ANNALS of the ORADEA UNIVERSITY.
Fascicle of Management and Technological Engineering

PRODUS PROGRAM PENTRU DETERMINAREA SUCCESIUNII OPTIME DE INTRARE A TIPURILOR DE REPERE INTR-UN SISTEM FLEXIBIL DE FABRICATIE

Sef lucr.dr.ing. Florin LUNGU
Universitatea Tehnica din Cluj-Napoca
Catedra Management si ingineria sistemelor
Tel.fax. 0264-415484
Email: flungu@yahoo.com

Cuvinte cheie: flexibilitate, programare, costuri de tranzitie, euristici

Abstract: The paper's aim is the achievement of a computer program to determine the optimum entrance sequence of the pieces' type into the flexible manufacturing system (FMS).

The paper involves a software application development in JAVA, which has to calculate the Hamiltonian route using those two heuristics from the graphs theory, presented in detail in bibliography.
Key words: flexibility, programming, transition costs, heuristics

Este firesc ca în managementul sistemelor flexibile de fabricatie (SFF) sa se tina seama de proprietatea fundamentala a acestui gen de sistem de productie: flexibilitatea.

Trecerea sistemului de la prelucrarea unui tip de produs la prelucrarea altui tip de produs, respectiv de la o stare la alta a sa, se numeste tranzitie si reprezinta, de fapt, comutarea sistemului de la o tehnologie de fabricatie la alta. Costul corespunzator acestei comutari se numeste cost de tranzitie si cuprinde costurile provenite de la diferite reglaje care trebuie realizate, schimbari de programe de prelucrare, schimbari de scule, montare/demontare dispozitive etc.

Problema succesiunii de intrare a produselor în SFF este echivalenta cu programarea succesiunii starilor sistemului, respectiv, cu problema succesiunii de comutare a tehnologiilor în SFF.

Termenul "programare" trebuie înteles mai aproape de sensul cuvântului din informatica decât de semnificatia lui generala, datorita faptului ca aici nu este vorba de o programare în raport cu anumite termene de livrare ci este o programare care urmareste optimizarea functionarii sistemului. Problema programarii SFF are si un considerabil continut tehnologic pentru ca implica activitatile de pregatire a sistemului, de adaptare a lui la sarcina de productie variabila topologic.

Teoretic, pot exista doua moduri de intrare a tipurilor de produse în SFF [2]:

1. Tipurile de produse intra în SFF într-o succesiune predeterminata. Succesiunea optima este aceea care realizeaza cele mai mici costuri de tranzitie pentru realizarea tuturor tipurilor de repere. Rezolvarea acestei probleme s-a facut cu ajutorul unui aparat matematic bazat pe teoria grafurilor. Este vorba, de determinarea drumului hamiltonian de lungime minima dintr-un graf. Acest mod de functionare a SFF genereaza cele mai mici costuri de tranzitie si realizeaza toate tipurile de produse din SFF. În revers, sistemul în aceasta maniera de functionare nu raspunde însasi menirii sale, adica de a reactiona rapid la cereri aleatoare de produse. Tipurile de produse se succed, aici, într-un ciclu bine determinat, iar dupa executia tuturor tipurilor de piese ciclul se reia.

2. A doua modalitate de lucru a SFF este cu intrari aleatoare ale tipurilor de produse în sistem.

În aceasta lucrare este simulata prima metoda de functionare a unui SFF. Obiectivul este de a se stabili o succesiune de intrare a tipurilor de produse în sistem astfel încât sa fie minimizat efortul global de flexibilitate al SFF.

Pentru aceasta s-au utilizat cele doua metode (euristici) propuse în [1,2].

Euristica nr. 1 se mai numeste si metoda descompunerii în componente tare conexe. Pentru a verifica calitatea solutiei obtinute prin euristica nr.1, se face o verificare utilizând problema cuplajului maximal.

Euristica nr. 2 este metoda matricilor latine.

Aplicatia aferenta acestei simulari este dezvoltata în mediul JAVA si implementeaza cele trei metode frecvent utilizate în domeniul teoriei grafurilor si anume: prima euristica (denumita si metoda lui Chen) pentru descompunerea unui graf în componente tare conexe, metoda de calcul a drumurilor hamiltoniene prin înmultirea latina, precum si metoda de calcul a cuplajului maximal de valoare minima a unui graf prin metoda acoperirii zerourilor.

Programul principal al aplicatiei este proba1.java, care prezinta o interfata prin intermediul careia se poate preciza dimensiunea grafului si se poate selecta una dintre metodele mai sus mentionate.

Aceasta interfata a fost realizata folosind pachetul javax.swing, pachet special al limbajului de programare JAVA ce contine componentele pentru interfete grafice. Dintre componentele puse la dispozitie, s-a utilizat componenta Jframe (fereastra în sine), JtextBox (mesajele si câmpul pentru citirea dimensiunii grafului) si Jbutton (pentru crearea celor trei butoane corespunzatoare celor trei metode).

Dupa introducerea dimensiunii matrice, se apasa unul dintre cele 3 butoane, pentru alegerea metodei dorite.

Fiecare metoda este implementata de câte o clasa separata. Astfel, pentru metoda descompunerii în componente tare conexe avem clasa Eur1, pentru calculul drumurilor hamiltoniene avem clasa MatriciLatine si pentru calcul cuplajului de valoare minima avem clasa Acoperire.

Clasa Eur1 realizeaza descompunerea grafului în componente tare conexe. Dupa cum este prezentat si mai jos, se preiau datele din formularul de completare a tabelului precum si valoarea maxima a arcelor care se vor valida. În rest, se completeaza cu valoarea 0.

Pasul 1. Se preia matricea patratica.

Pasul 2. Se preia valoarea arcului maxim.

Pasul 3. Se elimina arcele mai mari decât arcul maxim introdus;

Pasul 4. Se realizeaza o copie a tabelului

Pasul 5. Construirea vectorilor linie Vi'

Pasul 6. Construirea vectorilor coloana Vj'

Pasul 7. Calculul componentelor tare conexe Ci

Pasul 8. Se scrie matricea de adiacenta A

Pasul 9. Se calculeaza matricea drumurilor D

Daca exista un indice i cu $d_i = 1$ atunci graful are circuite, nu se poate aplica algoritmul si algoritmul se opreste. Daca nu, se trece la pasul 10.

Pasul 10. Se calculeaza puterile de atingere pentru fiecare componenta tare conexa.

Pasul 11. Daca nu se verifica relatia n*(n-1)/2 = numar $_i$ atunci graful nu are drumuri hamiltoniene si algoritmul se opreste, altfel se trece la pasul 12.

Pasul 12. Se ordoneaza nodurile în ordinea descrescatoare a puterilor lor de atingere si obtinem drumul hamiltonian cautat.

Daca nu se gaseste drum hamiltonian, se poate relua algoritmul, marind valoarea lungimii pentru care validam un arc sau întroducând alte cuplaje în matrice.

Si clasa Acoperire, la fel ca si clasa MatriciLatine va porni pe un nou fir de executie initiat de thread-ul AcoperireThread, pentru a nu se încurca cu meniul principal. Asemanator cu celelalte metode, si în acest caz se va introduce matricea grafului, dupa care la apasarea butonului Calculeaza, se porneste algoritmul de calcul al cuplajului de valoare minima.

Pasii algoritmului sunt:

Pasul 1. Se preia matricea patratica.

Pasul 2. Se scade din fiecare linie minimul acesteia apoi, în matricea obtinuta, din fiecare coloana minimul acesteia (se poate face si invers, rezultatul final va fi acelasi).

Pasul 3. În ordinea crescatoare a numarului de zerouri si de sus în jos (în cazul existentei mai multor linii cu acelasi numar de zerouri) se încadreaza pentru fiecare linie zeroul a carui coloana contine cele mai putine zerouri (primul de la stânga dintre acestea, în caz de egaliate) si se bareaza celelalte zerouri de pe linia si coloana acestuia. Pe parcursul algoritmului sunt luate în considerare la numarare doar zerourile neîncadrate si nebarate înca. În final, pe fiecare linie si pe fiecare coloana va fi cel mult un zero încadrat. Daca în final sunt n (= dimensiunea matricei) zerouri, atunci arcele corespunzatoare formeaza cuplajul cautat. Daca sunt mai putine, se trece la pasul 4.

Pasul 4. La acest pas se va stabili numarul minim posibil de linii si coloane care sa contina toate zerourile matricii. În acest sens vom proceda astfel:

a) se marcheaza liniile care nu au nici un zero încadrat;

b) se marcheaza coloanele care au un zero barat pe o linie marcata;

c) se marcheaza liniile care au un zero încadrat pe o linie marcata (daca exista);

Se repeta operatiile b) si c) pâna nu mai poate fi marcata nici o linie si nici o coloana.

Pasul 5. Se taie liniile nemarcate si coloanele marcate.

Pasul 6. Se împart elementele matricei în trei grupe:

G_1 = elemente aflate la intersectii de linii netaiate cu coloane netaiate;

G_2 = elemente situate la intersectii de linii taiate cu coloane netaiate sau de linii netaiate cu coloane taiate;

G_3 = elemente situate la intersectii de coloane taiate cu linii taiate

Pasul 7. Se gaseste minimul grupei G_1, care se scade din fiecare element al lui G_1 si se aduna la fiecare element al grupei G_3. Elementele grupei G_2 ramân neschimbate.

Pasul 8. Se reia algoritmul de la pasul 3.

Clasa MatriciLatine va fi pornita pe un fir de executie diferit de firul de executie al programului principal. Din aceasta cauza, am introdus thread-ul MatriciLatineThread, care initializeaza un nou fir de executie si porneste clasa MatriciLatine.

Aceasta clasa preia din programul principal dimensiunea grafului, si creaza o matrice patratica de aceasta dimensiune. Utilizatorului i se deschide o noua fereastra, în care va trebui sa întroduca matricea booleana a grafului. Componenta swing folosita pentru crearea tabelului este Jtable. Dupa acest pas, prin apasarea butonului Calculeaza, se porneste algoritmul de calcul a înmultirilor latine.

Pasii algoritmului sunt:

Pasul 1. Preluarea matricii.

Pasul 2. Construirea matricii booleene.

Pasul 3. Construirea matricii latine T, dupa legea:

T[i][j] = ij daca exista arcul ij

0 daca arcul nu exista

Pasul 4. Construirea matricii latine reduse T* pastrând doar cel de al doilea caracter din T

T*[i][j] = j daca exista arcul ij
0 daca arcul nu exista

Pasul 5. Se calculeaza succesiv matricile:

$T^2 = T$ L T*

$T^3 = T^2$ L T*

……

$T^{n-1} = T^{n-2}$ L T, unde n este dimensiunea matricii si L reprezinta înmultirea latina. Matricea T^{n-1} contine toate drumurile hamiltoniene din graf.

S-a ales solutia listelor dinamice. Astfel fiecare element din matrice este de fapt o lista.

Aceasta metoda a înmultirii latine (algoritmul lui Kaufmann) este utila, mai ales, în cazul grafurilor tare conexe, unde alti algoritmi nu sunt eficienti. Totusi, metoda este greu de aplicat pentru grafuri cu un numar mare de noduri.

Pentru a exemplifica, mai jos sunt prezentate de ferestrele de dialog ale aplicatiei.

Astfel, la intrarea în aplicatie apare o fereastra în care trebuie introdusa dimensiunea matricii si aleasa metoda de calcul a drumului hamiltonian.

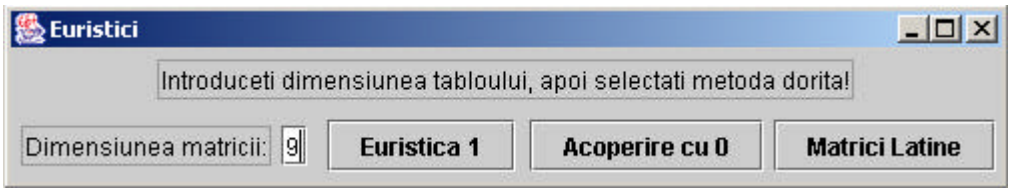

Figura 1. Fereastra de start a aplicatiei Euristici

Dupa cum se observa în Figura 2 înainte de accesarea butonului Calculeaza este necesar sa se introduca de la tastatura marimea arcului la care dorim facem calculul Euristica 1.

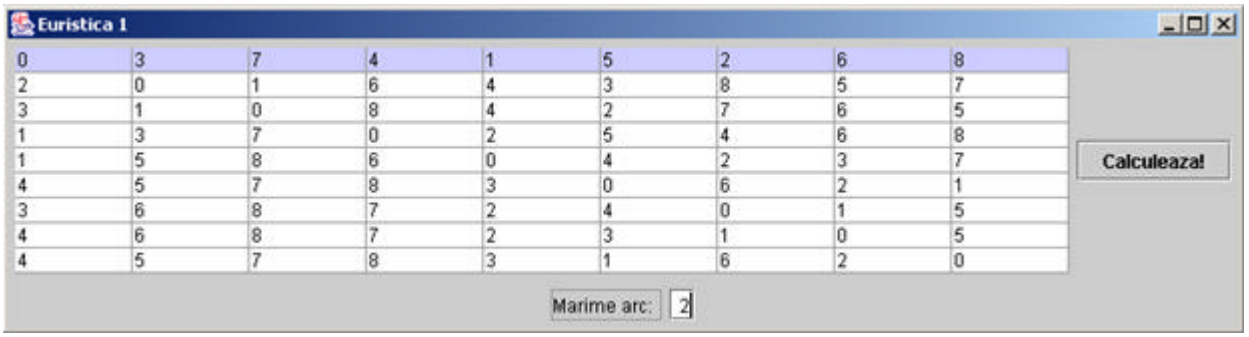

Figura 2. Fereastra de introducere a matricii pentru calculul Euristica 1

Afisarea rezultatului indica faptul ca nu exista drum hamiltonian.

Figura 3. Rezultatul calculului Euristica 1

Se introduce un cuplaj pe prima linie coloana a patra.

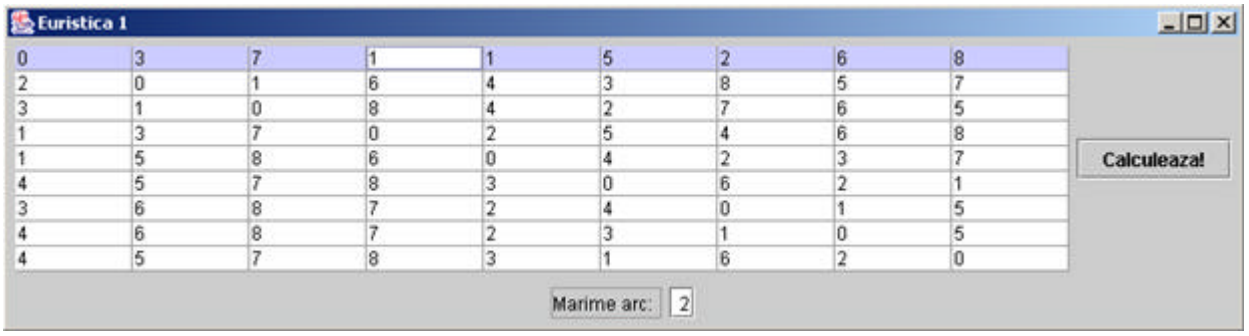

Figura 4. Introducerea unui cuplaj în calculul Euristica 1

Prin introducerea cuplajului aplicatia va genera un nou rezultat; de aceasta data s-a gasit drum hamiltonian.

Figura 5. Rezultatul dupa introducerea cuplajului în calculul Euristica 1

Calculul Acoperire cu 0 necesita introducerea unei matrici, care pe diagonala în loc de 8 va contine un numar mai mare decât cel mai mare cost de tranzitie continut de matrice.

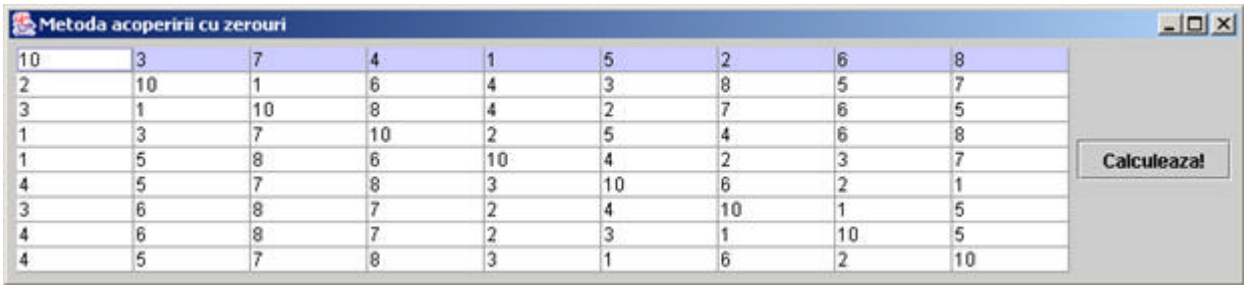

Figura 6. Fereastra de introducere a matricii pentru calculul Acoperire cu 0

Rezultatul generat de calculul Acoperire cu 0 este unul pozitiv, în sensul ca au fost gasite solutii.

Figura 7. Rezultatul calculului Acoperire cu 0

Pentru calculul Matrici Latine s-a introdus matricea din figura de mai jos.

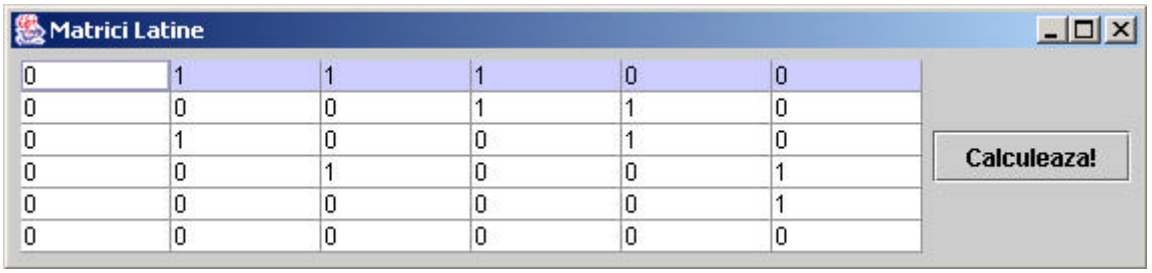

Figura 8. Fereastra de introducere a matricii pentru calculul Matrici Latine

S-au gasit doua solutii care sunt prezentate în Figura 9.

Figura 9. Rezultatul calculului Matrici Latine

În concluzie, utilizarea unei aplicatii software care sa rezolve problema calculului drumului hamiltonian conduce la economii importante de resurse, atât materiale cât si financiare, fara ca sa afecteze negativ calitatea procesului de calcul.

Portabilitatea aplicatiei Euristici este data de platforma folosita la dezvoltarea ei, si anume JAVA. Având o interfata intuitiva, aplicatia este usor de folosit.

BIBLIOGRAFIE

1. Abrudan, I., - *Economia proiectarii sistemelor flexibile de fabricatie – Îndrumator de proiectare*, Editura UTC-N, Cluj – Napoca, 1993.
2. Abrudan, I., - *Sisteme flexibile de fabricatie. Concepte de proiectare si management*, Editura Dacia, Cluj – Napoca, 1996
3. Ionescu, T., - *Grafuri – aplicatii Vol.I*, Editura Didactica si Pedagogica, Bucuresti, 1973.
4. Vaduva, M.,C., - *Programarea în JAVA*, Editura Microinformatica, Cluj – Napoca, 1999