### TC 6. Rezolvarea problemei planificării activităților de tip sisteme de producție bazate pe comandă (Job Shop Scheduling –JSS)

Pentru problema planificării activităţilor în varianta JSS datele şi ipotezele de lucru sunt următoarele:

* *J*, un set de *n* sarcini, ;
* *O*, un set de operaţii; fiecare sarcină este constituită dintr-o secvenţă de *m* activităţi (operaţii), *m* independent de , , ; evident, ;
* *M*, o mulţime de *m* maşini care execută operaţii din *O*; activităţile fiecărei sarcini se execută pe câte o maşină din *M* (în cadrul unei sarcini nu există două operaţii care să fie executate pe aceeaşi maşină);
* în cadrul fiecărei sarcini, activităţile respectă o relaţie de precedenţă din punctul de vedere al execuţiei (o ordine de execuţie), astfel încât două sau mai multe activităţi nu pot fi executate simultan; cu alte cuvinte, fiecare sarcină este definită de un set ordonat de operaţii, ordinea fiind predefinită;
* o funcţie care asociază fiecărei operaţii o durata de execuţie; durata efectuării unei operaţii este independentă de planificare;
* activităţile, odată programate, nu îşi pot întrerupe execuţia;
* la fiecare moment de timp, pe oricare maşină, poate fi executată o singură operaţie.

În literatura de specialitate există o serie de variante de rezolvare a problemei JSS în manieră genetică. (Qing-dao-er-ji, Wang, 2012; Chen, Wub, Chen, Chen, 2012)

În continuare este prezentată o variantă propusă de autori pentru obținerea unei soluții a JSS prin utilizarea unei metode proprii de determinare a proprietății unui plan de a fi fezabil.

Planificarea unei operaţii *o* revine la a îi aloca un timp de execuţie (pe maşina corespunzătoare lui *o*), un plan fiind o colecţie de astfel de alocări cu proprietatea că fiecare operaţie este inclusă cel mult o dată. Problema este de a găsi un plan cu proprietăţile:

* completitudine: planul conţine toate operaţiile mulţimii *O*;
* corectitudine: sunt satisfăcute toate restricţiile impuse prin ordinea de execuţie;
* optimalitate: durata totală a planului este minimă.

Rezolvarea problemei prin implementare GA este realizată prin menţinerea unei populaţii de genotipuri care reprezintă permutări ale mulţimii posibile de operaţii. Fie *n\*m* dimensiunea mulţimii de operaţii. O permutare, *pg*, corespunde ordinii de analizare a operaţiilor în scopul planificării lor şi are corespondent într-un plan (fenotip), *pf*, care constă în programarea fiecărei operaţii conform unei analize efectuate în ordinea în care aceasta apare în permutarea genotip. Planul corespunzător unei permutări *pg* poate fi construit pe baza următoarei proceduri.

Pentru efectuează:

* ;
* determină maşina pe care este executată , ;
* asignează timpul de start minim pentru , ţinând cont de ocuparea maşinii şi relaţiile de precedenţă pe care trebuie să le respecte relativ la operaţiile din cadrul sarcinii din care face parte şi care au fost deja planificate.

Funcţia de evaluare aplicată unui genotip furnizează durata de execuţie a planului corespunzător în spaţiul fenotipurilor. Scopul este de a minimiza funcţia de evaluare.

Alegerea operatorilor de variaţie este realizată astfel încât progeniturile să rămână în spaţiul genotipurilor. În cazul mutaţiei poate fi folosit de exemplu operatorul interschimbare. Deoarece relaţiile de adiacenţă nu au nici o semnificaţie în această problemă, operaţia de recombinare poate fi aleasă din oricare dintre cele prezentate în §3.4. În implementarea GA a fost utilizată recombinarea PMX.

În cazul acestei probleme operatorii de selecţie sunt, de asemenea, independenţi de reprezentarea cromozomială, deci pot fi utilizaţi oricare dintre cei prezentaţi în acest capitol, singurul scop fiind acela de a minimiza funcţia obiectiv. Operatorul de selecţie a părinţilor este de tip SUS cu distribuţia de probabilitate de selecţie FPS standard, în timp ce generaţia următoare este construită pe baza urmaşilor cu propagarea celui mai bun cromozom din generaţia curentă în locul celui mai slab urmaş, dacă progeniturile sunt sub acel cromozom din punct de vedere al funcţiei de evaluare.

Stabilirea populaţiei iniţiale este realizată aleator, iar condiţia terminală este de tip prag: de exemplu, dacă a fost atins un număr de iteraţii sau au fost evaluaţi un număr maxim de cromozomi.

În continuare este prezentată implementarea GA, în situaţia în care funcţia obiectiv, care trebuie maximizată, este definită astfel:

unde, pentru fiecare plan fezabil *x*, desemnează durata de execuţie a lui *x* (este întotdeauna nenulă).

Funcţiile *gen\_perm* şi *PMX* sunt identice cu cele prezentate în capitolele 2, respectiv 3 (mai puţin faptul că, în prezentarea PMX de mai sus părinţii sunt specificaţi direct în funcţie, în timp ce, pentru rezolvarea acestei probleme, ei sunt parametri de intrare). Funcţia *gen\_ini* este cea prezentată în problema TSP.

Evident, partea cea mai dificilă este aceea de a asocia unei permutări un plan care să respecte cerinţele de completitudine şi corectitudine, construcţia neasigurând, în acest caz, un plan optim, ci un plan cât mai bun conform permutării date. O variantă de rezolvare este următoarea. Fie *x* permutarea (genotipul) a cărui plan trebuie determinat (fenotipul) şi *n\*m* numărul de operaţii planificate. Vom presupune în continuare că şi aceasta este şi ordinea de efectuare a operaţiilor (ele sunt înscrise în sarcină în ordinea în care trebuie executate). Fiecare operaţie este codificată (reprezentată) prin numărul .

Iniţial, planifică toate operaţiile de la momentul 0 şi marchează fiecare activitate ca neexecutată (prin intermediul menţinerii unui vector binar, iniţializat cu toate valorile 0).

Cât timp nu au fost planificate toate activităţile

Pentru fiecare :

1. fie neplanificat, lista operatorilor care preced o, lista operatorilor care succed o; pot fi determinate astfel:
   * dacă face parte din sarcina şi este pe poziţia *r* în aceasta:

; evident, dacă , lista este vidă

* + altfel, face parte din sarcina şi este ultima operaţie:

;

* + , dacă , altfel este vidă (ultima operaţie din )

1. pentru fiecare
   * dacă *op* a fost executată, timpul de început al execuţiei operaţiei *o* trebuie sa fie după timpul de sfârşit al lui *op* (dacă o este planificată înainte, este resetat timpul de început cu timpul de sfârşit al lui *op*: );
2. fie plan*(o*) timpul de start calculat până la acest moment; dacă maşina pe care trebuie să se execute *o* nu este liberă, adică există un interval planificat pentru o altă activitate care se intersectează cu, atunci planifică *o* după acel interval; operaţia este repetată până când, pe maşina corespunzătoare, nu există programări conflictuale;
3. marchează *o* ca operaţie programată;
4. pentru fiecare
   * dacă *op* a fost executată, şi timpul de început al execuţiei operaţiei *op* este sub timpul de sfârşit al lui *o* (situaţie de programare eronată), atunci renunţă la planificarea lui *op* (este scoasă programarea de pe maşina care o execută) şi marchează *op* ca neplanificat.

Algoritmul furnizează, pe lângă un plan, costul acestuia şi asocierile la nivel de maşină (pentru fiecare operaţie este specificat timpul de început al execuţiei şi timpul la care operaţia se încheie).

function []=GA\_plan(dim,pc,pm,NMax);

[m,nm,durate,mo]=citeste\_date;

pop=gen\_ini(dim,nm);

Maxx=[];

perm=[];

Maxim1=0;

for i=1:NMax

[parinti,fob,fob1]=selectie\_SUS(pop,m,nm,durate,mo);

[popN1,fobN1]=crossover(parinti,m,nm,durate,mo,fob,pc);

[popN,fobN]=

mutatie\_perm\_inserare(popN1,m,nm,durate,mo,fobN1,pm);

[popN,fobN]=selectie\_generatie\_urmatoare(pop,popN,fob1,fobN);

[Maxim,k]=max(fobN);

[asociere,plan,cost]=

permutare\_plan(m,nm,durate,mo,popN(k,1:nm));

Maxx=[Maxx Maxim];

if(Maxim>Maxim1)

A=asociere;

Maxim1=Maxim;

end;

perm=[perm;popN(k,1:nm)];

pop=popN;

end;

[mmm,poz]=max(Maxx);

ppp=perm(poz,1:nm);

disp('Costul minim:');

disp(1/mmm);

disp('Permutarea');

disp(ppp);

for i=1:NMax

Mini(i)=1/Maxx(i);

end;

maximm=max(Mini);

disp('Planificarea');

for k=1:m

disp('Masina');

disp(k);

disp(' Start op Stop');

A{k}=sortrows(A{k},1);

disp(A{k});

end;

figure

i=1:NMax;

plot(i,Mini(i),'ks-');

hold on

plot(poz,1/mmm,'rs');

axis ([1 NMax 0 maximm+1]);

end

function [m,nm,durate,mo]=citeste\_date();

f=fopen('operatii-durata1.txt');

nm=fscanf(f,'%d',1);

durate=fscanf(f,'%f',nm);

fclose(f);

%disp(durate);

f=fopen('masini-operatii1.txt');

m=fscanf(f,'%d',1);

%disp(m);

mo=fscanf(f,'%d',nm);

%disp(mo);

fclose(f);

end

function [asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,x);

% pentru fiecare masina: asocierea start op, sfarsit op

asociere=cell(1,m);

% operatie

% executat(i)=0 daca operatia i nu a fost executata;

% altfel executat(i)=1

executat=zeros(1,nm);

% un plan este un vector care asociaza fiecarei operatii 1..n

% un timp de inceput, plan(o)

cost=0;

plan=zeros(1,nm);

gata=0;

while(~gata)

gata=1;

for i=1:nm

o=x(i);

if(~executat(o))

% determinarea operatiilor care preced

pr=[];cat=fix(o/m);rest=o-cat\*m;

if(rest)

for k=1:rest-1

pr=[pr cat\*m+k];

end;

else

for k=1:m-1

pr=[pr (cat-1)\*m+k];

end;

end;

succ=[];

if(rest)

for k=rest+1:m

succ=[succ cat\*m+k];

end;

end;

[tt,nrp]=size(pr);

% masina pe care se executa

masina=mo(o);

% stabilirea timpului de executie in functie de predecesori

for k=1:nrp

% op precede o;

op=pr(k);

% daca op a fost efectuata

if(executat(op))

if(plan(o)<plan(op)+durate(op))

plan(o)=plan(op)+durate(op);

end;

end;

end;

[d1,d2]=size(asociere{masina});

gata1=0;

while(~gata1)

gata1=1;

for tt=1:d1

% daca operatia nu pate fi planificata la plan(o)

% din cauza ocuparii masinii cautam, dupa plan(o),

% primul interval liber in care sa incapa

if(((asociere{masina}(tt,1)<plan(o))&&

(asociere{masina}(tt,3)>plan(o)))||

((asocire{masina}(tt,1)<plan(o)+durate(o))&&

(asociere{masina}(tt,3)>plan(o)+durate(o)))||

((asociere{masina}(tt,1)>=plan(o))&&

(asociere{masina}(tt,3)<=plan(o)+durate(o))))

plan(o)=asociere{masina}(tt,3);

gata1=0;

end;

end;

end;

asociere{masina}=[asociere{masina};

[plan(o),o,plan(o)+durate(o)]];

executat(o)=1;

if(cost<plan(o)+durate(o))

cost=plan(o)+durate(o);

end;

[tt,nrs]=size(succ);

% pentru toti succesorii executati deja, elimina,

% daca este cazul, planificarea lor

for k=1:nrs

op=succ(k);

if((executat(op))&&(plan(op)<plan(o)+durate(o)))

masina=mo(op);

% scot planificarea lui op de pe masina

[d1,d2]=size(asociere{masina});

for tt=1:d1

if(asociere{masina}(tt,1)==plan(op))

ind=tt;

break;

end;

end;

xx=asociere{masina}(1:tt-1,:);

yy=asociere{masina}(tt+1:d1,:);

asociere{masina}=[xx;yy];

executat(op)=0;

gata=0;

end;

end;

end;

end;

end;

% verificam corectitudinea; nu va fi afisata nici o

% pereche de operatii efectuate in ordine incorecta

incorect=[];

for k=1:nm/m

for kk=1:m-1

ope=[(k-1)\*m+kk (k-1)\*m+kk+1];

% ope(1) trebuie sa se termine inainte de ope(2),

% conditie incalcata de plan

if(plan(ope(1))+durate(ope(1))>plan(ope(2)))

incorect=[incorect;ope];

end;

end;

end;

disp(incorect);

end

function [parinti,fob,fob1]=selectie\_SUS(pop,m,nm,durate,mo);

[dim,nm]=size(pop);

fob1=zeros(dim,1);

for i=1:dim

[asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,pop(i,1:nm));

fob1(i)=1/cost;

end;

p=fob1;

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=pop;

fob=fob1;

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=q(i))

parinti(k,1:nm)=pop(i,1:nm);

fob(k)=fob1(i);

r=r+1/dim;

k=k+1;

end;

i=i+1;

end;

end

function [popN,fobN]=crossover(pop,m,nm,durate,mo,fob,pc);

popN=pop;

[dim,nm]=size(pop);

fobN=fob;

for k=1:2:dim

x1=pop(k,1:nm);

y1=pop(k+1,1:nm);

r=unifrnd(0,1);

if(r<=pc)

[x2,y2]=PMX(nm,x1,y1);

popN(k,1:nm)=x2;

popN(k+1,1:nm)=y2;

[asociere,plan,cost1]=permutare\_plan(m,nm,durate,mo,x2);

fobN(k)=1/cost1;

[asociere,plan,cost2]=permutare\_plan(m,nm,durate,mo,y2);

fobN(k+1)=1/cost2;

end;

end;

end

function popN,fobN]=

mutatie\_perm\_inserare(pop,m,nm,durate,mo,fob,pm);

popN=pop;

fobN=fob;

[dim,nm]=size(pop);

for i=1:dim

r=unifrnd(0,1);

if(r<pm)

p=zeros(1,2);

p(1)=unidrnd(nm);

p(2)=unidrnd(nm);

while(p(1)==p(2))

p(2)=unidrnd(nm);

end;

poz=sort(p);

popN(i,1:poz(1))=pop(i,1:poz(1));

popN(i,poz(1)+1)=pop(i,poz(2));

popN(i,poz(1)+2:poz(2))=pop(i,poz(1)+1:poz(2)-1);

popN(i,poz(2)+1:nm)=pop(i,poz(2)+1:nm);

[asociere,plan,cost]=

permutare\_plan(m,nm,durate,mo,popN(i,1:nm));

fobN(i)=1/cost;

end;

end;

end

function [rezultat,fob]=

selectie\_generatie\_urmatoare(pop,popN,fob1,fobN);

rezultat=popN;

fob=fobN;

[max1,i]=max(fob1);

[max2,j]=max(fobN);

if(max1>max2)

[min1,k]=min(fobN);

rezultat(k,:)=pop(i,:);

fob(k)=max1;

end;

end

*Exemplu de test*

Ca date de intrarea au fost folosite:

* numărul sarcinilor: 5;
* numărul maşinilor: 4, maşinile fiind identificate prin numere de la 1 la 4;
* numărul operaţiilor: 20, operaţiile fiind considerate numere de la 1 la 20;
* duratele de execuţie ale operaţiilor de la 1 la 20 (vectorul *durate*):

[3 2 5 1 3 5 9 4 3 7 3 5 4 4 7 6 4 9 2 7] ;

* alocarea operaţiilor (de la 1 la 20) pe maşini (vectorul *mo*: maşini-operaţii):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [2 1 4 3 | 2 1 3 4 | 1 2 3 4 | 4 1 2 3 | 2 1 3 4] |
| corespunzător | corespunzător | corespunzător | corespunzător | corespunzător |

* precedenţele: aşa cum am specificat în prezentarea modalităţii de generare a unui plan corespunzător unei permutări, în fiecare sarcină operaţiile sunt identificate prin , ordinea efectuării operaţiilor fiind de la cea etichetată cu valoarea minima (), crescător până la ultima operaţie (etichetată cu valoarea minima ()).

La un apel GA\_plan(300,0.8,0.02,80); pot fi obţinute rezultatele:

Costul minim: 33

Planificarea cea mai buna calculata de GA

Masina 1

Start op Stop

0 9 3

3 6 8

9 2 11

11 14 15

15 18 24

Masina 2

Start op Stop

0 5 3

3 1 6

6 17 10

10 10 17

17 15 24

Masina 3

Start op Stop

8 7 17

17 11 20

20 4 21

24 19 26

26 16 32

Masina 4

Start op Stop

0 13 4

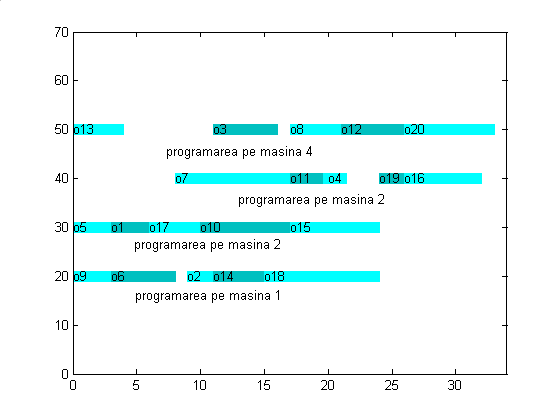
11 3 16

17 8 21

21 12 26

26 20 33

Rezultatul corespunde următoarei planificări, unde fiecare activitate *a* este marcată prin *oa*.



Evoluţia GA este prezentată în următorul grafic.

