软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):785−801 [doi: 10.13328/j.cnki.jos.004966]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于模式组合的粒子群优化测试用例生成方法*

姜淑娟[1,2], 王令赛[1], 薛 猛[1], 张艳梅[1,3], 于 巧[1], 姚慧冉[1]

[1](中国矿业大学 计算机科学与技术学院,江苏 徐州 221116)

[2](广西可信软件重点实验室(桂林电子科技大学),广西 桂林 541004)

[3](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 姜淑娟, E-mail: shjjiang@cumt.edu.cn

摘 要: 适应度函数的设计在基于搜索的测试用例生成技术中占据重要的位置,然而在某些特殊的程序中,如存在嵌套、非结构性跳转或因 return,break 等语句跳出循环的程序,已有的适应度函数无法评价到所有的分支.目前的方法是修改程序的源代码,以使每个分支得到评价.但修改源代码不但可能影响程序的原有结构、引入错误,而且很难实现自动化.针对这一问题,提出一种基于模式组合的粒子群优化测试用例生成方法.首先,将分支条件定义为"模式",即,一类具有相同特征且能提高适应度值的个体集合,并改变其分支函数的插桩方式,可解决分支条件不完全评价的问题;然后,设计一种新的交叉算子,寻找到所有使模式的分支函数值最小的个体,将这些个体中含有模式的部分通过交叉算子组合到一个个体上,既可防止模式在进化过程中被破坏,又可因多种模式的组合而提高个体的适应度值;最后,使用局部搜索策略对种群中的最优个体进行搜索,提高粒子群优化算法的局部搜索精度,进一步提高测试用例生成效率.为了评价该方法的有效性,基于一组基准程序和开源程序进行实验.实验结果表明:对于含有模式的程序,该测试用例生成方法与已有方法相比,在覆盖率和平均进化代数上均有明显优势.

关键词: 测试用例生成;粒子群优化算法;交叉算子;局部搜索策略

中图法分类号: TP311

中文引用格式: 姜淑娟,王令赛,薛猛,张艳梅,于巧,姚慧冉.基于模式组合的粒子群优化测试用例生成方法.软件学报,2016,27(4):785−801. http://www.jos.org.cn/1000-9825/4966.htm

英文引用格式: Jiang SJ, Wang LS, Xue M, Zhang YM, Yu Q, Yao HR. Test case generation based on combination of schema using particle swarm optimization. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):785−801 (in Chinese). http://www.jos.org.cn/1000-9825/4966.htm

Test Case Generation Based on Combination of Schema Using Particle Swarm Optimization

JIANG Shu-Juan[1,2], WANG Ling-Sai[1], XUE Meng[1], ZHANG Yan-Mei[1,3], YU Qiao[1], YAO Hui-Ran[1]

[1](School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

[2](Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology), Guilin 541004, China)

[3](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: The design of fitness function plays one of most important roles in search based test data generation. While in some special program structures, such as nested structure, unstructured jump statements, and return/break statements, the existing fitness functions can't evaluate all the branches. The currently used approaches are to change the source program so that the branches can be evaluated

* 基金项目: 国家自然科学基金(61502497); 广西可信软件重点实验室研究课题(kx201530); 南京大学计算机软件新技术国家重点实验室基金(KFKT2014B19)

Foundation item: National Natural Science Foundation of China (61502497); Guangxi Key Laboratory of Trusted Software (kx201530); State Key Laboratory for Novel Software Technology at Nanjing University (KFKT2014B19)

收稿时间: 2015-07-26; 修改时间: 2015-10-20; 采用时间: 2015-11-26; jos 在线出版时间: 2016-01-13

CNKI 网络优先出版: 2016-01-14 13:16:25, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1316.011.html

completely. Changing source program might not only affect the program structure and result in errors, but also be hard to implement automatically. To solve the problem, this paper presents an approach of test case generation based on combination of schema using particle swarm optimization. First, a definition called “schema” is presented for all the branches which are able to improve the fitness value, and the branch function of the schema is obtained, solving the problem of partial evaluation. Then, a crossover is designed to place search on all the individuals which have the minimum value of branch function of the schema. The crossover views each schema as a whole and combines all the schemata into a single individual, as a result the crossover can not only prevent schemata destroyed in the process of evolution, but also improve the fitness value of individuals because of the combination. Furthermore, a local search strategy is used for the best particle in each generation in the process of test case generation. Experiments on some benchmark programs and open source programs are performed. The experimental results show that the proposed approach has obvious advantages in average coverage and generations comparing with other methods.

Key words: test case generation; particle swarm optimization algorithm; crossover operator; local search strategy

基于搜索的测试用例生成技术[1]将测试用例生成问题通过适应度函数转化成了函数优化问题,进而利用启发式算法寻找最优解.因此,适应度函数的设计极为重要,它必须对测试用例进行正确的评价,启发式算法才可以有效地进行寻优.

然而,McMinn[2]发现:在某些特殊的程序结构中,如存在嵌套、非结构性跳转或因 return,break 等语句跳出循环的程序,原有的适应度函数无法评价到所有的条件.比如:存在嵌套的程序,如果外层分支未覆盖,适应度函数则无法评价内层分支;但是若内层分支并不受外层分支的影响,那么即使内层分支的条件已经满足,它依然无法提高适应度函数的值.如图 1(a)所示,该程序有 3 层嵌套分支,若最外层分支 $a==b$ 不能满足,那么即使最内层分支 $c\geqslant 100$ 的分支满足条件,该分支条件也无法被覆盖到.针对这种情况,McMinn 等人对源代码进行了修改,如图 1(b)所示:移除了嵌套结构,使每个分支均可被执行到,并且增加了一个中间变量来获取各个分支的信息,使每个分支都能得到完全评价.修改被测程序源码的方法虽在图 1 的示例程序中有效果,但仍具有一定的局限性.首先,针对大型程序很难识别出所有含有此类特殊结构的代码;其次,即使识别出来,也需要人工判断如何修改才能不影响到源代码的原有逻辑[2,3].因此,修改源码的方式自动化程度不高.

```
void nest (int a, int b, int c)
{
    if (a==b)
    {if (b≤5)
        {if (c≥100)
            {//target path}
        }
    }
}
```

(a) 原始程序

```
void nest (int a, int b, int c)
{
    int count=0;
if (a==b) count++;
if (b≤5) count++;
if (c≥100) count++;
    if (count==3)
{//target path}
}
```

(b) 修改后的程序

Fig.1 Sample program nest

图 1 示例程序 nest

本文通过改变适应度函数的插装方式,在不改变源代码的基础上,即可预先获取模式对应分支的适应度函数信息,进而可以评价到还未执行的内层分支,或 return,break 语句跳出时仍未执行到的分支.

模式的概念由 Holland[4]在二进制编码的遗传算法中提出,即,一组满足特定条件的染色体的抽象表示.对于一个 4 位二进制编码的染色体来说,1**0 即表示满足第 1 位是 1 且第 4 位是 0 的所有染色体的集合,他认为,满足某种模式的染色体有助于提高适应度值.而 McMinn[2]针对数值型的输入变量提出了模式的定义,即,满足分支条件且可以提高适应度值的一组染色体的集合,比如满足 $c\geqslant 100$ 的所有测试用例.他还表明:遗传算法由于存在交叉算子,因而这些模式可能组合到一起,有助于提高适应度函数的值.然而,由于遗传算法存在选择算子,若个体本身适应度值低,即使它携带了有利于提高适应度值的模式,也可能被淘汰而无法保存下来这些有潜力的模式.粒子群优化算法由于其个体在进化过程中会不断地向最优解学习,若个体携带了有利于提高适应度值的模式,则该模式可保存下来.因此,若在粒子群算法中使用交叉算子,则更有可能将这些模式组合到一起,进而提

升个体的适应度值.

本文设计了一种新的交叉算子,将模式作为一个整体进行交叉,并将所有的模式均组合到一个个体上,既可防止模式在进化过程中被破坏,又可因多种模式的组合而提高个体的适应度值,使种群中的其他个体均向该个体进行学习,加快种群收敛速度.

本文的贡献如下:

(1) 针对存在嵌套、非结构性跳转或因 return,break 等语句而跳出循环的程序,导致部分分支无法被评价的情况,本文改变原有的插桩方式,预先获取到模式所对应的分支函数,在不改变源代码的基础上,可以对原适应度函数未评价到的分支进行评价;

(2) 针对有潜力提高适应度值的模式而言,本文设计了一种交叉算子,既可防止模式在进化过程中被破坏,又可因多种模式的组合而提高个体的适应度值,使种群中的其他个体均向该个体进行学习,加快种群收敛速度;

(3) 对每代的最优粒子均使用局部搜索策略,可协调算法的全局和局部搜索性能,有助于提高测试用例生成效率.

1 基本概念

1.1 基于搜索的测试用例生成

基于搜索的测试用例生成技术[1]由 Harman 提出,并在测试用例自动生成领域取得了较多的成果.主要利用适应度函数将测试用例生成问题转化成函数优化问题,进而利用一些启发式搜索算法寻找最优解,最终达到程序所要求的覆盖标准.因此,启发式搜索算法是基于搜索的测试用例生成技术的核心.

1.2 标准粒子群优化算法(PSO算法)

粒子群优化算法作为启发式搜索算法中的一种,具有简单、高效、收敛速度快的特点.它模拟的是一个经过简化的鸟类群体运动的模型.该算法将每只鸟比作一个粒子,根据鸟类种群中各个粒子之间的合作和相互学习,最终达到算法优化的目的.在粒子群优化算法中,需要被优化的问题的解抽象为粒子,粒子通过不断地向种群中的最优解及其自身经历过的最优解学习,不断被优化进而找到最优解.

在 PSO 算法中[5],设种群含有的粒子数目为 N,搜索空间的维度为 D,第 i 个粒子的位置 $x_i=(x_{i1},x_{i2},\ldots,x_{iD})$,速度 $v_i=(v_{i1},v_{i2},\ldots,v_{iD})$,粒子历史上搜到的最优位置为 $p_i=(p_{i1},p_{i2},\ldots,p_{iD})$,即局部最优解.整个种群历史上搜索到的最优位置为 $g=(g_1,g_2,\ldots,g_D)$,即全局最优解.当种群进化到第 t 代时,粒子的速度和位置更新公式如下:

$$v_i^{t+1}=wv_i^t+c_1r_1(p_i^t-x_i^t)+c_2r_2(g^t-x_i^t) \tag{1}$$

$$x_i^{t+1}=v_i^{t+1}+x_i^t \tag{2}$$

其中,学习因子 c_1 和 c_2 均为常数,r_1 和 r_2 均是 0-1 之间的随机数,惯性权重 w 从 0.9 到 0.4 线性递减.

1.3 模　式

程序中的目标路径可以表示为多个条件组合的形式[2],如图 1 中要覆盖目标分支,即满足如下条件:

$$a==b\wedge b\leqslant 5\wedge c\geqslant 100.$$

定义 1(模式[2]). 若某条件存在于目标路径条件组合中,且有可能提高适应度值,则满足该条件的所有个体的集合称为模式.例如在图 1 中,$S_1=\{(a,b,c)|a==b\}$,$S_2=\{(a,b,c)|b\leqslant 5\}$,$S_3=\{(a,b,c)|c\geqslant 100\}$均可称为模式.

模式可理解为个体中的一个基因片段,只要满足条件,即可说明该个体存在此种模式.如个体{1,1,4}存在模式 S_1,同时其也存在模式 S_2.

模式也可同时存在于多个个体之中,如个体{1,1,4}和个体{2,2,4}均含有模式 S_1.

定义 2. $var(S)$表示模式 S 所包含的变量集合.例如在图 1 中,如 $var(S_1)=\{a,b\}$,即表示该模式所包含的变量为第 1 维和第 2 维的变量.

本文将所用到的模式定义为独立模式和互斥模式两种.

定义 3(独立模式[2]). 若在一个程序中存在两个不同的模式 m_1 和 m_2,且 $var(m_1)\wedge var(m_2)=\varnothing$,则称 m_1 与 m_2 互相独立.

例如在图 1 中,$S_1=\{(a,b,c)|a==b\}$和 $S_3=\{(a,b,c)|c\geqslant 100\}$互相独立,$S_2=\{(a,b,c)|b\leqslant 5\}$也和 $S_3=\{(a,b,c)|c\geqslant 100\}$互相独立.独立的模式结合有利于适应度值的增加,如满足 S_1 的测试用例 $t_1=(5,5,5)$与满足模式 S_3 的测试用例 $t_2=(5,10,100)$交叉,则有可能产生测试用例 $t_3=(5,5,100)$,适应度值得到了提高.

定义 4(互斥模式[2]). 若在一个程序中存在两个不同的模式 m_1 和 m_2,且 $var(m_1)\wedge var(m_2)\neq\varnothing$,则称 m_1 与 m_2 互斥.

例如在图 1 中,$S_1=\{(a,b,c)|a==b\}$和 $S_2=\{(a,b,c)|b\leqslant 5\}$互斥.比如一个满足模式 S_1 的测试用例 $t_1=\{10,10,10\}$与满足 S_2 的测试用例 $t_2=\{10,5,10\}$,每个模式都可能使适应度函数有所提高,但是 t_1 和 t_2 不能交叉,这是由于它们存在一个公共的变量 b,而 b 的取值在不同的测试用例中是不同的,导致存在互斥现象.

2 基于模式组合的粒子群优化测试用例生成方法

2.1 方法的总体框架

基于模式组合的粒子群优化测试用例生成方法主要针对存在独立模式的程序,由于原始的适应度函数无法评价到该类程序中的部分分支,从而对其插桩方式进行改变,预先获取模式的适应度值,再通过交叉算法将这些模式组合到一起,可以提高个体的适应度值,有助于加速种群的进化.另外,为了提高算法的局部搜索精度,该方法使用了局部搜索策略——即我们在前期工作[6]中所描述的 AVM(alternating variable method)方法.

测试用例生成的总体框架如图 2 所示,包含测试环境构造模块、测试运行模块和算法模块这 3 个模块.

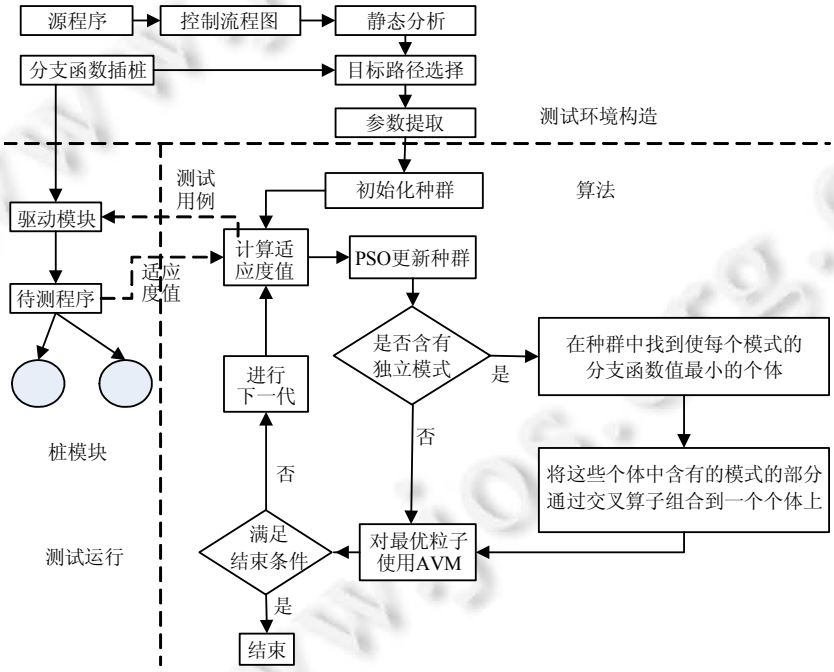


Fig.2 Overall framework of our approach

图 2 本文方法的总体框架

(1) 测试环境构造模块

该模块完成如下工作:对源程序进行静态分析、选择目标路径并对其进行插桩、提取有用的参数作为算法

模块的输入.

(2) 测试运行模块

该模块利用算法模块传递进来的测试用例驱动待测程序运行,进而计算出适应度值返回给算法模块.本文对原始的适应度函数的插桩方式进行了改进,针对含有独立模式的程序,在最外层分支对其进行插桩,可以预先获取到每个模式的适应度值,从而有利于交叉算子对最优模式进行组合.

(3) 算法模块

该模块分为模式组合和局部搜索策略两部分.

- 模式组合部分,针对存在独立模式的程序,通过获取每个模式的适应度值,找到所有含有最小适应度值的个体,并将其与含有最优模式最多的个体进行交叉,把最优的模式均集中到一个个体上,种群中其他个体即可向该最优个体学习,有利于加快 PSO 算法的进化过程;
- 局部搜索策略 AVM 算法,使用我们在前期工作[6]中所描述的算法,该算法对每一代的最优个体进行局部搜索,对其搜索范围限制为 L,有助于提高局部搜索精度.

2.2 适应度函数设计

本文使用分支距离法[7]来构造适应度函数,即:在各个分支节点都插入分支函数 f_i,表示当前测试用例与该分支为真的距离,具体的插桩方式如图 3 所示.若 f_i 小于等于 0,则表明覆盖了该分支,将 f_i 设为 0.若该目标路径含有 m 个分支节点,那么总的适应度函数 F 的计算方法如公式(3)所示.

$$F=\frac{\sum_{i=0}^{m-1}\frac{1}{f_i+1}}{m}\times 100\% \tag{3}$$

然而,若程序中存在嵌套、非结构性跳转或者过早退出循环的结构,将会导致某些分支无法得到评价,比如图 3 中的程序 nest,若不能满足最外层的 $a==b$ 的分支,那么程序不会进入下一层嵌套,而是直接结束.即使最内层分支 $c\geqslant 100$ 的分支满足条件,f_2 的值为 0,由于没有执行到该分支条件,而无法加入总的适应度函数 F 中,因此无法评价到最内层的分支.针对这种情况,我们对原始的分支函数的插桩方式进行改进,针对含有独立模式的程序,在最外层分支对其进行插桩,可以预先获取到每个模式的适应度值.原始插桩方式如图 3 所示,我们将独立模式的分支函数 f_1 和 f_2 放到最外层,则即使程序未能执行到该分支,我们依然可以获取到模式的适应度信息,改变后的插桩方式如图 4 所示.

```
void nest (int a, int b, int c)
{
    f0=Math.abs(a−b);
    if (a==b)
    {f1=b−5;
        if (b≤5)
        {f2=100−c;
            if (c≥100)
            {//target path}
        }
    }
}
```

Fig.3 Code of original instrument

图 3 原始插桩方式的代码

```
void nest (int a, int b, int c)
{
f0=Math.abs(a−b);
f1=b−5;
f2=100−c;
    if (a==b)
    {if (b≤5)
        {if (c≥100)
            {//target path}
        }
    }
}
```

Fig.4 Code of changed instrument

图 4 改变插桩方式后的代码

对于存在独立模式的程序,若模式 S 与若干个模式均相互独立,那么挑选其中含有变量个数最少的模式作为 S 的独立模式,以防止多个变量在测试用例生成过程中互相影响.如 $S_1=\{(a,b,c)|a==b\}$ 和 $S_3=\{(a,b,c)|c\geqslant 100\}$ 互相独立,$S_2=\{(a,b,c)|b\leqslant 5\}$ 也和 $S_3=\{(a,b,c)|c\geqslant 100\}$ 互相独立,但是因为 S_2 含有的变量个数少于 S_1,因此插桩时只需预先获取 S_3 和 S_2 所对应的分支函数即可.

2.3 交叉算子对最优模式的组合

定义 5(最优模式). 若在种群中存在个体 i,使得独立模式 S 对应的分支函数适应度值最小,则在个体为 i 的情况下,模式 S 称为最优模式,表示为 $S.bestNum=i$.

最优模式是指特定个体中的某个基因片段,该基因片段的适应度值在所有相同位置的个体中最小.如模式为 $S=\{(a,b,c)|a>b\}$,个体 1 为{2,1,1},个体 2 为{1,1,1},则个体 1 满足 2>1 的条件,适应度值为 0;而个体 2 不满足 1>1 的条件,其适应度值比个体 1 大,故个体 1 中的模式 S 称为最优模式.

程序中若含有多个模式,则每个模式在种群中均存在一个个体,使得该模式为最优模式,即,使该模式的分支函数适应度值最小.若将这些分散在不同个体中的最优模式加以组合,则有利于提高个体的质量,加快收敛速度.因此,本文使用交叉算子对最优模式进行组合.

定义 6(交叉算子). 当模式组合算法寻找到两个均含最优模式的个体之后,交叉算子将两个个体中的最优模式所包含的变量值进行互换,并将其相对应维度的速度值也进行互换,进而将模式作为一个整体集中到了一个个体之中.

2.3.1 模式组合算法描述

假设种群中含有 N 个粒子,程序中含有 M 个相互独立的模式,$i=1,2,\ldots,N$,$j=1,2,\ldots,M$,模式 S_j 所对应的变量为 $var(S_j)$,则如下矩阵表示种群中每个个体在每个模式下的适应度值:

$$\begin{pmatrix} f(1,S_1) & \ldots & f(1,S_M) \\ \vdots & f(i,S_j) & \vdots \\ f(N,S_1) & \ldots & f(N,S_M) \end{pmatrix},$$

其中,$f(i,S_j)$表示第 i 个个体在第 j 个模式 S_j 下的适应度值,每个行向量表示该个体中各个不同的模式所对应的适应度值,每个列向量表示每个模式在当前种群下每个个体的适应度值.

因此,每一列中均含有一个最优模式,即:表示每个模式在种群中一定存在一个个体,使其适应度值最小.我们将每个最优模式所对应的个体的编号 i 记录下来,即,优化函数如公式(4)所示.

$$\min(f(i,S_j)),j\in[1,M] \tag{4}$$

上述优化函数对于含有 M 个模式的程序中的每个模式 S_j,均需在个体编号 $i\in[1,N]$之间找到一个个体编号为 $bestNum$ 的个体,使得该模式的适应度值最小.那么,所有最优模式所对应的个体编号即为 $S_1.bestNum,S_2.bestNum,\ldots,S_M.bestNum$.若其中某个个体含有多个最优模式,即,存在 $S_j.bestNum=S_k.bestNum=S_m.bestNum=\ldots(j\neq k\neq m)$,那么即可找到含有最优模式最多的个体编号 $most\_S$.若最优模式均匀地分散在部分个体之中,无法找到含有最优模式最多的个体,则任选一个含有最优模式的个体作为 $most\_S$,将其他个体与之组合.模式组合的方式如下:

若程序中存在 M 个模式,个体编号为 $most\_S$ 的个体含有最多的最优模式,且个数为 $m(m\leqslant M)$,则对于其他 $M-m$ 个最优模式 S_j 来说,其个体所包含的模式 S_j 的变量为 $var(S_j)$,将该个体的这些变量值与 $most\_S$ 个体中所对应的模式 S_j 下的变量值进行交叉.通过交叉运算,个体 $most\_S$ 中不含最优模式的部分变量 $var(S_j)$,获得了来自于最优模式 S_j 最优个体的变量值.即:将所有的最优模式通过交叉运算,均集中到了个体 $most\_S$ 中.

原始的交叉算子可以实现个体间信息的交换,有可能产生更加优秀的个体,而我们则利用交叉算子对最优模式进行了组合,将最优模式通过交叉算子均集中在同一个个体 $most\_S$ 之中.然后,其他个体向该个体进行学习,加快了种群进化的速度.具体的算法如算法 1 所示(第 2.3.2 节).

2.3.2 模式组合算法

该程序中含有 M 个独立的模式,种群中有 N 个个体,新建一个 M 维数组 $best\_Individuals[M]$,保存含有最优模式的个体的编号;新建一个 N 维数组 $num\_BestSchema[N]$,保存每个个体所含有的最优模式的数目;将含有最优模式最多的个体编号表示为 $best\_individual$,则具体算法过程如算法 1 所示.

算法 1.

Input: All the individuals' variables, all the schema in each individual;

Output: The best individual after crossover.

```
1  begin
2    for each schema j in M do
3      for each individual i in N do
4        if (f(i,S_j) is the minimum) then
5          best_Individuals[j]=i
6          num_BestSchema[i]++
7        end if
8      end for individual
9    end for schema
10   for each i of num_BestSchema[i] in N do
11     if (num_BestSchema[i] is the maximum) then
12       best_individual=i
13     end if
14   end for num_BestSchema[i]
15   for each i of best_Individuals[i] in M do
16     if (best_Individuals[i]!=best_individual) then
17       swap(var(S_best_Individuals[i]),var(S_best_individual))
18     end if
19   end for best_Individuals[i]
20 end
```

算法分为 3 个阶段:

- 第 2 行~第 9 行:对每个模式 j 进行搜索,并在 N 个个体中找到个体 i 的适应度值 $f(i,S_j)$均比其他个体的适应度值要小,将个体 i 的标号保存在最优个体数组 *best_Individuals*[j]中,即,个体 i 包含了第 j 维最优模式.对 *num_BestSchema*[i]加 1,即,第 i 个个体含有最优模式的个数;
- 第 10 行~第 14 行:对 *num_BestSchema*[N]数组进行遍历,若 *num_BestSchema*[i]是整个数组中最大的元素,则表明第 i 个个体含有的最优模式的个数最多,将其记为 *best_individual*,则剩下的其他最优模式应通过交叉算子集中到第 i 个个体之中;
- 第 15 行~第 19 行:对最优个体数组 *best_Individuals*[M]进行扫描,若当前模式的最优个体编号不等于 *best_individual*,则将该模式所包含的所有变量与 *best_individual* 对应位置的变量进行交换,那么通过交叉算子,最优模式均集中到了最优秀的个体 *best_individual* 中.

2.3.3 模式组合算法示例

若一个程序中存在 3 个两两相互独立的模式:S_1,S_2 和 S_3,则其对应的变量:$var(S_1)=\{a,c\}$,$var(S_2)=\{b,d,e\}$,$var(S_3)=\{f\}$,相对应的每个模式的适应度函数分别为 f_1,f_2 和 f_3,种群中有 3 个个体 *individual*_1,*individual*_2 和 *individual*_3.

如图 5 所示,每一行表示每个个体其 3 个模式所对应分支函数的适应度值.如在 *individual*_1 中,3 个模式的分支函数值分别为 3,5,1.图 5 中的每一列分别表示该模式下所有个体的分支函数值.在所有的个体中,*individual*_1 的分支函数 f_1 和 f_3 均最小,即,*individual*_1 中含有 2 个最优模式 S_1 和 S_3.同理,*individual*_2 中的 f_2 也是所有个体中最小的,即,*individual*_2 中含有 1 个最优模式 S_2,而 *individual*_3 中由于没有最小的适应度函数值,所以不含最优模式.通过对最优模式进行交叉组合,图 6 所示为经过交叉之后的个体及其模式的适应度值,可以看出,最优模式均集中在 *individual*_1 中.

个体	f_1	f_2	f_3
*individual*_1	**3**	5	**1**
*individual*_2	4	**2**	2
*individual*_3	5	7	5

Fig.5 Fitness value before crossing

图 5 交叉之前适应度值

个体	f_1	f_2	f_3
*individual*_1	**3**	**2**	**1**
*individual*_2	4	5	2
*individual*_3	5	7	5

Fig.6 Fitness value after crossing

图 6 交叉之后适应度值

从图 5 和图 6 中可以看出:*individual*_2 中的最优模式 S_2 与 *individual*_1 中的最优模式通过交叉的方式进行组合,且适应度值比交叉之前有所减小.因 $var(S_2)=\{b,d,e\}$,即,S_2 所含有的变量为 b,d,e,因此,以这 3 个变量作为一个整体与 *individual*_1 进行交叉.图 7 和图 8 为 *individual*_1,*individual*_2 和 *individual*_3 具体到变量进行交叉的前后对比图,可以看出:原来在 *individual*_2 中的变量 b_2,d_2,e_2 通过交叉,与 *individual*_1 中的 b_1,d_1,e_1 进行了交换.

*individual*_1	a_1	b_1	c_1	d_1	e_1	f_1
*individual*_2	a_2	$\boldsymbol{b_2}$	c_2	$\boldsymbol{d_2}$	$\boldsymbol{e_2}$	f_2
*individual*_3	a_3	b_3	c_3	d_3	e_3	f_3

Fig.7 Values before crossing

图 7 交叉之前的变量

*individual*_1	a_1	$\boldsymbol{b_2}$	c_1	$\boldsymbol{d_2}$	$\boldsymbol{e_2}$	f_1
*individual*_2	a_2	b_1	c_2	d_1	e_1	f_2
*individual*_3	a_3	b_3	c_3	d_3	e_3	f_3

Fig.8 Values after crossing

图 8 交叉之后的变量

2.3.4 模式组合算法优势

本文设计的交叉算子相对于普通的交叉算子有如下优点:

(1) 普通交叉算子的交叉方式有单点交叉、双点交叉、均匀交叉等方式,但是由于交叉点是随机选取的,故在交叉的过程中容易破坏最优模式;而本文中的交叉算子把模式当作一个整体来进行交叉,可以使最优模式被完整地交换;

(2) 普通的交叉算子在进行完交叉操作之后,不一定能保证新生成的个体一定比原来的个体优秀;而本文的交叉算子通过交叉将所有最优模式集中到一个个体中,使种群中产生一个最优秀的个体.因为本文使用的算法为粒子群优化算法,则其他个体均可向这个最优秀的个体进行学习,加速了种群进化的速度;

(3) 普通的交叉算子按照交叉概率对所有的个体进行交叉,而本文设计的交叉算子只将含有最优模式的个体进行交叉,不含最优模式的个体(如 *individual*_3),并不参与交叉过程,因此在一定程度上降低了时间消耗.

2.4 局部搜索策略

局部搜索策略为交替变量法 AVM,它由 Korel[7]最早提出,是一种以爬山算法为基础的算法,可用于测试用例生成中,以提高算法的局部搜索能力.

AVM 的主要思想是:轮流调整每一维度的输入变量的值,直到适应度值不能再提高;然后,修改下一维度变量的值,直到调整完所有的变量值都无法提高适应度值为止.调整的方法采用爬山算法,即:仅在该变量的邻域范围内寻找最优解,若发现的新解比当前解优秀,则用新解替换当前解,并继续搜索新解的邻域,迭代地进行搜索,以发现最优解.

我们的前期工作[6]主要对原始的 AVM 方法进行了如下改进:

(1) 对每个变量以 2^n 为半径进行搜索,n 为局部搜索的次数.该算法实际上为二分搜索,为了防止因局部搜索次数过多而造成的效率降低,我们将搜索最大范围限制为 L,以协调算法的全局和局部搜索能力;

(2) 由于 AVM 搜索的效率与搜索的起始位置有直接关系,若起始位置较差,则需要更多的搜索次数,因此只对每代最优粒子使用 AVM,从而在一定程度上提高了搜索效率.

3 实 验

实验部分以开源程序为实验对象,提出两个研究问题来探索本文设计的交叉算子是否有效以及本文方法是否优于其他方法,收集实验结果并加以分析,以验证本文方法的有效性.具体问题如下.

问题 1. 本文设计的用于进行模式组合的交叉算子相对于其他交叉方式,如单点交叉、双点交叉、均匀交叉是否有优势?

本文设计了一种新的交叉算子,使所有最优模式通过交叉算子集中到一个个体上,加速了进化过程.为了探索本文提出的交叉算子是否有效以及效果如何,我们选择了单点交叉、双点交叉和均匀交叉这 3 种交叉算子作为对比,设计了实验,见第 3.4.1 节.

本组实验为了验证交叉算子的有效性,仅使用原始 PSO 方法加不同的交叉算子进行测试用例生成,交叉操作在 PSO 每代更新结束之后进行;且为了确保实验的公平性,在本组实验中不使用本文中的局部搜索策略 AVM,以用来排除其他因素的干扰,准确地观察交叉算子的设计对算法的影响.

问题 2. 本文方法相比于其他文献中的方法是否有优势?

本文方法将最优模式通过交叉算子进行了组合,且使用 AVM 算法提高了局部搜索精度.为了验证该方法相对于其他文献是否有优势,我们设计了一组实验,将本文方法与原始 PSO 方法、APSO 方法和我们前期工作中提出的 OL-PSO 方法进行了对比,主要是比较覆盖率和平均进化代数两个指标,实验的相关细节见第 3.4.2 节.

3.1 实验对象

实验选取了 8 个程序进行测试,其中 5 个程序为含有独立模式的程序,3 个为不含独立模式的程序.通过对这两类程序的实验进行比较,来验证本文设计的交叉算子是否有效.程序的详细信息见表 1.

Table 1 Program under test

表 1 测试程序

程序编号	程序名称	变量个数	搜索范围	独立模式数目	程序来源
1	Equals	20	$[-10^2,10^2]$	10	Java collections
2	Transform	10	[0,255]	10	文献[8]
3	luhnCheck	16	$[-10^3,10^3]$	16	Apache commons valid
4	nest	3	$[-10^4,10^4]$	2	文献[2]
5	NextDate	3	$[-10^4,10^4]$	0	文献[9]
6	Triangle	3	$[-10^4,10^4]$	0	文献[8]
7	density	4	$[-10^4,10^4]$	0	Apache commons math
8	iszeros	20	$[-10^3,10^3]$	20	文献[2]

程序 1 的目标路径是两个数组是否相等,本实验将两个数组的长度均设为 10,它包含了 10 个独立模式.程序 2 的目标路径为:输入的 10 个字符均为 16 进制的字符,且其转化为 10 进制之后的总和在 50~100 之间,它包含了 10 个独立模式,另外还有一个模式与这 10 个模式均互斥.程序 3 的目的是检测银行卡或者信用卡号码是否有效,本实验将号码长度固定为 16,若每个号码均在 0~9 之间,且满足 luhnCheck 函数的条件,即为目标路径,它包含了 16 个独立模式,另外还有一个模式与这 16 个模式均互斥.程序 4 为图 1 中的程序 nest,包含 2 个独立模式,1 个互斥模式.程序 5 的目标路径是年、月、日满足 1912 年~2050 年之间闰年 2 月 29 日,不包含独立模式.程序 6 的目标路径为等边三角形,不包含独立模式.程序 7 是三角形分布的概率密度函数,目标路径是某一随机变量 x 等于中位数 c,不包含独立模式.程序 8 的目标路径数组中所有的元素均为 0,它包含了 20 个独立模式.

3.2 评价标准

为了评价本文方法的测试用例生成效率,本文主要使用两个评价标准.

(1) 覆盖率:对每个程序都运行 100 次,若在最大进化代数 T=1000 之内,该程序的目标路径被覆盖,则记录下来.覆盖率为这 100 次运行中被覆盖次数的平均值,然后将其转化为百分数的形式来显示;

(2) 平均进化代数:每个程序均运行 100 次,若该程序在最大进化代数 T=1000 之内覆盖到了目标路径,则将该程序此时的进化代数记录下来;如果没有找到,则记录为 T.进化代数即为这些记录的平均值.

另外,为了更好地对比本文方法与其他 3 种方法的区别,我们对实验结果进行了统计检验和效应量分析.

3.3 实验设计

本文方法在 Eclipse 平台上,使用 Java 语言实现.具体实现流程如下.

Step 1. 构建初始种群;

Step 2. 对每一代的个体均利用适应度函数进行评价,若适应度值为 100,则表明获得最优解;否则,继续执行;

Step 3. 根据粒子群更新公式对个体的位置和速度进行更新;

Step 4. 判断该程序是否含有独立模式:若有,则利用第 2.3 节中的模式组合方法生成一个新的最优解;否则,进入 Step 5;

Step 5. 使用 AVM 方法对最优解进行局部搜索,转到 Step 2 中判断是否获得最优解.

针对问题 1,为了验证本文交叉算子的有效性,将本文交叉算子与其他交叉方式,如单点交叉、双点交叉、均匀交叉作对比.为了保证实验的公平性,本实验步骤仅进行 Step 1~Step 4,可保证 4 种交叉算子均只应用在粒子群优化算法上,而不是因为其他方面的优化(如局部搜索)才导致本交叉算子具有优势.

针对问题 2,为了验证本文方法的有效性,将其与其他 3 种算法进行测试用例生成的对比,4 种方法的实验参数设置见表 2.PSO 代表原始粒子群优化算法;APSO 表示文献[10]的方法;OL-PSO 表示应用了正交搜索和局部搜索策略相结合的方法,即,我们之前的工作[6];CL-PSO 表示本文方法,即,使用模式组合和局部搜索相结合的方法,实验步骤如 Step 1~Step 5 所示.

Table 2 Values of all the parameters in experiments

表 2 实验参数设置

参数 \ 算法	PSO	APSO	OL-PSO	CL-PSO
NUM	30	30	30	30
T	1 000	1 000	1 000	1 000
c_1,c_2	2	2	2	2
w	(0.4~0.9]	(0.4~0.9]	(0.4~0.9]	(0.4~0.9]
*v*_max	*x*_max/10	*x*_max/10	*x*_max/10	*x*_max/10
k	–	–	50	–
select_rate	–	–	0.2	–
L	–	–	8	8
tz	–	10	–	–

为了保证实验的公平性,4 种方法在参数设置上,比如种群数目、最大进化代数等参数值均设为相同,其他不同的参数均参考其他文献而设置.其实验参数设置如下:*NUM* 为种群数目,*T* 表示算法最大进化代数,*v*_max 为速度最大值,*x*_max 为搜索范围最大值,*k* 为两次奇异值分解之间相隔的进化代数,*select_rate* 为新生成的正交种群可以进入原始种群的比例,*L* 为局部搜索范围,*tz* 为局部收敛次数.

3.4 实验分析

3.4.1 多种交叉方式的比较

为了探索本文设计的交叉算子是否有效以及效果如何,本组实验选择了单点交叉、双点交叉和均匀交叉这 3 种交叉算子作为对比.表 3 分别为使用单点交叉、双点交叉、均匀交叉以及本文的交叉算子方式进行测试用例生成,比较 4 种交叉算子在测试用例生成中所完成的覆盖率和所需要的平均进化代数这两个指标.

Table 3 Average coverage and generations of different crossovers

表 3 交叉方式的平均覆盖率和平均进化代数

测试程序	单点交叉		双点交叉		均匀交叉		本文交叉算子	
	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数
1	100	287.95	100	235.53	100	348.09	100	15.31
2	74	721.9	86	582.3	81	714.93	100	3.8
3	44	857.06	45	804.44	90	742.13	100	43.44
4	100	46.82	100	47.75	100	50.52	100	38.06
8	37	907.86	53	827.98	68	856.14	100	42.11

由于本文的交叉算子是为了集中最优模式,因此对不含独立模式的程序 5(NextDate)、程序 6(Triangle)和程序 7(density),并不适合使用交叉算子,因此,这 3 个程序不在本组实验比较的范围内.

由表 3 可以看出:本文的交叉算子在含有独立模式的 5 个程序中,对于程序 2、程序 3 和程序 8 来说,本文交叉算子相对于单点交叉、双点交叉和均匀交叉方式在覆盖率上有明显提高;而在平均进化代数方面,除了程序 4 为 nest 程序,较为简单,导致 4 种交叉方式区别不大之外,在其他 4 个程序中,本文交叉算子的平均进化代数远远低于其他 3 种方法,说明本文设计的交叉算子在测试用例生成效率上具有明显优势.

而对于单点交叉、双点交叉和均匀交叉,它们在覆盖率上相差并不是很大.对于程序 1、程序 4 来说,这 3 种交叉算子的覆盖率均达到了 100%.而此时均匀交叉的平均进化代数比单点交叉和双点交叉方法的平均进化代数略高,推测是由于均匀交叉算子对模式的破坏较大.例如,程序 1 中一共含有 10 个独立模式,每个独立模式包含两个变量,但是两个变量的位置并不相邻,因此,均匀交叉的方法有可能破坏本来已经进化得较为优秀的模式.单点交叉方法同理,而双点交叉是交换中间一整块的变量,有可能对模式的破坏较小.

对于程序 3 和程序 8 来说,相对于单点交叉和双点交叉,均匀交叉算子在覆盖率上有所提高,主要是由于程序 3 和程序 4 中每个变量就是一个独立模式,使用均匀交叉不仅不会破坏模式;反之,可能因为与其他个体的组合而提高了自身的适应度值,导致其覆盖率相对较高.

为了更好地对比本文交叉算子和其他 3 种交叉算子的区别,我们以每种交叉方式收集到的 100 次进化代数为样本,对实验结果进行了统计检验和效应量分析.表 4 给出了利用 Wilcoxon 秩和检验[11]和 Cohen d[12]的效应量对各种交叉方式进行分析的结果.因覆盖率为一个平均值,故无法对其进行统计检验和效应量分析.

Table 4 *p-value* and effect size of different crossovers

表 4 交叉方式的 *p-value* 值和效应量值

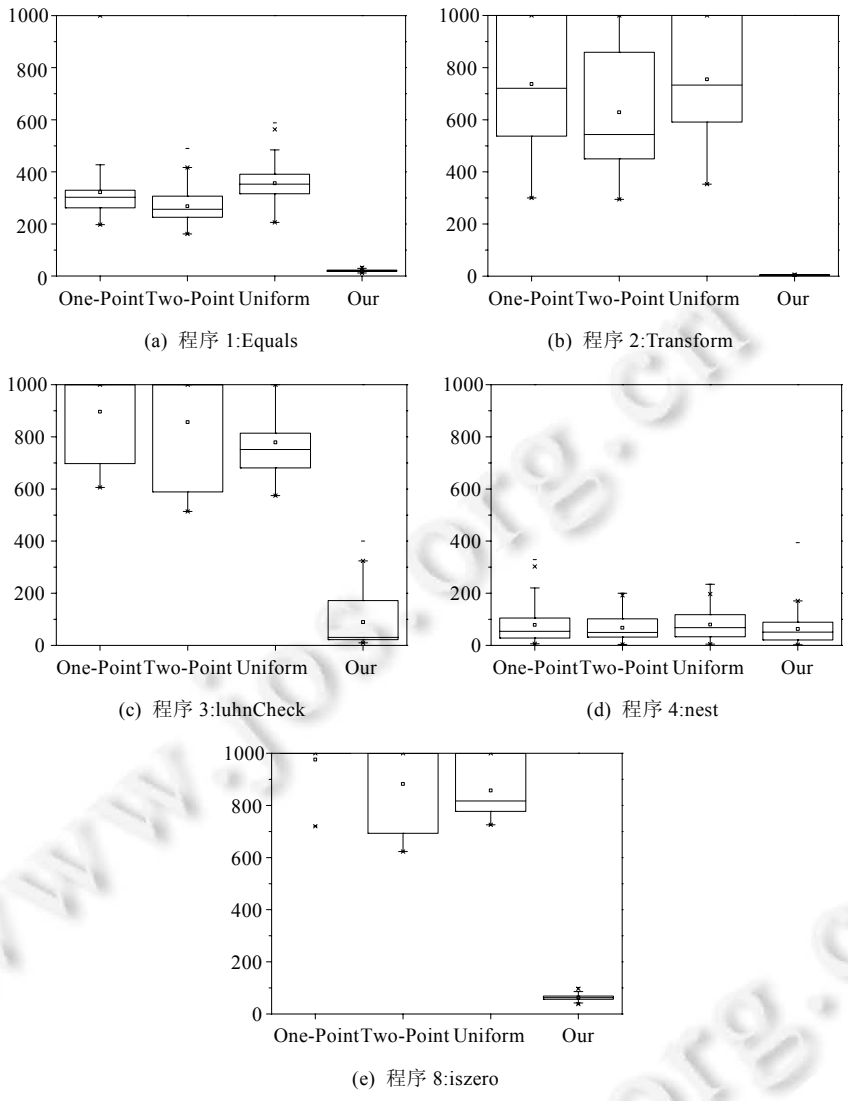
测试程序	单点交叉 vs 本文交叉		双点交叉 vs 本文交叉		均匀交叉 vs 本文交叉	
	p-value	Cohen *d*	*p-value*	Cohen *d*	*p-value*	Cohen *d*
1	<0.001	3.21 (L)	<0.001	6.14 (L)	<0.001	7.06 (L)
2	<0.001	4.79 (L)	<0.001	3.81 (L)	<0.001	5.24 (L)
3	<0.001	6.15 (L)	<0.001	4.79 (L)	<0.001	6.07 (L)
4	0.021	0.27 (S)	0.242	0.09	0.005	0.32 (S)
8	<0.001	17.22 (L)	<0.001	7.40 (L)	<0.001	11.22 (L)

在统计检验中,本文提出的空假设分别为单点交叉、双点交叉、均匀交叉的平均进化代数与本文交叉算子的平均进化代数差异不显著.若拒绝本文提出的空假设,则 *p-value* 需小于 0.05.由表 4 可知:基本上所有的测试程序,这 3 种假设的 *p-value* 值均远小于 0.05,故使用本文交叉算子所运行的平均进化代数与单点交叉、双点交叉和均匀交叉的平均进化代数差异显著,并且该结论具有统计学意义.

效应量是衡量处理效应大小的指标,与显著性检验不同,它不受样本容量影响.效应量反映两个样本间重叠的程度,Cohen 提出,d=0.8,d=0.5 和 d=0.2 分别对应大(L)、中(M)、小(S)这 3 种效应量.若效应量大于 0.8,即说明两个样本重叠的程度小,效应越明显.如表 4 所示,大部分程序中所有的效应量都大于 0.8,即效应较为明显.而对于程序 4 来说,因为程序本身较为简单,在表 3 中可以看到,单点交叉、双点交叉和均匀交叉这 3 种方式下覆盖率均为 100%,且它们的平均进化代数也与本文交叉算子的平均进化代数相近.所以样本重叠程度比较高,导致效应量值较小,效应不明显.

表 3 中的数据只反映出了 4 种交叉方式在平均覆盖率和平均进化代数上的差异,但并不直观.因此我们绘制了图 9,直观地显示出在 100 次实验中,在 4 种交叉方式下进行测试用例生成其平均进化代数的分布情况.图中横坐标代表 4 种不同的交叉方式,纵坐标表示平均进化代数.

图 9 所示为含有独立模式的 5 个程序,在不同的交叉算子下的平均进化代数分布情况图.由图 9 可知:除了程序 4(因其程序简单),4 种交叉算子的平均进化代数基本上无区分之外,在其他 4 个程序中,本文方法平均进化代数均远远低于单点交叉、双点交叉和均匀交叉的平均进化代数,说明本文设计的交叉算子效果显著.同时,本文方法的数据分布较为稳定,且数据较为集中;而其他 3 种交叉方式的数据分布比较分散,说明其稳定性也不如本文设计的交叉算子好.



(a) 程序 1:Equals (b) 程序 2:Transform

(c) 程序 3:luhnCheck (d) 程序 4:nest

(e) 程序 8:iszero

Fig.9 Comparision of generations between different crossovers

图 9 不同交叉方式平均进化代数对比图

3.4.2 本文方法与其他文献方法的比较

为了验证本文方法 CL-PSO 相对于其他方法是否有优势,将 CL-PSO 与原始 PSO 方法、APSO 和我们前期的 OL-PSO 方法进行了对比,表 5 为分别使用 4 种方法对每个测试程序执行 100 次后的覆盖率和平均进化代数.

由表 5 可以看出:OL-PSO 和本文方法 CL-PSO 明显优于 APSO 方法和 PSO 方法,将覆盖率提升至了 100%,尤其针对程序 2、程序 3 和程序 8 等较难覆盖的程序来说,其平均进化代数也有了较大程度的降低,极大地提高了测试用例生成效率.而针对原本就可达到 100%覆盖率的程序而言,OL-PSO 和 CL-PSO 方法的平均进化代数有一定程度的降低.

OL-PSO 是将正交搜索与局部搜索策略相结合的方法;而 CL-PSO 是利用交叉算子对最优模式进行组合,再使用局部搜索算法进行搜索.这两种方法在覆盖率上均为 100%,无差异.而在平均进化代数上,只有程序 2、程序 3 和程序 4 差异较大一些.这是由于这 3 个程序均含有独立模式,适合用交叉算子对其进行模式组合,因此适

合使用 CL-PSO 方法进行测试用例生成.而针对于不含独立模式的程序,比如程序 5、程序 6 和程序 7,二者差异不大,甚至 OL-PSO 的效果要稍微比 CL-PSO 方法好一些.这是由于,针对不含独立模式的程序,CL-PSO 无法发挥其交叉算子的作用,而 OL-PSO 依然可以对其进行正交搜索,使算法的全局搜索性能和局部搜索性能得以协调,因此,CL-PSO 对存在独立模式的程序而言更加有效.

Table 5 Average coverage and average generations of the four approaches

表 5 4 种方法的平均覆盖率和平均进化代数

测试程序	PSO		APSO		OL-PSO		CL-PSO	
	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数	覆盖率(%)	平均进化代数
1	81	320.9	100	185.87	100	3.07	100	4.01
2	65	578.24	62	479.36	100	120.76	100	6.47
3	39	783.07	63	728.71	100	283.7	100	25.9
4	100	403.56	100	177.33	100	67.82	100	7.49
5	100	267.1	100	156.18	100	134.5	100	138.14
6	100	377.04	100	159.05	100	63.49	100	70.9
7	100	63.07	100	32.23	100	8.42	100	9.1
8	19	921.47	54	935.5	100	22.7	100	23.61

为了更好地对比本文方法与其他 3 种方法的区别,我们以每种方法收集到的 100 次进化代数为样本,对实验结果进行了统计检验和效应量分析.表 6 为利用 Wilcoxon 秩和检验和 Cohen *d* 的效应量对各种方法进行分析的结果.

Table 6 *p-value* and effect size of the four approaches

表 6 4 种方法的 *p-value* 值和效应量值

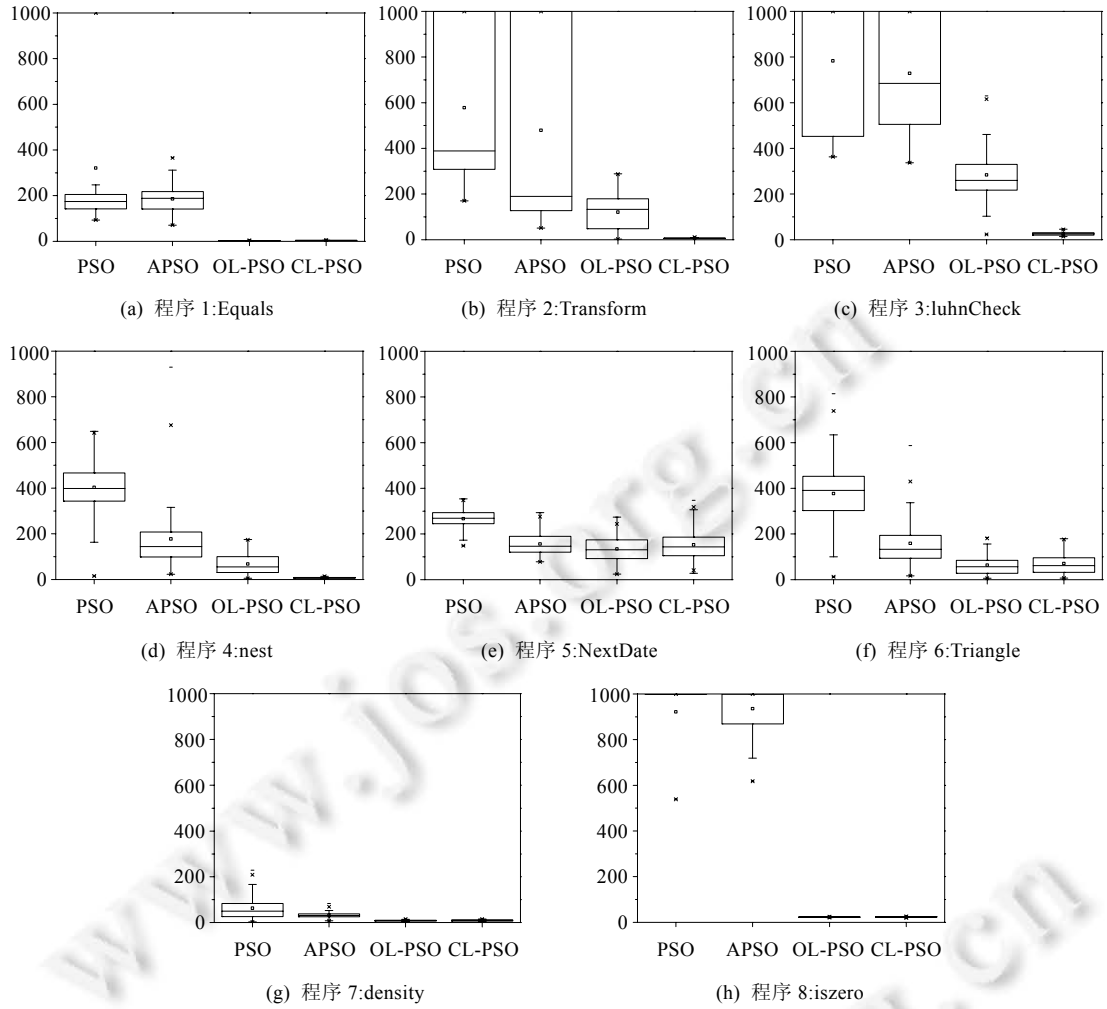
测试程序	PSO vs. CL-PSO		APSO vs. CL-PSO		OL-PSO vs. CL-PSO	
	p-value	Cohen *d*	*p-value*	Cohen *d*	*p-value*	Cohen *d*
1	<0.001	1.35 (L)	<0.001	4.06 (L)	1	−2.44 (L)
2	<0.001	2.52 (L)	<0.001	1.61 (L)	<0.001	2.03 (L)
3	<0.001	3.92 (L)	<0.001	4.05 (L)	<0.001	3.63 (L)
4	<0.001	5.09 (L)	<0.001	1.84 (L)	<0.001	1.87 (L)
5	<0.001	2.91 (L)	0.009	0.38 (S)	0.763	−0.07
6	<0.001	2.78 (L)	<0.001	1.16 (L)	0.827	−0.17
7	<0.001	1.64 (L)	<0.001	2.47 (L)	0.928	−0.23 (S)
8	<0.001	7.77 (L)	<0.001	13.33 (L)	1	−0.82 (L)

在表 6 中的统计检验部分,本文提出的空假设分别为 PSO,APSO,OL-PSO 的平均进化代数,与本文的 CL-PSO 方法差异不显著.而对于 PSO 和 APSO 方法来说,所有程序的 *p-value* 均小于 0.05,即可以拒绝原假设,说明本文方法 CL-PSO 的平均进化代数与 PSO 和 APSO 方法存在显著性差异.而对于 OL-PSO 方法来说,只有程序 2、程序 3 和程序 4 这 3 个程序的 *p-value* 小于 0.05,而其他 5 个程序都不能拒绝原假设,即,本文方法 CL-PSO 与 OL-PSO 差异不显著.

在效应量分析部分,d=0.8,d=0.5 和 d=0.2 分别对应大(L)、中(M)、小(S)这 3 种效应量.对于 PSO 和 APSO 来说,基本上大部分程序的效应量均大于 0.8,效应量程序较高.而对于 OL-PSO 来说,其中 5 个程序的效应量为负值,说明 OL-PSO 平均进化代数的平均值比本文方法 CL-PSO 小,并且在这 5 个程序中,有 3 个程序的效应量程度都是较小的,即样本重叠程度较高,说明本文方法与 OL-PSO 方法差异不大.

图 10 为 8 个程序分别在 PSO,APSO,OL-PSO 和 CL-PSO 这 4 种方法下的平均进化代数分布情况,其中,横坐标标识 4 种方法,纵坐标标识平均进化代数.

由图 10 可以看出,OL-PSO 和本文的 CL-PSO 方法在 8 个程序中平均进化代数都比 PSO 和 APSO 方法要低.同时,对于含有独立模式的程序 2、程序 3 和程序 4 来说,使用了交叉算子对最优模式进行组合的 CL-PSO 方法明显要比 OL-PSO 方法更有优势,说明 CL-PSO 针对于此类程序是有效的.



(a) 程序 1:Equals (b) 程序 2:Transform (c) 程序 3:luhnCheck

(d) 程序 4:nest (e) 程序 5:NextDate (f) 程序 6:Triangle

(g) 程序 7:density (h) 程序 8:iszero

Fig.10 Evolution algebra of different approaches

图 10 不同方法平均进化代数对比图

3.4.3 实验结果分析

通过前面两节的实验结果,我们对实验设计部分提出的两个问题有如下回答.

回答问题 1. 本文设计的用于进行模式组合的交叉算子相对于其他交叉方式,如单点交叉、双点交叉、均匀交叉是否有优势?

通过表 3 和图 9 可以看出:本文设计的交叉算子针对含有独立模式的程序效果均优于单点交叉、双点交叉和均匀交叉这 3 种交叉方式,且使用我们的方法也比其他 3 种交叉方式的结果要稳定一些,数据分布较为集中.

回答问题 2. 本文方法相比于其他文献的方法是否有优势?

从表 5 和图 10 可以看出:与原始 PSO 方法、APSO 方法相比,本文方法 CL-PSO 在覆盖率上有明显的优势;而对于 OL-PSO,二者覆盖率都达到了 100%,覆盖率上无差异.只有在含有独立模式的程序 2、程序 3 和程序 4 中可以看出:CL-PSO 比 OL-PSO 在平均进化代数上有明显的减小,在测试用例生成方面更具有优势.这是由于,CL-PSO 使用了交叉算子法对模式进行了组合.而对于其他程序而言,CL-PSO 和 OL-PSO 在平均进化代数上基本上无差异.

4 有效性分析

影响本文实验有效性的因素包括两个方面.

一方面是内部因素,即粒子群优化算法本身对实验结果的影响.因为粒子群优化算法是一种随机性算法,会使实验结果有一定的随机性.因此,为了降低算法随机性带来的影响,本文实验结果取 100 次重复实验的平均值;

另一方面是外部因素,如程序本身对实验结果的影响.目前,本文方法只适用于存在独立模式的程序,而独立模式的基础是假设所有分支都是独立的,若分支之间存在相互关系,则其无法满足独立模式的条件,即为互斥模式.因此,今后的工作将研究如何解决分支间相互影响的问题.

5 相关工作

基于搜索的测试用例生成是基于搜索的软件工程(search-based software engineering,简称 SBSE)[13]领域的一个重要分支,是软件测试用例生成领域的一个重要的研究方向.1992 年,Xanthakis 等人[14]首次将遗传算法应用于软件测试用例生成.Sthamer[15]在其博士论文中,使用遗传算法研究了分支测试、循环测试等问题.Wegener 等人[16]建立了基于遗传算法的测试用例生成环境,通过对一些工业程序的实验,其结果表明,基于遗传算法的测试用例生成较随机法具有更高的覆盖率.Fraser 和 Arcuri[17−19]介绍了一种针对 Java 程序的测试用例集自动生成工具——EvoSuite.该工具主要采用遗传算法产生测试用例集[18],文献[19]通过实验对随机选取的工业 Java 程序进行实验,结果表明,使用 EvoSuite 生成测试用例可以达到较高的分支覆盖率.在文献[20]中,Fraser 等人使用文化基因算法代替 EvoSuite 中的遗传算法,加入局部搜索算子,提高了分支覆盖率.相较于遗传算法在测试数据生成中的应用研究[21−24]较为完善有所不同,PSO 算法在测试数据生成中的研究才刚起步.Windisch 等人[5]将 PSO 算法应用到测试数据生成中,通过对一些程序进行实验研究,结果表明,PSO 算法在代码覆盖和执行效率上要优于遗传算法.毛澄映等人[25]以分支覆盖为准则,选取了 4 种典型 PSO 算法的变体与遗传算法和模拟退火算法进行测试数据生成效果的实证分析,实验结果表明,使用 PSO 算法在分支覆盖率等性能上要优于遗传算法和模拟退火算法.

针对程序因结构复杂而导致其分支条件无法得到评价这一问题,McMinn[2]对源代码进行了修改,通过中间变量得到内层分支的评价信息,可使所有的分支条件得到完全评价.而本文在不修改源代码的基础上,仅针对这些未能被评价到的条件的插桩方式进行了改变,将插桩语句放到最外层,可有效解决适应度函数不能完全评价所有分支的问题,且没有过多地引入时间消耗.

McMinn[2]对大量程序进行了实验,并证明:对于含有模式的程序,遗传算法由于其特有的交叉运算,可以对模式进行组合,生成更优的个体,因此优于其他算法.但由于遗传算法中的交叉算子在进行交叉时容易破坏程序中的模式,且由于遗传算法存在选择操作,含有最优模式的个体有可能因为其适应度值低而被淘汰.因此,本文将交叉算子引入到 PSO 中,并设计了一种新的交叉算子,将所有最优模式均集中到种群中最优秀的个体上,而其他个体均向该优秀个体进行学习,加快了进化过程.实验部分通过与其他的交叉方式进行对比,证明本文的交叉方式不但不会破坏个体中的最优模式,还可以因其他个体向最优个体的学习,从而较快地收敛到全局最优解.

Zhu 等人[10]提出了一种 APSO 算法进行测试用例生成,主要根据当前个体的适应度值对惯性权重进行调整,并且对个体进行早熟收敛判断,超过一定次数时,则重新初始化该粒子.史娇娇等人[26]提出了一种自适应 PSO 算法,并将其应用于测试用例生成中.我们的前期工作[6]提出了一种基于正交搜索和局部搜索相结合的粒子群优化测试用例生成方法,利用奇异值分解的方法预测种群进化方向,在其正交方向生成新种群,增强算法的全局搜索能力.此外,还使用局部搜索策略 AVM 来增强算法的局部搜索能力,使其全局和局部搜索能力相协调,更高效地生成测试用例.

上述文献中,应用 PSO 算法生成测试用例时,主要集中改进 PSO 算法,或者调节算法中的参数,或者增强算法的全局或局部搜索能力,然而并没有考虑程序的特殊结构对测试用例生成效率的影响.而本文主要针对含有独立模式的程序设计了一种新的交叉算子,并且通过实验与相关文献中的方法进行了比较.

6 结束语

本文首先针对存在独立模式的程序,改变了其分支函数插桩的方法,解决了其模式所对应的分支无法得到评价的问题;然后设计了一种新的交叉算子,通过获取的每个模式的适应度值,将所有个体中的最优模式通过交叉算子集中到一个个体中,再使用粒子群优化算法进行测试用例生成,其他的个体在进化过程中都可以向这个最优的个体学习,加速粒子群进化;并且在进化过程中,对于每代的最优个体还使用了局部搜索策略,以进一步提高测试用例生成效率;最后,为了评价我们设计的交叉算子对于含有独立模式的程序的有效性,在实验部分与单点交叉、双点交叉和均匀交叉这 3 种交叉方式进行对比,结果表明:我们的交叉算子在交叉过程中由于没有破坏最优模式,而是将其当作一个整体进行交叉,因此效果远好于其他 3 种交叉算子.并且还将本文的方法 CL-PSO 与 PSO,APSO 和 OL-PSO 进行了对比,实验结果表明:CL-PSO 方法在对含有模式的程序进行测试用例生成时,相对于其他方法在覆盖率和平均进化代数上均有一定的优势.

由于本文的 CL-PSO 方法只针对含有独立模式的程序,而对于含有互斥模式的程序,由于模式之间存在相互影响,故本文未对其进行分析和处理.因此,我们下一步工作将分析互斥模式之间的关系,探索互斥模式在程序中受哪些因素的影响,并设计新的组合方式针对含有互斥模式的程序进行测试用例生成.

致谢 在此,我们对审稿人和编辑表示感谢,对本文提出建议的同行表示感谢.

References:

[1] Harman M, Jones BF. Search based software engineering. Information and SoftwareTechnology, 2001,43(14):833−839.

[2] McMinn P. An identification of program factors that impact crossover performance in evolutionary test input generation for thebranch coverage of C programs. Information and Software Technology, 2013,55(1):153−172. [doi: 10.1016/j.infsof.2012.03.010]

[3] McMinn P, Binkley D, Harman M. Empirical evaluation of a nesting testability transformation for evolutionary testing. ACM Trans. on Software Engineering and Methodology, 2009,18(3):824−833. [doi: 10.1145/1525880.1525884]

[4] Holland JH. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1992.

[5] Windisch A, Wappler S, Wegener J. Applying particle swarm optimization to software testing. In: Lipson H, ed. Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation. New York: ACM Press, 2007. 1121−1128. [doi: 10.1145/1276958.1277178]

[6] Wang LS, Jiang SJ, Zhang YM, Yu Q. Test case generation based on orthogonal exploration and particle swarm optimization. Acta Electronica Sinica, 2014,42(12):2345−2351 (in Chinese with English abstract).

[7] Korel B. Automated software test data generation. IEEE Trans. on Software Engineering, 1990,16(8):870−879. [doi: 10.1109/32.57624]

[8] Xiao M, El-Attar M, Reformat M, Miller J. Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. Empirical Software Engineering, 2007,12(2):183−239. [doi: 10.1007/s10664-006-9026-0]

[9] Kifetew MF, Panichella A, Lucia AD, Oliveto R, Tonella P. Orthogonal exploration of the search space in evolutionary test case generation. In: Pezzè M, Harman M, eds. Proc. of the 2013 Int'l Symp. on Software Testing and Analysis. Lugano: ACM Press, 2013. 257−267. [doi: 10.1145/2483760.2483789]

[10] Zhu XM, Yang XF. Software test data generation automatically based on improved adaptive particle swarm optimizer. In: Zhang J, ed. Proc. of the 2010 Int'l Conf. on Computational and Information Sciences. Chengdu: IEEE CPS, 2010. 1300−1303. [doi: 10.1109/ICCIS.2010.321]

[11] Wilcoxon F. Individual comparisons by ranking methods. Biometrics Bulletin, 1945,1(6):80−83.

[12] Cohen J. Statistical Power Analysis for the Behavioral Sciences. 2nd ed., Lawrence Erlbaum Assoc. Inc., 1988.

[13] Harman M, Jia Y, Zhang Y. Achievements, open problems and challenges for search based software testing. In: Wotawa F, ed. Proc. of the IEEE Int'l Conf. on Software Testing, Verification & Validation. Graz: GUM Press, 2015. 1−12. [doi: 10.1109/ICST.2015.7102580]

[14] McMinn P. Search_Based software testing: Past, present and future. In: Schieferdecker I, Pretchner A, eds. Proc. of the 2011 4th Int'l Conf. on Software Testing, Verification and Validation Workshops. Berlin: CPS Press, 2011. 153−163. [doi: 10.1109/ICSTW.2011.100]

[15] Sthamer H. The automatic generation of software test data using genetic algorithms [Ph.D. Thesis]. Brirain: University of Glamorgan, 1996.

[16] Wegener J, Baresel A, Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 2001,43(14):841−854. [doi: 10.1016/S0950-5849(01)00190-2]

[17] Fraser G, Arcuri A. EvoSuite: Automatic test suite generation for object-oriented software. In: Gyimóthy T, Zeller A, eds. Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering (FSE). New York: ACM Press, 2011. 416−419. [doi: 10.1145/2025113.2025179]

[18] Fraser G, Arcuri A. Whole test suite generation. IEEE Trans. on Software Engineering, 2013,39(2):276−291. [doi: 10.1109/TSE.2012.14]

[19] Fraser G, Arcuri A. A large scale evaluation of automated unit test generation using EvoSuite. ACM Trans. on Software Engineering and Methodology (TOSEM), 2014,24(2):8:1−8:42. [doi: 10.1145/2685612]

[20] Fraser G, Arcuri A, Mcminn P. A memetic algorithm for whole test suite generation. Journal of Systems & Software, 2014,103:311−327. [doi: 10.1016/j.jss.2014.05.032]

[21] Michael C, McGraw G, Schatz M. Generating software test data by evolution. IEEE Trans. on Software Engineering, 2001,27(12):1085−1110. [doi: 10.1109/32.988709]

[22] Miller J, Reformat M, Zhang H. Automatic test data generation using genetic algorithm and program dependence graphs. Information and Software Technology, 2006,48:586−605. [doi: 10.1016/j.infsof.2005.06.006]

[23] Watkins A, Hufnagel EM. Evolutionary test data generation: A comparison of fitness functions. Software Practice and Experience, 2006,36(1):95−116. [doi: 10.1002/spe.684]

[24] Hermadi I, Lokan C, Sarker R. Genetic algorithm based path testing: Challenges and key parameters. In: Huang XH, Xu LD, Zhou ZD, eds. Proc. of the 2010 2nd WRI World Congress on Software Engineering. Wuhan: IEEE Computer Society Press, 2010. 241−244. [doi: 10.1109/WCSE.2010.82]

[25] Mao CY, Yu XX, Xue YZ. Algorithm design and empirical analysis for particle swarm optimization-based test data generation. Journal of Computer Research and Development, 2014,51(4):824−837 (in Chinese with English abstract).

[26] Shi JJ, Jiang SJ, Han H, Wang LS. Adaptive particle swarm optimization algorithm and its application in test data generation. Acta Electronica Sinica, 2013,41(8):1555−1559 (in Chinese with English abstract).

附中文参考文献:

[6] 王令赛,姜淑娟,张艳梅,于巧.基于正交搜索的粒子群优化测试用例生成方法.电子学报,2014,42(12):2345−2351.

[25] 毛澄映,喻新欣,薛云志.基于粒子群优化的测试数据生成及其实证分析.计算机研究与发展,2014,51(4):824−837.

[26] 史娇娇,姜淑娟,韩寒,王令赛.自适应粒子群优化算法及其在测试数据生成中的应用研究.电子学报,2013,41(8):1555−1559.

姜淑娟(1966－),女,山东莱阳人,博士,教授,博士生导师,CCF 会员,主要研究领域为软件分析与测试,编译技术.

张艳梅(1981－),女,博士,讲师,CCF 会员,主要研究领域为软件分析与测试.

王令赛(1989－),女,硕士,主要研究领域为软件测试.

于巧(1989－),女,博士生,主要研究领域为软件分析与测试.

薛猛(1979－),男,讲师,CCF 会员,主要研究领域为软件测试,测试数据生成.

姚慧冉(1989－),女,硕士生,主要研究领域为软件测试,测试数据生成.