O abordare evolutivă a planificării proceselor

Ionuț Balan, Sorin Vlad

Abstract—**This paper aims to emphasize the results obtained by using genetic algorithms in scheduling comparatively with other classical methods used in the same purpose. A scheduling problem is defined by a set of jobs and a set of resources. The jobs are constrained by the relation of precedence forcing some of the jobs to wait until the other jobs are completed. The goal is to find a schedule minimizing the overall completion time. As case study is considered the problem of scheduling n jobs on m machines, each job having a certain running time on each machine.**

Index Terms—**genetic , neh, offspring, flow shop, random**

I. INTRODUCERE

Datorită numeroaselor aplicații din industrie şi economie, problema flow-shop a fost studiată de către numeroşi cercetători care au impus diverse limitări problemei, au definit diverse funcții obiectiv şi au implementat diverse tehnici de optimizare.

La baza unei probleme flow-shop clasice stau două elemente: M maşini (în general, sau calculatoare în particular) şi N operații de prelucrat de către aceste maşini. Secvența de procese, ce compune fiecare din cele N operații, are aceeaşi ordine de execuție pe fiecare dintre cele M maşini. Fiecare operație poate fi executată pe o singură maşină la un moment dat şi fiecare maşină poate executa doar o singură operație la un moment dat. Fiecare operație este prelucrată o singură dată pe o singură maşină. Problema principală constă din determinarea ordinii şi a timpilor de execuție a operațiunilor astfel încât timpul total de execuție al acestora să fie minim. Se urmăreşte planificarea proceselor fiecărei operații pe fiecare dintre cele M maşini astfel încât să se evite cât mai mult posibil apariția unor procese care sunt nevoite să aştepte execuția pe o anumită maşină.

Dacă numărul de maşini este M=2 atunci problema se poate rezolva cu ajutorul algoritmului lui Johnson. Dacă numărul de maşini este M=3 atunci problema este NP complexă şi pune serioase probleme de calcul. În acest caz există $(n!)^m$ alternative de secvențiere a operațiilor.

Regula lui Johnson este cea mai cunoscută regulă optimă aplicabilă unei clase largi de probleme flow-shop. Regula lui Johnson afirmă că operația i precede operația j într-o secvență optimă dacă $\min\{t_{i1}, t_{j2}\} \le \min\{t_{i2}, t_{j1}\}$. Cu ajutorul regulii lui Johnson problema flow-shop pentru M=2 poate fi rezolvată astfel:

Pasul 1: Determină $\min_i \{t_{i1}, t_{i2}\}$

Pasul 21 Dacă operația cu timpul minim de prelucrare necesită maşina 1, plasează operația asociată pe prima poziție disponibilă din secvența de execuție. Salt la pasul 3.

Pasul 22 Dacă operația cu timpul minim de prelucrare necesită maşina 2, plasează operația asociată pe prima poziție disponibilă din secvența de execuție. Salt la pasul 3.

Pasul 3: Se şterge operația şi se revine la pasul 1 până când toate pozițiile din secvența de execuție sunt completate.

II. ALGORITMUL NEH

Nawaz a propus algoritmul NEH, ideea centrală a acestuia fiind aceea că unei operații cu un timp total de execuție mai mare pe toate maşinile trebuie să i se acorde o mai mare prioritate decât unei operații cu un timp total de execuție mai mic. În consecință, operațiile se vor aranja în ordinea descrescătoare a timpului total de execuție calculat, de exemplu pentru operația j ca fiind $\sum_{j=1}^{m} t_{ij}$.

Algoritmul NEH presupune apoi secvențierea parțială a operațiilor bazată pe prioritățile stabilite anterior. O secvență va fi construită prin introducerea, la un moment dat, a unei operații neplanificate în schema de planificare parțială. Dacă în schema parțială sunt k operații, în momentul introducerii fiecărei operații se alege cea mai bună secvență parțială dintre cele k+1 secvențe parțiale posibile. Noua operație poate fi plasată în una dintre cele k+1 poziții din secvența parțială. După alegerea celei mai bune poziții din secvență pentru operația respectivă, ținând cont şi de durata de execuție obținută, această porțiune din secvența parțială va rămâne fixă şi se va insera o nouă operație. Locul de plasare a operației în secvență se stabileşte în funcție de timpii de prelucrare ai tuturor maşinilor, luați în ordine descrescătoare

Algoritmul NEH poate fi prezentat pe scurt astfel:

*) Se ordonează cele n operații în ordinea descrescătoare a sumei timpilor de execuție pe cele m maşini.

*) Se iau primele două operații şi se ordonează astfel încât să se minimizeze timpul total de execuție. În această etapă se consideră că secvența este formată doar din cele două operații.

Ionuț Bălan este preparator în cadrul Catedra de Informatică a Facultății de Ştiințe Economice şi Administrație Publică din cadrul Universității „Ştefan cel Mare" Suceava. (email: ionutb@seap.usv.ro).

Sorin Vlad este lector în cadrul Catedrei de Informatică a Facultății de Ştiințe Economice şi Administrație Publică, Universitatea "Ştefan cel Mare" Suceava (email: sorinv@seap.usv.ro).

Pentru k= 1, n execută
*) Inserează operaţia k în poziţia corespunzătoare din schema parţială din cele k posibile care minimizează timpul parţial de execuţie.

Deşi algoritmul este de complexitate ($O(n^3m)$) soluţiile obţinute sunt bune.

În rezolvarea problemei flow-shop, algoritmul NEH este privit ca cea mai bună soluţie[4]. Algoritmul este aplicat de obicei pentru a determina o soluţie iniţială ce va fi folosită ca punct de plecare în metode de îmbunătăţire a rezultatelor a rezultatelor cum ar fi *tabu search* [4] sau algoritmii genetici [6].

Principalul inconvenient al algoritmului NEH constă din faptul că este necesară evaluarea unui număr de $[n(n+1)/2]-1$ secvenţe parţiale de execuţie. Timpul de execuţie creşte rapid odată cu creşterea dimensiunilor problemei.

Pentru a reduce timpul de execuţie, Taillard [5] a găsit o modalitate de îmbunătăţire a algoritmului prin calculul secvenţelor parţiale pentru fiecare iteraţie. Timpul de execuţie se reduce însă secvenţele parţiale vor trebui de asemenea reevaluate.

III. ALGORITMI GENETICI

Inteligenţa Artificială (AI) poate fi considerată ca fiind acel domeniu al Informaticii ce are ca obiectiv proiectarea sistemelor înzestrate cu anumite proprietăţi pe care în mod obişnuit le asociem inteligenţei umane: înţelegerea limbajului, învăţarea, raţionamentul, rezolvarea problemelor , demonstrarea teoremelor, etc.[1]

Algoritmii genetici reprezintă o importantă clasă din cadrul metodelor de căutare şi optimizare. Ei reprezintă o transpunere în programare a principiilor biologice.

Aceşti algoritmi se deosebesc de tehnicile tradiţionale de căutare prin următoarele caracteristici:

- optimizarea se face prin extragerea de noi puncte în spaţiul de căutare, precum şi exploatarea informaţiilor din punctele găsite;
- au proprietatea de a fi paralelizaţi. Efectul lor este echivalent cu o căutare de soluţii multiple din spaţiul avut la dispoziţie, fără a testa toate valorile acestui spaţiu;
- sunt algoritmi ai căror soluţii sunt obţinute prin aplicarea unor operatori specifici asupra anumite valori generate aleatoriu;
- acţionează asupra mai multor soluţii simultan.

Structura acestui tip de algoritm este următoarea:

Pasul 1. Generarea populaţiei iniţiale (generare făcută aleatoriu, această populaţie fiind supusă unor transformări pe parcursul procesului, după un anumit număr de paşi fiind obţinută soluţia finală);

Pasul 2. Selecţia – Algoritmul genetic selecţionează indivizii unei populaţii în funcţie de performanţele lor. Indivizii selecţionaţi vor reprezenta o populaţie intermediară, populaţie a cărei cromozomi vor reprezenta "părinţii" noii generaţii.

Pasul 3. Aplicarea operatorilor genetici – În acest pas o serie de operatori genetici sunt aplicaţi indivizilor selectaţi la pasul 2. Printre cei mai utilizaţi operatori sunt cei de mutaţie şi cei de încrucişare. În funcţie de problema studiată pot apărea şi alţi operatori, cum ar fi: operatorul de inversare, operatorul de reordonare, etc.

Pasul 4. Selectarea populaţiei care va părăsi sistemul;

Pasul 5. Repetarea paşilor 2,3,4, până la satisfacerea criteriului de optimalitate sau parcurgerea unui anumit număr de generaţii.

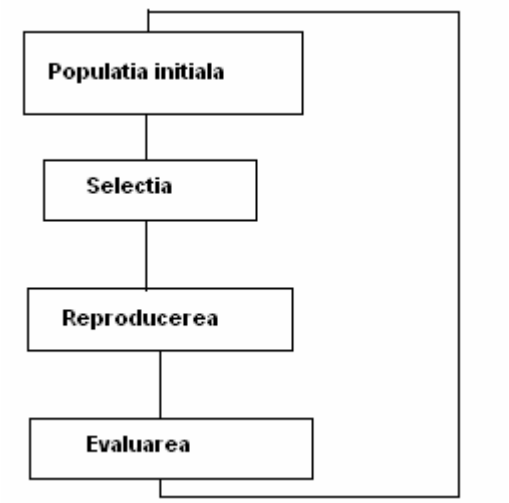

Fig. 1 Structura algoritmului genetic

Structura din figura 1 este una generală pentru acest tip de algoritmi, de aceea programatorul va trebui să adapteze acest model în funcţie de necesităţi. Pentru aceasta trebuie să cunoască rolul fiecărei noţiuni în obţinerea rezultatului final. Astfel vom avea:

Populaţia reprezintă un set de posibile soluţii pentru problemă. De exemplu, dacă dorim să obţinem minimul unei funcţii f: Rm → R, atunci fiecare obiect x ce aparţine mulţimii Rm poate fi o posibilă soluţie. De asemenea, orice subset P din mulţimea R poate reprezenta populaţia supusă prelucrărilor.

Indivizii (cromozomii) – reprezintă elementele componente ale populaţiei. Din definiţia populaţiei putem desprinde faptul că "x" ,fiind un punct din mulţimea Rm, reprezintă un individ (cromozom).

Urmaşii (*offsprings*)- sunt indivizi obţinuţi prin aplicarea operatorilor genetici asupra populaţiei studiate. Aceşti operatori diferă în funcţie de problema studiată. De exemplu, un operator de încrucişare aplicat următorilor părinţi:

$$(x1, x2,..., xk , xk+1,..., x\ m)$$
$$(y1, y2,..., yk , yk+1,..., y\ m),$$

poate duce la obţinerea următorilor urmaşi:

$(x1, x2,..., xk , yk+1,..., y m)$
$(y1, y2,..., yk , xk+1,..., x m).$

În exemplu de mai sus se observă faptul că încrucişarea s-a realizat pe poziția k+1.

Operatorii genetici reprezintă orice schimbare a vreunui cromozom. Operatorii pot fi aplicați asupra populației într-un mod aleatoriu, o mare influență asupra acestora având-o problema modelată.

Selecția reprezintă operația prin care anumiți indivizi vor fi aleşi drept părinți pentru generarea urmaşilor. Sunt selectați cei mai buni indivizi, ținându-se cont de funcția *fitness* folosită.

Finalitatea este determinată de scop. Algoritmul se poate opri după un anumit număr de paşi, sau în momentul în care este atins un prag de eroare.

În implementarea algoritmilor genetici trebuie să ținem seama şi de următoarele particularități[2]:

1. Algoritmii genetici sunt o clasă de algoritmi probabilişti care combină elemente de căutare dirijată şi căutare aleatoare. Ei realizează un echilibru aproape optim între explorarea spațiului stărilor şi exploatarea celor mai bune soluții găsite.
2. Algoritmii genetici sunt mai robuşti decât alte metode existente de căutare dirijată şi decât algoritmii clasici de optimizare.
3. Metodele de căutare bazate pe algoritmii genetici sunt caracterizate de faptul că ele mențin o populație de soluții potențiale. Metodele clasice de căutare acționează la un moment dat asupra unui singur punct din spațiul de căutare.
4. De regula, algoritmii genetici lucrează cu o codificare a elementelor din spațiul stărilor problemei şi nu acționează direct asupra elementelor acestui spațiu.
5. Algoritmii genetici folosesc funcții de performanță obținute prin transformări simple ale funcției obiectiv. Nu este necesar ca funcția obiectiv să fie derivabilă. Nu sunt necesare nici proprietăți speciale de convexitate ale funcției obiectiv.
6. Algoritmii genetici sunt simplu de folosit.
7. Algoritmii genetici pot găsi soluții optime(sau aproape optime) cu o mare probabilitate.

IV. Rezultate obținute

În cele ce urmează vom prezenta rezultatele obținute în urma aplicării algoritmilor genetici asupra unor matrici ce conțin timpii de rulare a unor procese pe anumite maşini. De asemenea se face şi o comparație a rezultatelor obținute prin folosirea acestui algoritm cu rezultatele obținute prin algoritmul NEH, prezentat mai sus.

Operatorii genetici folosiți pentru optimizarea problemei studiate sunt mutația şi încrucişarea. În paragrafele următoare se prezintă principalii operatori ce determină regenerarea populației inițiale.

Considerăm reprezentarea din figura 2 ca fiind cromozomul inițial, supus mutațiilor.

6 1 9 3 4 5 0 7 8 2

Fig. 2 Cromozomul inițial

Primul operator de mutație îşi va face efctul în urma generării aleatoare a două gene ce alcătuiesc un cromozom. Prin interschimbarea acestor două gene va rezulta cromozomul copil. În figura 3 avem cromozomul copil obținut prin interschimbarea genei 4 cu gena 9 din cromozomul inițial.

6 1 **4** 3 **9** 5 0 7 8 2

Fig. 3 Cromozomul copil obținut prin interschimbarea dintre două gene ale cromozomului inițial

Pentru un al doilea operator de mutație pe care îl vom folosi vom recurge la găsirea timpului maxim de aşteptare de pe ultimul procesor. Vom căuta acest timp maxim dorind să recurgem la eliminarea lui, prin poziționarea genei ce l-a provocat ca primă genă a cromozomului. Cromozomul obținut va avea forma celui din figura 4, considerând că timpul maxim de aşteptare a fost atins la gena identificată prin 3.

3 6 1 9 4 5 0 7 8 2

Fig. 4 Cromozomul obținut prin aplicarea celui de-al doilea operator de mutație

Tot pentru a elimina acel timp maxim putem recurge la rearanjarea genelor începând cu ultima situată înainte de locația unde se atinge timpul maxim şi până la sfârşitul cromozomului. Rezultatul obținut poate fi unul de forma celui din figura 5.

6 1 0 2 4 **9** 5 **3** 7 8

Fig. 5 Cromozom copil obținut prin mutație.

Pe lângă operatorul de mutație, un alt operator des folosit este cel de încrucişare. În exemplu nostru generăm doi cromozomi pe care îi vom supune operației de încrucişare. Cromozomii inițiali sunt cei din figura 6, iar rezultatele sunt prezentați în figura 7.

6 1 9 3 **4** 5 0 7 8 2
0 1 5 9 **2** 3 4 6 7 8

Fig. 6 Cromozomi supuşi operației de încrucişare.

6 1 9 3 **0 5 2 4 7 8**
0 1 5 9 **6 3 4 7 8 2**

Fig. 7 Cromozomi obținuți prin încrucişare.

După aplicarea operatorilor genetici asupra populației, datele trebuie ordonate ținând cont de valoarea funcției fitness calculată pentru fiecare cromozom. De asemenea, la fiecare

pas se recomandă generarea aleatoare a unui anumit număr de indivizi pentru a evita obținerea unui optim local pentru funcția de evaluare a cromozomilor.

Probabilitatea de mutație a unui cromozom este de 0,5 , în timp ce cea de încrucișare ajunge la 0,2.

În tabelul 1 sunt prezentate rezultatele obținute la planificarea proceselor pe un anumit număr de maşini cu ajutorul algoritmilor genetici şi algoritmului NEH.

Tabel 1 Rezultatele obținute, pentru diferite cazuri, folosind algoritmii genetici şi NEH

Nr. Procese x Nr. Maşini	NEH	Cel mai bun rezultat obținut cu AG	Medie obținută cu AG	Număr indivizi din populație	Număr generații
10 x 2	663	663	663	20	100
10 x 4	728	728	728	20	100
20 x 5	1373	1350	1350	40	100
20 x 10	1638	1611	1630	40	200
30 x 10	2179	2121	2153	50	200
50 x 10	3190	3132	3142	60	500
50 x 20	4082	3947	3975	100	1000
100 x 10	5898	5898	5898	100	1000
100 x 20	7064	6807	6875	200	1000
100 x 40	7064	6788	6827	400	1000

Se observă că în 3 dintre cazurile studiate rezultatele obținute prin aplicarea celor doi algoritmi coincid, în timp ce în alte 7 cazuri rezultatele obținute prin aplicarea algoritmilor genetici sunt mai bune. În funcție de problemă numărul de indivizi din populația studiată, precum şi numărul de generații diferă. Datele din tabelul 1 sunt reprezentate grafic în figura 8.

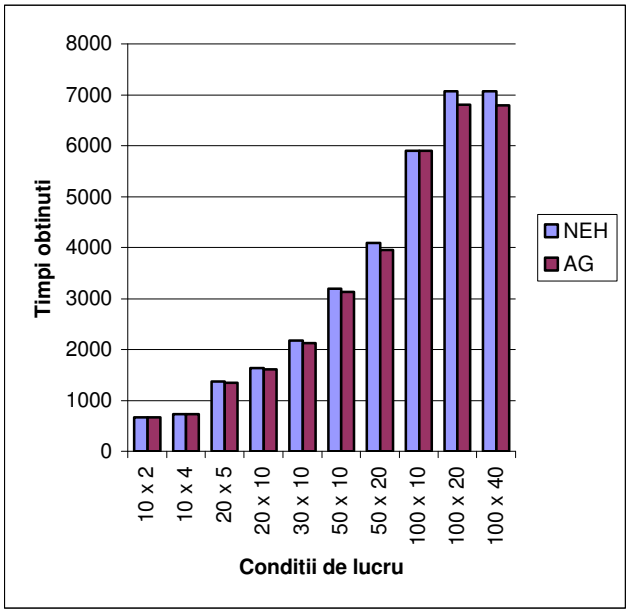

Fig. 8 Rezultatele celor 2 algoritmi.

Marele dezavantaj al algoritmilor genetici este dat de timpul de rulare mult mai mare decât timpul execuției unui algoritm clasic.

În figura 9 se observă viteza de convergență a algoritmilor genetici în cazul planificării a 50 de procese pe 20 de maşini , folosindu-se 500 de generații.

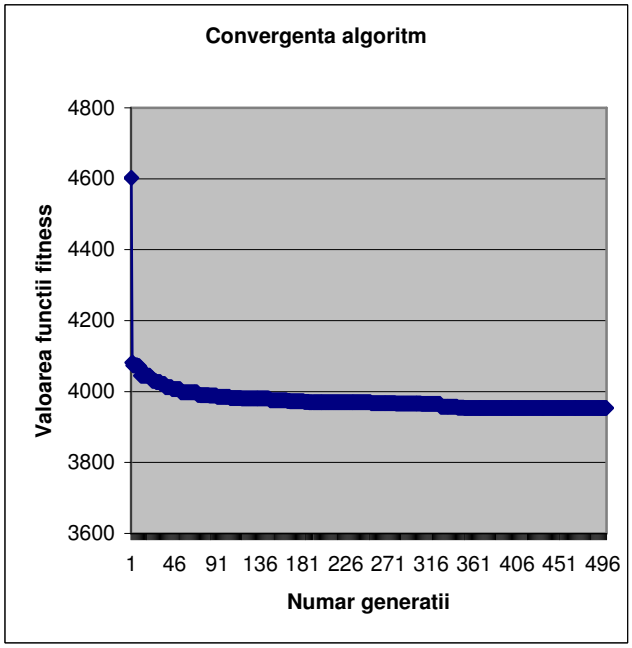

Fig. 9. Rezultate obținute pentru 50 procese x 20 maşini (500 generații).

V. CONCLUZII

Din cele prezentate mai sus se observă că algoritmii genetici furnizează rezultate destul de bune la anumite probleme de planificare. În cazul problemelor NP-complexe este recomandată folosirea acestui tip de algoritmi în defavoarea altor algortimi clasici. Problema flow-shop urmăreşte să planifice un număr de procese pe un anumit număr de maşini astfel încât timpul de rulare a acestora să fie cât mai mic. Pentru un număr mic de maşini există algoritmi care oferă rezultate foarte bune, cum ar fi algoritmul lui Johnson. Dar prin creşterea numărului de maşini problema devine din ce în ce mai complexă, acest lucru ducând la obținerea de rezultate nesatisfăcătoare în cazul folosirii unor metode clasice. În acest caz se recomandă folosirea algoritmilor genetici, chiar dacă timpul de rulare creşte destul de mult. Marea majoritate a problemelor de planificare se pot rezolva cu ajutorul acestor tehnici inspirate din selecția naturală. Algoritmii genetici presupun efectuarea unui număr mare de calcule şi acest lucru limitează utilizarea lor în anumite aplicații. Efortul de calcul este dependent de valorile setate pentru parametrii algoritmului: dimensiunea populației, presiunea de selecție, etc. Diminuarea, de exemplu, a dimensiunii populației ar determina reducerea timpului de execuție, dar ar putea afecta dramatic acuratețea rezultatelor algoritmului. O soluție pentru reducerea timpilor de execuție, care să nu afecteze acuratețea şi viteza de convergență a algoritmului, ar fi paralelizarea algoritmilor genetici. Acest lucru va duce, în final, la micşorarea timpului de rulare a aplicațiilor ce realizează planificarea dorită, în funcție de viteza de comunicație dintre procesoare. Paralelizarea poate fi realizată urmărind trei mari

direcţii: algoritmi genetici globali, algoritmi genetici bazaţi pe migrare şi algoritmi genetici bazaţi pe migraţie.

REFERINŢE

[1] Dumitrescu, D., Principiile inteligenţei artificiale, Ed. Albastră, 2002

[2] Dumitrescu, D., Algoritmi genetici şi strategii evolutive- apucaţii în Inteligenţa Artificială şi în domenii conexe, Ed. Albastră, 2006

[3] J. Grabowski, M. Wodecki**,** A very fast tabu search algorithm for the permutation flow shop problem with makespan criterium,, Computer & Operation Research, 31 (2004), 1891-1909

[4] Rubén Ruiz, Concepción Maroto, Javier Alcaraz, Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics, European Journal of Operational Research 165(1): 34-54 (2005)

[5] É.D. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research, 1990

[6] Wang L, Zheng DZ, A modified evolutionary programming for flow shop scheduling, International Journal of Advanced Manufacturing Technology, 2003