Programowanie ekspresji genów

Matejko Marek, Mazur Krzysztof, Paszkot Dawid

25 stycznia 2022

Spis treści

1 Wprowadzenie

Naszym celem było zapoznanie się z programowaniem ekspresji genów.Podczas gdy algorytmy genetyczne i programowanie genetyczne są dobrze znane w literaturze, programowanie ekspresji genów jest mało znane.

Programowanie ekspresji genów jest algorytmem genetycznym genom-fenom, który łączy w sobie prostotę algorytmów genetycznych i możliwości programowania genetycznego.W pewnym sensie programowanie ekspresji genów jest uogólnieniem algorytmów genetycznych i programowania genetycznego. Programowanie ekspresji genów różni się od programowania genetycznego, ponieważ eliminuje się koszt zarządzania strukturą drzewa i zapewniania poprawności programów.

Programowanie ekspresji genów to algorytm tworzący programy lub modele. Te programy są złożonymi strukturami drzewiastymi, które uczą się i przystosowują, zmieniając swoje rozmiary, kształty i skład, podobnie jak żywy organizm. Gen to symboliczny ciąg mający głowę i ogon. Chromosom to zbiór genów.

2 Materiały i metody

Materiały dydaktyczne

W celu zapoznania się z tematem i postawionymi przed nami zadaniami, przeczytaliśmy dostarczoną literaturę GeneExpresion.pdf).
Posłużyliśmy się również materiałami dostępnymi w internecie: różne artykuły związane z tematem zadania na stronie *(en.wikipedia.org)* posłużyły za uzupełnienie naszej wiedzy oraz *(translate.google.com)* w celu sprawdzania tłumaczenia z wyżej podanych materiałów.

Metody

Programy pisaliśmy w języku C++ w środowiskach Visual Studio Code. Użyliśmy bibliotek *<iostream>*, *<cstdlib>* , *<ctime>* , *<cmath>* , *<cstring>*.
Używamy Gene Expression Programming do wykonywania regresji symbolicznej w celu uzyskania map.
Spodziewamy się, że logistyczna mapa zostanie znalezione, ponieważ ograniczamy funkcje w implementacji do wielomianów.
Stosujemy programowanie ekspresji genów, dopóki nie znajdziemy funkcji g, która:

$$fitness(x) < \varepsilon \text{ dla } \varepsilon > 0. \text{ Typowa wartość } \varepsilon = 0.001.$$

3 Implementacja

Funkcja evalr() pobiera ciąg znaków i oblicza odpowiednią funkcję w danym punkcie za pomocą rekurencji. Funkcja eval() używa evalr() do oceny bez modyfikowania argumentu wskaźnika.

```
double evalr(char*& e, double x)
{
    switch (*(e++))
    {
    case '1': return 1.0;
    case 'x': return x;
    case 'y': return pi * x;
    case 'c': return cos(evalr(e, x));
    case 's': return sin(evalr(e, x));
    case '+': return evalr(e, x) + evalr(e, x);
    case '-': return evalr(e, x) - evalr(e, x);
    case '*': return evalr(e, x) * evalr(e, x);
    default: return 0, 0;

    }

}
```

Rysunek 1: Funkcja evalr()

```
double eval(char* e, double x)
{
    char* c = e;
    return evalr(c, x);
}
```

Rysunek 2: Funkcja eval()

Podobnie printr() i print() są odpowiedzialne za wyprowadzanie wyrażeń symbolicznych w czytelnej postaci.

Tablica znaków i tablica typu double są przekazywane do funkcji fitess(). Tablica double składa się z trójek: punktu oceny, oczekiwanej wartości i typu porównania. Jeśli typ porównania wynosi zero, to celem jest równość, Jeśli mniej niż zero, to rzeczywista wartość powinna być mniejsza niż wartość oczekiwana, Jeśli większa niż zero, wtedy wartość rzeczywista powinna być większa niż wartość oczekiwana.

```
void printr(char*& e)
{
    switch (*(e++))
    {
    case '1': cout << '1'; break;
    case 'x': cout << 'x'; break;
    case 'y': cout << "pi*x"; break;
    case 'c': cout << "cos("; printr(e); cout << ")"; break;
    case 's': cout << "sin("; printr(e); cout << ")"; break;
    case '+': cout << '('; printr(e); cout << '+'; printr(e);
        cout << ')'; break;
    case '-': cout << '('; printr(e); cout << '-'; printr(e);
        cout << ')'; break;
    case '*': cout << '('; printr(e); cout << '*'; printr(e);
        cout << ')'; break;

    }
}
```

Rysunek 3: Funkcja printr()

```
double eval(char* e, double x)
{
    char* c = e;
    return evalr(c, x);
}
```

Rysunek 4: Funkcja eval()

```
double fitness(char* c, double* data, int N)
{
    double sum = 0.0;
    double d;
    for (int j = 0; j < N; j++)
    {
        d = eval(c, data[3 * j]) - data[3 * j + 1];
        if (data[3 * j + 2] == 0) sum += fabs(d);
        else if (data[3 * j + 2] > 0) sum -= (d > 0.0) ? 0.0 : d;
        else if (data[3 * j + 2] < 0) sum -= (d < 0.0) ? 0.0 : d;
    }
    return sum;
}
```

Rysunek 5: Funkcja fitness()

Funkcja gep() implementuje algorytm programowania ekspresji genów. Jako argumenty gep() przyjmuje punkty danych, tj. liczbę punktów danych N, wielkość populacji i pożądaną kondycję eps. Funkcja strncpy jest używana do kopiowania określonych regionów łańcuchów znaków reprezentujących

chromosomy. Musimy określić długość, ponieważ chromosomy nie są zakończone zerem.

Używamy 10 symboli dla części głowy reprezentacji, więc całkowita długość genu wynosi 21. Ponieważ używamy tylko x jako symbolu końcowego, otrzymujemy wielomiany rzędu do 11, tj. wielomian najwyższego rzędu obsługiwany przez reprezentację do x^11 . Mamy 0.1 dla prawdopodobieństwa mutacji, 0.4 dla prawdopodobieństwa insercji, 0.7 dla prawdopodobieństwa mutacji. W każdej iteracji algorytmu eliminuje się gorsza połowa populacji.

```
void gep(double* data, int N, int P, double eps)

{
    int i, j, k, replace, replace2, rlen, rp;
    int t = h * (n - 1) + 1;
    int gene_len = h + t;
    int pop_len = P * gene_len;
    int iterations = 0;
    char* population = new char[pop_len];
    char* elim = new char[P];
    int toelim = P / 2;
    double bestf, f;
    double sumf = 0.0;
    double pmutate = 0.1;
    double pinsert = 0.4;
    double precomb = 0.7;
    double r, lastf;
    char* best = (char*)NULL;
    char* iter;
```

Rysunek 6: Funkcja gep()

4 Wyniki

Stwierdzamy, że w większości przypadków najlepiej dopasowaną mapą jest -4(x - 1)x i x(1 - x)(2x + 3)

- ((1 - x) ((1 + (1 + 1)) + 1)) x = -4(x - 1)x
- ((x*(1-x))*(((1+1)+1)+1)) = -4(x - 1)x
- ((x+(((((((1-1)-1)-x)-x)*x)+1)*x))+x) = x(1 - x)(2x + 3)
- (((1-x)-(x-1))*(x+x)) = -4(x - 1)x
- ((((1-(1*(((1+x)+x)*x)))*x)+x)+x) = x(1 - x)(2x + 3)
- (((x-((x+x)*((x+x)-1)))+x)*1) = -4(x - 1)x
- ((1-x)*(x+(((x+x)*1)+x))) = -4(x - 1)x

5 Dyskusja

Udało nam się otrzymać rezultaty zgodne z przewidywaniami teoretycznymi.