Министерством образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Костромской Государственный Университет»
(ФГБОУ ВО «КГУ»)
Институт физико-математических и естественных наук
Направление подготовки/Специальность:
21-ИББо-6б «Информационная безопасность»
Дисциплина: «Технологии индивидуального анализа данных»

Курсовая работа

"Реализация генетического алгоритма в Scilab"

Выполнил: студент Телегин Е.С.
Группа 21-ИББо-6б
Проверил: к.т.н. Алексеев Д.С
Оценка _____
Подпись преподавателя _____

Кострома 2023

Оглавление

Введение....3
1. Реализация в среде Scilab....4
1.1 Разбор кода....4
1.2 Листинг работы кода....9
Заключение....13
Список литературы....14
Приложение....15

Введение

Цель:

- Реализация генетического алгоритма в программно-математической среде Scilab.

Задачи:

- Написание функций реализующих различные этапы алгоритма
- Оптимизация функции: нахождение глобального максимума функции.

Я выбрал данную тему, потому что мне она показалась довольно интересной, так как генетический алгоритм это интерпретация эволюции природы. Также меня эта тема заинтересовала, так как генетические алгоритмы применяются при разработке программного обеспечения, в системах искусственного интеллекта, оптимизации, искусственных нейронных сетях и в других отраслях знаний. Генетические алгоритмы также применяются совместно с нечеткими системами.

Для реализации генетического алгоритма будем рассматривать алгоритм из приложения 1. А также буду применять различные встроенные в Scilab функции, без применения библиотек. В качестве условия проверки конечности алгоритма я выбрал перебор поколений (или же количество итераций). Селекцию буду производить с советующим уменьшением популяции вдвое. В качестве целевой функции выбрал sin(x)*x.

1. Реализация в среде Scilab

1.1 Разбор кода

Определяем начальные параметры алгоритма:

```
PopSize=160; // размер популяции
Crossing=0.9 //вероятность скрещивания
Mutation=0.5; //вероятность мутации
Pokol=5; //количество популяций
Chrom_s = 0; //размеры хромосомы
```

Инициализация начальной популяции. Формируем популяцию исходя из начальных параметров размера, в которой каждая хромосома будет иметь значение в интервале от 1 до 265. Для этого воспользуемся функцией grand и дискретным равномерным распределением (uin) ,которое формирует случайные целые числа, равномерно распределённые между значениями Low и High (включительно). Также находим длину максимальной по значению хромосомы с помощью функции length (функция length возвращает в Chrom_s длину элемента символьной строки max(population))

```
population = grand(PopSize, 1, "uin", 1, 265);
```

Получение длинны максимальной хромосомы. В дальнейшем пригодится для таких этапов как скрещивание и мутация

```
Chrom_s = length(dec2bin(max(population)));
```

Функция приспособленности. Для расчета приспособленности воспользуемся целевой функцией sin(x)*x, выбранной ранее. Для работы функции в качестве параметра передается матрица популяции.

Приведенный ниже код рассчитывает и записывает в нулевую матрицу(**adapted**) значения приспособленности для каждой хромосомы.

```
function adapted=f_adaptet(popul)
    adapted = zeros(size(popul));
    for i =1:size(popul,1)
        adapted(i) = sin(popul(i))*popul(i);
```

```
    end
endfunction
```

Функция селекции хромосом. Для работы функции при ее вызове нужно передать матрицу популяции и матрицу приспособленности хромосом, а также количество отбираемых индивидуумов. После создания нулевой матрицы(**selection**, рассчитываем вероятности выпадения каждой хромосомы(prob) для рулетки и создаем вектор суммированных вероятностей(cumulative_prob).

Вектор суммированных вероятностей (cumulative_probabilities) используется для выбора хромосомы с помощью рулетки. Каждый элемент вектора представляет собой сумму вероятностей всех хромосом до этого индекса включительно. Например, если вероятности выбора хромосом равны [0.2, 0.3, 0.1, 0.4], то вектор суммированных вероятностей будет равен [0.2, 0.5, 0.6, 1.0].

При выборе хромосомы с помощью рулетки генерируется случайное число r от 0 до 1. Затем находится индекс j первой хромосомы, для которой сумма вероятностей до этого индекса больше или равна r. Это позволяет выбирать хромосомы пропорционально их вероятностям, так как более вероятные хромосомы имеют больший вклад в вектор суммированных вероятностей и, следовательно, более вероятно будут выбраны при генерации случайного числа r.

```
function selection=f_select(pop, adapt_mass, num_sel)
    selection = zeros(num_sel,1);
    prob = adapt_mass/sum(adapt_mass);
    // создаем вектор суммированных вероятностей (для рулетки)
    cumulative_prob = cumsum(prob);
    for i = 1:num_sel
        r = rand();
        j = find(cumulative_prob >= r, 1);
        selection(i) = pop(j) ;
```

```
    end
endfunction
```

Функция скрещивания. Для работы функции в качестве параметра передается матрица родителей(**parents)** полученная в результате селекции, а также вероятность скрещивания(**crossover_rate)**. Формируется нулевая матрица(**offspring)** для записи хромосом после скрещивания. Начинаем цикл for начиная с 1 и до размера матрицы с шагом 2(такой шаг нужен тк получаем сразу 2 потомка).

Производим отбор родителей(par1, par2) для скрещивание случайным образом. Задаем точку скрещивания в пределах от 2 до размеров хромосомы(Chrom_s).

Проверяем, если вероятность скрещивания будет больше чем случайное число, то производим скрещивание следующим образом:

i-тому потомку матрицы offspring присваиваем значение полученное путем замены индексов родителя par1 начиная с первого бита до abs(point_cross- Chrom_s) (вычитание длины хромосомы нужно тк функции bitset(см приложение 2) и bitget(см приложение 3) считают биты с правого конца), на вырезаемые биты родителя par2 методом bitget[2]. Аналогично получаем и i+1 потомка.

В другом случае оставляем хромосомы неизменными, они и будут являться потомками.

```
function offspring=crossover(parents, crossover_rate)
    offspring = zeros(size(parents, 1),1);
    for i = 1:2:size(parents,1)
        //выбираем родителей
        rand_1 = floor(rand()*size(parents,1))+1;
        par1 = parents(rand_1);
        parents(rand_1)=[];
        rand_2 = floor(rand()*size(parents,1))+1;
        par2 = parents(rand_2);
```

```
            parents(rand_2)=[];
        if rand() < crossover_rate
            //получаем потомков
            point_cross=grand(1,1,"uin",1, Chrom_s);
            offspring(i) = bitset(par1,[1:abs(point_cross-
Chrom_s)],[bitget(par2,1:abs(point_cross- Chrom_s))]);
            offspring(i+1) = bitset(par2,[1:abs(point_cross-
Chrom_s)],[bitget(par1,1:abs(point_cross- Chrom_s))]);
        else
            offspring(i)= par1;
            offspring(i+1) = par2;
        end
    end
end
```

Функции мутации хромосом. Для работы функции в качестве параметра передается популяция(**popul)** и вероятность мутации(**ver_mutation)**. Создается нулевая матрица мутации хромосом(**mutation)**. Выбирается случайное точка мутации, и проводится проверка на нахождение элемента в этой точке(0 или 1). Проводится мутация путем конструкции if , если случайное число меньше вероятности мутации то мутация производится, иначе хромосома остается неизменной и записывается матрицу мутации. Сама мутация проходит следующим образом: если в этой точке находится 0 то к мутируемому гену присваивается 1 и наоборот. После происходит замена в гена в точке на матируемый ген с помощью bitset (см приложение 2).

```
function mutation=f_mut(popul, ver_mutation)
    mutation = zeros(size(popul,1),1);
    for i=1:size(popul,1)
        if rand() <= ver_mutation
            point_mut=grand(1,1,"uin",1, Chrom_s);
            gen_mut = 0;
```

```
            if bitget(popul(i),abs(point_mut-Chrom_s)+1:abs((point_mut-
Chrom_s)+1))==1
                get_mut = 0;
            else get_mut = 1
            end
            mutation(i)=bitset(popul(i),[point_mut],[gen_mut]);
        else
            mutation(i)=popul(i);
        end
    end
endfunction
```

Функция отбора наилучшей хромосомы. Для работы функции в качестве параметра передается конечная популяция(**pop**). Обозначим лучшую адаптивность(best_fit) например как -5, после рассчитываем приспособленность полученных хромосом конечной популяции(fit) и запускаем цикл for для перебора хромом и отбора той у которой адаптивность будет максимальной.

```
function best_ch=f_best(pop)
    best_fit = -5;
    best_ch = zeros(size(pop));
    fit = f_adaptet(pop)
    disp(fit)
    for i =1:size(pop,1)
        disp(pop(i))
     if fit(i) > best_fit
         best_fit = fit(i);
         best_ch = pop(i);
     end
    end
endfunction
```

Сам генетический алгоритм выглядит следующим образом:

```
// Инициализация начальной популяции
population = grand(PopSize, 1, "uin", 1, 265);
   Chrom_s = length(dec2bin(max(population)));
   for i = 1:Pokol
      // Оценка приспособленности каждого индивидуума
      fitness = f_adaptet(population);
      // Выбор родителей для скрещивания /селекция
         parent = f_select(population,fitness,floor(size(fitness,2)/2));
      // Скрещивание родителей
      offsp = crossover(parent, Crossing);
      input("Скрещиваем хромосомы.Нажмите enter для продолжения:"")
      // Мутация потомства
      mutated_offspring = f_mut(offsp, Mutation);
      population = mutated_offspring;
   end
   // Нахождение лучшего индивидуума в популяции
   best = f_best(population)
```

1.2 Листинг работы кода

Инициализируем популяцию (слева хромосомы в десятичной системе счисления, справа в двоичной)

```
  33.       "000100001"
  58.       "000111010"
  190.      "010111110"
  16.       "000010000"
  20.       "000010100"
  142.      "010001110"
  145.      "010010001"
  106.      "001101010"
  89.       "001011001"
  259.      "100000011"
  166.      "010100110"
  26.       "000011010"
  171.      "010101011"
  137.      "010001001"
  140.      "010001100"
  1.        "000000001"
  250.      "011111010"
  255.      "011111111"
```

Рис 1.1 Инициализация популяции

Узнаем длину максималььной хромосомы

```
Узнаем размер хромосомы.Нажмите

   9.
```

Рис 1.2 Расчет длины хромосомы

Рассчитываем приспособленность каждой хромосомы

```
       column 1 to 17
 32.997091   57.586614   189.58186  -4.6064531   18.258905  -83.466891
       column 18 to 33
-129.12987  -88.190239  -23.525282  -10.832078  -140.19855  -123.46519
       column 34 to 49
 2.847667  -38.362154   195.79247  -173.10083   91.347093   3.1150506
       column 50 to 65
-35.766314   61.587987  -175.47797  -230.00305   17.645654  -31.471212
       column 66 to 82
-137.5584   40.09032   100.0964   163.6803   76.546177   189.17577   11
```

Рис 1.3 Расчет приспособленности

Производим селекцию

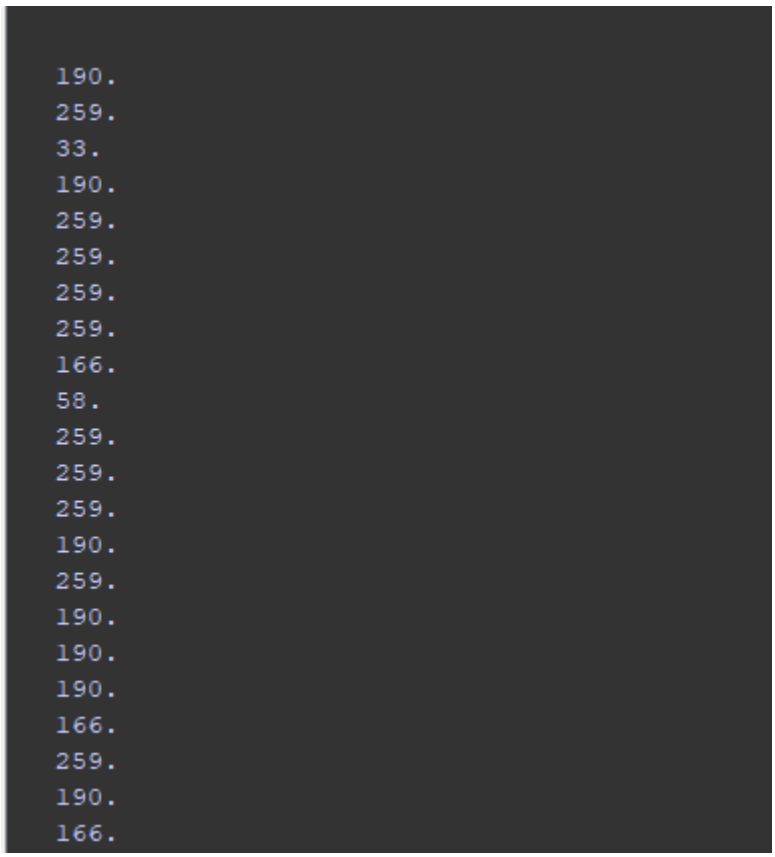


Рис 1.4 Селекция хромосом

Скрещиваем хромосомы (слева популяция до скрещивания в десятичном и двоичном представлении, справа после скрещивания)

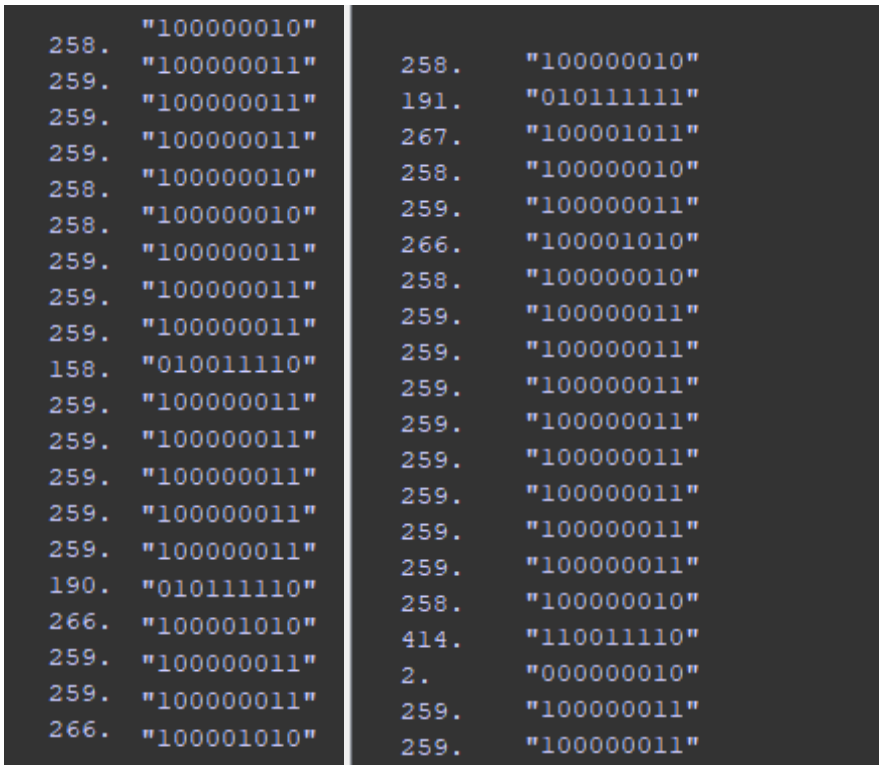


Рис 1.5 Скрещивание хромосом

Производим мутацию

```
 258.          "100000010"
 191.          "010111111"
 267.          "100001011"
 2.            "000000010"
 257.          "100000001"
 266.          "100001010"
 258.          "100000010"
 258.          "100000010"
 3.            "000000011"
 259.          "100000011"
 259.          "100000011"
 259.          "100000011"
 259.          "100000011"
 259.          "100000011"
 257.          "100000001"
 258.          "100000010"
 158.          "010011110"
 2.            "000000010"
 259.          "100000011"
 257.          "100000001"
```

Рис 1.6 Мутация хромосом

Выбираем наилучшую хромосому В результате работы, самой наилучшей оказалась хромосома "259".

```
 97.945992   254.75063   97.945992   97.945992   254.75063

 258.

 259.

 258.

 258.

 259.

 259.

"100000011"
```

Рис 1.7 Отбор наилучшей хромосомы

Заключение

В результате работы над темой был написан алгоритм для нахождения максимального значения хромосомы, в котором производится уменьшение вдвое популяции с каждым поколением.

Работать над данной темой мне понравилось. Наиболее сложным, на мой взгляд, показалось реализовать метод рулетки в селекции, так как не сразу понял, как рассчитать размерность места на рулетке для хромосомы.

Список литературы

Технологии интеллектуального анализа данных - Алексеев Д. С., Щекочихин О. В.

https://help.scilab.org/docs/6.1.1/ru_RU/index.html

Приложение 1

Классический генетический алгоритм

Классический генетический алгоритм состоит из следующих шагов:

1. Инициализация, или выбор исходной популяции хромосом.

2. Оценка приспособленности хромосом в популяции – расчет функции приспособленности для каждой хромосомы.

3. Проверка условия остановки алгоритма.

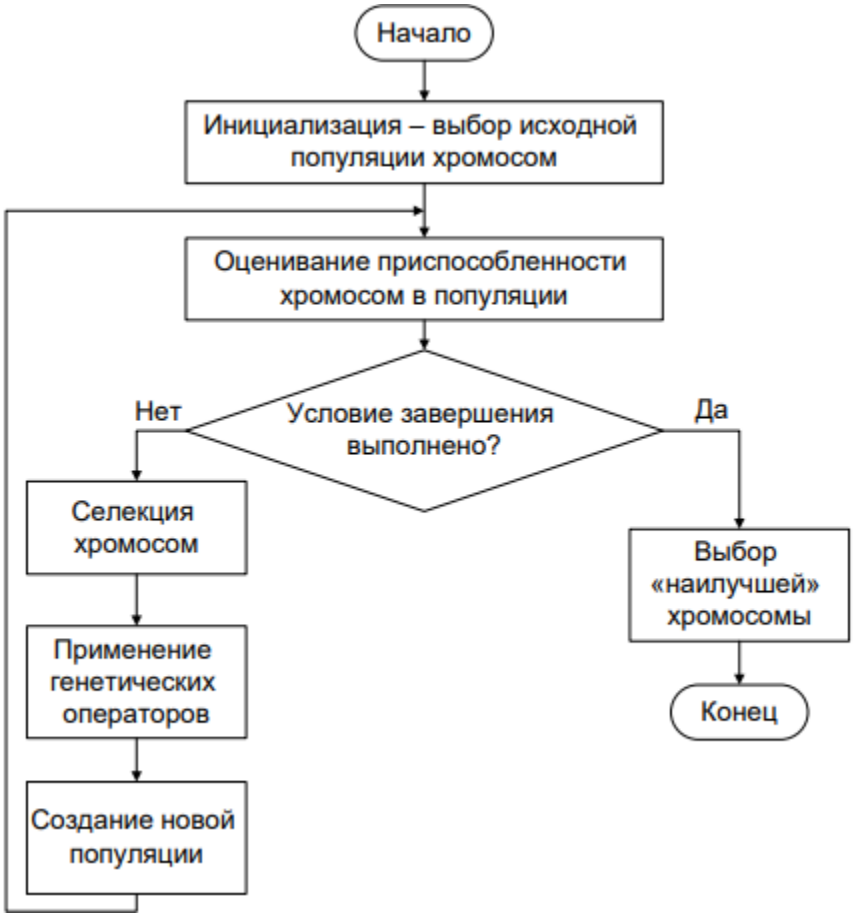
4. Селекция хромосом – выбор тех хромосом, которые будут участвовать в создании потомков для следующей популяции.

5. Применение генетических операторов – мутации и скрещивания.

6. Формирование новой популяции.

7. Выбор «наилучшей» хромосомы.

Рис. 1.8. Блок-схема основного генетического алгоритма



Функция bitset

```
y = bitset(x, bitInd, bitVal)
```

Устанавливает в целых числах биты по указанным индексам

x - положительные десятичные или кодированные целые числа (поддерживаются все типы целых чисел), в которых нужно установить биты.

bitInd - Индексы битов, которые должны быть установлены: массив положительных десятичных или кодированных целых чисел (поддерживаются все типы целых чисел), чьи значения находятся в интервале [1 bitmax], где bitmax - это максимальный индекс битов для типа x

bitVal - Массив значений 0 или 1 в виде десятичных или кодированных целых чисел

Например:

Возьмем матрицу A = [112 215]

И например заменим 1 и 4 индексы элементов матрицы на 1, а также заменим с 1 по 4 индексы на 1111

```
A= [112 215]
disp(dec2bin(A))
ch1 = bitset(A,[1,4],1);
ch2 = bitset(A,[1:4],[1 1 1 1]);
disp(dec2bin(ch1))
disp(dec2bin(ch2))
```

результат выполнения:

```
"01110000"  "11010111"

"01110001"  "11011111"

"01111111"  "11011111"
```

Рис 1.9 результат выполнения функции bitset

Функция biget

```
y = bitget(x, pos)
```

Извлекает из целых чисел биты по указанным индексам

X - Скаляр, вектор, матрица или гиперматрица положительных десятичных или кодированных целых чисел.

pos - Скаляр, вектор, матрица или гиперматрица десятичных или кодированных целых чисел в [1, bitmax], где bitmax - это максимальный индекс битов для типа переменной x: индексы битов, которые следует извлечь.

y - Скаляр, вектор, матрица или гиперматрица из 0 и 1 типа переменной x

Например:

Возьмем ту же матрицу из приложения 2. Извлечём из первого элемента с 1 по 8 индексы, а из второго с 3 по 5

```
A= [112 215]
disp(dec2bin(A))
ch1 = bitget(A(1),1:8);
ch2 = bitget(A(2),3:5);
disp(ch1)
disp(ch2)
```

```
"01110000"  "11010111"

 0.   0.   0.   0.   1.   1.   1.   0.

 1.   0.   1.
```

Рис 1.10 результат выполнения функции bigset