# 遗传算法

## 实验目的

用遗传算法找到最适合当前环境的最优个体

## 二、算法分析

遗传算法（Genetic Algorithm）是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码，初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代（generation）演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度（fitness）大小选择（selection）个体，并借助于自然遗传学的遗传算子（genetic operators）进行组合交叉（crossover）和变异（mutation），产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码（decoding），可以作为问题近似最优解。

## 三、实验题目

1.评估每条染色体所对应个体的适应度。  
      2.按照适应度越高，选择概率越大的原则，从种群中选择两个个体作为父方和母方。  
      3.抽取父母双方的染色体，进行交叉，产生子代。  
      4.对子代的染色体进行变异。  
      5.重复2，3，4步骤，直到新种群的产生。

## 四、实验代码

**package** lab\_8;

**public** **class** GA {

**private** **int** ChrNum = 10; //染色体数量

**private** String[] ipop = **new** String[ChrNum]; //一个种群中染色体总数

**private** **int** generation = 0; //染色体代号

**public** **static** **final** **int** ***GENE*** = 46; //基因数

**private** **double** bestfitness = Double.***MAX\_VALUE***; //函数最优解

**private** **int** bestgenerations; //所有子代与父代中最好的染色体

**private** String beststr; //最优解的染色体的二进制码

/\*\*

\* 初始化一条染色体（用二进制字符串表示）

\*/

**private** String initChr() {

String res = "";

**for** (**int** i = 0; i < ***GENE***; i++) {

**if** (Math.*random*() > 0.5) {

res += "0";

} **else** {

res += "1";

}

}

**return** res;

}

/\*\*

\* 初始化一个种群(10条染色体)

\*/

**private** String[] initPop() {

String[] ipop = **new** String[ChrNum];

**for** (**int** i = 0; i < ChrNum; i++) {

ipop[i] = initChr();

}

**return** ipop;

}

/\*\*

\* 将染色体转换成x,y变量的值

\*/

**private** **double**[] calculatefitnessvalue(String str) {

//二进制数前23位为x的二进制字符串，后23位为y的二进制字符串

**int** a = Integer.*parseInt*(str.substring(0, 23), 2);

**int** b = Integer.*parseInt*(str.substring(23, 46), 2);

**double** x = a \* (6.0 - 0) / (Math.*pow*(2, 23) - 1); //x的基因

**double** y = b \* (6.0 - 0) / (Math.*pow*(2, 23) - 1); //y的基因

//需优化的函数

**double** fitness = 3 - Math.*sin*(2 \* x) \* Math.*sin*(2 \* x)

- Math.*sin*(2 \* y) \* Math.*sin*(2 \* y);

**double**[] returns = { x, y, fitness };

**return** returns;

}

/\*\*

\* 轮盘选择

\* 计算群体上每个个体的适应度值;

\* 按由个体适应度值所决定的某个规则选择将进入下一代的个体;

\*/

**private** **void** select() {

**double** evals[] = **new** **double**[ChrNum]; // 所有染色体适应值

**double** p[] = **new** **double**[ChrNum]; // 各染色体选择概率

**double** q[] = **new** **double**[ChrNum]; // 累计概率

**double** F = 0; // 累计适应值总和

**for** (**int** i = 0; i < ChrNum; i++) {

evals[i] = calculatefitnessvalue(ipop[i])[2];

**if** (evals[i] < bestfitness){ // 记录下种群中的最小值，即最优解

bestfitness = evals[i];

bestgenerations = generation;

beststr = ipop[i];

}

F = F + evals[i]; // 所有染色体适应值总和

}

**for** (**int** i = 0; i < ChrNum; i++) {

p[i] = evals[i] / F;

**if** (i == 0)

q[i] = p[i];

**else** {

q[i] = q[i - 1] + p[i];

}

}

**for** (**int** i = 0; i < ChrNum; i++) {

**double** r = Math.*random*();

**if** (r <= q[0]) {

ipop[i] = ipop[0];

} **else** {

**for** (**int** j = 1; j < ChrNum; j++) {

**if** (r < q[j]) {

ipop[i] = ipop[j];

}

}

}

}

}

/\*\*

\* 交叉操作 交叉率为60%，平均为60%的染色体进行交叉

\*/

**private** **void** cross() {

String temp1, temp2;

**for** (**int** i = 0; i < ChrNum; i++) {

**if** (Math.*random*() < 0.60) {

**int** pos = (**int**)(Math.*random*()\****GENE***)+1; //pos位点前后二进制串交叉

temp1 = ipop[i].substring(0, pos) + ipop[(i + 1) % ChrNum].substring(pos);

temp2 = ipop[(i + 1) % ChrNum].substring(0, pos) + ipop[i].substring(pos);

ipop[i] = temp1;

ipop[(i + 1) / ChrNum] = temp2;

}

}

}

/\*\*

\* 基因突变操作 1%基因变异

\*/

**private** **void** mutation() {

**for** (**int** i = 0; i < 4; i++) {

**int** num = (**int**) (Math.*random*() \* ***GENE*** \* ChrNum + 1);

**int** chromosomeNum = (**int**) (num / ***GENE***) + 1; // 染色体号

**int** mutationNum = num - (chromosomeNum - 1) \* ***GENE***; // 基因号

**if** (mutationNum == 0)

mutationNum = 1;

chromosomeNum = chromosomeNum - 1;

**if** (chromosomeNum >= ChrNum)

chromosomeNum = 9;

String temp;

String a; //记录变异位点变异后的编码

**if** (ipop[chromosomeNum].charAt(mutationNum - 1) == '0') { //当变异位点为0时

a = "1";

} **else** {

a = "0";

}

//当变异位点在首、中段和尾时的突变情况

**if** (mutationNum == 1) {

temp = a + ipop[chromosomeNum].substring(mutationNum);

} **else** {

**if** (mutationNum != ***GENE***) {

temp = ipop[chromosomeNum].substring(0, mutationNum -1) + a

+ ipop[chromosomeNum].substring(mutationNum);

} **else** {

temp = ipop[chromosomeNum].substring(0, mutationNum - 1) + a;

}

}

//记录下变异后的染色体

ipop[chromosomeNum] = temp;

}

}

**public** **static** **void** main(String args[]) {

GA Tryer = **new** GA();

Tryer.ipop = Tryer.initPop(); //产生初始种群

String str = "";

//迭代次数

**for** (**int** i = 0; i < 1000000; i++) {

Tryer.select();

Tryer.cross();

Tryer.mutation();

Tryer.generation = i;

}

**double**[] x = Tryer.calculatefitnessvalue(Tryer.beststr);

str = "最小值" + Tryer.bestfitness + '\n' + "第"

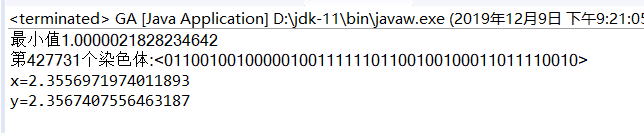
+ Tryer.bestgenerations + "个染色体:<" + Tryer.beststr + ">" + '\n'

+ "x=" + x[0] + '\n' + "y=" + x[1];

System.***out***.println(str);

}

## }五、实验结果：



## 六、实验总结

遗传算法是一种模拟自然界生物进化过程，就是遵照“物竞天择适者生存”的原则，通过不断的筛选，选出最适合当前环境当前模型的目标。自然选择的过程从选择群体中最适应环境的个体开始。后代继承了父母的特性，并且这些特性将添加到下一代中。如果父母具有更好的适应性，那么它们的后代将更易于存活。迭代地进行该自然选择的过程，最终，我们将得到由最适应环境的个体组成的一代。