Modelarea EXPRESIILOR folosind tehnici EVOLUTIVE

Mihai Oltean, Crina Groșan

Rezolvarea problemelor care implică folosirea unor expresii matematice a fost intens studiată în ultimele decenii. În acest articol este prezentată o tehnică evolutivă pentru rezolvarea de probleme dificile care conțin expresii matematice.

Algoritmi evolutivi

Rezolvarea problemelor folosind metode evolutive este un domeniu de tradiție. El își are rădăcinile în anii '50, când mai mulți cercetători au preluat metafora evolutivă din natură, aplicând-o la rezolvarea problemelor din diverse domenii. Metafora evolutivă este transpusă, informatic, într-un număr foarte mare de algoritmi. Totuși, câteva considerente generale pot fi indicate.

Se lucrează cu o populație (vector, listă) de posibile soluții. Acestea se mai numesc uneori și *indivizi*, *cromozomi* etc. Indivizii populației inițiale sunt generați aleator. Cu ajutorul selecției și al operatorilor genetici indivizii evoluează și, eventual, vor converge către soluție.

Indivizii sunt evaluați pe baza unei așa-zise funcții de *fitness* (calitate). *Fitness* înseamnă speranță de viață. Cu cât un individ este mai performant, cu atât speranța lui de viață este mai mare.

Asupra indivizilor din populație se aplică operatori genetici. Acești operatori sunt dependenți de reprezentare. Dintre cei mai răspândiți operatori genetici amintim *încrucișarea* și *mutația*. Prin încrucișare, de obicei, doi părinți dau naștere a doi fii. Prin mutație se aplică o mică "perturbație" a valorii unui individ.

De obicei, nu toți indivizii sunt aleși pentru aplicarea operatorilor genetici. Modul și ordinea în care se aplică operatorii genetici asupra indivizilor variază foarte mult. De exemplu, într-un algoritm genetic din populație sunt selectați indivizi în funcție de *fitness*-ul lor. Indivizii selectați sunt introduși într-o populație intermediară. Indivizii din această populație intermediară sunt supuși operatorilor genetici de încrucișare și mutație. Indivizii rezultați în urma acestor operații constituie "următoarea generație", adică o nouă populație care va fi din nou supusă selecției și operatorilor genetici. Acest proces va continua pentru un anumit număr de generații.

Programare genetică

Programarea genetică este o ramură a algoritmilor evolutivi care se ocupă cu evoluarea codurilor sursă.

Se dorește, de exemplu, evoluarea unui program *Pascal* sau C++ care să rezolve problema drumului *hamiltonian* sau problema ieșirii dintr-un labirint. Să presupunem că nu știm cum să rezolvăm această problemă. Așadar, nu ne interesează să obținem o soluție pentru un set oarecare de date, ci dorim să obținem un program sursă care să genereze o soluție cât mai aproape de cea corectă pentru orice date de intrare. Cu alte cuvinte, ne interesează să obținem ca rezultat un program asemănător cu cel pe care l-am fi putut scrie dacă am fi știut rezolva problema.

Din punct de vedere evolutiv, abordarea unor astfel de probleme se face generând o mulțime (populație) aleatoare de coduri sursă care apoi sunt selectate pe baza funcției de *fitness* și sunt evoluate cu ajutorul unor operatori genetici specifici.

În primul rând, trebuie să atribuim o funcție de calitate (funcția *fitness*) fiecărui program generat. Această funcție de *fitness* trebuie să reflecte performanțele programului căruia îi este atașată.

De obicei, atașarea unei funcții de *fitness* se face rulând programul respectiv și măsurând calitatea soluției în raport cu o soluție care se cunoaște a fi optimă. Un program va avea o calitate mai mare, dacă soluția generată va fi mai apropiată de soluția optimă. Dacă nu se cunoaște o soluție optimă, atunci această situație nu implică apariția unor dificultăți suplimentare, deoarece noi dorim să maximizăm sau să minimizăm valoarea *fitness*-ului.

Evoluarea programelor sursă se realizează prin intermediul operatorilor genetici specifici. De exemplu, un operator de recombinare poate însemna alipirea unor secvențe dintr-un cod sursă cu secvențe din alt cod sursă. Un operator de mutație ar putea însemna inserarea de noi instrucțiuni în codul sursă. De notat este faptul că alegerea operatorilor folosiți este la latitudinea programatorului. Totul depinde de imaginația celui care propune o metodă. Multe articole științifice prezintă noi operatori genetici cu performanțe superioare celor existenți.

Evident, în urma aplicării acestor operatori genetici pot fi generate coduri sursă care conțin greșeli de sintaxă. De asemenea, sunt generate secvențe de cod sursă nefolositoare. Un exemplu în acest sens este secvența de instrucțiuni:

```
i := i + 1;
i := i - 1;
```

După executarea acestor instrucțiuni valoarea variabilei `i` rămâne neschimbată, deci uneori ar fi utilă eliminarea unora dintre aceste secvențe. Eliminarea nu este recomandată întotdeauna pentru că ulterior o altă aplicare a operatorilor genetici ar putea insera și alte instrucțiuni între cele două de mai sus, ducând astfel la eliminarea redundanțelor.

Evoluarea programelor sursă în limbajele *Pascal* sau *C* este o muncă dificilă și rezultatele sunt foarte puțin spectaculoase. Obținerea prin evoluție a unui cod sursă *Pascal* sau *C* pentru rezolvarea problemei drumului *hamiltonian* este un fel de *Fata Morgana*. *John Koza*, unul dintre pionierii domeniului, a declarat că acasă are o rețea cu 500 de calculatoare cu ajutorul căreia evoluează coduri sursă complexe. Și nici în aceste condiții nu a obținut rezultate semnificative.

De obicei, se lucrează cu reprezentări mai simple ale programelor și anume cu arbori. Generarea unui arbore atașat unui program nu este o muncă dificilă. În plus, există limbaje de programare (de exemplu *LISP*) în care programele sunt scrise sub forma unor liste ușor transformabile în arbori.

Expresii aritmetice

Tot în domeniul programării genetice intră și generarea unor funcții matematice care îndeplinesc anumite criterii. Se dorește găsirea unei funcții `f` care satisface cât mai bine o mulțime de perechi `(x, f(x))` date.

Ne propunem să lucrăm cu expresii aritmetice. Problema care apare este cum reprezentăm eficient indivizii astfel încât în urma aplicării operatorilor genetici să rezulte expresii aritmetice valide. De exemplu, considerând expresiile aritmetice:

E_1 = (**a** + **b** – **d**) * **c** – **d** * **a**

E_2 = x *(c + d * (a - b))

și considerând operatorul genetic clasic de recombinare ca fiind recombinarea cu un singur punct de tăietură, obținem expresiile:

F_1 = (**a**(c + d *(a-b)) și

F_2 = x * + **b** – **d**) * **c** – **d** * **a**.

Aici punctul de tăietură a fost ales după poziția a doua. Se observă că cele două expresii aritmetice rezultate nu sunt valide.

Aceleași dificultăți se întâlnesc și în cazul operatorului de mutație.

O soluție ar fi căutarea unor operatori genetici noi care să poată fi aplicați asupra expresiilor aritmetice. Aceștia ar trebui să fie destul de puternici pentru a efectua o căutare completă în spațiul posibilelor soluții și suficient de rapizi pentru ca rezolvarea problemei să fie obținută într-un timp scurt. O modalitate ar fi reprezentarea sub formă de arbori a expresiilor aritmetice. Acest mod de reprezentare este specific *programării genetice*.

Arborii sunt structuri complexe de reprezentare a datelor. Operatorul genetic de încrucișare folosit pentru evoluarea arborilor constă, de obicei, în schimbul reciproc de subarbori.

Operatorul de mutație modifică un arbore prin adăugarea sau ștergerea unor subarbori. În urma aplicării operatorilor genetici pe arbori vor rezulta arbori valizi, care generează expresii aritmetice corecte.

Această metodă prezintă dificultăți datorate, în principal, aplicării greoaie a operatorilor genetici asupra arborilor (schimbarea între ei a doi subarbori este o sarcină destul de dificilă).

O altă problemă care apare când se lucrează cu reprezentări arborescente este dimensiunea arborilor. Fără un control strict al dimensiunii, se poate întâmpla ca arborii rezultați să crească excesiv, lucru care duce la o manipulare greoaie și, mai ales, la ocuparea completă a memoriei.

Gene expression programming

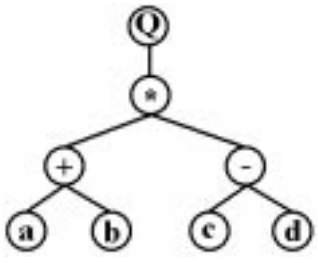
O altă soluție ar fi căutarea unor reprezentări mai eficiente pentru expresiile aritmetice. O astfel de reprezentare este prezentată în cele ce urmează. Acest tip de codificare a unei expresii aritmetice apare în [1].

De obicei, când se lucra cu expresii aritmetice, cele mai des folosite parcurgeri erau traversările în inordine, postordine si preordine. Codificarea propusă de *Ferreira* este o extensie a parcurgerii pe niveluri a unui arbore.

Fie expresia matematică:

$$F = \sqrt{(a + b) \times (c - d)}.$$

Reprezentarea sub formă de arbore a acestei expresii este:



Aici "`Q`" simbolizează rădăcina pătrată. Parcurgerea pe niveluri a acestui arbore generează șirul: `Q*+-abcd`.

În cele ce urmează vom vedea care sunt avantajele acestei reprezentări atunci când expresiile matematice sunt evoluate cu ajutorul unor mecanisme genetice.

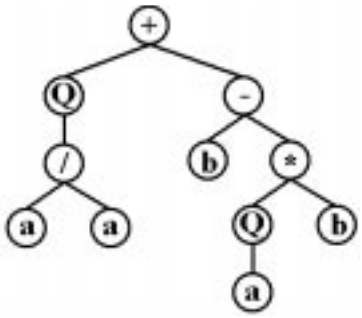
Scopul nostru este de a obține întotdeauna expresii matematice valide. De aceea vom impune câteva restricții asupra reprezentării și operatorilor.

Un cromozom este alcătuit dintr-un cap și o coadă. Capul conține simboluri care reprezintă atât operanzi cât și operatori. Coada conține simboluri care reprezintă doar operanzi. Pentru fiecare problemă în parte este aleasă o valoare `h` pentru lungimea capului. Lungimea cozii `t` este calculată în funcție de lungimea capului și a numărului de argumente `n` al operatorului cu cele mai multe argumente, astfel: `t = (n - 1) * h + 1`.

Să considerăm un cromozom care conține operatorii `{Q, +, -, *, /}` și operanzii `{a, b}`. În acest caz `n = 2`. Pentru lungimea capului cromozomului `h = 10`, rezultă că lungimea cozii este `t = 11`. Un posibil cromozom care verifică aceste cerințe este următorul (coada este reprezentată folosind caractere îngroșate):

`+Q-/b*aaQ`**`baabaabbaaab`**

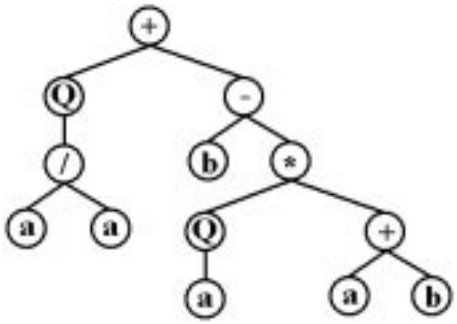
Acest șir codifică arborele:



Se observă că, deși cromozomul are lungimea 21, doar primele 10 caractere codifică arborele reprezentat mai sus. Restul de caractere sunt nefolosite.

Să presupunem că o mutație apare în poziția 10, schimbând caraterul "`b`" în "`+`". Este obținut următorul cromozom: `+Q-/b*aaQ+`**`aabaabbaaab`**

În acest caz, arborele codificat de acest cromozom este:

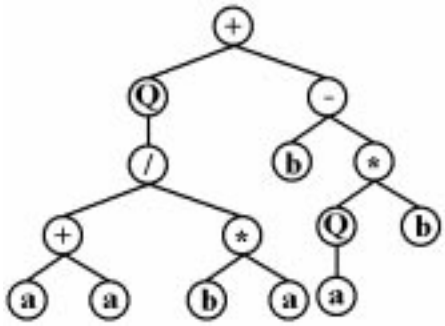


Se observă că lungimea șirului de caractere util este în acest caz 12.

Să mai facem o modificare asupra cromozomului. Să schimbăm pozițiile 7 și 8 (care conțin două caractere "`a`") în "`+`" respectiv "`*`". Se obține următorul cromozom:

`+Q-/b*+*Q+`**`aabaabbaaab`**

Arborele codificat de acest cromozom este:

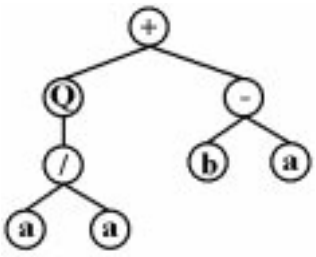


Se observă că pentru acest arbore numărul caracterelor folosite din cromozom este mult mai mare.

Până în acest moment am făcut doar mutații care creșteau dimensiunea arborelui; este posibilă și situația opusă care duce la scăderea dimensiunii. Dacă pe poziția 6 din arbore schimbăm caracterul "`*`" cu "`a`" obținem următorul cromozom:

`+Q-/ba+*Q+`**`aabaabbaaab`**

Acest cromozom codifică arborele:



Se observă că arborele obținut are cu trei noduri mai puțin decât arborele inițial. Chiar dacă fiecare cromozom are o lungime fixă, acesta poate codifica arbori de o mare varietate, de la cel mai simplu care conține doar un nod (în acest caz pe prima poziție a cromozomului se află un operand), până la cel mai complex care valorifică toate caracterele cromozomului (în acest caz toate elementele capului sunt operatori de aritate maximă).

Din exemplele anterioare se observă că, oricât de profunde ar fi modificările într-un cromozom, acesta va codifica întotdeauna un arbore valid. Pentru ca acest lucru să fie adevărat trebuie ca structura cromozomului să rămână nealterată, adică să nu se permită inserarea în coadă a unor operatori.

Decodificarea informației

Decodificarea informației înseamnă trecerea de la un cromozom la arborele corespunzător. În cromozom arborele este codificat sub forma rezultată în urma parcurgerii pe niveluri. Această reprezentare este mai greu de manipulat atunci când se dorește evaluarea arborelui. Dacă avem un arbore binar care codifică o expresie aritmetică, atunci pentru evaluarea ei trebuie să parcurgem arborele în postordine. De aceea trebuie realizată trecerea de la forma conținută în cromozom la o altă formă ușor utilizabilă pentru parcurgerea în postordine.

De exemplu, dacă arborele codificat în cromozom este binar, atunci se poate construi ușor o reprezentare în *Heap* a acestuia. O posibilitate este alegerea unei reprezentări statice: se folosește un tablou cu trei coloane și N linii (N este numărul de noduri din arbore), în care pe fiecare linie în prima coloană s-a reținut informația atașată unui nod, iar pe celelalte două poziții s-au reținut pozițiile din care încep subarborii stâng și drept a nodului curent.

Translatarea începe întotdeauna de la prima poziție din cromozom și se continuă atât timp cât arborele care se construiește nu devine complet. Restricțiile impuse cromozomului garantează întotdeauna că dintr-un cromozom se poate construi un arbore valid.

Fitness-ul unui cromozom

Se evaluează expresia codificată într-un cromozom. Cu cât valoarea rezultată este mai aproape de rezultatul dorit, cu atât *fitness*-ul cromozomului este mai mare.

Vom nota cu f funcția de *fitness*. Aceasta este definită pe mulțimea posibilă a cromozomilor și are, de obicei, valori în mulțimea numerelor reale pozitive.

Selecția

Pentru selecție vom folosi metoda turnirului binar. Aceasta constă în selectarea aleatoare a doi indivizi și compararea lor directă. Cel mai performant dintre aceștia este inserat în populația intermediară, care va constitui rezerva de indivizi pentru încrucișare și mutație. Populația intermediară se mai numește și *piscină de încrucișare*.

Încrucișarea

Prin încrucișare, din doi indivizi numiți părinți, se obțin doi fii. Deoarece indivizii sunt șiruri de caractere, se poate folosi încrucișarea cu un singur punct de tăietură. Spre exemplu să considerăm doi cromozomi:

$x = x_1\ x_2 \dots x_k\ x_{k+1} \dots x_r$

și

$y = y_1\ y_2 \dots y_k\ y_{k+1} \dots y_r$.

În urma recombinării (încrucișării) se schimbă între cei doi cromozomi secvențele aflate la dreapta punctului de tăietură k. Cromozomii fii vor fi:

$x' = x_1\ x_2 \dots x_k\ y_{k+1} \dots y_r$

și

$y' = y_1\ y_2 \dots y_k\ x_{k+1} \dots x_r$.

Mutația

Operatorul de *mutație* constă în înlocuirea unui simbol cu un alt simbol. În secvența care alcătuiește capul cromozomului, fiecare simbol poate fi transformat în orice alt simbol. În secvența care alcătuiește coada cromozomului, doar operanzii pot fi transformați în alți operanzi.

De obicei, nu toate simbolurile din cromozom sunt supuse mutației. De fapt, mutația este aplicată asupra unui număr relativ mic de simboluri din cromozom.

Se consideră o probabilitate de mutație p_m pe baza căreia sunt schimbate simbolurile din cromozom. Algoritmul folosit pentru realizarea mutațiilor este descris în continuare:

Pentru fiecare poziție a cromozomului se execută:

P_1. Se generează un număr aleator q în intervalul [0, 1].

P_2. Dacă $q < p_m$, atunci se execută mutația poziției respective, schimbând un simbol cu un altul; în caz contrar ($q \geq p_m$), poziția respectivă nu se schimbă.

Problema 1: Expresie de valoare dată

Se dau constantele întregi C_1, C_2, ..., C_N, S. Se cere să se construiască o expresie aritmetică E care conține operatorii {+, -, *} și eventual paranteze. Operanzii admiși sunt constantele C_1, C_2, ..., C_N. Fiecare constantă poate apărea o dată, de mai multe ori, sau niciodată în construcția expresiei E. Valoarea expresiei E trebuie să fie cât mai apropiată de S.

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie sunt scrise valorile N și S,
- pe următoarea linie sunt scrise constantele C_1, C_2, ..., C_N, despărțite prin spații.

Rezolvare

Pentru rezolvarea acestei probleme vom folosi un algoritm genetic standard. Cromozomii reprezintă posibilele soluții ale problemei și sunt codificați în forma descrisă anterior. Pentru simplificare cromozomii sunt vectori de valori întregi din intervalul [0, N + 3]. Numerele 0, 1, 2 codifică operatorii +, -, respectiv *, iar numerele de la 3 la N + 3 reprezintă indicii constantelor C_1, C_2, ..., C_N (numărul 3 corespunde constantei C_1, numărul 4 corespunde constantei C_2 etc.).

Lungimea capului cromozomilor este un parametru al algoritmului și poate fi fixat de către utilizator. De asemenea, parametri ai algoritmului mai sunt și dimensiunea populației, numărul de iterații și numărul celor mai buni indivizi care sunt trecuți automat în populația următoare (Nr_Elitism). De obicei, un număr de 100 de iterații și un număr de 100 de indivizi într-o populație sunt suficienți.

Codul sursă poate fi descărcat de la următoarea adresă: http://www.ginfo.ro/revista/12_2/index.shtml.

Exemplu

Intrare

```
10 4
3 5 2 6
```

Ieșire

```
*2563
```

Șirul furnizat la ieșire corespunde expresiei 2*5.

Discuție

Se poate lucra și cu numere reale. În acest caz se poate introduce și operatorul /.

Se pot introduce restricții asupra folosirii operanzilor. De exemplu, se poate introduce restricția ca fiecare ope-

rand să se folosească o singură dată. Această restricție va afecta puțin felul în care sunt codificate expresiile.

Cei care doresc pot implementa o variantă a algoritmului genetic, folosind însă codificarea sub formă de arbori binari alocați dinamic. Se va observa diferența clară între viteza de execuție a acestui program și viteza de execuție și cantitatea de memorie alocată de programele care folosesc direct codificarea sub formă de arbore.

Problema 2: Regresie simbolică

Fie o funcție `f` care depinde de un singur parametru `x`. Dându-se o mulțime de perechi $S = \{(x_i, f(x_i)), i = 1, \ldots, N\}$, se cere să se determine o funcție `f` care satisface cât mai bine aceste restricții. Funcția nu poate să conțină constante.

Datele de intrare se citesc dintr-un fișier text având următoarea structură:

- pe prima linie se află numărul `N` al perechilor `(x, f(x))`;
- pe fiecare dintre următoarele `N` linii se află câte o pereche sub forma `x f(x)`.

Rezolvare

În primul rând definim mulțimea operatorilor pe care dorim să îi folosim. Aceștia pot fi cei aritmetici `(+, -, *, /)`, dar și alte funcții matematice `(sin, cos, tg, log, sqrt` etc.). Deci, într-o expresie vom întâlni operatorii definiți și operandul `x`. Astfel, structura cromozomului va fi asemănătoare cu cea de la problema precedentă.

Diferența care apare este la calcularea funcției de *fitness*. Dacă la problema precedentă calitatea unei soluții era dată de diferența între valoarea ei și valoarea țintă `S`, aici problema este puțin mai complicată, deoarece pentru diferite valori ale lui `x` vom avea diferite valori țintă `f(x)`. O variantă posibilă ar fi să însumăm (pentru fiecare valoare x_i) diferențele între valoarea obținută și valoarea țintă.

Discuție

Restricția privitoare la apariția constantelor într-o expresie va fi discutată în problema următoare.

Deoarece se lucrează cu numere reale, se cere ca expresia rezultată să se evalueze la valori apropiate cu o precizie dată față de valorile țintă.

Problema 3: Ghicește următorul număr

Se dau primii `M` termeni din șirul de constante $(a_N)_{N\geq 0}$. Acest șir este alcătuit pe baza unei reguli. Regula este o expresie de un singur parametru `(N)`. Se cere să se determine următorul număr din secvență (a_{M+1}) și eventual regula după care a fost alcătuit.

Exemplu

Pentru secvența `1, 15, 129, 547, 1593, 3711, 7465, 13539, 22737, 35983` algoritmul trebuie să ghicească următorul număr ca fiind `54321`.

Șirul este construit conform regulii: $a_N = 5N^4 + 4N^3 + 3N^2 + 2N + 1$.

Rezolvare

Această problemă este un caz particular al problemei precedente, deoarece domeniul variabilei `N` este acel al numerelor întregi nenegative. Totuși, problema este și mai complicată deoarece implică utilizarea de constante.

Rezolvarea este asemănătoare cu soluțiile propuse la problemele precedente. Lângă fiecare apariție a variabilei `N` se mai adaugă și o constantă care va evolua cu ajutorul operatorilor genetici. Se poate folosi mutația ca singurul operator care evoluează constantele. În acest caz, mutația unei constante înseamnă adăugarea sau scăderea unei mici valori la aceea constantă.

Probleme propuse

În cele ce urmează vom propune spre rezolvare câteva probleme ale căror soluții sunt bazate pe tehnicile prezentate în cadrul acestui articol.

Găsește recurența

Se dau primii `M` termeni ai unei recurențe $a_N = a_1 * a_{N-k} + a_2 * a_{N-k-1} + \ldots + a_p * a_{N-k-p} + 1$. Se cere să se determine relația de recurență care satisface acest șir. Termenii a_i sunt numere reale.

Funcții trigonometrice

Se consideră o expresie care conține funcțiile trigonometrice `sin`, `cos`, `tg` și `ctg`, precum și diferite alte funcții aritmetice. Se cere să se determine alte expresii echivalente cu cea dată.

Teoreme

Se consideră anumite valori pentru catetele și ipotenuza unui triunghi dreptunghic. Să se determine expresia teoremei lui *Pitagora* folosind tehnici evolutive.

Să se aplice aceeași metodă pentru determinarea altor expresii pentru diferite teoreme din fizică și matematică.

Bibliografie

[1] Cândida Ferreira, *Gene Expression Programming: a New Adaptive Algorithm for Solving Problems*, *Complex Systems* (în curs de apariție).

[2] Mihai Oltean, *Proiectarea și Implementarea Algoritmilor*, Editura Computer Libris Agora, Cluj-Napoca, 2000.

[3] D. Dumitrescu, *Algoritmi genetici și strategii evolutive*, Editura Albastră, Grupul MicroInformatica, Cluj-Napoca, 2000.

[4] J. Koza, *Genetic Programming: On the programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[5] Garey Johnson, *Computers and Intractability: A Guide to NP-Completeness*, Freeman & Co, San Francisco

[6] R. Motwani, *P. Raghavan, Randomized Algorithms*, Cambridge University Press, 1995.

Mihai Oltean și Crina Groșan sunt masteri și doctoranzi ai Universității Babeș-Bolyai din Cluj-Napoca. Pot fi contactați prin e-mail la adresele moltean@cs.ubbcluj.ro, *respectiv* cgrosan@cs.ubbcluj.ro.