Simularea unei linii tehnologice supuse procesului de echilibrare

Lect. Cristina COCULESCU
Universitatea Româno-Americana Bucuresti

An assembly line is a sequence of work-stations, connected together by a material handling system, which is used to assemble components into a final product. The assembly process consists of a sequence of tasks or work-elements. A task consists of some elemental operations which are tied together because of the use of a common tool, jig or fixture. Accordingly, tasks cannot be sub-divided and must be completed at their assigned work- station. The tasks in an assembly process are typically ordered i.e. there may be precedence requirements that must be enforced.

The assembly line balancing problem is the problem of assigning tasks to work -stations. Because tasks may require widely different times, the assignment of task-times to work - stations is rarely equal. This leads to idle time at work - stations.

One of the objectives of the assembly line balancing problem is to minimize this idle time. This article deals with the simulation of process line balancing for a single sample and for determinist manufacture times.

Keywords: *assembly line, balancing problem, simulation, tasks, work - stations, precedence relations, cycle time.*

În numeroase cazuri, experimentele pe obiecte reale sunt greu de realizat si necesita mari fonduri si resurse umane pentru a fi realizate corespunzator. În prezent, datorita facilitatilor de calcul pe care le ofera calculatoarele electronice are loc perfectionarea metodelor de modelare, un loc aparte ocupând modelarea sistemelor complexe cu tehnici de simulare. Acestea ne ofera informatii despre "sistem" înainte ca el sa fie realizat în mod concret prin construirea unor modele matematice si logice care descriu comportarea sistemului real (sau a unor componente ale sale) de-a lungul unei perioade mari de timp.

Cu ajutorul simularii se genereaza "intrarile" si tinând seama de starile interne ale sistemului, prin algoritmi adecvati, se determina "iesirile" si se descrie evolutia în timp a starilor interne ale sistemului. Desi ofera solutii suboptimale, simularea este o tehnica eficienta pentru studierea problemelor complexe la nivel de firma.

Fazele procesului tehnologic trebuie executate într-o anumita ordine în sensul ca anumite faze nu pot fi executate înaintea altora. Relatiile de ordine dintre faze reflecta desfasurarea reala, din punct de vedere tehnologic, a procesului si se numesc relatii de precedenta. Matematic ele formeaza un digraf aciclic.

Vom nota cu $F=\{1,2,\ldots,n\}$ multimea fazelor procesului tehnologic ce urmeaza a fi simulat si cu $t=\{t(1),t(2),\ldots,t(n)\}$ vectorul timpilor de executie asociati fazelor. Problema echilibrarii procesului tehnologic presupune gruparea fazelor în statii de lucru astfel încât suma timpilor de executie ai fazelor ce se executa într-o statie sa fie mai mica sau egala cu o valoare R numita ritm. Pentru modelul problemei de echilibrare abordat R este fixat si este acelasi pentru toate statiile. Obiectivul echilibrarii este determinarea celui mai mic numar de statii cu proprietatea anterioara.

În acest context, ne propunem sa construim o functie cu proprietatea ca are probabilitati egale de generare a valorilor sale. Consideram astfel functia ***rnd*** definita astfel: pentru un numar natural n , *rnd(n)* este un numar natural oarecare cuprins între *0* si *n-1*. Daca dorim o valoare reala, sa zicem un numar cuprins între -2 si 3, cu doua zecimale exacte, folosim o expresie de tipul: *rnd(500)/100 – 2*. În exemplul de simulare, numarul maxim de faze este 200. Atunci numarul de faze este *rnd(200) + 1*. Timpul de executie al fiecarei faze este cel mult egal cu 4 unitati de timp

deci pentru faza *i*, *t(i)=rnd(400)/100+0.01* (se calculeaza cu cel mult doua zecimale exacte si nu poate fi nul). Consideram în continuare ca operatiile aritmetice ce vor fi folosite la generarea unei astfel de functii, se efectueaza pe 32 de biti, adica orice rezultat al unei operatii aritmetice se considera ca restul împartirii rezultatului real la 2^{32}. Aceasta valoare a fost aleasa pentru ca este suficient de mare pentru nevoile obisnuite si este usor de folosit în programare. Astfel se considera sirul $\{x_n\}_{n\geq 0}$ definit astfel:

$$\begin{cases} x_0 = c \\ x_{n+1} = a*x_n + b \text{ pentru } n \geq 1 \end{cases}$$

unde *a* si *b* sunt numere impare (*a* este un numar de forma *a = 4*k + 1*). Un astfel de sir, mai ales daca *a* si *b* sunt numere prime, are o periodicitate extrem de lunga si este suficient de bun pentru nevoile obisnuite. Daca la valoarea lui se adauga si cea a functiei standard generatoare de numere aleatoare folosita în bibliotecile diverselor limbaje de programare si ea este setata dupa contorul de timp si ceasul intern al calculatorului, rezultatul va putea fi de o mare varietate.

Programul RNDGEN simuleaza un proces tehnologic în sensul generarii aleatoare a caracteristicilor principale ale fazelor de lucru, ce vor fi considerate la rezolvarea problemei echilibrarii unei linii tehnologice pe care se asambleaza un singur model al unui produs, timpii de executie ai fazelor sunt deterministi iar ritmul de functionare al liniei este fixat. Se definesc constantele utilizate. v10000 reprezinta numarul maxim de încercari nereusite consecutive de adaugare a unei relatii de precedenta. În continuare se definesc variabilele si functiile utilizate. Semnificatia lor se va vedea din descrierea programului.

```
FILE* denfis;
FILE* rezultat;
FILE* denumiri;
DWORD a,b,c,x;
int rnd(int);
void main()
{
 int nrfaze;
 int restcomp;
 int i,j,k;
 int ind[400],faza[400],codut[400];
 int fazapr[200][20],fazapr_s[200][20];
 int fazaprf[200][20],
     fazaprf_s[200][20];
 int nrpred[200],nrpred_s[200];
 float timp[200],ponderf[200],
       ponderf_s[200];
 float tmax,suma;
 int numar_utilaje,p1,p2,p3;
 int erori,absenta;
 char denfaze[400][20];
 char car;
 int numar_incercari_nereusite=0;
 int modif=0;
```

Se initializeaza generatorul standard de numere aleatoare din C. Apoi se cere introducerea de la tastatura a 3 numere, pe baza carora se va genera sirul de numere aleatoare ce va fi combinat cu cel din generatorul standard. Se alege apoi numarul de utilaje, dupa care se creeaza fisierul faze1.dat în care se trece numarul de faze si considerarea sau ignorarea restrictiilor de tip utilaj, dupa cum se trage la sorti.

```
 numar_utilaje=rnd(10)+1;
 _lcreat("faze1.dat",0);
 denfis=fopen("faze1.dat","r+");
 nrfaze=20+rnd(80)+1;
 fprintf(denfis,"numãr faze= %3d\n",
         nrfaze);
 restcomp=rnd(2);
 fprintf(denfis,"considerare restrictii
        de tip utilaj =%2d\n",restcomp);
 fclose(denfis);
```

Se citesc apoi denumirile fazelor. Ele sunt standard si au fost construite manual, în fisierul **denfaze.dat** (fisier text care a fost editat cu programul NOTEPAD). Aceste denumiri sunt valabile pentru orice rezultat al programului RNDGEN.

```
eti1:
 denumiri=fopen("denfaze.dat","r+");
 for(i=0;i<nrfaze;i++) {
  for(j=0;j<20;j++)
   denfaze[i][j]=32;
 j:=0;
efz2:
 fscanf(denumiri,"%c",&car);
 if(car==lf) goto efz1;
 denfaze[i][j]=car;
 j++;
 goto efz2;
efz1: ; }
fclose(denumiri);
```

Generam apoi pentru fiecare faza, numarul utilajului pe care se executa si timpul de executie. Consideram ca nu avem relatii de precedenta. Ele vor fi adaugate pe parcursul executiei programului.

```
 for(i=0;i<nrfaze;i++)
 { faza[i]=i;
   codut[i]=rnd(numar_utilaje)+1;
   timp[i]=rnd(400)/100.0;
 }
 for(i=0;i<nrfaze;i++)
 { for(j=0;j<20;j++)
    fazapr[i][j]=0;
eti3:  ; }
```

Urmeaza generarea unei relatii de precedenta. Ea este relativ simpla. Daca se va dovedi ca nu este buna, va fi anulata , dar nu acum.

```
reluare:
 p1=rnd(nrfaze);
 p2=rnd(20);
 p3=rnd(nrfaze);
 if(p1==p3) goto reluare;
 if(fazapr[p1][p2] != 0) goto reluare;
 fazapr[p1][p2] = 1+p3;
```

Matricea ce contine relatiile de precedenta a fost initializata cu zerouri. Pe parcursul generarii relatiilor, apar valori nenule. Pentru fiecare faza, se pun în evidenta aceste valori (relatii de precedenta) si câte sunt. Pentru faza i, ind[i]=2 daca faza nu are ponderea calculata. Fazele fara predecesori au ponderea calculata din start ca fiind egala cu timpul lor de executie si ind[i]=1 daca faza i nu are predecesori (pondere calculata).

```
 erori=false;
 for(i=0;i<nrfaze;i++)
 { ind[i]=2;nrpred[i]=0;}
 for(i=0;i<nrfaze;i++)
 { for(j=0;j<20;j++)
    { if(fazapr[i][j] == 0) goto eti9;
      nrpred[i]++;
     fazaprf[i][nrpred[i]-1]=
                          fazapr[i][j];
eti9: ;      }      ;
 if(nrpred[i] != 0) goto eti100;
 ind[i]=1;
 ponderf[i]=timp[i];
eti100: ;}
j=0;
```

Se calculeaza apoi timpul total de operare si timpul maxim de operare pentru o faza (maximul timpilor de operare ai fazelor generate).

```
eti110:
 tmax=0;suma=0;
 for(i=0;i<nrfaze;i++)
 { suma=suma+timp[i];
   if(timp[i]>tmax) tmax=timp[i];
 }
```

Primul pas este sa vedem daca este macar o faza cu ponderea necalculata. Daca s-a facut calculul la toate, este finalizat si încercarea de introducere a relatiei de precedenta a reusit.

```
debut_calcul_ponderi:
 modif=0;
 erori=false;
 for(i=0;i<nrfaze;i++)
   if(ind[faza[i]-1]==2) erori=true;
 if(erori==false)
   goto final_calcul_ponderi;
```

Se scaneaza multimea fazelor. Daca ponderea unei faze a fost calculata, trecem la faza urmatoare. Altfel se considera egala cu timpul fazei la care se adauga ponderile predecesorilor. Aceste ponderi sunt fie calculate, fie initializate cu timpul de lucru al fazei respective. Daca valoarea obtinuta este mai mare ca timpul total de operare, cel putin o faza a fost luata în calcul de doua ori, deci graful este ciclic. În acest caz relatia de precedenta trasa la sorti va fi revocata si încercarea este nereusita.

Daca unul dintre predecesori are ponderea necalculata, ponderea fazei nu poate fi calculata. Ea va putea fi calculata la o noua parcurgere a listei fazelor. Daca la o parcurgere a listei fazelor, reusim sa calculam ponderea unei faze, **modif** creste si daca în final ramâne nul, înseamna ca nu s-a mai putut calcula nici o pondere, ceea ce înseamna ca orice faza fara pondere calculata are macar un predecesor cu pondere necalculata. La o reluare a listei, nu s-ar mai putea calcula ponderea nici unei faze si de aceea, daca în final **modif** ramâne nul, este eroare si relatia de precedenta recent introdusa este gresita si va fi eliminata.

```
 i=0;
eti139:
 if(ind[i] != 2) goto eti136;
 ind[i]=0;
 ponderf[i]=timp[i];
 for(j=0;j<20;j++)
 { if(fazapr[i][j]==0) goto eti135;
   ponderf[i]=ponderf[i]+
              ponderf[fazapr[i][j]-1];
   if(ind[fazapr[i][j]-1] == 2)ind[i]=2;
eti135: ; }
 if(ind[i]==0) modif++;
eti136:
 if(ponderf[i]>suma) goto eti144;
 i++;if(i<nrfaze) goto eti139;
```

```
 if(modif==0) goto eti144;
 goto debut_calcul_ponderi;
eti144:
 numar_încercari_nereusite++;
```

Secventa de la debut_calcul_ponderi nu poate cicla la infinit. Daca toate ponderile au fost calculate, ea se termina imediat. Daca o ponderе depaseste suma timpilor tuturor fazelor, calculele se termina si relatia de precedenta recent introdusa este gresita. Daca graful relatiilor de precedenta nu este ciclic, exista cel putin o permutare admisibila a fazelor. Daca (x_1, x_2, …, x_n) este o astfel de permutare, dupa calcularea ponderilor fazelor x_1, x_2, …., x_i se poate calcula ponderea fazei x_{i+1} deoarece toti predecesorii fazei x_{i+1} sunt printre fazele x_1, …., x_i ale caror ponderi au fost calculate. Pâna la urma se vor calcula ponderile tuturor fazelor.

Daca un asemenea calcul este imposibil, raman mereu cel putin o faza a carei ponderе nu a fost calculata. Dupa un numar de cicluri se va ajunge în situatia în care la o parcurgere a fazelor nu este nici o noua faza cu ponderе calculata. Atunci graful fazelor este gresit si ultima relatie de precedenta va fi anulata. În acest caz se reface matricea relatiilor de precedenta de dinaintea tragerii la sorti nefavorabile. La atingerea a 10000 de încercari nereusite consecutive, nu se va mai introduce o noua relatie de precedenta.

```
 for(i=0;i<nrfaze;i++)
  for(j=0;j<20;j++)
    fazapr[i][j]=fazapr_s[i][j];
 if(numar_incercari_nereusite==v10000)
    goto listare_rezultat;
 goto reluare;
```

Daca tentativa a reusit, matricea relatiilor de precedenta va fi salvata pentru a se putea reveni dupa o încercare nereusita.

```
final_calcul_ponderi:
 numar_încercari_nereusite=0;
 for(i=0;i<nrfaze;i++)
 { nrpred_s[i]=nrpred[i];
   ponderf_s[i]=ponderf[i];
   for(j=0;j<20;j++)
   { fazapr_s[i][j]=fazapr[i][j];
     fazaprf_s[i][j]=fazaprf[i][j];
   }
 }
 goto reluare;
listare_rezultat:
 for(i=0;i<nrfaze;i++)
 { nrpred[i]=nrpred_s[i];
   ponderf[i]=ponderf_s[i];
   for(j=0;j<20;j++)
   { fazaprf[i][j]=fazaprf_s[i][j];
   }
 }
```

Creem apoi fisierul de rezultat **faze2.dat** si scriem capul de tabel. Pentru fiecare faza se va scrie denumirea, numarul utilajului pe care se executa, timpul de executie si ponderea.

```
for(i=0;i<nrfaze;i++)
{fprintf(rezultat,"|%4d|",faza[i]+1);
 for(j=0;j<20;j++)
  fprintf(rezultat,"%1c",denfaze[i][j]);
 fprintf(rezultat,"|%2d|%8.2f!%11.2f!",
         codut[i],timp[i],ponderf[i]);
```

Daca nu sunt predecesori, acest fapt va fi pus în evidenta printr-un mesaj. Daca sunt, ei vor fi listati. Daca sunt mai mult de 10, vor fi trecuti urmatorii pe alt rând cu o discontinuitate în tabel. Acest lucru este practic imposibil. Este extrem de greu ca o faza sa aiba mai mult de 5 predecesori directi generati cu acest program. În simularile facute nu s-a obtinut nici un caz de executie a blocului de instructiuni cuprins între eti131 si eti132.

```
 if(nrpred[i] != 0) goto eti131;
 fprintf(rezultat," faza fara predece
                  sori\n");
 goto eti130;
eti131:
 if(nrpred[i] <= 10) goto eti132;
 for(j=0;j<10;j++)
 { fprintf(rezultat,"%3d ",
            fazaprf[i][j]); }
   fprintf(rezultat," \n");
   fprintf(rezultat,"| | | | |");
   for(j=10;j<nrpred[i];j++)
   { fprintf(rezultat,"%3d ",
              fazaprf[i][j]); }
     fprintf(rezultat," \n");
     goto eti130;
eti132:
 for(j=0;j<nrpred[i];j++)
 { fprintf(rezultat,"%3d ",
           fazaprf[i][j]); }
   fprintf(rezultat," \n");
eti130: ; }
```

În final se trec concluziile.

Bibliografie

1. ARCUS, A. L., *Assembly line balancing by computer*, Berkeley,1962

2. ARCUS, A. L., *A computer method of sequencing operations for assembly lineas,* New-York,1966
3. ANDREICA, M., STOICA, M., *Modelarea si simularea proceselor economice,* LITO ASE, Bucuresti, 1994
4. BOLDUR-LATESCU, Gh., SACUIU, I., TIGANESCU, E., *Cercetare operationala cu aplicatii în economie*, Editura Didactica si Pedagogica, Bucuresti, 1979
5. CIOBANU, Gh., *A branch and bound algorithm to solve an assembly line balancing problem*, Economic Computation and Economic Cybernetics Studies and Research, 4, 1977
6. CIOBANU, Gh., NICA, V., MUSTATA, F., MARACINE, V., *Cercetari operationale cu aplicatii în economie*, Editura Matrix Rom, Bucuresti, 1996
7. HARTULARI, C., *Un algoritm euristic pentru echilibrarea liniilor de asamblare,* Studii si cercetari de calcul economic si cibernetica economica, 3, 1980, pag. 74-79
8. HELGERSON, W. P., BIRNIE, D. P., *Assembly line balancing using the ranked positional weight technique,* 1961
9. JACKSON, J. R., *A computing procedure for a line balancing problem,* Management Science, 1956
10. MANSOOR, E. M., *Assembly line balancing-an improvement of the ranked positional technique,* Journal of Industrial Engineering, 1964
11. RATIU-SUCIU, C., *Modelarea si simularea proceselor economice,* Editia a II-a, Editura Didactica si Pedagogica, Bucuresti, 1997
12. RADULESCU, D., GHEORGHIU, O., *Optimizarea flexibila si decizia asistata de calculator*, Editura Stiintifica, Bucuresti, 1992
13. RUSU, C., BRUDARU, O., *Proiectarea liniilor de fabricatie flexibile*, Editura Tehnica, Bucuresti, 1990
14. SNEIDOVICH, M., *Analysis of a preference order assembly line problem,* Management Science, 27, 9, 1981
15. TOMESCU, I., *Probleme de combinatorica si teoria grafurilor,* Editura Didactica si Pedagogica, Bucuresti, 1981
16. VADUVA, I., STOICA, M., ODAGESCU, I., *Simularea proceselor economice*, Editura Tehnica, Bucuresti, 1983