Laborator 2

Probleme matematice de optim local/global

1. Obiectivele laboratorului

- Însușirea modului de abordare a unei probleme matematice de optim local/global, folosind algoritmi genetici.
- Însușirea unor metode de evitare a blocării algoritmului într-o vecinătate a unui minim local
- Însușirea modului de abordare pentru găsirea celei mai bune aproximații polinomiale a unei funcții date
- Studiul convergenței, a preciziei soluției și a eficienței algoritmului

2. Minimul unei funcții matematice

2.1. Formularea problemei

De multe ori ne întâlnim în inginerie cu problema găsirii minimului (sau al maximului) unei funcții matematice, pentru care nu există întotdeauna o metodă analitică de abordare, mai ales când spatiul unde se caută soluția (mulțimea valorilor posibile) este prea mare. O problemă des întâlnită la algoritmii de optimizare este găsirea minimului global în cazul în care există și minime locale.

Abordăm mai jos o problemă matematică relativ simplă de optimizare, pentru a trece prin toți pașii rezolvării ei folosind algoritmi genetici:

Se dă funcția $f : R \rightarrow R$, $f(x) = x^4 - 2x^2 - x$, se cere să se găsească $x_m \in [-2, 2]$ pentru care f este minimă. Pentru x_m se cere o precizie de $\delta = 0.001$.

2.2. Rezolvarea problemei

Pentru rezolvarea problemei vom face referire la aspectele teoretice din laboratorul 1, capitolul 3. Algoritmul folosit va fi cel prezentat în Figura 1.

Stabilirea structurii cromozomului

Soluția acestei probleme (individul) este un număr real în intervalul [-2, 2]. Având în vedere precizia cerută δ, putem spune că spațiul de căutare al soluției este mulțimea

$S = \{-2, -1.999, -1.998, -1.997, ..., 1.999, 2\}$, cu $x_m \in S$

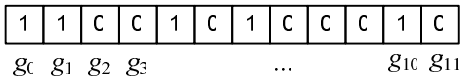
Cardinalul mulțimii S este $|S| = [2 - (-2)] / \delta + 1 = 4001$ valori.

Alegerea modului de construire a cromozomului s-ar putea face folosind o singură genă, având alela un număr fracționar aparținând mulțimii S, dar atunci nu ar exista suficientă diversitate, ceea ce ar împiedica operatorii de încrucișare și mutație să funcționeze eficient. Alegem deci genele de tip *boolean*, cu valorile în mulțimea $A = \{0, 1\}$. Putem privi deci fiecare genă ca un bit, iar întreg cromozomul va codifica un număr în intervalul cerut. Avem nevoie deci de:

$N = log_2 |S| = 11.96614$ biți

Nu putem folosi un număr de N' = 11 biți (gene de tip boolean), pentru că atunci precizia ar deveni δ' = [2 – (-2)] / $2^{N'}$ = 0.00195, ceea ce nu este suficient. Se vor folosi deci N = 12 gene.

Genotipul este mulțimea G a tuturor cromozomilor posibili. Un exemplu de cromozom este următorul:



În figura de mai jos am indicat graficul funcției și noțiunile de bază (cromozom, mapping, individ, funcția de performanță, precizie, etc), cu referire directă pentru această problemă. Graficul funcției este arătat doar cu scop demonstrativ. Se observă că funcția prezintă un minim local și un minim global, diferite, incluse în intervalul [-2, 2].

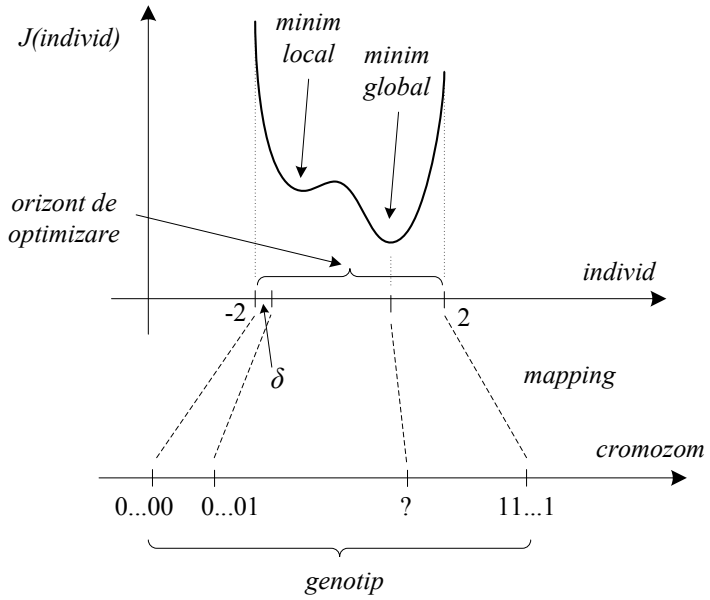


Figura 3. Minimul unei funcții matematice - concepte și noțiuni

Stabilirea funcției de mapping

Se caută o funcție de mapping $M : G \rightarrow [-2, 2]$. Cea mai simplă abordare este să considerăm mulțimea genelor cromozomului un număr binar (în baza 2), să-l transformăm în baza 10, și apoi să-l scalăm la intervalul cerut. Avem deci:

$$M'(chr) = \frac{\sum_{k=0}^{11} g_k \cdot 2^k}{2^{12} - 1} \in [0, 1]$$

$$M(chr) = M'(chr) \cdot 4 - 2 = \frac{\sum_{k=0}^{11} g_k \cdot 2^k}{2^{12} - 1} \cdot 4 - 2 \text{, cu } M(chr) \in [-2, 2]$$

De exemplu, valoarea individului corespunzător cromozomului dat mai sus este:

$$M(chr_1) = \frac{3234}{2^{12}-1} \cdot 4 - 2 = 1.15897436 \in [-2, 2]$$

Stabilirea funcției de performanță

Funcția de performanță $J : [-2, 2] \rightarrow R_+$ poate fi considerată chiar funcția inițială, $J(x) = f(x) = x^4 - 2x^2 - x$, pentru că se caută chiar minimul acestei funcții. Totuși, constrângerile pachetului JGAP impun ca valoarea returnată de funcția de performanță să fie pozitivă, deci vom adăuga o valoare *bias* stabilită empiric la funcția J: $J(x) = f(x) + 5 = x^4 - 2x^2 - x + 5$

2.3. Implementare

Oferim mai jos codul aferent problemei de optimizare specificată.

Secvența de cod 1: Minimizarea unei funcții matematice

```
import org.jgap.*;
import org.jgap.impl.*;

public class FuncGrIV {
  private static final int MAX_ALLOWED_EVOLUTIONS = 10;
  public static void main(String[] args) throws
       InvalidConfigurationException{
    Configuration conf = new DefaultConfiguration();
    Configuration.resetProperty(Configuration.PROPERTY_FITEVAL_INST);
    //low value for fitness = better
    conf.setFitnessEvaluator(new DeltaFitnessEvaluator());
    conf.setPreservFittestIndividual(true);
    conf.setKeepPopulationSizeConstant(true);
    conf.setFitnessFunction(new FitnessFunctionGrIV());

    int nrGenes = 12;
    IChromosome sampleChromosome = new Chromosome(conf,
        new BooleanGene(conf), nrGenes);
    conf.setSampleChromosome(sampleChromosome);

    conf.setPopulationSize(20);
    Genotype population = Genotype.randomInitialGenotype(conf);

    for (int i = 0; i < MAX_ALLOWED_EVOLUTIONS; i++) {
      population.evolve();
      IChromosome bestChrSoFar = population.getFittestChromosome();
      System.out.println(
        "Fitness: " + bestChrSoFar.getFitnessValue() + ", " +
        "individual: " + FitnessFunctionGrII.Mapping((bestChrSoFar)));
    }
  }
}

public class FitnessFunctionGrIV extends FitnessFunction {
  private final double POSITIVE_BIAS = 5;
  public double evaluate(IChromosome chr) {
    double individ = Mapping(chr);
    double fitness = Math.pow(individ,4) - 2 * Math.pow(individ,2) -
individ;
    return fitness + POSITIVE_BIAS;
  }
```

```
   public static double Mapping(IChromosome chr) {
      double base10 = 0;
      for (int i=0; i<chr.size(); i++) {
         boolean allele =
((Boolean)chr.getGene(i).getAllele()).booleanValue();
        if (allele)
           base10 += Math.pow(2, i);
      }
      double individ = base10 / (Math.pow(2, 12) - 1) * 4 - 2;
      return individ;
     }
}
```

2.4. Testarea programului și măsurarea performanțelor

Precizia soluției

Rulând programul, obținem destul de repede (de cele mai multe ori, la generația 6 sau 7) soluția $x_m \approx 1.1071$. Pentru verificarea preciziei, putem folosi resurse web (de exemplu www.wolframalpha.com) pentru găsirea soluției exacte x_m'. Pentru această problemă, $x_m' \approx 1.10716$, deci se confirmă faptul că soluția găsită este în intervalul de precizie cerut față de soluția exactă, adică $| x_m' - x_m| < \delta$

Convergența

Convergența este asigurată datorită configurației de păstrare a celui mai bun individ din fiecare generație și a operatorului de selecție,. Pentru observarea directă a convergenței, recomandăm reprezentarea grafică a abaterii soluției în funcție de iterație.

În unele cazuri, datorită caracterului aleator al căutării, populația de indivizi rămâne „blocată" în vecinătatea minimului local. Aceasta este o problemă des întâlnită și dificil de rezolvat pe caz general la algoritmii de optimizare. Pentru a ocoli această situație se poate apela la unele artificii computaționale, dintre care amintim:

- asigurarea unei diversități suficiente în populația inițială astfel încât să existe cel puțin un individ în vecinătatea convexă a fiecărui minim local. Aceasta se poate asigura modificând mărimea populației.
- folosirea unui operator de selecție care menține în populație și indivizi mai puțin performanți, eventual folosind o probabilitate (de exemplu, selecția de tip turneu sau ruletă),
- generarea, în fiecare generație (sau când, timp de 5 sau 10 generații nu apare o îmbunătățire a soluției), a unui număr de soluții aleatoare care să aducă diversitate în populație. Această funcție este de obicei preluată de operatorul de mutație. Dacă nu sunt performante, soluțiile generate nu vor supraviețui în generația următoare.

Eficiența programului

Se observă ca s-a ales mărimea populației 20 și un număr de 10 de generații. Rezultă deci că, în total, se vor face un număr de $20 \times 10 = 200$ de evaluări într-un spațiu cu $2^{12} = 4096$ soluții posibile (deci aprox. 4.9 % dintre soluțiile posibile sunt evaluate). Putem observa deci că algoritmii genetici, deși au o componentă aleatoare specifică algoritmilor de *brute force*, sunt totuși mult mai performanți decât aceștia din urmă.

3. <u>Problemă propusă: aproximarea unei funcții cu o funcție polinomială</u>

Dezvoltarea funcțiilor în serie Taylor este deseori utilă pentru că reduce problema unei funcții complicate la un polinom ușor calculabil, util de exemplu când avem la dispoziție o putere de calcul redusă (microcontrolere, etc). Totuși, seria Taylor aproximează funcția *în jurul unui punct*, nu *pe un anumit interval*.

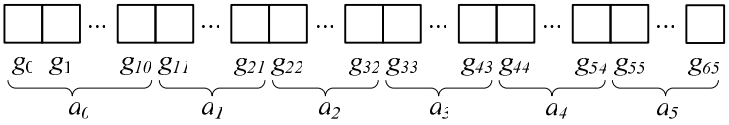
Ne propunem să găsim o funcție polinomială p: $R \rightarrow R$ de grad maxim 5, care aproximează cel mai bine funcția $f : R \rightarrow [-1, 1]$, f(x) = $sin(x)$, pe intervalul $[0, \pi]$.

<u>Stabilirea structurii cromozomului</u>

Notăm funcția polinomială de grad 5 cu $p(x) = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$. Problema devine deci de a găsi coeficienții a_5, a_4, a_3, a_2, a_1, a_0 pentru care aproximarea este optimă (individul este deci funcția $p(x)$). Similar cu problema precedentă, considerăm o precizie $\delta = 0.01$ pentru coeficienți (două zecimale), și căutăm coeficienții numai în intervalul [-10, 10]. Pentru fiecare coeficient avem nevoie deci de:

$$N = \log_2 \frac{10-(-10)}{\delta} = \log_2 2000 \leq 11 \text{ biți.}$$

Cromozomul va avea în consecință 66 de gene cu alela în mulțimea {0, 1} (gene de tip *boolean*), grupate câte 11 pentru fiecare coeficient în parte, astfel:



<u>Stabilirea funcției de mapping</u>

Caută o funcție de mapping $M : G \rightarrow P$, unde G este genotipul care a fost descris mai sus și P este mulțimea tuturor polinoamelor de grad 5. Considerăm cromozomul format din 6 secvențe de câte 11 biți (corespunzătoare fiecărui coeficient), iar apoi, în mod similar cu problema precedentă, transformăm fiecare secvență în baza 10 și o scalăm la intervalul [-10, 10]. Avem deci:

$$M(chr) = \left(\frac{\sum_{k=0}^{10} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^5 + \left(\frac{\sum_{k=11}^{21} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^4 + \ldots + \left(\frac{\sum_{k=55}^{65} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^0$$

<u>Stabilirea funcției de performanță</u>

Performanța unui polinom $p(x)$ se măsoară în funcție de ”cât de bine aproximează funcția dată”. Mai exact, abaterea dintre $p(x)$ și $sin(x)$ trebuie să fie minimă în orice punct pe intervalul $[0, \pi]$. Aceasta o putem formula matematic astfel: căutăm coeficienții $a_5, \ldots, a_0$ pentru care se atinge $M_a = \min_{a5,\ldots,a0} \int_0^{\pi} \left| p(x) - \sin(x) \right| dx$

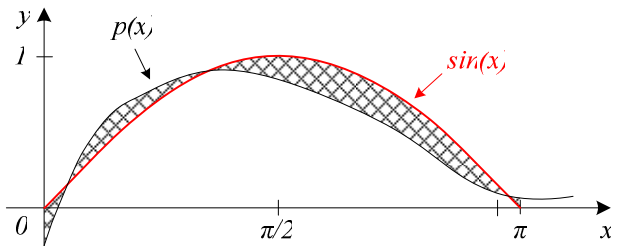


Figura 4. Funcția obiectiv și funcția polinomială

Integrala de mai sus reprezintă de fapt penalizarea pe care o aplicăm unui polinom $p(x)$. Diferența este luată în modul pentru că trebuie penalizat și cazul în care $p(x)$ este sub graficul $sin(x)$ (adică o distanță negativă). Problema de optimizare este deci una de minimizare a funcției de performanță. Pentru a facilita calculul performanței unui polinom, discretizăm integrala și o aproximăm cu o sumă de penalizări individuale. Considerăm un număr de 1000 de puncte intervalul [0, π] și calculăm diferența între valoarea polinomului și funcția dată numai în acele puncte.

$$J\left(p(x)\right)=\sum_{k=0}^{1000}\left|p(\frac{k\cdot\pi}{1000})-\sin(\frac{k\cdot\pi}{1000})\right|$$

Evident, în cazul acestei probleme este nevoie de o populație mai numeroasă și un număr de generații ridicat din cauză că spațiul de căutare este mai mare.

4. Exerciții

1. Rulați programul dat în capitolul 2. Observați precizia soluției și convergența algoritmului. Modificați parametrii programului și notați observațiile.
2. Construiți un program care să reprezinte grafic abaterea soluției găsite în fiecare generație la exercițiul anterior în funcție de numărul generației. Analizați și discutați graficul. Măsurați timpul mediu necesar rulării pentru o generație.
3. Reduceți mărimea populației și generați toată populația inițială în vecinătatea convexă a minimului local. După câte generații depășește algoritmul zona de minim local și din ce cauză ? Cât mai durează până găsește minimul global ?
4. Implementați un program Java care să rezolve problema de la capitolul 3 folosind algoritmi genetici. Pe baza modelului dat în capitolul 2, modificați structura cromozomului și funcția de performanță. Pentru aceasta din urmă, implementați mai întâi o metodă care calculează valoarea unui polinom dat pentru un argument x_0.
5. Analizați spațiul de căutare al soluției pentru problema din capitolul 3. Care este eficiența algoritmului ? Încercați mai multe variante pentru numărul de generații și mărimea populației. Cum afectează fiecare dintre ele performanța algoritmului ?