MODELOWANIE DANYCH POMIAROWYCH ZA POMOCĄ ALGORYTMÓW GENETYCZNYCH (EWOLUCYJNYCH)

dr inż. Igor Skalski

Gdańsk 2017

Spis treści

1. Wstęp

Algorytm genetyczny [1, 2] stanowi sposób na znalezienie optymalnego zestawu wartości, w szczególności zestawu parametrów równania, oparty na naśladowaniu działania zjawiska ewolucji.

Odpowiednio zaprogramowany algorytm genetyczny – w odróżnieniu od metod gradientowych – jest uniwersalny, odporny na występowanie nieciągłości zależności i ekstremów lokalnych. Algorytm genetyczny szybko znajduje przybliżone rozwiązanie i potrafi nadążać za zmiennością danych albo funkcji w czasie. Na szczególną uwagę zasługuje łatwość implementacji funkcji przystosowania, która może zawierać ograniczenia w przestrzeni rozwiązań.

Algorytm genetyczny, jak większość algorytmów optymalizujących, nie gwarantuje osiągnięcia rozwiązania optymalnego, a osiągniecie dokładnych wyników wymaga znacznych mocy obliczeniowych i długiego czasu.

W opisach algorytmów genetycznych stosuje się następujące pojęcia zaczerpnięte z nauki o dziedziczeniu – genetyki:

- populacja – zbiór osobników w jednym pokoleniu
- fenotyp – cechy osobnika mające wpływ na ocenę jego przystosowania
- genotyp – opis cech osobnika zawarty w genach
- chromosom – struktura zawierająca genotyp osobnika
- gen – element składowy chromosomu zawierający pojedynczą wartość
- krzyżowanie – kombinowane łączenie fragmentów chromosomów dwóch osobników
- mutacja – losowa modyfikacja wartości genu
- funkcja przystosowania – funkcja określająca przystosowanie osobnika do środowiska
- błąd przystosowania – określa błąd przystosowania osobnika do środowiska

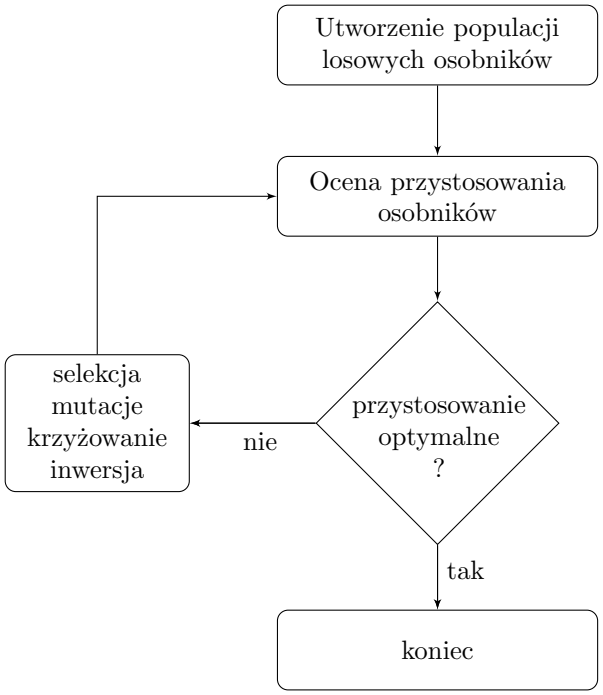
Algorytmy genetyczne znajdują zastosowania w optymalizacji parametrów równań empirycznych. W szczególności są stosowane do optymalizacji sztucznych sieci neuronowych. Algorytmy genetyczne służą też do rozwiązywania trudnych problemów inżynierskich.

2. Podstawy teoretyczne

W praktyce, dla optymalizacji funcji zmiennych rzeczywistych, korzystną formą chromosomów w algorytmie genetycznym jest wektor genów – wartości zmiennoprzecinkowych. Taka postać genów pozwala na optymalizację równań zapisanych w postaci analitycznej, w szczególności równań empirycznych.

Obliczenia polegają na stworzeniu populacji osobników, zawierających chromosomy złożone z genów. Na podstawie funkcji przystosowania (funkcji błędu) z osobników zostają wybrane te, które posiadają najlepsze cechy. Najlepsze osobniki są wykorzystywane do reprodukcji, polegającej na krzyżowaniu chromosomów. Podczas reprodukcji geny poddaje się też losowym mutacjom, a chromosomy ulegają inwersji.

2.1. Algorytm


Utworzenie populacji losowych osobników
Ocena przystosowania osobników
przystosowanie optymalne ?
nie
selekcja
mutacje
krzyżowanie
inwersja
tak
koniec

2.2. Mutacja

Mutacja chromosomu oznacza dokonanie operacji arytmetycznej na wartości losowo wybranego genu:

1,23	2,11	0,67	5,01	3,14	0,07	3,53	3,15	8,46	2,62

$+$

1,21

$=$

1,23	2,11	0,67	5,01	3,14	0,07	4,74	3,15	8,46	2,62

Mutacje mogą również przebiegać jednopunktowo albo wielopunktowo.

Mutacja pozwala na uzyskanie potomstwa znacząco różnego od rodziców. Większość zmian jest niekorzystna. Mutacje często pozwalają na znalezienie bardziej globalnego rozwiązania w sytuacji, gdy dotychczasowa praca algorytmu polegała na optymalizacji w rejonie rozwiązania lokalnego.

2.3. Krzyżowanie

Krzyżowanie chromosomów polega na łączeniu ich losowo wybranych fragmentów, n. p.:

1,23	2,11	0,67	5,01	3,14	0,07	3,22	7,89	2,93	7,57

$+$

7,35	6,47	2,55	2,65	3,18	5,42	3,53	3,15	8,46	2,62

$=$

1,23	2,11	0,67	5,01	3,14	0,07	3,53	3,15	8,46	2,62

W powyższym przykładzie przedstawiono krzyżowanie jednopunktowe.

Krzyżowanie pozwala na uzyskanie osobników potomnych podobnych do ich najbardziej optymalnych rodziców. Większość uzyskanych w ten sposób osobników ma cechy mniej korzystne niż ich rodzice.

2.4. Inwersja

Inwersja jest rzadziej spotykaną operacją, polegającą na odwróceniu kolejności genów we fragmencie chromosomu:

1,23	2,11	0,67	5,01	3,14	0,07	3,53	3,15	8,46	2,62
				↑				↑	

$\Downarrow$

				↓				↓	
1,23	2,11	0,67	5,01	8,46	3,15	3,53	0,07	3,14	2,62

Inwersja stanowi znaczne zaburzenie zestawu genów, pozwalające na opuszczenie lokalnego minimum funkcji przystosowania.

Tablica 1. Dane pomiarowe

x	y
1	10
2	6
3	5
4	4
5	2
6	1
7	2
8	1
9	1
10	0

2.5. Zakończenie obliczeń

Trudnym elementem algorytmu genetycznego jest sposób określenia warunku zakończenia obliczeń. Najczęściej stosowane metody są następujące:

- metoda ekspercka – ręczne przerwanie pracy programu w oparciu o obserwacje wyników obliczeń
- przerwanie po wykonaniu określonej liczby cykli obliczeniowych
- ograniczenie czasu pracy programu
- określenie liczby kroków obliczeniowych wykonywanych po osiągnięciu stanu, w którym nie następuje zmniejszenie błędu przystosowania

3. Przykład użycia algorytmu genetycznego

Algorytmu genetycznego użyto do optymalizacji zależności danych eksperymentalnych przedstawionych w tablicy 1. Wartości zawarto w pliku o nazwie `dane.dat`.

Do odwzorowania zależności użyto równania empirycznego:

$$y = a + b \cdot x^c \tag{1}$$

3.1. Implementacja algorytmu genetycznego

Program zapisany w języku C należy rozpocząć od przyłączenia wykorzystywanych bibliotek oraz zadeklarowania niektórych wykorzystywanych funkcji.

```
#include<stdio.h>
#include<stdlib.h>
 long int random(void);
 void srandom(unsigned int seed);
#include<math.h>
 int isnan(double x);
#include<values.h>
```

Trzy zmienne stanowią parametry pracy algorytmu genetycznego.

- `crosmut` – stanowi stosunek występowania krzyżowania i mutacji może przyjmować wartość z zakresu (0...1),
- `mutrnd` – określa maksymalną wartość mutacji,
- `mutval` – określa prawdopodobieństwo wystąpienia mutacji genu; może przyjmować wartość z zakresu (0...1).

```
double crosmut=0.5;
double mutrnd=2.0;
double mutval=0.5;
```

Do tablic `x` i `y` zostaną wczytane wartości. Po tej operacji zmienna `n` będzie zawierała wartość określającą liczbę zestawów danych.

```
#define DANEN 100

double x[DANEN];
double y[DANEN];
int n;
```

Stała `WSPN` określa liczbę parametrów równania. Zmiana tej wartości wymaga dokonania istotnych zmian w programie.

```
#define WSPN 3
```

Stała `GAPOP` określa liczbę osobników w populacji.

```
#define GAPOP 10
```

Każdy chromosom zawiera tablicę wartości rzeczywistych – genów i ostatnio obliczoną wartość błędu przystosowania.

```
typedef struct
{
  double gene[WSPN];
  double error;
}chromosome_t;

chromosome_t chromosome[GAPOP];
```

Funkcja `get_random_int` zwraca wartość całkowitą z zakresu [`min`...`max`].

```
int get_random_int(int min,int max)
{
  double r=(double)min+
           ((double)rand()/(double)(RAND_MAX))*(((double)max-(double)min));
  if(r<0&&r-(int)r<-0.5)
    r--;
  else
  if(r>0&&r-(int)r>=0.5)
    r++;
  return((int)r);
}
```

Funkcja `equation` zawiera funkcję błędu przystosowania i zwraca wartość funkcji dla x=`x`. Zmienne `a`, `b` i `c` stanowią parametry funkcji.

```
double equation(double x,double a,double b,double c)
{
  return(a+b*pow(x,c));
}
```

Funkcja `errfun` oblicza ogólny błąd przystosowania osobnika/chromosomu `cr` do wszystkich danych.

```
double errfun(int cr)
{
  int i;
  double d=0;
  for(i=0;i<n;i++)
  {
    d+=pow(equation(x[i],
                    chromosome[cr].gene[0],
                    chromosome[cr].gene[1],
                    chromosome[cr].gene[2])-y[i],2.0);
  }
  return(d);
}
```

Funkcje `read_ga` i `write_ga` służą do odczytu i zapisu stanu algorytmu. Dzięki ich zastosowaniu możliwe jest sekwencyjne prowadzenie obliczeń.

```
int read_ga()
{
  int i,j;
  FILE *f;
  if(!(f=fopen("ga.dat","r")))
    return(1);
  for(i=0;i<GAPOP;i++)
```

```
    for(j=0;j<WSPN;j++)
      fscanf(f,"%lg",&chromosome[i].gene[j]);
  if(fclose(f))
    return(1);
  fprintf(stderr,"ok.\n");
  return(0);
}

int write_ga()
{
  int i,j;
  FILE *f;
  if(!(f=fopen("ga.dat","w+")))
    return(1);
  for(i=0;i<GAPOP;i++)
    for(j=0;j<WSPN;j++)
      fprintf(f,"%0.12f ",chromosome[i].gene[j]);
  if(fclose(f))
    return(1);
  return(0);
}
```

Przed rozpoczęciem obliczeń chromosomy wszystkich osobników powinny zawierać wartości losowe. Do tego celu stosowana jest funkcja `ga_randomize_chromosomes`.

```
void ga_randomize_chromosomes()
{
  int i,m;
  FILE *fd;
  if((fd=fopen("/dev/random","r")))
  {
    srandom(getc(fd));
    if(fclose(fd));
  }
  for(m=0;m<GAPOP;m++)
    for(i=0;i<WSPN;i++)
      chromosome[m].gene[i]=(2.0*(double)rand()/(double)RAND_MAX-1.0);
}
```

Krok obliczeniowy składa się z obliczenia błędów przystosowania wszystkich osobników i sortowania według wartości błędu przystosowania.

Na tym etapie możliwe jest przerwanie obliczeń w przypadku spełnienia określonego kryterium. W przedstawionym programie nie zastosowano żadnego kryterium, a przerwania obliczeń dokonywano ręcznie, zatrzymując pracę algorytmu.

Po sortowaniu dokonuje się krzyżowania albo mutacji połowy osobników. W przedstawionym rozwiązaniu połowa najlepiej przystosowanych osobników przechodzi bez zmian do następnego pokolenia.

```
void ga_step()
{
  int i,j;

  /* obliczanie błędów */
  for(i=0;i<GAPOP;i++)
    chromosome[i].error=errfun(i);

  /* sortowanie bąbelkowe */
  for(j=0;j<GAPOP-1;j++)
    for(i=0;i<GAPOP-1-j;i++)
    {
      if((chromosome[i].error<chromosome[i+1].error)||
         (isnan(chromosome[i+1].error)))
      {
        chromosome_t tchromosome=chromosome[i];
        chromosome[i]=chromosome[i+1];
        chromosome[i+1]=tchromosome;
      }
    }

  /* krzyżowanie i mutacje */
  for(i=0;i<GAPOP/2;i++)
  {
    if(((double)rand()/(double)RAND_MAX)>crosmut)
    /* crossover */
    {
      int r=(int)((double)1.0+(double)(WSPN-1)
                  *(double)rand()/(double)RAND_MAX);
      for(j=0;j<r;j++)
        chromosome[i].gene[j]=chromosome[i+GAPOP/2-1].gene[j];
      for(j=r;j<WSPN;j++)
        chromosome[i].gene[j]=chromosome[i+GAPOP/2].gene[j];
    }
    else
    /* mutation */
    {
      for(j=0;j<WSPN;j++)
        if(((double)rand()/(double)RAND_MAX)>mutval)
          chromosome[i].gene[j]+=((mutrnd*(double)rand()/
                                   (double)RAND_MAX)-mutrnd/2.0);
    }
  }
}
```

W głównej części programu następuje kolejno odczyt danych z pliku `dane.dat`, inicjalizacja algorytmu i wykonanie kroku obliczeniowego. Kod pomocniczy służy cyklicznemu wyświetlaniu osiągniętych wyników i zapisywaniu do pliku `dane.dat.ga` danych wykorzystywanych do sporządzenia wykresu.

```
int main(int argc,char *argv[])
{
  int i,j;
  FILE *fd;

  if(!(fd=fopen("dane.dat","r")))
    exit(1);

  i=0;
  {
    while(fscanf(fd,"%lg%lg",
                 &x[i],
                 &y[i])==2)i++;
  }
  n=i;

  fclose(fd);

  fprintf(stderr,"Data number: %d\n",n);

  if(read_ga())
  {
    fprintf(stderr,"read_ga(): failed.\n");
    ga_randomize_chromosomes();
  }

  j=0;
  for(i=0;i<10000000;i++)
  {
    j++;

    ga_step();

    if(j==100000)
    {
      int k;
      char f[100];

      fprintf(stderr,"a=%g b=%g c=%g\n",
              chromosome[GAPOP-1].gene[0],
              chromosome[GAPOP-1].gene[1],
              chromosome[GAPOP-1].gene[2]);

      for(k=0;k<n;k++)
      {
        fprintf(stderr,"%g ",x[k]);
        fprintf(stderr,"%g ",y[k]);
        fprintf(stderr,"%g \n",
                equation(x[k],
                         chromosome[GAPOP-1].gene[0],
                         chromosome[GAPOP-1].gene[1],
```

```
                        chromosome[GAPOP-1].gene[2]));
      }
      fprintf(stderr,"\n\n");

      write_ga();

      j=0;

      {
        double v;
        sprintf(f,"%s.ga","dane.dat");
        if(!(fd=fopen(f,"w+")))
        {
          fprintf(stderr,"ERROR: Can not open output file.\n");
          exit(1);
        }
        for(v=x[0];v<x[n-1];v+=0.01)
        {
          fprintf(fd,"%g %g\n",v,equation(v,
                                          chromosome[GAPOP-1].gene[0],
                                          chromosome[GAPOP-1].gene[1],
                                          chromosome[GAPOP-1].gene[2]));
        }
        fclose(fd);
      }
    }
  }
  return(0);
}
```

Program, zapisany w pliku `ga.c`, kompilowano w systemie Unix–owym za pomocą kompilatora `gcc`:

```
gcc -ansi -pedantic -Wall -lm -O3 ga.c -o ga
```

3.2. Inwersja

Ze względu na niewielką liczbę użytych genów, w powyższym programie nie zastosowano inwersji. Kod realizujący tę operację może mieć następującą postać:

```
    else
    /* inversion */
    {
      int j,k;
      do
      {
        j=get_random_int(1,GENN-2);
        k=get_random_int(1,GENN-2);
      }while(j==k);

      if(j>k)
```

```
    {
      int i=k;
      k=j;
      j=i;
    }

    do
    {
      double d=chromosome[i].gene[j];
      chromosome[i].gene[j]=chromosome[i].gene[k];
      chromosome[i].gene[k]=d;
      j++;
      k--;
    }while(j<k);
  }
```

3.3. Wyniki pracy programu

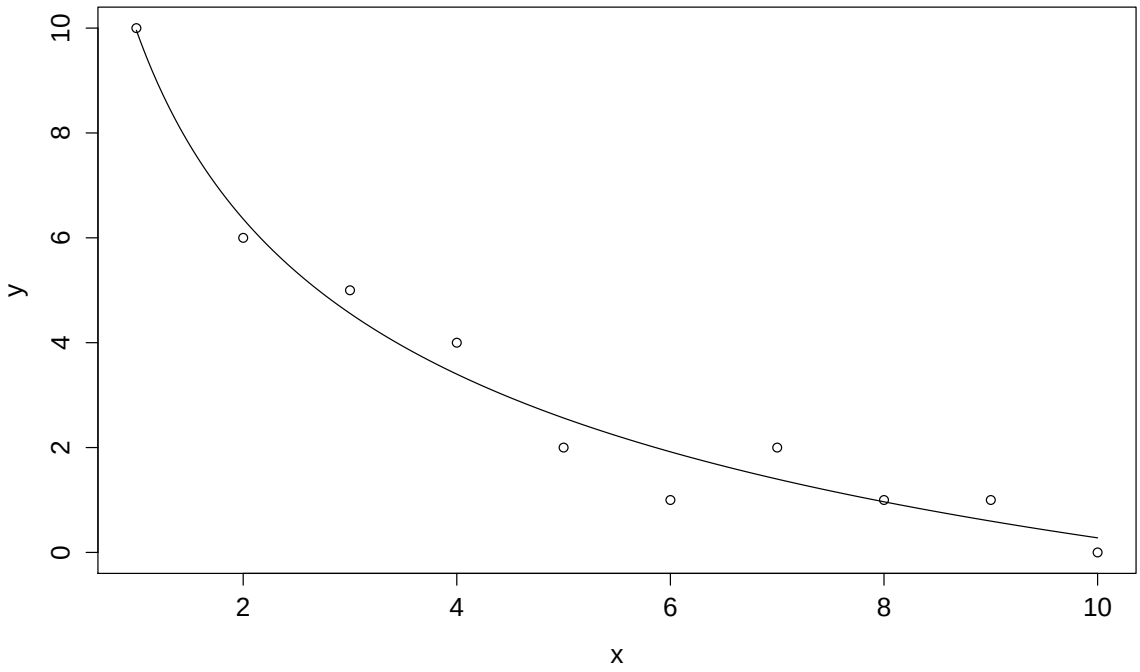
Wywołany program `ga` po chwili wyświetlił przybliżone wartości. Po kilkudziesięciu sekundach wartości te nie ulegały już istotnym zmianom:

```
a=-10.3157 b=20.2743 c=-0.281902
1 10 9.95862
2 6 6.36005
3 5 4.55889
4 4 3.40021
5 2 2.56399
6 1 1.91874
7 2 1.39848
8 1 0.965722
9 1 0.597293
10 0 0.27793
```

Równanie 1, opisujące przybliżoną zależność ma następującą postać

$$y = -10.3157 + 20.2743 \cdot x^{-0.281902} \tag{2}$$

Zależność w postaci graficznej przedstawiono na rys. 1.



Rys. 1. Doświadczalna i modelowana zależność wartości y od x

4. Prawa autorskie

Kod programu przedstawionego w niniejszej pracy może być wykorzystywany zarówno do celów niekomercyjnych, jak i komercyjnych zgodnie z licencją GNU GPL.

5. Literatura

[1] Neville M., Sibley A.,Developing a generic genetic algorithm.

[2] Kowal T., Wojtkiewicz M.,Zastosowanie algorytmłw genetycznych w sieciach neuronowych. 5 stycznia 2000.