ALGORITMI GENETICI

Aplicaţii în testarea programelor

Raluca Lefticaru

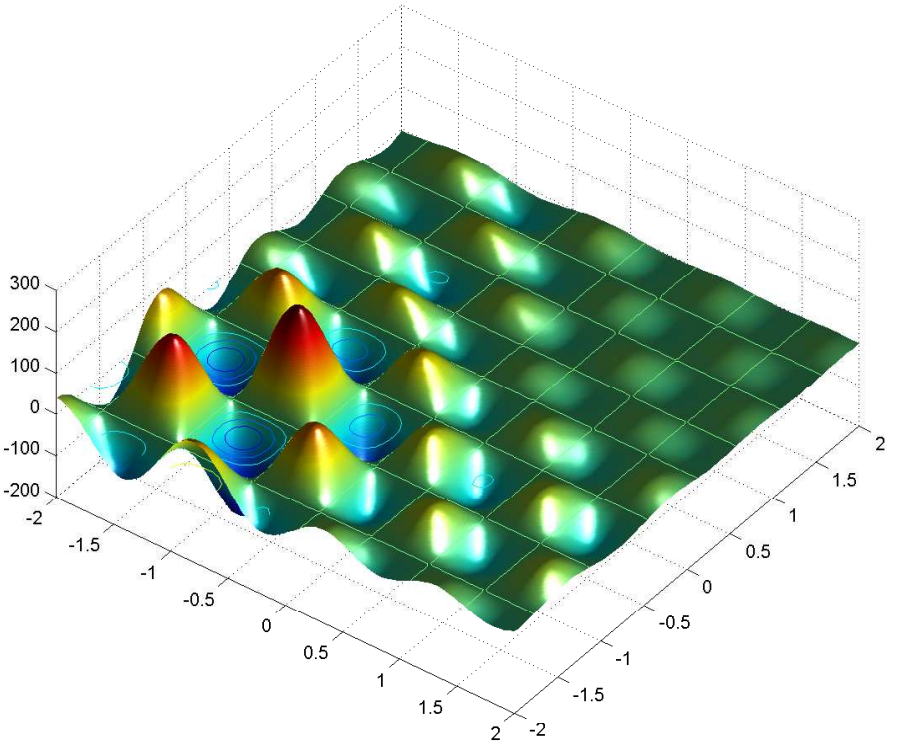
Algoritmi genetici

- Inspiraţi din principiul selecţiei naturale (C. Darwin)
- Sunt algoritmi probabilişti, care combină elemente de căutare dirijată şi căutare aleatoare, realizând un echilibru aproape optimal între explorarea spaţiului stărilor şi exploatarea celor mai bune soluţii găsite.
- Lucrează cu o populaţie de indivizi (soluţii potenţiale ale problemei) - *căutare globală*.
- Algoritmul simulează un proces de evoluţie: la fiecare generaţie, soluţiile bune se reproduc, iar cele relativ slabe nu supravieţuiesc.
- De regulă elementele populaţiei sunt codificate în formă binară.
- Operatorul de încrucişare este cel principal, cel de mutaţie are un rol secundar.

Algoritmi genetici

- Sunt folosiţi în special în rezolvarea problemelor de optimizare combinatorială.
- Folosesc funcţii de performanţă obţinute prin transformări simple ale funcţiei obiectiv.
- Nu este necesar ca funcţia obiectiv să fie derivabilă sau să îndeplinească proprietăţi speciale de convexitate.
- Pot găsi soluţii optime sau aproape optime cu o mare probabilitate.

Exemplu de funcţie obiectiv




```
plotobjective(@shufcn,[-2 2; -2 2]);
```

Generalităţi AG

- X *spaţiul de căutare* (spaţiul stărilor problemei)
- *Individ* (genotip, cromozom, string) al unei populaţii = un element din X, adică o soluţie posibilă a problemei.
- *Cromozomul* (purtătorul informaţiei genetice) este o structură liniară formată din *gene* (trăsături, caractere).
- Genele se gasesc in cromozom pe anumite poziţii (*loci*)
- *Alelele* reprezintă valorile genelor.
- *Genotipul* reprezintă o soluţie potenţială a problemei, iar *fenotipul* valoarea acestuia.
- $P(t) = \{x_1^t, \ldots, x_n^t\}$ populaţia de indivizi de la momentul t. $P(t)$ reprezintă o *generaţie*.

Generalităţi AG

- Evaluarea calităţilor indivizilor se face cu ajutorul unei funcţii de *fitness* (performanţă).
- Operatorul de *recombinare* $R : X^p \longrightarrow X^q$ realizează o transformare de tipul (p, q) în care din p părinţi iau naştere q descendenţi.
- Operatorul de *mutaţie* este o transformare unară $m : X \longrightarrow X$, care realizează mici perturbări ale indivizilor.
- *Supravieţuirea* determină în ce măsură indivizii unei generaţii supravieţuiesc în următoarea.

Algoritmi genetici

Algoritmi genetici **standard**	Algoritmi genetici **hibrizi**
codificare binară	codificare reală
lungimea cromozomului fixă	lungimea cromozomului variabilă
mărimea populaţiei fixă	mărimea populaţiei variabilă

Structura unui algoritm genetic

```
procedure genetic_algorithm
begin
t ⟵ 0
initialize P(t)
while (not termination-condition) do
 begin
 evaluate P(t)
 t ⟵ t + 1
 select P(t) from P(t-1)
 recombine P(t)
 mutate P(t)
 end
end
```

Descriere

- $P(t) = \{x_1^t, \ldots, x_n^t\}$ populaţia de indivizi de la iteraţia t.
- Fiecare soluţie posibilă x_i^t este evaluată, valoarea rezultată reprezentând *fitness*-ul acesteia.
- La iteraţia $t + 1$ este formată o nouă populaţie prin selectarea unor indivizi din populaţia anterioară (pasul `select`).
- Câţiva membri ai acestei populaţii suportă transformări prin intermediul operatorilor genetici (mutaţie şi recombinare), rezultând noi indivizi.
- Procedeul este repetat până când criteriul de oprire este satisfăcut.

Algoritmul Monte Carlo (alg. ruletei)

- Se calculează valorile fitness $f(x_i)$ pentru fiecare individ x_i, $i = 1 \ldots n$, $f \geq 0$ de maximizat.
- Se determină fitness-ul total al populaţiei $F = \sum_{i=1}^{n} f(x_i)$
- Se calculează probabilitatea de selecţie p_i a fiecărui individ: $p_i = \frac{f(x_i)}{F}$
- Se calculează probabilitatea cumulată q_i a fiecărui individ: $q_i = \sum_{j=1}^{i} p_j$

Algoritmul Monte Carlo (alg. ruletei)

Pentru a selecta noua populaţie, se repetă următorul procedeu de n ori:

- Se generează aleator un număr real r din intervalul $[0..1]$.
- Dacă $r < q_1$ atunci se selectează primul individ x_1, altfel se alege individul x_i pentru care $q_{i-1} < r \leq q_i$.

Observaţie: unii indivizi vor avea copii multiple în noua populaţie, alţii nu vor aparea deloc.

Alte mecanisme de selecţie

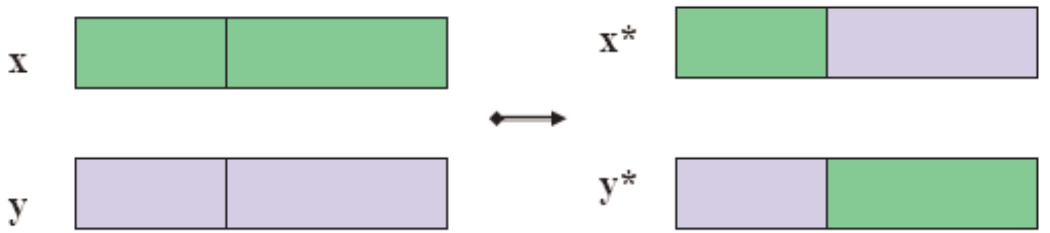
- Schimbarea de scală a funcţiei de fitness: liniară sau exponenţială; statică (aceeaşi pentru toate generaţiile) sau dinamică.
- Selecţia bazată pe ordonare, unde n = nr. de indivizi, r_i rangul individului i, $q \in [1, 2]$ *presiunea de selecţie*

$$p_i = \frac{1}{n}\left[q - \frac{(r_i - 1)2(q-1)}{(n-1)}\right]$$

- Selecţia de tip turnir.
- Strategii elitiste.

Recombinarea

Combină trăsăturile a doi cromozomi părinţi rezultând urmaşi care moştenesc aceste trăsături. Cei doi urmaşi rezultă cel mai des prin interschimbarea anumitor segmente din părinţi. Tipuri: încrucişare cu un punct de tăietură; cu mai multe puncte de tăietură; încrucişare adaptivă; segmentată; cu amestec; uniformă.



Încrucişarea cu un punct de tăietură

- p_c probabilitatea încrucişării, parametru al algoritmului genetic; numărul aşteptat de cromozomi care vor participa la încrucişare este $p_c \times n$
- pentru a selecta indivizii care vor fi încrucişaţi:
  - pentru fiecare individ, se generează un număr aleator r în intervalul $[0..1]$
  - dacă $r < p_c$ atunci selectează individul respectiv pentru încrucişare
- pentru a realiza efectiv încrucişarea (cu un punct de tăiere): alegem câte o pereche de indivizi, selectăm aleator un punct de tăiere şi interschimbăm segmentele.

Mutaţia

- alterează una sau mai multe gene alese arbitrar dintr-un cromozom
- p_m probabilitatea mutaţiei, parametru al algoritmului genetic; numărul aşteptat de gene mutate este $p_m \times m \times n$, m lungimea unui cromozom;
- mutaţia unei gene (bit): modificarea bitului din 0 in 1 sau invers
- pentru fiecare cromozom, şi fiecare genă a acestuia:
  - se generează un număr aleator r din intervalul $[0..1]$
  - dacă $r < p_m$ atunci se modifică valoarea genei
- pastrează variabilitatea în populaţie

Bibliografie AG

- D. Dumitrescu, Algoritmi genetici şi strategii evolutive - Aplicaţii în inteligenţa artificială şi în domenii conexe, Editura Albastră, Cluj-Napoca, 2000;
- D. Goldberg, Genetic Algorithms in Search, Optimization and Ma- chine Learning, Addison-Wesley, 1989.
- D. Whitley, A genetic algorithm tutorial, Statistics and Computing, 4: 6585, 1994.

Biblioteci AG

- **GAlib** - Biblioteca(de componente pentru Algoritmi Genetici (C++)
- **GAUL** - Genetic Algorithm Utility Library (C)
- **ECJ** - A Java-based Evolutionary Computation Research System
- **Genetic Algorithm Library for Processing** (Java)
- **VectorGA** - A Vectorized Implementation of a Genetic Algorithm in Matlab
- **JAGA** - Java API for Genetic Algorithms
- **JGAP** - Java Genetic Algorithms Package

Aplicaţii ale AG

- **Testare structurală:** Control Flow Graph (CFG) $\rightarrow$ Control Dependence Graph (CDG). Scopuri: atingerea nodurilor, căilor etc. *Pargas et al., Tonella, Tracey, McMinn etc.*
- **Testare funcţională:** Funcţia obiectiv obţinută din precondiţie şi non postcondiţie, pornind de la o specificaţie Z, *Jones et al.*
- **Grey-Box Testing:** Violarea assert-urilor, obţinerea excepţiilor.
- **Testare non-funcţională** WCET de maximizat, iar BCET de minimizat. *Wegener et al.*

Funcţiile obiectiv ale lui Tracey

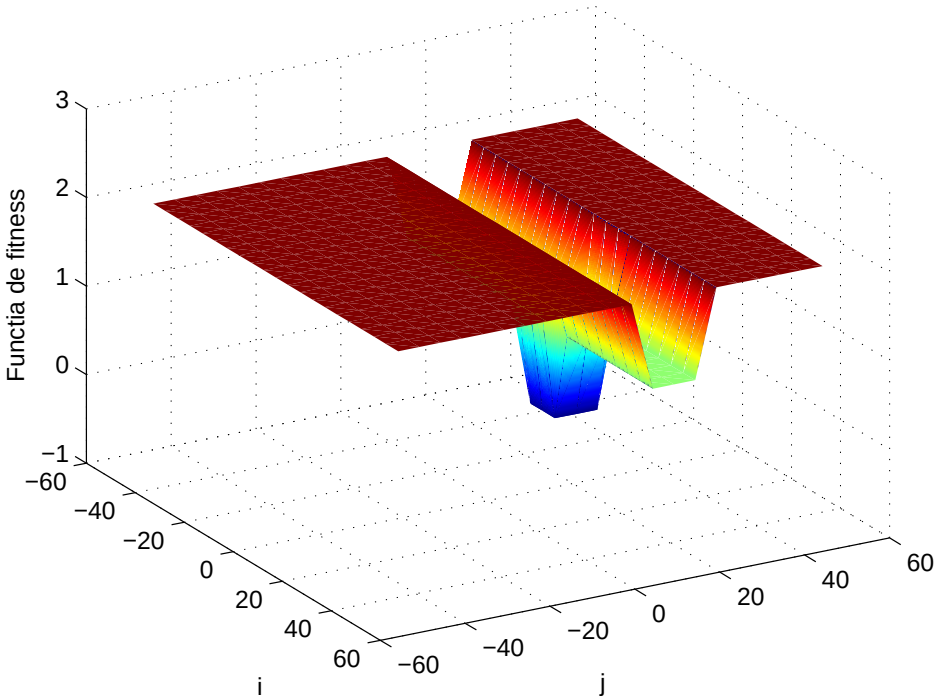
Relational predicate	Objective function obj
Boolean	if $TRUE$ then 0 else K
$a = b$	if $abs(a-b) = 0$ then 0 else $abs(a-b) + K$
$a \neq b$	if $abs(a-b) \neq 0$ then 0 else K
$a < b$	if $a - b < 0$ then 0 else $(a-b) + K$
$a \leq b$	if $a - b \leq 0$ then 0 else $(a-b) + K$
$a > b$	if $b - a < 0$ then 0 else $(b-a) + K$
$a \geq b$	if $b - a \geq 0$ then 0 else $(b-a) + K$
$\neg a$	Negation is moved inwards and propagated over a

Connective	Objective function obj
$a \wedge b$	$obj(a) + obj(b)$
$a \vee b$	$min(obj(a), obj(b))$
a xor b	$obj((a \wedge \neg b) \vee (\neg a \wedge b))$

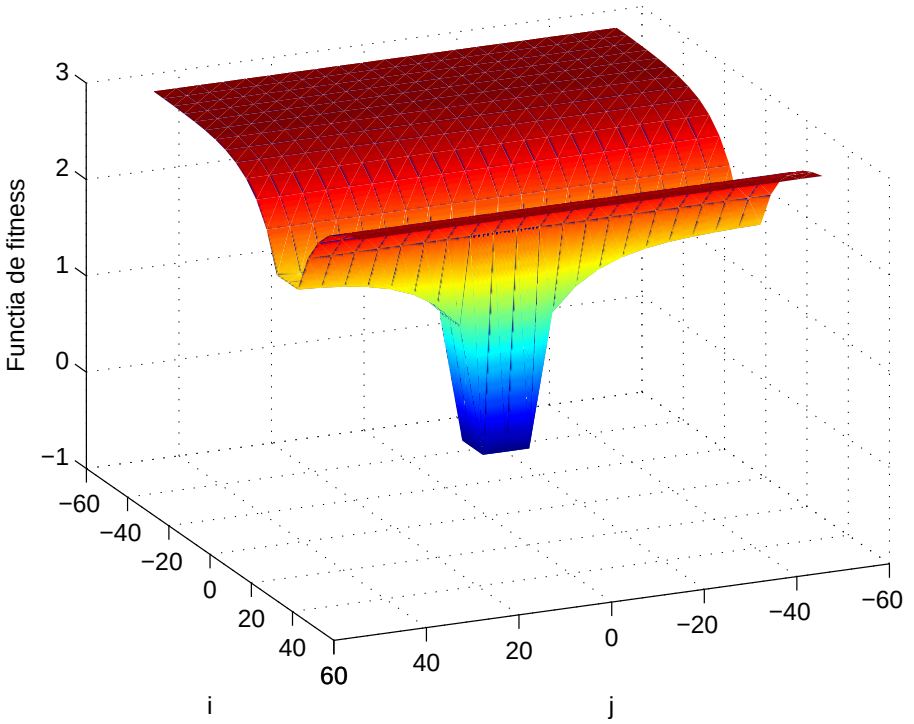
Exemplu

Nod CFG	
	`void example(int i, int j)`
	`{`
1	`  if (i >= 10 && i <= 20)`
	`  {`
2	`  if (j >= 0 && j <= 10)`
	`  {`
3	`  // target statement`
	`  // ...`
	`  }`
	`  }`
	`}`

Funcţie de fitness (Pargas)

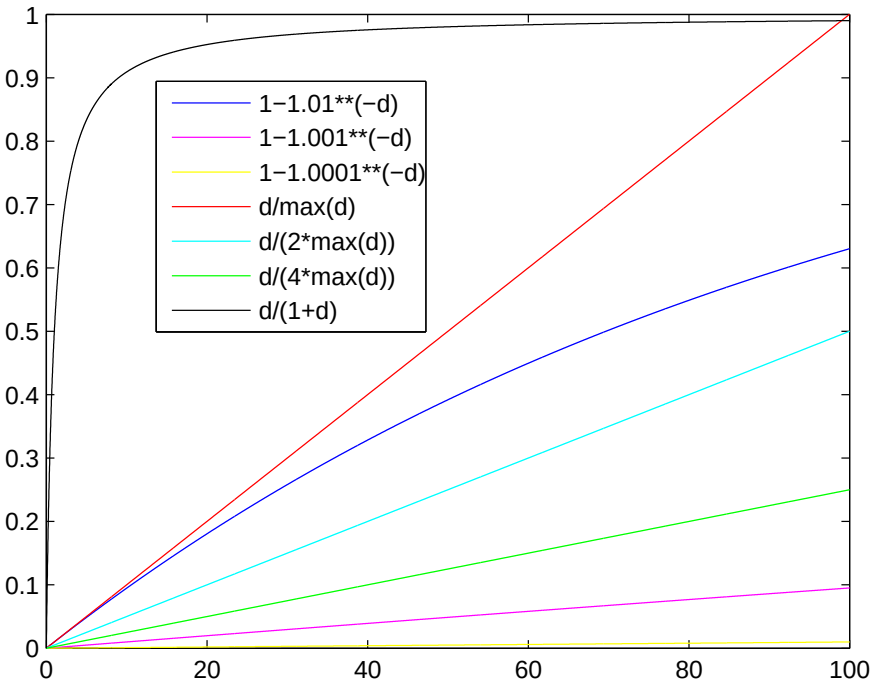


Tracey, Wegener, McMinn



$$(dependent - executed - 1) + n\_branch\_dist$$

Funcţii de normalizare



Bibliografie

- P. McMinn Evolutionary Search for Test Data in the Presence of State Behaviour. PhD Thesis, University of Sheffield, January 2005.
- N. Tracey. A Search-Based Automated Test-Data Generation Framework for Safety Critical Software. PhD thesis, University of York, 2000.
- P. Tonella. Evolutionary testing of classes. In Proceedings of the International Symposium on Software Testing and Analysis, pages 119 128, Boston, USA, 2004. ACM Press.
- P. McMinn, Search-Based Software Test Data Generation: A Survey. Software Testing, Verification and Reliability, 14(2), pp. 105156, 2004.