Maşala Alexandra
Grupa 1052
Seria D

Rezolvarea unei probleme folosind algoritm genetic

Vom folosi un algoritm stohastic pentru rezolvarea exercitiului 4 si anume, algoritmul genetic.

Enunt

Problema de transport presupune urmatoarele: exista o oferta de produse pe care o putem defini drept un vector notat cu O, cu 2 elemente, in cazul nostru. Fiecare furnizor ofera o anumita cantitate printr-un bun. Exista corespunzator o cerere formata din mai multi consumatori la care vor fi transportate acele bunuri. Cererea va fi notata ca un vector C cu 3 elemente. Bunurile sunt transportate de la surse catre consumatori, adica de la fabrici la depozite.

Se cunoaste o matrice cu costuri de transport, avand m linii si n coloane (2 x 3). Pe fiecare linie vom reprezenta oferta, pe fiecare coloana vom reprezenta o cerere. Fiecare element x_{ij} din aceasta matrice, ca urmare, va indica costul transportului unei unitati de la sursa i la destinatia j.

In tabelul de mai jos am notat cu F1 şi F2 furnizorii din Bucuresti si Craiova, iar cu D1, D2 si D3 depozitele din Ploiesti, Pitesti si Cluj.

	D1	D2	D3	Oferta
F1	50	70	90	120
F2	60	70	100	140
Cerere	100	60	80	

Oferta totala=	260
Cerere totala=	240
Oferta>Cerere	20

Conform cerintei, putem observa ca numarul total de tone de la F1 va fi mai mic sau egal cu 120, iar numarul total de tone de la F2 va fi mai mic sau egal cu 140.

Numarul total de tone stocat de D1 va fi mai mare sau egal cu 100, cel stocat de D2 mai mare sau egal cu 60 iar cel stocat de D3 mai mare sau egal cu 80.

Costul minim de transport va fi egal cu produsul scalar dintre numarul de tone si costul de transport al unei tone.

Ipoteza de lucru

Ca ipoteza de lucru, cererea este mai mica sau egala cu oferta, ceea ce inseamna ca suma cantitatilor din vectorul de cerere mai este mica sau egala cu suma cantitatilor din vectorul de costuri (o parte din oferta va ramane neconsumata).

Cerinta

Ni se cere sa gasim un plan de transport, adica o matrice de coordonate m x n in care fiecare elememt x_{ij} indica ce cantitate se transporta de la sursa i la sursa j.

Acest plan de transport trebuie sa acopere integral cererea, pe coloane, si o parte din oferta, pe linii.

Maşala Alexandra
Grupa 1052
Seria D

Astfel, intentia este de a consuma o parte din oferta si de a acoperi integral cererea.

Ni se cere acel plan cu costul minim. Costul transportarii unei unitati de la sursa i la destinatia j este dat de matricea de transport. Pentru a afla costul total al transportului de la sursa i la destinatia j, vom inmulti elementul corespunzator din matricea de transport cu cantitatea transportata de la sursa i la destinatia j si vom aduna toate aceste costuri. (avem produs scalar intre 2 matrice).

Exemplu

Vom avea exemplele de date de intrare in seturi de 3 fisiere. Un fisier ne da vectorul de oferte, unul vectorul de cereri si altul matricea de costuri:

Oferta: `120 140` `T_oferta.txt` `Costuri: 50 70 90` `T_costuri.txt`

Cererea: `100 60 80` `T_cerere.txt` `60 70 100`

Exemplu de apel:

```
s,c=GTR.Ex4_Transport('T_oferta.txt','T_cerere.txt','T_costuri.txt',10,50,0.8
,0.1)
```

Pentru datele reprezentate, avem un astfel de apel in care se transmit ca parametri cele 3 fisiere, 10 indivizi in dimensiunea populatiei curente, 50 de iteratii, 0.8 probabilitate de recombinare, 0.1 probabilitate de mutatie.

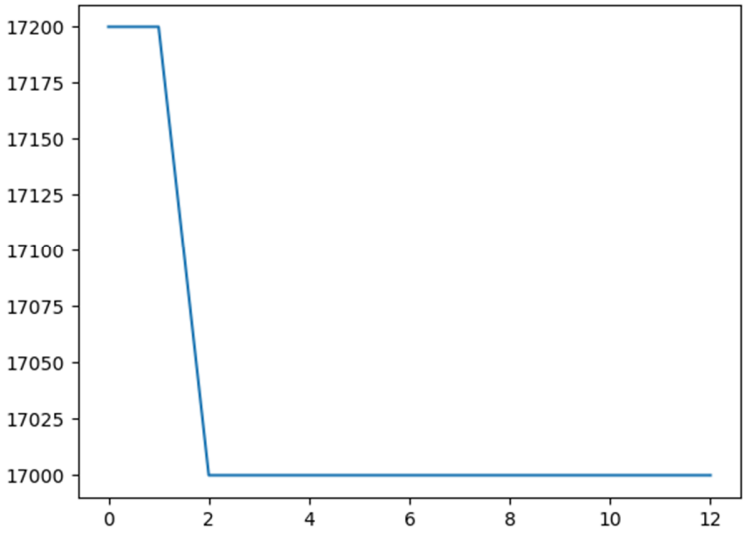
Cu acesti parametri de control si pe datele initial primite se pot obtine urmatoarele rezultate:

1.
```
Cel mai bun cost găsit:  17000.0
Soluția de transport:
[[40.  0. 80.]
 [60. 60.  0.]]
Oferta rămasă: [ 0. 20.]
O parte din ofertă va rămâne neconsumată.
Cerere rămasă: [0. 0. 0.]
Cererea acoperită perfect
```

Asadar, un cost total de transport minim gasit e 17000, iar planul de transport este matricea rezultata. De exemplu, prima casuta, indica faptul ca de la prima fabrica se transporta 40 de tone catre primul depozit, si tot asa mai departe. Fiind o problema nebalansata, cererea a fost integral acoperita dar oferta ramasa nu s-a consumat integral, ramanand 20 de tone.

Mesajele afisate sunt pentru verificarea automata din cadrul programului, dupa ce se gaseste solutia.

Maşala Alexandra
Grupa 1052
Seria D

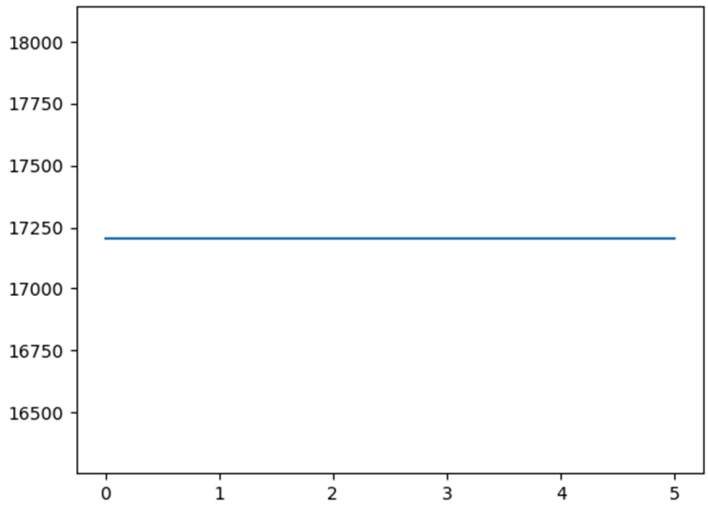


Graficul arata cum a evoluat calitatea celui mai bun individ din fiecare generatie de-a lungul celor 10 generatii.

Resursele utilizate sunt destul de mici in acest exemplu deoarece setul de intrare este de asemenea mic si nu asigura obtinerea solutiei optime in mod consistent.

2.

```
Cel mai bun cost găsit: 17200.0
Soluția de transport
[[100.   0.   0.]
[  0.  60.  80.]]
Oferta rămasă [20.  0.]
O parte din ofertă va rămâne neconsumată.
Cerere rămasă: [0. 0. 0.]
Cererea acoperită perfect
```



Maşala Alexandra
Grupa 1052
Seria D

Rezultat non-optim posibil: cost 17200.

Reprezentarea – codificare

In rezolvarea acestei probleme de transport vom proceda in felul urmator:

De la fiecare ofertant pleaca o cantitate care va fi reprezentata intr o celula a matricei plan de transport, iar acea cantitate e trimisa catre un consumator .

Pe masura ce alocam valori fiecarui element din matricea plan de transport, evident, se diminueaza oferta si se acopera o parte din cerere. Vom actualiza vectorul oferta si vectorul cerere, si le vom numi oferta ramasa si cerere ramasa.

La un moment dat se alege o valoare pentru o celula din matricea plan de transport. Valoarea x_{ij} nu poate sa depaseasca oferta ramasa de la furnizorul i si nici cantitatea ceruta de consumator j. Astfel, valoarea x_{ij} poate sa fie maxim minimum dintre oferta i si cerere j. Se vor actualiza ambii vector.

Acest proces se va repeta pentru fiecare element al matricei de transport pana cand nu mai ramane nimic din cerere sau oferta.

Ordinea de alocare va fi aleatoare, pe rand, cate un element, si va trebui memorata. Liniarizam matricea de transport si reprezentam rangul fiecarui element. Vom actualiza celulele si le vom da valori in ordinea indicata de o astfel de permutare. La un moment dat analizam elementul de rang k, p_k, din reprezentarea liniarizata. Elementul p_k e numarul celulei din matricea de transport pentru care alegem o valoare. Valoarea lui p_k o vom transforma intr o pereche de indici, numar de linii si numar de coloane:

$$p_k = (i-1) \times n + j, \text{ m= 2, n=3}$$

Rezolvare

Vom avea o functie principala `def Ex4_Transport(fo,fc,fcost,dim,nmax,pr,pm)` in care vor fi apelate toate metodele de rezolvare ale problemei de transport:

```
def Ex4_Transport(fo,fc,fcost,dim,nmax,pr,pm):
    # rezolvarea problemei generale de transport

    # fenotip: matricea de transport
    # genotip: ordinea in care se aloca elementele din matricea de transport <=>
permutare, ind. mereu fezabili
    # I: fo - fisier oferta (text, 1xm)
    #    fc - fisier cerere (text, 1 x n)
    #    fcost - fisier costuri (text, m x n)
    #    dim - dimensiune populatie
    #    nmax - numar maxim de iteratii
    #    pr - probabilitate de recombinare
    #    pm - probabilitate de mutatie
    # E: sol - solutia de transport gasita
    #    cost - cost de transport calculat
```

Maşala Alexandra
Grupa 1052
Seria D

```
# initializari - parametri de intrare
oferta=numpy.genfromtxt(fo)
cerere=numpy.genfromtxt(fc)
costuri=numpy.genfromtxt(fcost)


# generare populatie initiala
pop = gen_pop(dim, oferta, cerere, costuri)
v = [min(1000. / pop[:, -1])]

# bucla GA
ok=True
t=0
while t<nmax and ok:
    # selectie parinti
    parinti = s_ruleta_SUS(pop)
    # recombinare
    desc = recombinare(parinti, pr, oferta,cerere,costuri)
    # mutatie
    descm = mutatie(desc, pm, oferta,cerere,costuri)
    # selectie generatie urmatoare
    pop = s_elitista(pop, descm)
    # alte operatii
    # retine cea mai buna solutie
    vmax = min(1000./pop[:,-1])
    i = numpy.argmin(pop[:, -1])
    best = pop[i][:-1]
    v.append(vmax)
    t+=1
    ok=max(pop[:,-1])!=min(pop[:,-1])

print("Cel mai bun cost găsit: ",vmax)
print("Soluția de transport:")
sol=gen_alocare(best,oferta,cerere)
print(sol)
fig=grafic.figure()
grafic.plot(v)
verificare(sol,oferta,cerere)
return (sol,vmax)
```

Reprezentare – decodificare

Partea de decodificare nu se aplica doar la final, ci de fiecare data cand apare un individ nou deoarece am convenit ca evaluam indivizii imediat ce sunt generate.

Initializam:

CR - cererea totala ramasa; se initializeaza cu suma elementelor din vectorul cerere preluat din fisier

c_r - vector cerere ramasa; se initializeaza cu vectorul cerere preluat din fisierul text si se actualizeaza pe parcursul calculelelor scazand permanent din elementele sale cantitatile alocate.

o_r - oferta ramasa; se initializeaza cu vectorul oferta preluat din fisierul text si pe masura ce se fac alocari se diminueaza valorile sale

Maşala Alexandra
Grupa 1052
Seria D

La finalul algoritmului de construire a planului de transport, vectorii vor contine elemente 0.

`i=0`: Incepem analiza de la prima gena a cromozomului. Se initializeaza planul de transport x ca matrice m x n cu zerouri.

Cat timp mai exista cerere de alocat (CR>0) se analizeaza elementul p_k din cromozomul curent. Elementul p_k se transforma in o pereche de indici cu linia, respectiv coloana corespunzatoare. Este de fapt catul si restul impartirii la n.

Apelul `lin,col=numpy.unravel_index(int(permutare[i]),(m,n))`, in care functiei `unravel_index` ii dam rangul elementului p_k si dimensiunile matricei corespunzatoare. Functia ne da inapoi perechea de indici linie si coloana corespunzatoare rangului pe care l-am transmis ca parametru.

Dupa care avem alocarea propriu-zisa: elementul x din matricea plan de transport cu coordonate lin si col primeste ca valoare minimul dintre oferta ramasa corespunzatoare liniei si cererea ramasa coloanei: `x[lin,col]=min([o_r[lin],c_r[col]])`

Se actualizeaza oferta si cererea ramasa scazand valoarea alocata din elementul corespunzator din fiecare din cei 2 vectori:

```
o_r[lin]-=x[lin,col]
c_r[col]-=x[lin,col]
```

Actualizam cererea totala ramasa:

```
CR-=x[lin,col]
```

`i+=1`: Trecem la urmatoarea alela a cromozomului curent, repetam alocarea pentru fiecare alela atat timp cat mai exista cerere ramasa. In clipa in care s-a consumat toata cererea, restul elementelor din matricea de transport vor ramane 0.

```
def gen_alocare(permutare,oferta,cerere):
    # I: permutare - genotip, ordinea de alocare
    #    oferta, cerere - restrictii
    # E: x - alocarea obtinuta, sub forma de matrice

    m=len(oferta)
    n=len(cerere)
    x=numpy.zeros((m,n))
    i=0
    CR=sum(cerere)
    o_r=oferta.copy()   #oferta ramasa
    c_r=cerere.copy()   #cerere ramasa
    while CR>0:
        lin,col=numpy.unravel_index(int(permutare[i]),(m,n))
        x[lin,col]=min([o_r[lin],c_r[col]])
        o_r[lin]-=x[lin,col]
        c_r[col]-=x[lin,col]
```

Maşala Alexandra
Grupa 1052
Seria D

```
        CR-=x[lin,col]
        i+=1
    return x
```

Functia de evaluare

Cromozomii trebuie evaluati intr-un algoritm genetic, iar functia de evaluare necesita decodificarea cromozomului.

Functia obiectiv

Costul e o valoare care trebuie minimizata. In algoritmii genetici preferam sa facem implementarea in sensul maximizarii functiei obiectiv. Varianta aleasa in acest caz este inversarea costului.

```
def f_obiectiv(x,oferta,cerere,costuri):

    # I: x - cromozom evaluat
    #    oferta, cerere - restrictii
    #    costuri - matricea costurilor de transport
    #    E: c - calitate (1000/cost transport)

    a=gen_alocare(x,oferta,cerere)
    c=1000./numpy.sum(a*costuri)
    return c
```

Generare populatie initiala

Dupa ce s-a stabilit modul de reprezentare, codificarea si decodificarea implicit si functia de evaluare: populatia initiala se genereaza aleator, stabilita folosind reprezentarea cu permutari. Vom genera aleator permutari de dimensiune m x n.

```
def gen_pop(dim,oferta,cerere,costuri):
    # I: dim - dimensiune populatie
    #    oferta - oferta de transport (1xm)
    #    cerere = cerere de transport (1xn)
    #    costuri - costuri de transport (mxn)
    # E: pop - populatia generata

    m=len(oferta)
    n=len(cerere)
    pop=numpy.zeros((dim,m*n+1))
    for i in range(dim):
        x=numpy.random.permutation(m*n)
        pop[i, :-1] = x
        pop[i, -1] = f_obiectiv(x, oferta, cerere, costuri)
    return(pop)
```

Conditia de terminare

Dupa ce fiecare individ generat se decodifica, il evaluam si memoram calitatea sa langa reprezentare in matricea populatie.

Conditie de terminare al algoritmului:

Maşala Alexandra
Grupa 1052
Seria D

1. Sa avem mereu un numar limitat de iteratii, stabilit de noi ca parametru de control al algoritmului genetic.

2. Sa avem minim 2 calitati diferite in cadrul populatie curente.

Operatori - Selectia parintilor

In interepretare am folosit, ruleta de tip SUS, cu probabilitate de selectie din modelul FPS cu sigma scalare deoarece se poate obtine optimul folosind mai putine generatii si o dimensiune mai mica a populatiei.

```
def d_FPS_ss(pop,c):
    # distributia de selectie FPS cu sigma scalare

    # I: pop - bazinul de selectie
    #    c - constanta din formula de ajustare. uzual: 2
    # E: p - vector probabilitati de selectie individuale
    #    q - vector probabilitati de selectie cumulate

    m,n=numpy.shape(pop)
    medie=numpy.mean(pop[:,n-1])
    sigma=numpy.std(pop[:,n-1])
    val=medie-c*sigma
    g=[numpy.max([0, pop[i][n-1]-val]) for i in range(m)]
    s=numpy.sum(g)
    p=g/s
    q=[numpy.sum(p[:i+1]) for i in range(m)]
    return p,q

def s_ruleta_SUS(pop):
    # selectia tip ruleta multibrat

    # I: pop - bazinul de selectie
    # E: rez - populatia selectata

    m,n=numpy.shape(pop)
    p,q=d_FPS_ss(pop,2)                 #sau alta distributie
    rez=pop.copy()
    i=0
    k=0
    r=numpy.random.uniform(0,1/m)
    while k<m:
        while r<=q[i]:
            rez[k,:n]=pop[i,:n]
            r+=1/m
            k+=1
        i+=1
    return rez
```

Recombinare

Folosim operatorul CX impreuna cu schema de aplicare pentru probleme fara restrictii:

```
def r_CX(x,y,pr):
    # operatorul de recombinare Cycle Crossover pentru permutari
```

Maşala Alexandra
Grupa 1052
Seria D

```
    # I: x,y - cromozomii parinti
    #    pr - probabilitatea de recombinare
    # E: a,b - descendenti

    a=x.copy()
    b=y.copy()
    r=numpy.random.uniform(0,1)
    if r<pr:
        m=len(x)
        c,nrc=cicluri(x,y)
        for t in range(2,nrc+1,2):
            for i in range(m):
                if c[i]==t:
                    a[i]=y[i]
                    b[i]=x[i]
    return a, b

def cicluri(x,y):
    # determinare cicluri pentru CX

    # I: x, y - cromozomi
    # E: c - vector cu indicii ciclurilor
    #    cite - numarul de cicluri

    m=len(x)
    c=numpy.zeros(m,dtype=int)
    continua=1
    i=0
    cite=1
    while continua:
        a=y[i]
        c[i]=cite
        while x[i]!=a:
            j=list(x).index(a)
            c[j]=cite
            a=y[j]
        try:
            i=list(c).index(0)
            cite+=1
        except:
            continua=0
    return c,cite

def recombinare(parinti,pr,oferta,cerere,costuri):
    # etapa de recombinare

    # I: parinti - multiseutl parintilor
    #    pr - probabilitatea de recombianre
    #    oferta, cerere, costuri - paramatri ai problemei
    # E: desc - descendentii creati

    dim,n=numpy.shape(parinti)
    desc=numpy.zeros((dim,n))
```

```
    #selectia aleatoare a perechilor de parinti
    perechi=numpy.random.permutation(dim)
    for i in range(0,dim,2):
        x = parinti[perechi[i], :n - 1]
        y = parinti[perechi[i + 1], :n - 1]
        d1, d2 = r_CX(x, y, pr)
        desc[i, :n - 1] = d1
        desc[i][n - 1] = f_obiectiv(d1, oferta,cerere,costuri)
        desc[i + 1, :n - 1] = d2
        desc[i + 1][n - 1] = f_obiectiv(d2, oferta,cerere,costuri)
    return desc
```

Mutatie

Pentru mutatie avem operatorul de interschimbare care schimba ordinea de analiza intr-un mod mai natural, mai exact, schimba 2 celule intre ele.

Pentru ca nu avem restrictii folosim schema de mutatie pentru probleme fara constrangeri.

```
def m_perm_schimb(x,pm):
    # operatorul de mutatie prin interschimbare pentru permutari

    # I: x - individul supus mutatiei
    #    pm - probabilitatea de mutatie
    # E: y - individul rezultat

    y=x.copy()
    r=numpy.random.uniform(0,1)
    if r<pm:
        m = len(x)
        p = numpy.random.randint(0, m, 2)
        while p[0] == p[1]:
            p[1] = numpy.random.randint(0,m)
        p.sort()
        y[p[1]]=x[p[0]]
        y[p[0]]=x[p[1]]
    return y

def mutatie(desc,pm,oferta,cerere,costuri):
    # etapa de mutatie

    # I: desc - descendentii obtinuti in etapa de recombianre
    #    pm - probabilitatea de mutatie
    #    oferta,cerere,costuri - restrictiile problemei
    # E: descm - descendenti modificati

    dim,n=numpy.shape(desc)
    descm=desc.copy()
    for i in range(dim):
        x=descm[i,:n-1]
        y=m_perm_schimb(x,pm)
        descm[i,:n-1]=y
        descm[i,n-1]=f_obiectiv(y,oferta,cerere,costuri)
    return descm
```

Maşala Alexandra
Grupa 1052
Seria D

Selectia generatiei urmatoare

Selectia generatiei urmatoare este elitista (ne asigura ca de la o generatie la alta calitatea celui mai bun individ nu scade), facand o inlocuire generationala cu limita de varsta 1. Toti descendentii trec in noua generatie inlocuind indivizii populatiei curente, dar daca niciun descendent nu e mai bun decat cel mai bun individ al populatiei curente, il salvam pe acesta si eliminam unul dintre descendenti (de obicei eliminam individul cel mai slab)

```
def s_elitista(pop,desc):
    # selectia elitista a generatiei urmatoare

    # I: pop - populatia curenta
    #    desc - descendentii populatiei curente
    # E: noua - matricea descendentilor selectati

    noua=desc.copy()
    dim,n=numpy.shape(pop)
    max1=max(pop[:,n-1])
    i=numpy.argmax(pop[:,n-1])
    max2=max(desc[:,n-1])
    if max1>max2:
        k=numpy.argmin(desc[:,n-1])
        noua[k,:]=pop[i,:]
    return noua
```

Dupa ce se decodifica cel mai bun individ din populatia finala si se ajunge la un plan de transport optim, se incheie proiectarea problemei de transport.