentru linile din tabel colorate cu rosu.



(a) (b) (c)

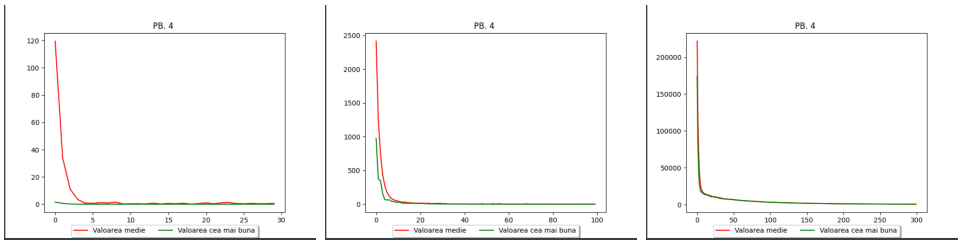
In cazul de fata, in care s-a folosit incrucisarea continua, se observa ca atunci cand creste numarul de generatii si solutia cea mai buna se imbunatateste. Probabilitatea scade cu fiecare generatie pentru a explora la inceput, iar dupa aceea sa exploatam solutile.

Comparand cu algoritmul de mai sus, se observa ca nu sunt valorile la fel de bune cu acest algoritm. Acestaa obtine valori mai mari si intr-un timp mai mare. Diferentele de timp nu sunt mari, dar valorile de best sunt.

In tabelul de mai jos este algoritmul evolutiv care foloseste rank selection pentru parinti si incrucisare simpla:

Valori AE cu incrucisare simpla si rank selection								
Dimensiunea populatiei	Dim.	K = nr. pentru turnir	NR. Rulari	NR. Generati	Alpha	Best	AVG	Timp (secunde)
10	2	-	10	10	0.3	4.9574e-05	1.0896	0.214
100	2	-	10	20	0.5	8.069e-10	0.072	4.7554
100	2	-	10	20	0.3	1.1486e-09	0.5884	4.776
100	2	-	10	30	0.4	2.3460e-14	2.781	7.3786
1000	2	-	10	40	0.3	6.94690e-23	0.6084	118.9426
10	10	-	10	10	0.4	63.119	107.535	0.3023
10	10	-	10	100	0.5	0.2909	.2909	3.242
100	10	-	10	100	0.5	0.00995	0.16259	31.5039
1000	10	-	10	100	0.6	0.000138	3.2059	393.919
100	100	-	10	300	0.4	267.4353	329.3969	388.472
100	100	-	10	300	1	565.709	597.0647	373.6149
10	100	-	10	1000	0.5	97.709	98.2478	113.345
10	2	-	10	1000	0.5	1.0825e-08	1.0825e-08	22.691

Figurile de sub tabel sunt reprezentarile grafice pentru linile din tabel colorate cu rosu.



(a) (b) (c)

La rank selection se observa o imbunatatire buna la timpul de rulare si are o tendinta de a avea valori mai bune decat selectia turnir.

In tabelul de mai jos sunt valorile pentru PSO

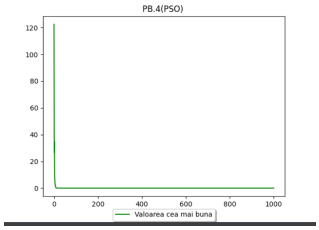
Valori PSO									
Dimensiunea populatiei	Dim.	NR. Iteratiir	NR. Rulari	c1	c2	w	Factor Racire	Best	—Timp (secunde)
10	2	100	10	2	2	1	0.999	9.61187	—0.1262
10	2	1000	10	2	2	1	0.99	3.2820e-126	—1.159
10	2	1000	10	3	1	0.3	0.999	3.2820e-57	—1.198
10	2	1000	10	2.5	1.5	0.3	0.999	4.563e-34	—1.14
10	2	1000	10	2	2	0.3	0.999	0.0	—1.187
50	2	100	10	2	2	1	0.99	6.2041e-15	—0.56
50	2	1000	10	2	2	0.7	0.999	7.15254e-247	—5.26
50	2	2000	10	2	2	0.5	0.999	0.0	—12.40
10	10	1000	10	2	2	1	0.999	6.794	—4.55
10	10	1000	10	2	2	0.7	0.999	0.0023	—4.58
100	10	1000	10	2	2	0.7	0.999	5.560e-95	—42.833
10	10	2000	10	2	2	0.5	0.999	0.0227	—8.961
10	10	2000	10	2	2	1	0.999	1.1519	—8.516
50	10	2000	10	2	2	0.3	0.999	3.6151e-07	—42.084
100	10	2000	10	2	2	0.4	0.999	3.402e-48	—84.419
10	30	1000	10	2	2	0.3	0.999	3083.903	—13.039
10	30	2000	10	2	2	0.3	0.999	2640.499	—24.207
10	30	1000	10	2	2	0.7	0.999	10649.9188	—12.487
10	30	1000	10	2	2	0.2	0.999	5005.9096	—12.574
100	30	2000	10	2	2	0.3	0.999	142.8971	—257.32

Analizand datele din tabel se observa ca atunci cand factorul de inertie(w) este mai mic se obtin valori mai bune sau chiar optime in cazul cu 2 dimensiuni. De asemenea un rol important il are si factorul de racire care scade factorul de inertie cu fiecare iteratie. Acesta daca e mare factorizeaza cautarea globala si cand scade cauta local mai mult. Ceea ce ne permite o explorare a spatiului mult mai buna.

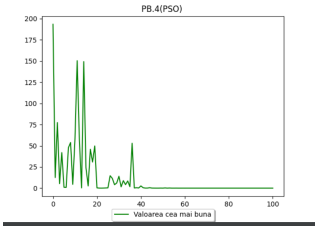
Se observa ca si numarul de iteratii joaca un rol destul de important in explorarea spatiului si gasirea solutiei optime. In unele cazuri in care numarul de iteratii creste iar numarul populatiei nu, acesta nu aduce imbunatatiri radicale la solutie. De asemenea cresterea numarului de indivizi din populatie ajuta la explorarea spatiului.

Dupa cum se mai observa se obtin soltutii mai bune atunci cand factorul de invatare cognitiva si factorul de invatare sociala sunt egale cu 2.

Figurile de mai jos sunt reprezentarea grafica pentru primele doua linii rosii din tabel, mai exact pentru dimensiunea populatiei 10 respectiv 50 in dimensiunea R^2

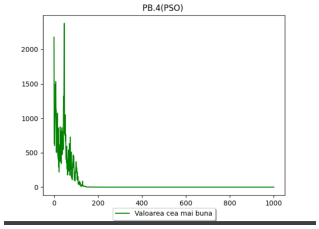


(a)

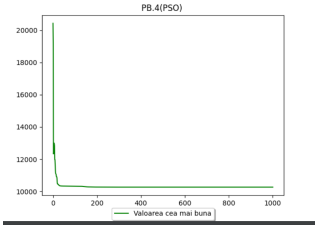


(b)

Figurile de mai jos sunt reprezentarea grafica pentru ultimele doua linii rosii din tabel, mai exact pentru dimensiunea populatiei 100 respectiv 10 in dimensiunea R^10 si respectiv R^30



(a)



(b)