Algoritmi genetici seriali şi paraleli

Ing. Octavian PAIU
Catedra de Informatică Economică, A.S.E. Bucureşti
Dr.ing. Vladimir DUMITRESCU
Departamentul de Cercetări Economice, A.S.E. Bucureşti

În România, în ultima perioada de timp, a început să se manifeste un interes crescut în ceea ce priveste sistemele distribuite, acestea reprezentând cea mai adecvată alternativă la supercalculatoare, din punct de vedere al raportului preţ/performanţă. Mai ales dacă se ţine cont şi de faptul că achiziţia unor astfel de echipamente ramâne, deocamdată, prohibitivă din cauza preţului ridicat. Pe de altă parte, există în România un număr apreciabil de servere şi staţii de lucru puternice, conectate în reţele locale; dar, numărul de aplicaţii realizate este restrâns, în special în domeniul graficii, prelucrării/recunoaşterii de imagini şi aplicaţii multimedia.
În străinătate există deja standarde larg acceptate, pachete de programe şi instrumente specializate (cum ar fi MPI, PVM, TRAPPER) pentru generarea de aplicaţii distribuite. S-a acumulat o mare experienţă şi în domeniul algoritmilor genetici, o clasă de tehnici adaptive de căutare, în care munca de pionierat a fost efectuată acum mai bine de 20 ani de catre J. Holland şi studenţii săi. Aceşti algoritmi s-au dovedit a fi extrem de eficienţi pentru un larg domeniu de probleme de optimizare şi căutare (proiectarea aripilor de avion, cercetarea operaţională, robotica, optimizarea topologiilor de reţele neuronale, urmărirea stocurilor de piaţă, etc.).
Articolul de faţă reprezintă o sinteză a cercetărilor efectuate de autori în domeniul general al sistemelor distribuite, cu focalizare în special pe algoritmi genetici seriali şi paraleli (distribuiţi). Aceste cercetări s-au efectuat în cadrul unui grant acordat autorilor în anii 1996 şi 1998 de Ministerul Cercetării şi Tehnologiei.
Cuvinte cheie: *algoritm, evoluţie, operator genetic, adecvare, euristică, programare, încrucişare, mutaţie, genetică, populaţie, genă, cromozom, software, reproducere, simulare, optimizare, schemă, codificare, paralelizare.*

1. Prezentare generală

Termenul de **algoritm evoluţionar (AE)** (evolutionary algorithm) este folosit *generic* pentru a descrie sisteme computerizate de rezolvare a problemelor cu modele de calcul care au la baza proiectării lor mecanismele cunoscute ale *evoluţiei*.
Există o *gamă* relativ largă de AE printre care putem menţiona: algoritmii genetici; programarea evoluţionară; strategiile de evoluţie; sistemele clasificatoare; programarea genetică.
Toate aceste clase de algoritmi şi tehnici de programare au o bază conceptuală comună: *simularea evoluţiei* unor structuri individuale prin procesele de reproducţie, selecţie şi mutaţie. Aceste procese depind de *performanţa* structurilor individuale, performanţă definită într-un anume mediu.

Ca regulă quasi generală, AE definesc o *populaţie* de structuri individuale care evoluează conform operatorilor (regulilor) de selecţie sau altor operatori denumiţi uneori operatori de căutare (sau *operatori genetici*) cum sunt recombinarea şi mutaţia. Fiecare individ din populaţie primeşte o măsură a *adecvării* (fitness) la mediul înconjurător definit. Reproducerea îşi focalizează atenţia asupra indivizilor cu adecvare cât mai bună la mediu. Operatorii genetici de recombinare şi mutaţie perturbă indivizii, asigurând in acest fel *euristica* generală pentru explorare şi căutare.
Algoritmii genetici (AG) reprezintă atât o clasă de algoritmi evoluţionari cât şi tehnici de optimizare care au la bază conceptele AE. În acest mod de abordare variabilele sunt reprezentate ca genele dintr-un cromozom. Algoritmii genetici pot scoate în evidenţă un grup de soluţii candidat (populaţie) pe supra-

fața de răspuns. Se pot găsi cromozomii cu cea mai bună adecvare (fitness) prin selecție naturală sau prin utilizarea operatorilor genetici, cum sunt reproducerea, mutația, recombinarea.

Avantaje ale utilizării algoritmilor genetici: nu necesită cunoștințe precise sau informații de gradient despre suprafața de răspuns; eventualele discontinuități de pe suprafața de răspuns au un efect minimal asupra performanțelor globale ale optimizării; sunt rezistenți la blocajele, uzuale pentru alte metode de optimizare, într-un optim "local"; sunt foarte performanți pentru probleme de optimizare de mari dimensiuni; pot fi utilizați pentru o gamă largă de probleme de optimizare.

Dezavantaje ale utilizării algoritmilor genetici: în multe cazuri este dificilă determinarea exactă a optimului global; necesită, de regulă, un mare număr de evaluări ale funcției de răspuns (adecvare); configurarea AG este uneori dificilă și necesită stabilirea unui număr mare de parametri.

Un model de funcționare a unui AG simplu poate fi descris ca având patru pași sau etape:

1. Se creează **populația inițială de cromozomi**, fie prin generare aleatoare, fie prin perturbarea unui cromozom de "intrare";
2. Se calculează funcția de adecvare. Scopul funcției de adecvare este codificarea cantitativă (numerică) a performanței unui cromozom oarecare. Pentru aplicațiile de optimizare din lumea reală alegerea "corectă" a funcției de adecvare reprezintă pasul cel mai critic.
3. În această etapă cromozomii cu valorile (indice de adecvare) cele mai mari ale funcției de adecvare se plasează o dată sau de mai multe ori, semialeator, într-o submulțime de împerechere. Cromozomii cu indici de adecvare mici sunt, în continuare, înlăturați din populație.
4. Explorarea constă de regulă din operatori de recombinare (împerechere), uneori numai reproducere, și eventual aplicarea operatorului de mutație.

După încheierea pasului (etapei) de explorare s-a obținut o nouă generație (de fapt o nouă populație) formată din cromozomi nou creați (urmași) și eventual cromozomi părinți reproduși fără împerechere. Se pot repeta etapele 2, 3, 4 până la obținerea unui număr suficient de generații care au caracteristici (gene) din ce în ce mai bune.

Modul de reprezentare (sau codificare) a variabilelor care se optimizează prin AG are o importanță deosebită asupra performanțelor de exploatare (căutare), deoarece procesul de optimizare se aplică reprezentărilor variabilelor și nu valorilor "matematice". Cele mai obișnuite metode de reprezentare (codificare), *codificarea binară și codificarea în numere reale* diferă între ele în principal prin modul de aplicare a operatorilor genetici - crossover sau/și mutație.

În codificarea binară fiecare cromozom este un vector de biți, fiecare bit reprezentând o genă. Modul de reprezentare prin numere reale este mai simplu din punct de vedere conceptual. Fiecare variabilă care se optimizează este reprezentată ca un număr în virgulă mobilă convențional. Se construiește astfel o relație de tipul *o genă - o variabilă în virgulă mobilă.*

Metodele euristice de căutare a soluțiilor folosite în AG au la bază *teorema schemelor.* Se *definește* o **schemă *H*** ca fiind o machetă (șablon, template) care descrie o submulțime de cromozomi (indivizi din populație) cu secțiuni genetice similare. Schemele care au un ordin de definiție mic și au indicele de adecvare peste medie sunt "preferate" și sunt denumite *blocuri fundamentale* (building blocks).

În contextul utilizării efective a AG pentru rezolvarea unei anumite probleme specifice de optimizare se poate pune o întrebare esențială: *este oare posibilă utilizarea cu succes a unui AG pentru rezolvarea "acelei" probleme ?*

Mulți cercetători au scris despre avantajele AG în comparație cu alte metode și tehnici de optimizare. Totuși, dacă se iau în considerare și cele scrise de "practicanții" unor alte metode, se poate constata că, în multe cazuri, avantajele AG nu sunt chiar atât de "puternice". Există unele probleme, denumite **AG-dificil**, care prezintă dificultăți deosebit de mari la implementarea unui AG corespunzător (cu performanțe cel puțin acceptabile). Una din direcțiile principale de studiu al AG este *depistarea unor metode de*

determinare apriori a dificultății AG pentru o problemă dată.

Pentru problemele la care nu se știe prea mult despre suprafața de răspuns și la care calculul gradientului este fie mare consumator de timp de calcul, fie instabil numeric, mulți cercetători preferă să utilizeze metode de optimizare cum sunt AG, optimizarea simplex ș.a, care nu necesită informații de gradient pe suprafața de răspuns. Este preferabilă utilizarea AG datorită *versatilității și flexibilității acestora.*

Un exemplu pentru care AG nu sunt recomandabili îl reprezintă problemele de optimizare în care calculul vectorului gradient este precis și rapid. AG funcționează și pentru astfel de aplicații, dar atingerea regiunii optimale se realizează mult mai lent.

2. Teorema fundamentală a algoritmilor genetici

Se consideră că indivizii unei populații de dimensiunea n sunt reprezentați de șiruri de o anumită lungime, l, construite pe alfabetul binar $V = \{0,1\}$. De exemplu, șirul binar de 7 biți $A = 0111010$ poate fi reprezentat simbolic prin $A = a_1a_2a_3a_4a_5a_6a_7$.

Pentru discutarea și clasificarea similitudinii șirurilor unei populații se utilizează conceptul de șablon de similitudine (schema) care este un șir "generic" de aceeași lungime cu șirurile populației și care reprezintă toate șirurile care au aceleași caracteristici în anumite poziții. O schemă se reprezintă pe alfabetul $V+ = \{0, 1, *\}$, unde simbolul * poate fi 0 sau 1, în pozițiile în care apare. De exemplu, fie schema de 7 biți, $H = *11*000$. Șirul $A = 0111000$ este un exemplu al schemei H, deoarece valorile genelor a_i sunt identice cu valorile h_i în pozițiile fixe 2, 3, 5.

Pentru un șir binar de lungime l se pot defini 3^l scheme diferite. In general, pentru alfabete de cardinalitate k există $(k + 1)^l$ scheme. Într-o populație de șiruri cu n membri există $n*2^l$ scheme conținute în populație, deoarece fiecare șir este el însuși un exemplu (un reprezentant) a 2^l scheme. Toate aceste considerații asupra numărului de scheme posibile pun în evidență ordinul de mărime al informației care trebuie procesată de algoritmii genetici.

Nu toate schemele sunt identice, unele sunt mai "specifice" decât altele. De exemplu, schema 011*1** este o expresie a unei cerințe de similitudine mult mai puternice decât în schema 0******.

Pentru cuantificarea acestor aspecte se introduc două proprietăți ale schemelor: *ordinul și lungimea de definire. Ordinul* unei scheme H, notat prin $o(H)$, este numărul de poziții fixe (numărul de biți 0 și 1) prezente în șablon. *Lungimea de definire* a unei scheme H, notată $\delta(H)$, este distanța între prima și ultima poziție specifică în șir.

Efectul **reproducerii** asupra unui număr așteptat de scheme într-o populație poate fi determinat astfel:

Fie m numărul de exemple, la momentul (generația) t, ale unei scheme particulare H conținută într-o populație $A(t)$. Putem scrie $m=m(H,t)$ (m poate fi diferit la diferite momente de timp t). Pe durata reproducerii, un șir este "copiat" funcție de adecvarea sa (fitness), sau mai exact, un șir A_i este selectat cu probabilitatea $p_i = f_i/\Sigma f_j$.

Ecuația de creștere prin reproducere a unei scheme devine:

$$m(H, t+1) = m(H, t).f(H)/\Sigma f$$

O anumită schemă crește cu raportul între adecvarea medie a schemei și adecvarea medie a populației. Această comportare se desfășoară în paralel pentru toate schemele H conținute într-o anumită populație A.

Se poate obține și ecuația:

$$m(H,t) = m(H,0).(1+c)^t$$

Aceasta reprezintă forma discretă a funcției exponențiale (progresia geometrică) și exprimă faptul că efectul reproducerii constă în alocarea exponențială crescătoare (descrescătoare) a numărului de încercări peste (sub) schemele medii.

Numai reproducerea singură nu promovează explorarea unor noi regiuni ale spațiului de căutare; prin simpla copiere a unor structuri vechi, fără modificarea acestora, nu pot fi încercate altele noi.

Încrucișarea (Crossover) este un schimb de informații structurat, dar încă aleatoriu, între șiruri. El creează structuri noi, fără a afecta serios strategia de alocare dictată de reproducerea în sine.

Se poate calcula o limită minimă a probabilității, p_s, de supraviețuire după încrucișare,

pentru orice schemă. Deoarece o schemă supraviețuiește atunci când zona de încrucișare se află în afara lungimii de definire, probabilitatea de supraviețuire după încrucișare simplă este $p_s = 1 - \delta(H)/(l-1)$. Dacă se ia în considerare efectul *combinat* al reproducerii și încrucișării suntem "interesați" în calcularea numărului așteptat de exemplare, în generația următoare, al unei scheme particulare *H*. Presupunând că operațiile de reproducere și încrucișare sunt *independente*, se obține estimarea:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1}\right]$$

Mutația reprezintă o alterare aleatoare, cu probabilitatea p_m, a unei singure poziții dintr-un șir. Pentru ca o schemă *H* să supraviețuiască, toate pozițiile sale fixe trebuie să supraviețuiască.

Pentru o schemă particulară *H* numărul așteptat de copii, primit în generația următoare, după operațiile *de reproducere, încrucișare și mutație, luate împreună*, este dat de ecuația:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - o(H)p_m\right]$$

Această ecuație, cunoscută și sub denumirea de *Teorema Schemelor* sau *Teorema fundamentală a Algoritmilor Genetici*, se exprimă concis astfel: *schemele cu lungime de definire scurtă, de ordin redus, aflate peste medie, obțin un număr de încercări exponențial crescător în generațiile următoare.*

Holland a demonstrat că, deși, un AG procesează, în fiecare generație, numai *n* structuri, el procesează un număr mult mai mare de scheme (aproximativ n^3). Mai mult aceste scheme sunt procesate în paralel. Se spune că un AG realizează, din acest punct de vedere, un *paralelism implicit*. Deși în fiecare generație se efectuează calcule aproximativ proporționale cu mărimea populației, în realitate un AG prelucrează în paralel aproximativ n^3 scheme, fără necesitatea unei evidențe speciale sau memorii, alta decât populația însăși.

Schemele scurte, de ordin redus și cu adecvare ridicată sunt extrase, recombinate și extrase din nou pentru a forma șiruri cu un potențial ridicat de adecvare. Astfel se reduce complexitatea problemei de rezolvat; în loc de a construi șiruri de performanță ridicată, încercând toate combinațiile, se construiesc șiruri din ce în ce mai "bune", plecând de la cele mai bune soluții parțiale ale ultimelor încercări. AG caută performanța apropiată de optim tocmai prin juxtapunerea acestor scheme scurte denumite ***blocuri fundamentale***.

3. Simularea principiilor de operare ale AG

Considerăm că un studiu al AG trebuie să conțină cel puțin un instrument software, ușor de utilizat, care să ofere cercetătorului un cadru pentru experimentarea modului de lucru al unui AG, plecând de la conceptele teoretice (populație, cromozomi și reprezentarea acestora, funcția de adecvare, operatorii genetici etc). Un astfel de software demonstrativ (**GA Demo**) a fost realizat de **Shaffer** (1996) și face parte din domeniul public (freeware). Programul a fost *extins cu noi funcții de autorii lucrării de față. (funcția sigmoidă și afișarea timpului de execuție a AG).* Programul simulează principiile de operare al unui AG, pornind de la o *configurație* definibilă de utilizator cu privire la: populația inițială de cromozomi; operatorii genetici utilizați de AG; funcțiile de adecvare; tipul de optim dorit (maxim, minim); numărul maxim de generații etc.

Cromozomii sunt reprezentați în program ca *șiruri binare* (gene cu valoarea 0 sau 1) de o lungime dată (specificată de utilizator). Dacă funcția de adecvare aleasă este de mai multe variabile *reale*, cromozomii sunt considerați ca fiind compuși prin concatenarea unor subșiruri binare asociate variabilelor. Aceste subșiruri sunt convertite în numere reale în vederea calculării valorilor funcției de adecvare (plecând de la expresia ei analitică) pentru fiecare cromozom în parte.

Programul GA Demo încorporează, ca exemple, trei funcții de adecvare. *Autorii*

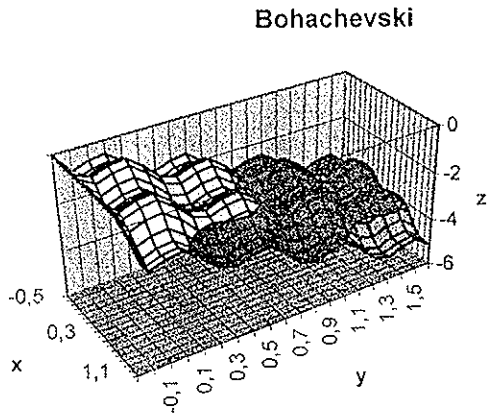
*lucrării de față au extins programul cu **funcția sigmoidală**.*

1) Însumarea binară. Această funcție definește valoarea de adecvare a unui cromozom ca fiind *suma valorilor genelor*. Pentru această funcție AG încearcă să creeze un cromozom cu toți biții "1" sau "0" (maxim sau minim). Acest tip de funcție este utilă pentru a urmări modul în care AG "promovează" cromozomii cei mai buni, de la o generație la următoarea, pentru atingerea optimului dorit.

2) Funcția lui Bohacevski, definită analitic de expresia:

$$f(x,y) = -(x^2 + 2y^2 - 0{,}3\cos(3\pi x) - 0{,}4\cos(4\pi y) + 0{,}7)$$

Pentru această funcție subșirurile binare sunt convertite în numere reale, și folosite ca variabile reale în ecuația anterioară. Există un maxim global al funcției, localizat la x=0; y=0. O reprezentare grafică a funcției pentru domeniul x,y∈[-0.5,1.5] este dată în figura care urmează.



3) Funcția lui Rosenbrock, definită analitic de expresia:

$$f(x,y) = 100(y - x^2) + (1 - x)^2$$

4) Funcția Sigmoida (sigmoidală), definită analitic de expresia:

$$f = final + \frac{initial - final}{1 + (abs\frac{x - offset}{contraction})^{degree}}$$

Această funcție poate fi considerată de o singură variabilă (x) și 5 parametri (*inițial, final, offset, contraction, degree*). Funcția este utilizată, printre altele, în probleme de interpolare a seturilor de date experimentale, acolo unde metodele de interpolare simple (gen regresie liniară) nu dau rezultate. În astfel de probleme se urmărește determinarea setului de valori ai parametrilor pentru care funcția aproximează cel mai bine evoluția setului de date între punctele măsurate, adică:

$$\sum_i (f_i - \overline{f}_i)^2 \rightarrow \min$$

unde f_i sunt valorile calculate ale sigmoidei în x_i iar $\overline{f}_i$ sunt valorile ordonatelor măsurate experimental în x_i. Aceasta conduce de fapt la o problemă de optimizare multidimensională care poate fi abordată prin metodele clasice de optimizare (Nedler și Mead, Powell etc). Am considerat util să abordăm optimizarea acestei funcții și prin metoda AG. Din punct de vedere al programului GA Demo funcția Sigmoidă poate fi considerată de adecvare cu 5 variabile reale (întrucât programul permite maximum 5 variabile, considerăm parametrul *degree* constant și egal cu 1).

În aceste condiții funcția Sigmoidă se exprimă prin:

$$f(x,y,z,u,v) = x + (y - x)/(1 + abs(z - u)/v)$$

Atât alegerea funcțiilor de adecvare menționate anterior, cât și a altor parametri care definesc configurația AG, se realizează în programul GA Demo, înaintea oricărei execuții, prin intermediul unei interfețe grafice.

4. Algoritmi genetici paraleli

AG sunt, în general, capabili să găsească soluții în durate convenabile de timp, dar pe măsură ce sunt aplicați unor probleme mai mari și mai dificile, timpul necesar găsirii unei soluții adecvate crește. În consecință, eforturile de a mări viteza AG s-au multiplicat, una dintre cele mai promițătoare alegeri fiind utilizarea unor *implementări paralele.* Cei mai "populari" AG paraleli prelucrează populații multiple ce evoluează separat în majoritatea timpului și schimbă indivizi doar ocazional. Acest tip de AG paralel se numește AG cu granulație mare sau distribuit.

Combinând mai multe tehnici de paralelizare se obțin algoritmi care combină avantajele fiecărei componente astfel încât să se obțină performanțe mai bune decât în cazul oricăreia dintre componente în parte.

Metoda de *paralelizare globală* lucrează cu o populație *unică,* evaluarea indivizilor și/ sau aplicarea operatorilor genetici efectuându-se *în paralel.* La fel ca în cazul AG seriali, fiecare individ intră în competiție cu toți ceilalți membri ai populației și are șansa de a se împerechea cu oricare alt individ. Cea mai frecvent utilizată operație care este paralelizată este cea de evaluare a indivizilor, deoarece gradul de fitness (adecvare) al unui individ este independent de restul populației și deci nu este nevoie de interacțiune pe durata acestei faze.

În cazul *AG paraleli cu granulație fină,* populația este divizată în *multe deme cu populații mici*, între aceste deme existând o *comunicație intensă.* Comunicația masivă furnizează un mijloc de diseminare a soluțiilor bune în întreaga populație. Selecția și împerecherea pot avea loc numai în interiorul unei deme.

O metodă destul de frecvent utilizată constă în plasarea indivizilor unui AG paralel cu granulație fină într-o grilă bidimensională deoarece în multe calculatoare *masiv paralele,* elementele de procesare sunt conectate folosindu-se această tehnologie. Majoritatea acestor calculatoare dispun, de asemenea, și de un *ruter global* care poate trimite mesaje oricărui alt procesor din rețea (implicând un cost mai mare de comunicație), astfel încât pot fi simulate peste grilă orice alte tipuri de topologii.

Din combinarea metodelor de paralelizare ale algoritmilor a rezultat o nouă clasă: cea a *algoritmilor genetici paraleli hibrizi.* Atunci când se combină metodele de paralelizare a AG, se formează o ierarhie. Pentru majoritatea AG paraleli hibrizi, la nivelul superior se află un AG cu granulație mare. O altă metodă de a hibridiza un AG paralel este cea de a forma o *paralelizare globală peste fiecare din demele unui AG cu granulație mare.* Migrația are loc între deme la fel ca și în cazul algoritmului cu granulație mare, dar evaluarea indivizilor se face în paralel. Această abordare poate fi folositoare când se lucrează cu aplicații complexe cu funcții obiectiv care necesită o cantitate considerabilă de timp de calcul. O a treia metodă de hibridizare a AG paralel ar putea fi folosirea unui AG cu granulație mare atât la nivelul superior cât și la nivelul inferior. Ideea constă în forțarea unei mixări panmictice la nivelul inferior prin folosirea unei rate mari de migrație și a unei topologii dense și o rată mică de migrație la nivelul superior. Acest hibrid ar fi, de asemenea, echivalent din punct de vedere al complexității cu un AG cu granulație mare dacă se consideră grupurile de subpopulații ca o singură demă.

Pentru testarea AG paraleli autorii lucrării au "descărcat" un sistem software "freeware" GALLOPS (**G**enetic **A**lgorithm **O**ptimized for **P**ortability and **P**arallelism **S**ystem) Release 3.2. Sistemul GALOPPS asigură paralelism hardware, inclusiv pentru utilizatorii unei rețele de PC-uri și, de asemenea, poate simula paralelismul pe un singur procesor (calculator). A fost proiectat, în ansamblul său, ca un AGP cu granulație mare (de tip "insulă"), fie real, fie simulat. S-a implementat, pentru experimentări, varianta GALOPPS Release 3.2 cu rulare pe un singur procesor - calculator PC sub MS-Windows95. S-a rulat aplicația de maximizare a funcției *x**10* "propusă" de

GALOPPS, atât în varianta AG serial (cu o singură populație), cât și în varianta AG paralel cu granulație mare. Aplicația (propusă inițial de Goldberg) trebuie să găsească maximul funcției *x**10*, normalizată în domeniul [0,1]. Funcția obiectiv este scrisă mai complicat decât este necesar tocmai pentru a demonstra cum pot fi manipulați direct cromozomii de către utilizator. Cromozomul este compus din *10 câmpuri, fiecare de câte un bit* (și nu ca un cromozom cu un *singur câmp de 10 biți*) astfel încât încrucișarea (crossover) să poată avea loc la *nivel de bit*, în orice poziție din cromozom.

5. Concluzii

Pe măsură ce AG se aplică unor probleme de căutare mai mari și mai dificile, devine necesară proiectarea unor algoritmi mai rapizi prin care să se poată obține și soluții acceptabile. AG paraleli sunt capabili să combine viteza și eficiența, studierea lor în continuare fiind deosebit de promițătoare.
Mai mult, existența unor instrumente software performante și flexibile de tip GALOPPS permite studierea și utilizarea unor algoritmi genetici seriali și paraleli pe o largă varietate de platforme hardware/ software: arhitecturi paralele de orice tip, rețele de calculatoare eterogene, sisteme stand-alone mono sau multiprocesor. În acest fel este posibilă obținerea unor rezultate notabile ale cercetării în acest domeniu chiar și cu investiții relativ reduse pentru achiziționarea platformelor de calcul.

Bibliografie

1. Davidor, Y. (1991) *Foundations of Genetic Algorithms*
2. Davies, L. (1991) *Handbook of Genetic Algorithm,* Van Nostrand Reinhold
3. Goldberg, D.E (1989) *Genetic Algorithm in Search, Optimization and Machine Learning*, New York, Addison-Wesley
4. Goldberg D.E. *Genetic algorithms and evolutionary algorithms come of age;* Communications of the ACM, Nr.37/3, 1994, pg. 113-119.
5. Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press
6. Shaffer, R. (1997) *Practical Guide to Genetic Algorithms*, Naval Research Laboratory
7. ***
http://chem1.nrl.navy.mil/~shaffer/gademo.html
8. ***
http://chem1.nrl.navy.mil/~shaffer/practga.html