应用遗传算法快速寻找游戏装备的最优组合

广州一方信息资讯有限公司

邢兴

08250178

摘要：本文介绍了应用遗传算法解决游戏装备的最优组合问题。首先会简单介绍什么是游戏装备的最优组合问题，以及研究该问题的实际意义。并介绍了解决该问题的现有方法——穷举法。然后给出了游戏装备的最优组合问题的数学语言描述，以及针对该问题应用遗传算法所需关键点，如编码方式、评估函数等。大量实验数据表明寻找游戏装备的最优组合遗传算法要优于穷举法。

关键字：遗传算法，组合问题，游戏装备

内容目录

1 简介.....1
2 游戏装备问题.....2
2.1 数学模型.....2
2.2 遗传算法.....3
2.3 求解步骤.....4
3 实验.....5
3.1 装备库种类 n←2、最大装备数量 m←50.....6
3.2 装备库种类 n←4、最大装备数量 m←100.....7
3.3 装备库种类 n←6、最大装备数量 m←100.....8
4 总结.....9
引用.....10
课程感言.....10
代码.....11

1 简介

大型多人在线游戏开发过程中，装备数值设定是个令人头痛的问题。例如游戏中可以选

择头、手、足、胸等多个部位多种装备进行组合，在增强游戏角色的某种能力的同时削弱另一种能力。游戏玩家在游戏过程中收集不同装备进行搭配，获得高于其他玩家的能力。而开发人员也希望了解当前的装备数值设定是否真正能够达到预期的目的。其中游戏装备中最优组合对游戏中装备等级的划分，装备的持有率，以及游戏平衡性都有重要的指导作用。特别是“极品装备”对于游戏运营的效益有极为密切的联系。

现有开发中，在寻找游戏装备最优组合时最常用的是穷举法。计算出所有可能的装备组合，并比较寻找最优组合。这种方法能保证寻找到的是真正的最优组合。但是在电子游戏迅速发展的今天，伴随着大型多人在线游戏的迅速扩张，游戏装备品种越来越多，数量越来越大，等级越来越细。穷举法的计算时间会随着游戏装备的品种、数量、等级的增长以几何级数增加。这使得游戏数值的调整、新的游戏装备的添加所需要的验证时间越来越长。大量的时间浪费在穷举计算的等待上。

使用近似计算寻找游戏装备的最优组合，可以极大的提高计算效率。加快近似最优组合的求解。虽然近似计算得到的可能的最优组合不一定是真正的最优组合，但是求解本身是为了给游戏开发和运营提供参考，所以也是具有实际意义的。这其中遗传算法采用的编码、交叉、选色，整个过程可以无须大的修改隐射到游戏装备最优组合的寻找问题中。本文将全面描述应用遗传算法寻找游戏装备的最优组合问题（以下简称“装备问题”）。

第 2 章将对装备问题进行描述，同时会给出该问题的数学定义、装备问题遗传算法中基因编码、交叉、选则的方法、和求解步骤。第 3 章会给出一个用于实验的“装备问题”。为了更好的说明遗传算法应用，用于实验的“装备问题”经过了简化。删除了与计算无关的旁支末节，保留了“装备问题”的核心内容。同时实验环境、实验内容、实验结果和与穷举法的对比也在第 3 章进行了阐述。

2　游戏装备问题

2.1 小节描述了“装备问题”的数学模型。2.2 小节概述了一般性的遗传算法，并针对“装备问题”的数学模型设计了遗传算法的几个要素：基因编码、交叉、选择。2.3 小结描述了应用遗传算法求解“装备问题”的主要步骤。

2.1　数学模型

设某游戏提供 n 种装备库 $E \leftarrow \{E_1 \ldots E_n\}$，游戏角色提供编号为 1...n 插槽分别设置这些装

备。每个插槽只能且必须安装对应编号装备库中的一只装备。每种装备库中有 m 个可选装备 $E_i \leftarrow \{x_i^1 \ldots x_i^m\}$，其中 $1 \leqslant i \leqslant n$，$1 \leqslant m \leqslant MAX$。合理的装备方案 $X \leftarrow \{x_1 \ldots x_n\}$，$x_i$ 来自对应的 E_i。每个装备为游戏角色提供了分值 $S_i^j \leftarrow Obj(x_i^j)$，$1 \leqslant j \leqslant m$。现求一种装备方式，使得 $\sum S_i$ 取得最大值，其中 $1 \leqslant i \leqslant n$。

标准模型:

$$\max \quad \sum S_i$$

$$\text{s.t.} \quad S_i = Obj(x_i)$$

$$x_i^j \in E_i$$

$$E_i \in E$$

$$1 \leqslant j \leqslant count(E_i)$$

$$1 \leqslant i \leqslant n$$

2.2 遗传算法

遗传算法（Genetic Algorithm）是模拟达尔文的遗传选择和自然淘汰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，它是有美国 Michigan 大学 J.Holland 教授于 1975 年首先提出来的，并出版了颇有影响的专著《Adaptation in Natural and Artificial Systems》，GA 这个名称才逐渐为人所知，J.Holland 教授所提出的 GA 通常为简单遗传算法（SGA）。

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。

应用遗传解决“装备问题”：将设置装备的插槽 1...n 分别作为基因位（Locus），每个基因位选择对应序号的装备库中的装备编号作为等位基因（Alletes）。这 n 个基因组成的 1...n 序列为一条染色体。

由于每个装备只可以设置在对应的插槽中，所以进行最简单的单点交叉就可保证交叉后新解的合理性。

	1	2	3		i		n-2	n-1	n
父代 X	x_1	x_2	x_3	……	x_i	……	x_{n-2}	x_{n-1}	x_n
父代 Y	y_1	y_2	y_3	……	y_i	……	y_{n-2}	y_{n-1}	y_n
子代 a	x_1	x_2	x_3	……	y_i	……	y_{n-2}	y_{n-1}	y_n
子代 b	y_1	y_2	y_3	……	x_i	……	x_{n-2}	x_{n-1}	x_n

表 1

评估函数 eval(x) ← Obj(x) + A，即目标函数 Obj(x) ← ΣS_i 加上一个足够大的正数 A，A 总是满足 A > $|\Sigma S_i|$，以保证 eval(x) 总是为正。

特别的，为了避免在已经寻找到可能最优解的情况下，仍然完成剩余的迭代，增加了最优解稳定次数，作为第二个退出条件。在经过 s 次迭代后，都没有找到一个比当前记录的最优解评估函数更好的解时，退出迭代，将当前记录的最优解作为可能最优解输出。而这个 s 即为最优解稳定次数。这样就可以将迭代次数尽可能加大，扩大最优解搜索范围的同时，不浪费计算时间。

2.3 求解步骤

Step 0: 初始化 选择种群大小 P；初始化可能装备组合 $X_1...X_p$，其中 X_i = $\{x_1...x_n\}$；交叉概率 p_c；变异概率 p_m；子代数限制 t_{max}；当前代数 t，并设置 t←0；最优解稳定次数限制 s_{max}；当前最优解稳定次数 s，并设置 s←0；从初始种群中选择评估函数值最大的解，令 X_{best}←X_i。

Step 1: 终止条件 如果 t = t_{max} 或 s = s_{max} 时，X_{best} 即为可能的最优装备组合。输出并终止计算。

Step 2: **产生新子代**

Substep 1: **交叉** 根据交叉概率 p_c，在当前种群中选取 P * p_c 条装备组合进行单点交叉，其中第 1 条同第 P * p_c 条交叉，其余第 i 条同第 i - 1 条交叉。得到新的 P * p_c 条染色体放入待选池中。

Substep 2: **变异** 根据变异概率 p_m，在当前种群中选取 P * p_m 条染色体进行变异，改变这 P * p_m 条染色体每条染色体的随机一位基因。得到的新的 P * p_m 条染色体放入待选池中。

Substep 3: **选择** 从当前种群中选择评估函数 eval(X) 最大的装备组合 X_b，加入新的子代；将待选池中的装备组合按照评估函数 eval(X) 的值从高到低排列，选择前 P - 1 条加入新的子代。

Substep 4: **更新最优** 在新子代中选择评估函数 eval(X) 最大的染色体 X_{nb}，如果 $eval(X_{nb}) > eval(X_{best})$，则更新 $X_{best} \leftarrow X_{nb}$；否则 s←s+1。

Step 3: **迭代** 增加 t←t+1，并跳转到 Step1。

3 实验

在实验中，虚拟了一个机器人大战游戏。该游戏提供 n 种装备库分别装备机器人的头、胸和四肢。每个部位只能且必须安装对应编号装备库中的一只装备。每种装备库中的装备数量在 1 到 m 之间。每个装备为机器人提供了一定的分值，为了说明问题，这里简化成一个线性分值。装备提供的分值可能为正，也可能为负。分值越高，战斗力越强。要寻找一种装备方式，使得机器人在这种装备方式下战斗力最强。

实验软件环境使用 Linux 系统，Java 6.0SE。硬件环境使用 Intel Centrino 1.6 GHz CPU，512M 内存。

令:

子代数 t←1E+5;

最优解稳定次数 s←1E+3;

交叉概率 $p_c \leftarrow 0.8$;

变异概率 $p_m \leftarrow 0.1$;

种群数量　　P←100

3.1　装备库种类 n←2、最大装备数量 m←50

遗传算法计算:

序号	eval(x)	计算时间 (ms)	子代数	退出条件
1	188.9334441363862	482	1004	$s = s_{max}$
2	188.9334441363862	478	1002	$s = s_{max}$
3	188.9334441363862	480	1004	$s = s_{max}$
4	188.9334441363862	478	1001	$s = s_{max}$
5	188.9334441363862	483	1003	$s = s_{max}$
6	188.9334441363862	486	1008	$s = s_{max}$
7	188.9334441363862	487	1001	$s = s_{max}$
8	188.9334441363862	480	1002	$s = s_{max}$
9	188.9334441363862	483	1001	$s = s_{max}$
10	188.9334441363862	490	1007	$s = s_{max}$
平均值	188.9334441363862	482.7	1003.3	
标准差	0	3.77	2.37	
答优率	100.00%			

表2

穷举计算:

序号	eval(x)	计算时间 (ms)	计算次数
1	188.9334441363862	76	315
2	188.9334441363862	87	315
3	188.9334441363862	87	315
4	188.9334441363862	102	315
5	188.9334441363862	94	315
6	188.9334441363862	76	315

7	188.9334441363862	94	315
8	188.9334441363862	78	315
9	188.9334441363862	92	315
10	188.9334441363862	87	315
平均值	188.9334441363862	87.3	315
标准差	0	8.19	0
答优率	100.00%		

表 3

从 3.1 实验数据可以得出：

1．在 n、m 规模较小时，遗传算法和穷举法都可以寻找到最优解。

2．遗传算法在获得了可能的最优解后花费了大量的时间来验证这个可能最优解是否稳定，所以在 n、m 规模较小时，穷举法所消耗的计算总时间远小于遗传算法。遗传算法的 10 次计算退出条件全部是达到可能最优解稳定次数上限。同时实际计算次数同可能最优解稳定次数上限接近。例如遗传算法第 7 次计算，在第一代中已经寻找到了可能的最优解。然后继续计算了 1000 次，以便验证这个可能的最优解是否稳定。

3.2　装备库种类 n←4、最大装备数量 m←100

遗传算法计算：

	eval(x)	计算时间（ms）	子代数	退出条件
1	387.3774374339101	614	1142	$s = s_{max}$
2	387.3774374339101	634	1181	$s = s_{max}$
3	387.3774374339101	723	1349	$s = s_{max}$
4	387.3774374339101	582	1064	$s = s_{max}$
5	387.3774374339101	890	1677	$s = s_{max}$
6	387.3774374339101	632	1177	$s = s_{max}$
7	387.3774374339101	821	1546	$s = s_{max}$
8	387.3774374339101	574	1060	$s = s_{max}$
9	387.3774374339101	717	1341	$s = s_{max}$
10	387.3774374339101	596	1103	$s = s_{max}$

平均值	387.3774374339101	678.3	1264	
标准差	0	101.9	200.03	
答优率	100%			

表 4

穷举计算：

	eval(x)	计算时间 (ms)	计算次数
1	387.3774374339101	1475	3527415
2	387.3774374339101	1518	3527415
3	387.3774374339101	1493	3527415
4	387.3774374339101	1502	3527415
5	387.3774374339101	1507	3527415
6	387.3774374339101	1500	3527415
7	387.3774374339101	1535	3527415
8	387.3774374339101	1482	3527415
9	387.3774374339101	1499	3527415
10	387.3774374339101	1489	3527415
平均值	387.3774374339101	1500	3527415
标准差	0	16.5	0
答优率	100.00%		

表 5

从 3.2 实验数据可以得出：

1. 在适当增加了 n、m 的数值，整体规模增加的时候，遗传算法计算的优势已经体现出来。
2. 穷举计算在计算时间上是稳定的，每次计算所花费的时间基本一致。
3. 遗传算法在计算时间上并不稳定，每次计算所花费的时间差异很大。但仍然能保证计算结果收敛于正确结果。

3.3 装备库种类 n←6、最大装备数量 m←100

遗传算法计算：

	eval(x)	计算时间 (ms)	子代数	退出条件

1	567.9881772350686	749	1239	$s = s_{max}$
2	567.9881772350686	1780	3024	$s = s_{max}$
3	567.9881772350686	719	1197	$s = s_{max}$
4	567.9881772350686	1966	3324	$s = s_{max}$
5	567.9881772350686	815	1364	$s = s_{max}$
6	567.9881772350686	1629	2783	$s = s_{max}$
7	567.9881772350686	1013	1708	$s = s_{max}$
8	567.9881772350686	1002	1682	$s = s_{max}$
9	567.9881772350686	958	1596	$s = s_{max}$
10	567.9881772350686	818	1368	$s = s_{max}$
平均值	567.9881772350686	1144.9	1928.5	
标准差	0	440.28	757.69	
答优率	100%			

表 6

穷举计算：

	eval(x)	计算时间（ms）	计算次数
1	567.9881772350686	-	1108672512

表 7

从 3.3 实验数据可以得出：

1．穷举法在计算规模进一步增大后，计算次数陡然增大了 314 倍，未能在可接受的时间里计算出正确结果。

2．遗传算法在计算规模进一步增大后，在计算次数和计算时间上略微增加。

3．遗传算法计算结果的标准差进一步增加，但仍然能保证计算结果收敛于正确结果。

4 总结

综合上述实验表明，计算规模增大时，穷举法的计算开销（时间、计算次数）会承几何级数增长。而遗传算法的计算开销（时间、计算次数）增加平稳。但是由于遗传算法本身的设计思路，遗传算法无法保证每次的计算都能够寻找到最优解。且每次计算所花费的时间差

异明显。但是对于如“装备问题”等需要指导性数据的实际问题中，一个可能的最优解即可对游戏数值调校加以引导。

随着游戏产业的发展，游戏玩家要求提高，游戏内容也越来越丰富。游戏中出现的各种组合问题也越来越复杂。实际开发中，装备库种类 n 达到十几或者几十种，而每个库中最大装备数量 m 可能数百甚至上千。采用穷举计算寻找游戏中装备的最优组合是异常困难。实验表明，在寻找游戏中装备的最优组合问题上，遗传算法能够发挥其解决复杂组合问题的优势，快速的寻找到游戏中装备的最优组合。

对于其他类型游戏，如策略游戏。在寻找最佳发展策略这个问题上，遗传算法也可以发挥其威力。这是我们下一步要进行的研究。

引用

[1] Go1dberg, David E (1989)，遗传算法：搜索、优化和机器学习，K1uwer Academic Publishers, Boston, MA.

[2] Mitche11, Me1anie (1996)，遗传算法概论，MIT Press, Cambridge, MA.

[3] Vose, Michae1 D (1999)，简单遗传算法：基础和理论，MIT Press, Cambridge, MA.

[4] Rona1d L. Rardin (2007)，运筹学：优化模型与算法，电子工业出版社，pp. 695-697. ISBN 978-7-121-04925-5.

[5] 衣杨 (2008)，数据建模与优化课程课件：第 4 章，pp. 47-67.

[6] 衣杨 (2008)，数据建模与优化课程课件：第 5 章，pp. 66-83.

代码

/src/ga4robot/Chromosome.java

```
 1 package ga4robot;
 2
 3 import java.util.ArrayList;
 4 import java.util.Random;
 5
 6 /**
 7  * 染色体
 8  * @author 邢兴
 9  */
10 public class Chromosome {
11     /**
12      * 基因组
```

```
13      */
14     private ArrayList<Gene> genes = null;
15     /**
16      * 染色体构造函数
17      * @param n     基因位数目
18      */
19     public Chromosome(int n) {
20         genes = new ArrayList<Gene>(n);
21     }
22
23     private Chromosome() {
24
25     }
26
27     /**
28      * 在 id 位上设置基因 gene
29      * @param id    基因位
30      * @param gene  基因
31      */
32     public void setGene(int id, Gene gene) {
33         genes.set(id, gene);
34     }
35
36     /**
37      * 在 id 位上添加基因 gene
38      * @param id    基因位
39      * @param gene  基因
40      */
41     public void addGene(int id, Gene gene) {
42         genes.add(id, gene);
43     }
44     /**
45      * 获取 id 位的基因
46      * @param id    基因位
47      * @return      基因
48      */
49     public Gene getGene(int id) {
50         return genes.get(id);
51     }
52
53     /**
54      * 重载 clone 方法
55      * @return      染色体
56      */
57     @Override
58     public Chromosome clone() {
59         Chromosome chromosome = new Chromosome();
60         chromosome.genes = (ArrayList<Gene>)genes.clone();
61         return chromosome;
62     }
63     /**
64      * 交叉
65      * @param chromosome    与当前染色体交叉的令一染色体
66      * @return              交叉得到新的两条染色体
67      */
68     public ArrayList<Chromosome> crossover(Chromosome chromosome) {
69         Random r = new Random();
70         int x = r.nextInt(genes.size());
71         ArrayList<Chromosome> list = new ArrayList<Chromosome>(2);
72         Chromosome c1 = new Chromosome(genes.size());
73         Chromosome c2 = new Chromosome(genes.size());
74         for(int i = 0; i < genes.size(); i ++) {
```

```
75              if (i < x) {
76                  c1.addGene(i, genes.get(i));
77                  c2.addGene(i, chromosome.getGene(i));
78              } else {
79                  c1.addGene(i, chromosome.getGene(i));
80                  c2.addGene(i, genes.get(i));
81              }
82          }
83          list.add(c1);
84          list.add(c2);
85          return list;
86      }
87      /**
88       * 变异
89       */
90      public void mutation() {
91          Random gr = new Random();
92          // 选取随机一个位置的基因
93          int id = gr.nextInt(genes.size());
94          GeneGroup group = GeneGroup.getGroup(id);
95          //从基因组中随意抽取一个，替换所选基因
96          genes.set(id, group.getRandomGene());
97      }
98      /**
99       * 评估函数
100      * @return 评估函数值
101      */
102     public double evaluation() {
103         double value = 0;
104         for(int i = 0; i < genes.size(); i ++) {
105             value += genes.get(i).getValue();
106         }
107         // 确保评估函数 eval(x) > 0
108         value += genes.size() * 30;
109         return value;
110     }
111 }
112
113
```

/src/ga4robot/Config.java

```
 1 package ga4robot;
 2
 3 import java.io.FileInputStream;
 4 import java.io.FileOutputStream;
 5 import java.io.IOException;
 6 import java.util.Properties;
 7
 8 /**
 9  * 配置文件
10  * @author 邢兴
11  */
12 public class Config {
13     /**
14      * 种群大小
15      */
16     private int population = 0;
17     /**
18      * 基因位数
19      */
20     private int geneCount = 0;
```

```
21      /**
22       * 交叉概率
23       */
24      private double crossoverPossible = 0;
25      /**
26       * 变异概率
27       */
28      private double mutationPossible = 0;
29      /**
30       * 子代数
31       */
32      private int generationCount = 0;
33      /**
34       * 最优解稳定次数
35       */
36      private int maxEqualsCount = 0;
37      /**
38       * 装备最大数量
39       */
40      private int maxGenes = 0;
41
42      /**
43       * 配置构造函数
44       * @throws java.io.IOException
45       */
46      public Config() throws IOException {
47          Properties p = new Properties();
48          p.load(new FileInputStream("config.property"));
49          generationCount =
Integer.parseInt(p.getProperty("generationCount"));
50          maxEqualsCount = Integer.parseInt(p.getProperty("maxEqualsCount"));
51          population = Integer.parseInt(p.getProperty("population"));
52          geneCount = Integer.parseInt(p.getProperty("geneCount"));
53          crossoverPossible =
Double.parseDouble(p.getProperty("crossoverPossible"));
54          mutationPossible =
Double.parseDouble(p.getProperty("mutationPossible"));
55          maxGenes = Integer.parseInt(p.getProperty("maxGenes"));
56      }
57
58      /**
59       * 初始化
60       * @throws java.io.IOException
61       */
62      public static void init() throws IOException {
63          Properties p = new Properties();
64          p.setProperty("generationCount", "100000");
65          p.setProperty("maxEqualsCount", "500");
66          p.setProperty("population", "100");
67          p.setProperty("geneCount", "4");
68          p.setProperty("crossoverPossible", "0.8");
69          p.setProperty("mutationPossible", "0.1");
70          p.setProperty("maxGenes", "50");
71          p.store(new FileOutputStream("config.property"), "for GA");
72      }
73
74      /**
75       *
76       * @return 子代数量
77       */
78      public int getGenerationCount() {
79          return generationCount;
```

```
 80     }
 81
 82     /**
 83      *
 84      * @return 可能最优解稳定次数
 85      */
 86     public int getMaxEqualsCount() {
 87         return maxEqualsCount;
 88     }
 89     /**
 90      *
 91      * @return 种群数量
 92      */
 93     public int getPopulation() {
 94         return population;
 95     }
 96
 97     /**
 98      *
 99      * @return 基因位数
100      */
101     public int getGeneCount() {
102         return geneCount;
103     }
104
105     /**
106      *
107      * @return 交叉概率
108      */
109     public double getCrossoverPossible() {
110         return crossoverPossible;
111     }
112
113     /**
114      *
115      * @return 变异概率
116      */
117     public double getMutationPossible() {
118         return mutationPossible;
119     }
120
121     /**
122      *
123      * @return 每基因位可选最大基因数
124      */
125     public int getMaxGenes() {
126         return maxGenes;
127     }
128
129     /**
130      * 单件模式
131      */
132     private static Config config = null;
133
134     public static Config singlation() throws IOException {
135         if (config == null) {
136             config = new Config();
137         }
138         return config;
139     }
140 }
```

/src/ga4robot/Gene.java

```
 1 package ga4robot;
 2
 3 /**
 4  * 基因
 5  * @author 邢兴
 6  */
 7 public class Gene {
 8     /**
 9      * 基因值
10      */
11     private double value = 0;
12     /**
13      * 基因位
14      */
15     private int id = 0;
16     /**
17      * 构造函数
18      * @param id        基因位
19      * @param value     基因值
20      */
21     public Gene(int id, double value) {
22         this.id = id;
23         this.value = value;
24     }
25
26     /**
27      *
28      * @return 基因值
29      */
30     public double getValue() {
31         return value;
32     }
33
34     /**
35      *
36      * @return 基因位
37      */
38     public int getId() {
39         return id;
40     }
41 }
```

/src/ga4robot/GeneGroup.java

```
 1 package ga4robot;
 2
 3 import java.io.IOException;
 4 import java.io.InputStream;
 5 import java.io.OutputStream;
 6 import java.util.ArrayList;
 7 import java.util.Random;
 8 import javax.xml.parsers.*;
 9 import org.w3c.dom.Document;
10 import org.w3c.dom.NodeList;
11 import org.xml.sax.SAXException;
12
13
14 /**
15  * 代选基因组
16  * @author 邢兴
17  */
18 public class GeneGroup {
```

```
19
20      /**
21       * 基因组
22       */
23      private ArrayList<Gene> genes = null;
24      /**
25       * 基因位
26       */
27      private int id = 0;
28
29      /**
30       * 基因组构造函数
31       * @param id    基因位
32       * @param n     基因组大小
33       */
34      public GeneGroup(int id, int n) {
35          genes = new ArrayList<Gene>(n);
36          this.id = id;
37      }
38
39      /**
40       * 设置基因
41       * @param id    基因位
42       * @param gene  基因
43       */
44      public void setGene(int id, Gene gene) {
45          genes.set(id, gene);
46      }
47      /**
48       * 添加基因
49       * @param id    基因位
50       * @param gene  基因
51       */
52      public void addGene(int id, Gene gene) {
53          genes.add(id, gene);
54      }
55
56      /**
57       * 获取基因
58       * @param id    基因位
59       * @return      基因
60       */
61      public Gene getGene(int id) {
62          return genes.get(id);
63      }
64
65      /**
66       * 随机获取基因
67       * @return  基因
68       */
69      public Gene getRandomGene() {
70          Random r = new Random();
71          return genes.get(r.nextInt(genes.size()));
72      }
73
74      /**
75       * 基因组大小
76       * @return
77       */
78      public int getSize() {
79          return genes.size();
80      }
```

```
 81
 82      /**
 83       * 单件模式
 84       */
 85      private static ArrayList<GeneGroup> group = null;
 86
 87      /**
 88       * 随机初始化测试数据
 89       * @param n 基因位数
 90       * @throws java.io.IOException
 91       */
 92      public static void initGroup(int n) throws IOException {
 93          group = new ArrayList<GeneGroup>(n);
 94          for(int ggi = 0; ggi < n; ggi ++) {
 95              Random ggr = new Random();
 96              int ggmax = ggr.nextInt(Config.singlation().getMaxGenes()) + 1;
 97              // 每个基因组有 1 .. MAX_GENE_IN_GROUP 个待选基因
 98              GeneGroup g = new GeneGroup(ggi, ggmax);
 99              for(int gi = 0; gi < ggmax; gi ++) {
100                  Random gr = new Random();
101                  // 每个基因价值在 -30 到 70 之间
102                  double value = (gr.nextDouble() - 0.3) * 100;
103                  Gene gene = new Gene(gi, value);
104                  g.addGene(gi, gene);
105              }
106              group.add(ggi, g);
107          }
108      }
109
110      /**
111       * 保存基因组
112       * @param config    输出流
113       * @throws java.io.IOException
114       * @throws org.xml.sax.SAXException
115       * @throws javax.xml.parsers.ParserConfigurationException
116       */
117      public static void saveGroup(OutputStream config) throws IOException,
SAXException, ParserConfigurationException {
118          config.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?
>\n<root>\n".getBytes());
119          for(GeneGroup gg: group) {
120              config.write("\t<genegroup>\n".getBytes());
121              for(Gene g : gg.genes) {
122                  config.write(("\t\t<gene>" + g.getValue() +
"</gene>\n").getBytes());
123              }
124              config.write("\t</genegroup>\n".getBytes());
125          }
126          config.write("</root>\n".getBytes());
127      }
128
129      /**
130       * 加载基因组
131       * @param config    输入流
132       * @throws java.io.IOException
133       * @throws org.xml.sax.SAXException
134       * @throws javax.xml.parsers.ParserConfigurationException
135       */
136      public static void loadGroup(InputStream config) throws IOException,
SAXException, ParserConfigurationException {
137          Document doc = DocumentBuilderFactory.newInstance().
138                  newDocumentBuilder().parse(config);
```

```
139         NodeList ggNode = doc.getElementsByTagName("genegroup");
140         group = new ArrayList<GeneGroup>(ggNode.getLength());
141         for(int ggi = 0; ggi < ggNode.getLength(); ggi ++) {
142             NodeList gNode = ggNode.item(ggi).getChildNodes();
143             int gmax = gNode.getLength();
144             GeneGroup g = new GeneGroup(ggi, gmax);
145             int i = 0;
146             for(int gi = 0; gi < gmax; gi ++) {
147                 if (gNode.item(gi).getNodeName().equals("gene")) {
148                     double value =
Double.parseDouble(gNode.item(gi).getTextContent());
149                     Gene gene = new Gene(gi, value);
150                     g.addGene(i, gene);
151                     i ++;
152                 }
153             }
154             group.add(ggi, g);
155         }
156     }
157 
158     /**
159      * 获取组
160      * @param id    基因位
161      * @return  基因组
162      */
163     public static GeneGroup getGroup(int id) {
164         return group.get(id);
165     }
166 
167     /**
168      *
169      * @return 基因组数，即基因位数
170      */
171     public static int getGroupSize() {
172         return group.size();
173     }
174 }
```

/src/ga4robot/Generation.java

```
 1 package ga4robot;
 2 
 3 import java.io.IOException;
 4 import java.io.InputStream;
 5 import java.util.ArrayList;
 6 import java.util.Random;
 7 import javax.xml.parsers.ParserConfigurationException;
 8 import org.xml.sax.SAXException;
 9 
10 /**
11  * 子代
12  * @author 邢兴
13  */
14 public class Generation {
15     /**
16      * 种群大小
17      */
18     private int population = 0;
19     /**
20      * 基因位数
21      */
22     private int geneCount = 0;
```

```
23      /**
24       * 交叉概率
25       */
26      private double crossoverPossible = 0;
27      /**
28       * 变异概率
29       */
30      private double mutationPossible = 0;
31      /**
32       * 染色体组
33       */
34      private ArrayList<Chromosome> chromosomes = null;
35      /**
36       * 历史最优解
37       */
38      private Chromosome bestC = null;
39
40      /**
41       * 构造函数，加载配置文件，随机初始化基因组
42       * @throws java.io.IOException
43       */
44      public Generation() throws IOException{
45          loadConfig();
46          GeneGroup.initGroup(geneCount);
47          init();
48      }
49
50      /**
51       * 构造函数
52       * @param config    基因组输入流
53       * @throws java.io.IOException
54       * @throws org.xml.sax.SAXException
55       * @throws javax.xml.parsers.ParserConfigurationException
56       */
57      public Generation(InputStream config) throws IOException, SAXException,
58              ParserConfigurationException {
59          loadConfig();
60          GeneGroup.loadGroup(config);
61          init();
62      }
63      /**
64       * 从配置文件中加载数据
65       * @throws java.io.IOException
66       */
67      private void loadConfig() throws IOException{
68          population = Config.singlation().getPopulation();
69          geneCount = Config.singlation().getGeneCount();
70          crossoverPossible = Config.singlation().getCrossoverPossible();
71          mutationPossible = Config.singlation().getMutationPossible();
72      }
73      /**
74       * 初始化
75       */
76      private void init() {
77          chromosomes = new ArrayList<Chromosome>(population);
78          for(int i = 0; i < population; i++) {
79              Chromosome chromosome = new Chromosome(geneCount);
80              for(int j = 0; j < geneCount; j ++) {
81                  chromosome.addGene(j,
GeneGroup.getGroup(j).getRandomGene());
82              }
83              chromosomes.add(chromosome);
```

```
 84         }
 85     }
 86     /**
 87      * 新子代
 88      */
 89     public void newGeneration() {
 90         // 选取进行交叉，越优秀机会越高
 91         ArrayList<Chromosome> crossPop = selection(crossoverPossible);
 92         for(int i = 0; i < crossPop.size(); i ++) {
 93             if (i == 0) {
 94 
chromosomes.addAll(crossPop.get(0).crossover(crossPop.get(crossPop.size() -
1)));
 95             } else {
 96                 chromosomes.addAll(crossPop.get(i).crossover(crossPop.get(i
- 1)));
 97             }
 98         }
 99 
100         // 变异
101         Random r = new Random();
102         int mutationCount = (int)(chromosomes.size() * mutationPossible);
103         for(int i = 0; i < mutationCount; i ++) {
104             chromosomes.get(r.nextInt(chromosomes.size())).mutation();
105         }
106         // 选择
107         chromosomes = selection(population);
108         Chromosome tmpC = getBest();
109         if (bestC == null || bestC.evaluation() < tmpC.evaluation()) {
110             bestC = tmpC;
111         }
112     }
113     /**
114      *
115      * @return 历史最优
116      */
117     public Chromosome getHistoryBest() {
118         return bestC;
119     }
120 
121     /**
122      *
123      * @return 当前最优
124      */
125     public Chromosome getBest() {
126         int max = 0;
127         for(int i = 1; i < chromosomes.size(); i ++) {
128             if (chromosomes.get(max).evaluation() <
chromosomes.get(i).evaluation()) {
129                 max = i;
130             }
131         }
132         return chromosomes.get(max);
133     }
134 
135     /**
136      *
137      * @return 评估函数合
138      */
139     private double totalEval() {
140         double value = 0;
141         for(Chromosome c: chromosomes) {
```

```
142             value += c.evaluation();
143         }
144         return value;
145     }
146 
147     /**
148      * 选择
149      * @param percent   选择概率
150      * @return  染色体组
151      */
152     private ArrayList<Chromosome> selection(double percent) {
153         double total = totalEval();
154         double p[] = new double[chromosomes.size()];
155 
156         for(int i = 0; i < chromosomes.size(); i ++) {
157             if (i == 0) {
158                 p[i] = chromosomes.get(0).evaluation() / total;
159             } else {
160                 p[i] = p[i-1] + chromosomes.get(i).evaluation() / total;
161             }
162         }
163 
164         int crossoverCount = (int)(chromosomes.size() * percent);
165         Random r = new Random();
166         ArrayList<Chromosome> result = new
ArrayList<Chromosome>(crossoverCount);
167         result.add(getBest());
168         while(result.size() < crossoverCount) {
169             double k = r.nextDouble();
170             for(int j = chromosomes.size() - 1; j >= 0 ; j --) {
171                 if (k > p[j]) {
172                     result.add(chromosomes.get(j));
173                     break;
174                 }
175             }
176         }
177         return result;
178     }
179 
180     /**
181      * 选择
182      * @param count 总数
183      * @return
184      */
185     private ArrayList<Chromosome> selection(int count) {
186         double total = totalEval();
187         double p[] = new double[chromosomes.size()];
188 
189         for(int i = 0; i < chromosomes.size(); i ++) {
190             if (i == 0) {
191                 p[i] = chromosomes.get(0).evaluation() / total;
192             } else {
193                 p[i] = p[i-1] + chromosomes.get(i).evaluation() / total;
194             }
195         }
196 
197         Random r = new Random();
198         ArrayList<Chromosome> result = new ArrayList<Chromosome>(count);
199         result.add(getBest());
200         while(result.size() < count) {
201             double k = r.nextDouble();
202             for(int j = chromosomes.size() - 1; j >= 0 ; j --) {
```

```
203                 if (k > p[j]) {
204                     result.add(chromosomes.get(j));
205 
206                     break;
207                 }
208             }
209         }
210         return result;
211     }
212 }
```

/src/ga4robot/Main.java

```
  1 package ga4robot;
  2 
  3 import java.io.BufferedReader;
  4 import java.io.FileInputStream;
  5 import java.io.FileNotFoundException;
  6 import java.io.FileOutputStream;
  7 import java.io.IOException;
  8 import java.io.InputStream;
  9 import java.io.InputStreamReader;
 10 import java.io.OutputStream;
 11 import java.util.Date;
 12 import javax.xml.parsers.ParserConfigurationException;
 13 import org.xml.sax.SAXException;
 14 
 15 /**
 16  * 主程序
 17  * @author 邢兴
 18  */
 19 public class Main {
 20 
 21     /**
 22      * 开始时间
 23      */
 24     private static long startTime = 0;
 25 
 26     /**
 27      * 记录开始时间
 28      */
 29     private static void startTime() {
 30         Date d = new Date();
 31         startTime = d.getTime();
 32     }
 33 
 34     /**
 35      *
 36      * @return 返回运行时间
 37      */
 38     private static long endTime() {
 39         Date d = new Date();
 40         return d.getTime() - startTime;
 41     }
 42 
 43     /**
 44      * 生成配置文件
 45      * @throws java.io.FileNotFoundException
 46      * @throws java.io.IOException
 47      * @throws org.xml.sax.SAXException
 48      * @throws javax.xml.parsers.ParserConfigurationException
 49      */
 50     private static void makeConfig() throws FileNotFoundException,
```

```
IOException,
 51             SAXException, ParserConfigurationException {
 52         GeneGroup.initGroup(Config.singlation().getGeneCount());
 53         OutputStream os = new FileOutputStream("config.xml");
 54         GeneGroup.saveGroup(os);
 55         os.close();
 56     }
 57 
 58     /**
 59      * 穷举
 60      * @return 最优解评估函数值
 61      * @throws java.io.FileNotFoundException
 62      * @throws java.io.IOException
 63      * @throws org.xml.sax.SAXException
 64      * @throws javax.xml.parsers.ParserConfigurationException
 65      */
 66     private static double exhaustingCount() throws FileNotFoundException,
 67             IOException, SAXException, ParserConfigurationException {
 68         InputStream config = new FileInputStream("config.xml");
 69         GeneGroup.loadGroup(config);
 70 
 71         // 遍历键值
 72         int key[] = new int[GeneGroup.getGroupSize()];
 73         // 总数
 74         int total = 1;
 75         for (int i = 0; i < GeneGroup.getGroupSize(); i++) {
 76             key[i] = 0;
 77             total *= GeneGroup.getGroup(i).getSize();
 78         }
 79         System.out.println("Total: " + total);
 80         double result = 0;
 81 
 82         for (int i = 0; i < total; i++) {
 83             Chromosome c = new Chromosome(GeneGroup.getGroupSize());
 84              for (int k = 0; k < GeneGroup.getGroupSize(); k++) {
 85                  c.addGene(k, GeneGroup.getGroup(k).getGene(key[k]));
 86             }
 87             if (result < c.evaluation()) {
 88                 result = c.evaluation();
 89             }
 90 
 91             for(int j = 1; j <= GeneGroup.getGroupSize(); j ++) {
 92                 key[j - 1] ++;
 93                 if (key[j - 1] == GeneGroup.getGroup(j - 1).getSize()) {
 94                     key[j - 1] = 0;
 95                 } else {
 96                     break;
 97                 }
 98             }
 99         }
100 
101         return result;
102     }
103 
104     /**
105      * 遗传算法
106      * @return 最优解评估函数值
107      * @throws java.io.FileNotFoundException
108      * @throws java.io.IOException
109      * @throws org.xml.sax.SAXException
110      * @throws javax.xml.parsers.ParserConfigurationException
111      */
```

```
112     private static double runGA() throws FileNotFoundException, IOException,
113             SAXException, ParserConfigurationException {
114         //begin-初始化
115         InputStream config = new FileInputStream("config.xml");
116         Generation g = new Generation(config);
117         double best = 0, newbest = 0;
118         Chromosome gbest = null;
119         int i = 0, j = 0;
120         //end-初始化
121         startTime();
122         while ((i < Config.singlation().getGenerationCount()) &&
123                 (j < Config.singlation().getMaxEqualsCount())) {
124             g.newGeneration();
125             gbest = g.getBest();
126             best = gbest.evaluation();
127             if (best != newbest) {
128                 newbest = best;
129                 j = 0;
130             } else {
131                 j++;
132             }
133             //System.out.println(best);
134             i++;
135         }
136         System.out.println("Last best: " + best);
137         System.out.println("History best: " +
g.getHistoryBest().evaluation());
138         System.out.println("Generation Number: " + i);
139         System.out.println("Count of Equals: " + j);
140         System.out.println("Time usage: " + endTime() + " ms");
141 
142         return best > g.getHistoryBest().evaluation() ? best :
g.getHistoryBest().evaluation();
143     }
144 
145     /**
146      * 10 次 GA
147      * @throws java.io.FileNotFoundException
148      * @throws java.io.IOException
149      * @throws org.xml.sax.SAXException
150      * @throws javax.xml.parsers.ParserConfigurationException
151      */
152     public static void variance() throws FileNotFoundException, IOException,
153             SAXException, ParserConfigurationException {
154         double ga[] = new double[10];
155         double total = 0;
156         for (int i = 0; i < 10; i++) {
157             ga[i] = runGA();
158             total += ga[i];
159         }
160         double average = total / 10;
161         double result = 0;
162         for (int i = 0; i < 10; i++) {
163             result += Math.pow(Math.abs(ga[i] - average), 2);
164             System.out.println(i + ": " + ga[i]);
165         }
166         System.out.println("Average:" + average);
167         System.out.println("Variance:" + result / 9);
168     }
169 
170     /**
171      * @param args the command line arguments
```

```
172      */
173     public static void main(String[] args) {
174         try {
175             System.out.println("Select something to do:");
176             System.out.println("\t1. Making config.xml");
177             System.out.println("\t2. Exhausting Count");
178             System.out.println("\t3. Computing GA");
179             System.out.println("\t4. Variance");
180             System.out.print("[1-4]:");
181             BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
182             switch (Integer.valueOf(br.readLine())) {
183                 case 1:
184                     makeConfig();
185                     break;
186                 case 2:
187                     startTime();
188                     System.out.println(exhaustingCount());
189                     System.out.println("Time usage: " + endTime() + " ms");
190                     break;
191                 case 3:
192                     runGA();
193                     break;
194                 case 4:
195                     variance();
196                     break;
197                 case 99:
198                     Config.init();
199                     break;
200                 default:
201                     System.out.println("Only 1 to 4 accepted!");
202                     break;
203             }
204         } catch (Exception e) {
205             e.printStackTrace();
206         }
207     }
208 }
```