Pairwise Testing Generation based on Particle Swarm Optimization Algorithm

Cyrus Lee, JIA WEI

Abstract: generation of test data is a key problem of combining test data. In order to solve this important problem, we use particle swarm optimization algorithm. Therefore, how to cover the higher or all test items in the test set with fewer pairwise test cases under the condition that the test case set can be generated automatically is the problem we need to study this time. This effort not only improves productivity by reducing the size of the tests, but also frees workers from the task of generating test sets. This paper proposes two solutions to this kind of problem. The first one is to generate pairwise test set based on the basic particle swarm optimization algorithm. The second method is based on one test at a time to generate pairwise pairing of particle swarm. In the process of algorithm implementation, we limit the threshold value of the given parameters to ensure the stable operation of the particles and the efficiency of the algorithm. And use Python to write the corresponding algorithm. The results show that the method presented in this paper is effective, feasible, and in a certain data scale has the advantage of operation time.

Key words: particle swarm optimization algorithm; Pairwise testing; Fitness; Pairwise

目录

1 绪论

本章介绍课题的研究背景以及发展事项，给出相关领域内，相对应问题的思路和解决办法，并简明阐述本文的研究内容和组织结构。

1.1 研究背景及现状

随着科技的进步和发展，人们对于软件的功能要求也日益增多。与此同时，伴随的相关问题也日渐陡增。因此，解决与之相伴问题一直以来都是经久不衰的研究话题。众所周知，软件的开发并非简单的写一段程序运行，其建立在各种各样的基础和需求之上。这使得通过软件测试来确保其运行的安全性和可靠性变得由为重要。

现如今，智能设备的不断更新迭代，层出不穷的各种软件，无一不展示出广大人民群众对于便捷的更高要求。从上世纪的第一代计算机 ENIAC 到现如今的掌上手机和电脑，人民群众的生活也变得更加方便快捷。购物出行，吃饭付款，各方面方便快捷的背后都有软件的支撑。因此，确保软件在不同平台稳定并高效的运行是我们如今最为关注的问题之一。因此，软件测试在软件上架运营之前由为重要。

1.2 组合测试背景及发展

在软件测试过程中，有时由于数据集合数量过大或输入参数数量较多等各种因素的影响，在软件测试的过程中进行穷尽测试大多数时间没有必要也不可能。因此，根据实际问题的相关需求，可以使用较少的测试数据集合去覆盖尽可能多的各种组合，并检测它们之间共同对软件系统造成的影响。因此，我们根据软件测试集合覆盖程度的不尽相同，从而将其划分为多种组合覆盖，比如两两组合覆盖。我们在应用不同种类的覆盖测试数据进行测试时。发现了许多传统方法难以发现的问题[1]，有研究发现大约有 70%的软件故障是由一个或两个参数的相互作用而引发的[2]，故而两两组合配对测试有相当重要的研究意义。

最小两两组合覆盖测试数据的生成问题是一个 NP-hard 问题, 因此我们会经常性的使用启发式算法来进行相关的求解[3]或者使用代数构造方法, 以此为依据来生成规模尽可能小的测试集合[1-15]和与其相关的测试数据[4]。早些年，大多数生成测试集合的主要是基于正交试验设计的，用此种方法来生成对应的测试数据集合。虽然该种方法主要以正交表为基础进行测试集合生成，但正交表在其存在性和构造方法等方面也有许多未解决的难题，因此常常具有一定的难度。在此之后，Kobayashi 等人[5]和 Williams 分别给出了[6]两种代数方法，可以在一定程度上有效地生成测试数据。除了以上所提及的相关代数方法外，美国贝尔实验室的 Cohen 等人[7]提出了一种测试数据的启发式生成方法，并以此其所研究的启发式算法为基础，在此之上开发了相应的测试数据自动生成系统 AETG，申请了专利。美国喷气推进实验室的 Tung 等人[8]也提出了一种相关的两两组合测试数据集启发式生成算法，并以此为基础，在其之上开发了名为 TCG 的测试数据生成工具。而 Lei 等人[3]提出的测试数据自动生成系统 PAIRTEST，是一种基于所给出参数顺序之上，进行渐进扩充的两两组合测试覆盖测试数据生成方法。

近年来，国内在组合测试数据生成领域也进行了相关系统的研究，并以此为依据提出了多种数学与启发式生成算法。中国科学院的严俊等人基于 SAT 求解工具以及回溯法生成两两组合覆盖测试数据[9]；南京大学的陈翔等人基于蚁群算法生成变力度的交互组合测试数据[10]；东南大学的查日军，南京大学的张德平等人，在这一领域内同样作了相关深入的研究，他们研究并在其基础上提出了有关基于网络图的两两组合测试数据生成算法[11]，并且在其基础上还对 AETG 方法进行拓展，以此

为依据给出了一种支持高维组合测试数据集生成方法[12]。在组合覆盖测试模型的基础上，将所有可用测试数据表示为一棵解空间树，利用回溯法搜索解空间树，便可在解空间树中选择出一个规模最小的路径集，以实现对测试参数的两两组合覆盖[13]。除此之外，利用组合分析的方法，针对二水平多因素系统给出两两组合测试数据生成算法[14]，对一类只在相邻因素之间存在相互作用的系统，提出了相邻因素组合测试的概念，给出了相邻因素两两组合覆盖表的生成算法[15]。

本文由于作为启发式搜索方法的一种补充，故利用了以仿生学为基础的粒子群优化算法来解决相关的两两组合测试数据生成问题，对核心问题的求解进行了文字说明并给出流程图。在此基础上，给出了两种算法的相关伪代码来进一步对解决问题的办法进行描述。

1.3 基于搜索的测试数据生成办法

基于搜索的算法有全局搜索和局部搜索。全局搜索算法包括但不限于本文中所提到的粒子群算法，局部搜索还包括其他算法例如模拟退火算法等，这类基于搜索的算法在生成最终的测试用例上主要方法是构造适应度函数，即将该问题转化为优化问题。便可用启发式算法来解决此类问题。

1.4 论文的研究内容和组织结构

本文的研究内容主要分为以下几个方面来进行说明，一方面是针对于粒子群算法本身而言，另一方面是将粒子群算法和两两配对组合测试数据生成问题进行结合。首先对粒子群算法本身进行研究分析，对其功能和解决问题的方法进行阐述。其次是为两问题的结合构造适应度函数，即将另一问题转换为优化问题并进行求解最后得出测试用例。

本文总共分为六章，以下是对每个章节内容的概括描述：

第一章为绪论部分，主要对本课题的研究背景和发展趋势进行相应的介绍和简单的说明，并交代相关领域内的研究成果。其次简要说明解决本文题目所提出问题的思路。最后对本文的内容和结构进行说明。

第二章介绍粒子群算法。讲述其运行原理，并对该算法的特性进行分析并评估该算法的优劣性。为第四章解决问题进行铺垫

第三章介绍两两组合配对测试的相关内容。包括该种方法的理论由来，并简单举例说明问题的实际应用，然后分析该问题的相关要点。为下一章转化为优化问题提供思路。

第四章将本文所提出的问题转化为优化问题并引入两种解决方案。首先是适应度函数的选取问题详解，即如何将两两配对组合测试问题转化为优化问题。并给出解决该问题的思路以及相关的算法。随即引入两种解决方案，在基于粒子群算法的基础上。算法二使用了 one test at a time 的策略。并与算法 1 形成对比，分析两种方法的优劣。并给出相关实验数据。最后总结本章

第五章为以上两种算法设计相关的实验，将两种算法进行对比并在一定条件下对两种算法的优劣性进行评判。再使用较为优秀的算法与其他算法生成的数据在规模和时间上进行对比，从各角度对算法本身的优劣进行评估。

第六章对以上所有的实验数据结果和算法进行总结，对两算法基于测试得出优劣性的原因进行分析，并对两种算法的特性进行总结。

最后对本文的研究方法做出总结，说明所做的研究成果，指出当前研究的不足之处，以及相关领域目前存在的缺陷和未来研究的趋势。

2 粒子群优化算法的相关理论

本章将会对粒子群优化算法的基本原理和实现算法的流程进行说明，并给出相应的粒子群优化算法流程图，以及将该算法应用于成对测试后的算法流程。

2.1 粒子群优化算法基本原理

粒子群优化算法（particle Swarm Optimization，PSO）的起源来自于对简单社会系统的模拟，最初是模拟鸟群体觅食过程[16]该算法是由 Kennedy 和 Eberhart 等[17]在 1995 年提出的一种基于种群搜索的自适应演化计算技术。该算法通过迭代来计算最优值，通过粒子的位置来表示待优化问题的解。我们通过计算待优化问题目标函数的适应度来评价一个粒子的优劣。此外，每个粒子还具有速度这一属性。这决定粒子本身的飞行方向和速率大小。而pbest和gbest分别用来记录当前历史上每个粒子个体的最优和迭代次数至当前一代时全局的最优。故前者称为个体极值，后者称为全局极值。

设在一个d维的目标搜索空间中，粒子数量有n个，则当迭代到t次时。粒子i的位置如下所示

$$X_i^t = (X_{i1}^t，\ X_{i2}^t，\ X_{i3}^t，\ \ldots，\ X_{id}^t)$$

与其相应的粒子飞行速度可表示为

$$V_i^t = (X_{i1}^t，\ X_{i2}^t，\ X_{i3}^t，\ \ldots，\ X_{id}^t)$$

在$t+1$代迭代时，粒子i的速度和位置的更新规则为

$$v_{ij}^{t+1} = \omega v_{ij}^t + C_1\gamma_1\left(pbest_{ij}^t - x_{ij}^t\right) + c_2\gamma_2\left(gbest_{ij}^t - x_{ij}^t\right) \qquad (1)$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \qquad (2)$$

$$\mathrm{i}=1,2,\ldots,\mathrm{n},\ \mathrm{j}=1,2,\ldots,\mathrm{d},$$

其中，C_1和C_2为两个学习因子，C_1代表对粒子自身历史的学习，C_2代表对粒子群体信息的学习；r_1和r_2代表两个在（0，1）均匀分布的随机数；ω为惯性权重，其决定了粒子对于原速度的记忆保持程度。在粒子群优化算法的参数中，ω的选取较为重要，较大则会使算法对未探测空间的搜索能力增强，但对局部细节调整的能力较弱，而较小则会增强其局部细节的调整能力，但探测能力则会相应减弱。因此在全局寻优和局部寻优两者之间，我们采取其他针对粒子群优化算法论文所推荐[18]的$C_1=C_2=1.49$，$\omega=0.729$。

在公式（1）速度更新公式中，该公式含义为：此代粒子速度=上一代粒子自身运动速度+上一代粒子朝自己最佳方向+上一代粒子朝全局最佳方向。

在公式（2）位置更新公式中，该公式含义为：此代粒子位置=上一代粒子位置+此代粒子自身运动速度。

2.2 粒子群优化算法基本流程

在该算法中，我们每次初始化粒子种群，并根据相对应的目标函数所抽象出的适应度函数来评估种群中每个粒子的优劣，随后进入迭代循环，直到达到最大迭代次数或全局最优后从循环中跳出。每次迭代时均需要根据算法本身的原理特性，利用相对应的优劣评价准则对粒子的速度与位置进行更新。除此之外，我们在每次更新粒子自身的参数之后，还需要更新历史最优位置和当前最优位置。以此来为基准使得粒子群优化算法的运行趋向于收敛。

粒子群优化算法基本算法流程如下图：

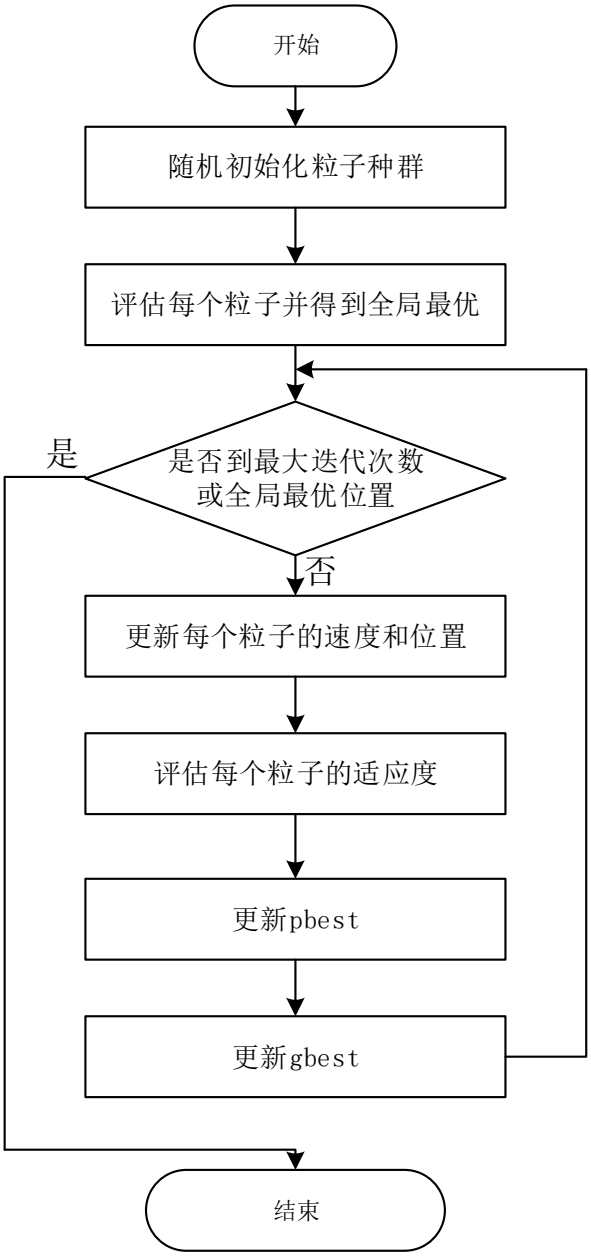

图 2.1 粒子群优化算法流程图

2.3 基于粒子群优化算法的组合测试数据生成流程

在本文中我们需要解决两两组合配对测试问题，因此我们将该算法应用两两于组合测试中。此处我们用每一个粒子来代表一个测试数据。则初始化随机生成的粒子种群中粒子在每一维度的位置代表其当前测试用例中的一个测试数据，在此基础上应用粒子群优化算法，使得问题得以解决。

该算法可用以下流程所表示：

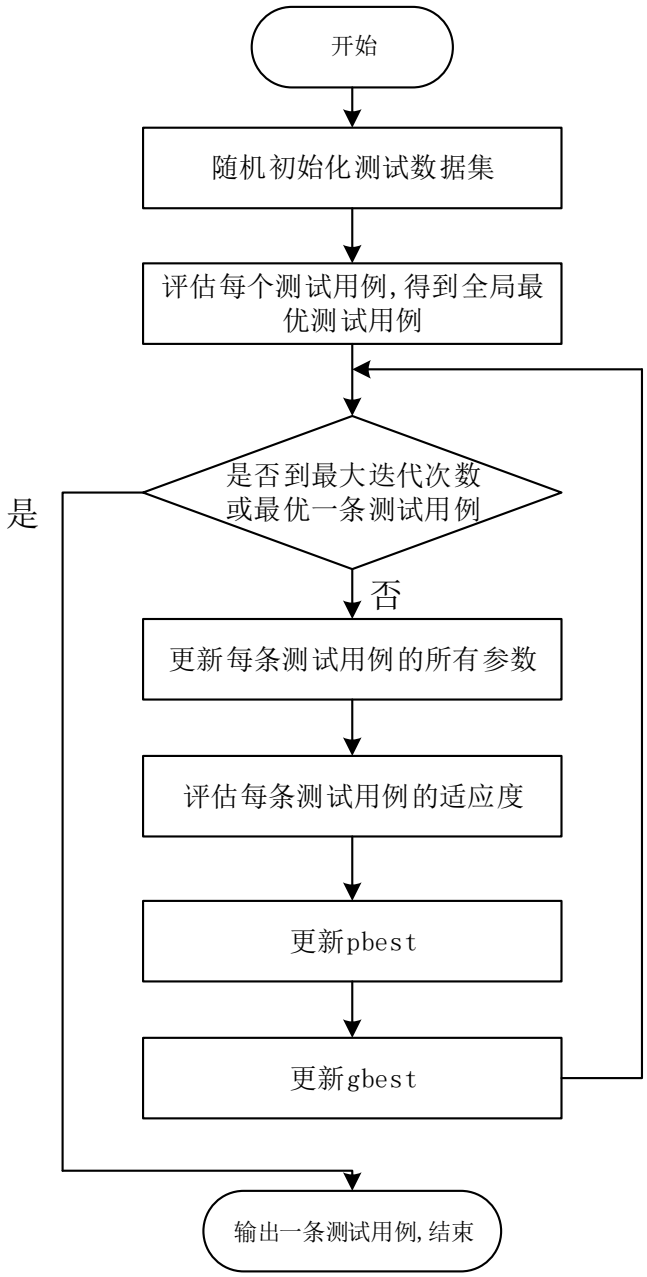

图 2.2 基于粒子群优化算法的组合测试数据生成流程图

3 pairwise testing 的相关介绍

本章将给出有关 pairwise testing 相关的原理介绍以及对问题的分析，举例说明并解释 pairwise testing。引入问题并在下一章节给出相应的解决办法。

3.1 pairwise testing 的原理

我们对于待测软件系统有如下假设，假设被测系统有k个参数，构成参数集$P = \{p_1, p_2, \ldots, p_k\}$，其中参数$p_i$有$n_i$个取值，取值集为$V_i = \{v_{i1}, v_{i2}, \ldots, v_{in_i}\}$。$TS$为所需的覆盖所有参数的不同取值的测试数据集。$TS$是由若干个$t$(测试数据）构成的集合，每个$t$可表示为一个$k$元组$t = (v_1, v_2, \ldots, v_k)\ (v_1 \in V_1, v_2 \in V_2, \ldots, v_k \in V_k)$ [19, 20]。所谓两两组合测试是在此基础上使得测试数据集中每条测试数据t中的v_i和v_j所组成的二元组集合均包含集合$V_i \times V_j$中的所有元素[19, 21]。对于一个 3 参数系统，每个参数有 2 个可选值，生成完全的测试数据需要$2^3 = 8$ 条测试数据，但采用两两组合测试，仅需要测试其中的 4 条即可对其中的任意两个参数进行所有取值组合，我们将举例说明这一方法。

组合测试用例集合可以用一个矩阵来表示，矩阵的每一行表示一个测试用例，每一列代表系统的一个参数，每一项代表测试用例的相应参数的取值。[22]

假设有 3 个维度，每个维度有 2 个因子。对于每一个维度中的所有因子，我们对其使用简写名称，即取其首字母来代表该参数，如下所示：

表 3.1 名称及其参数简写

名称	参数	简写
浏览器	Microsoft edge	M
	Opera	O
操作系统	Windows	W
	Linux	L
语言	Chinese	C
	English	E

我们给出了上文所假设的 3 个参数，每个有 2 个可选值的待测软件系统。其全部测试用例如下表所示：

表 3.2 所有组合测试

序号	浏览器	操作平台	语言
1	M	W	C
2	M	W	E
3	M	L	C
4	M	L	E
5	O	W	C
6	O	W	E
7	O	L	C
8	O	L	E

在不使用 pairwise 方法时，我们需要对该系统的 8 个用例进行全部测试。该方法测试覆盖率为 100%，但当问题规模过大时，我们需要测试的数据量将会呈指数级别增长。因此我们使用 pairwise 进行测试，从最下方一个 8 号开始向上去除重复用例，8 号的两两组合是 OL，OE ，LE。其中 OL 在

7 号出现过，OE 在 6 号出现，LE 在 4 号出现。所以 8 这个 case 就可以舍去，以此类推直到所有测试用例均被检测，则可得出下表。

下表共计 4 组测试用例，相比较不使用 pairwise 节省了 50%的测试。

表 3.3 使用 pairwise

序号	浏览器	操作平台	语言
2	M	W	E
3	M	L	C
5	O	W	C
8	O	L	E

同理我们也可以从上往下，最终得出的结果如下表所示：

表 3.4 使用 pairwise

序号	浏览器	操作平台	语言
1	M	W	C
4	M	L	E
6	O	W	E
7	O	L	C

如表中所示，剩下 4 个测试用例，但是具体内容不同。pairwise 算法最终剩下的测试用例个数肯定相同，但是可以有不同的用例组合。

3.3 pairwise testing 相关分析

根据以上原理描述和算法的讲解，并给出具体实例分析可得出如下结论。首先，两两组合配对测试数据集合的生成，是在所有因素的两两组合配对数据集合中找到规模最小的测试数据集合。由于该问题已被证明为 NP-C 问题[23]，因此我们选用启发式算法中的粒子群优化算法来在一定程度上解决此问题。

其次，如何评定一个测试数据集合为优秀的测试数据集合，即对于所生成的测试数据集，我们需要依据来判断所生成的测试数据是否足够小，是否对所有测试数据达到全覆盖。为此，我们可以先行针对问题规模进行初始化，得到相应的两两组合配对覆盖矩阵，并以此为依据在每次生成相应的测试用例时对矩阵进行标注，用此种方法进行求解。

最后，将该问题转化为优化问题并对使用粒子群优化算法其进行编码求解，该内容将在下一章进行详细阐述。

4 将 pairwise 转为优化问题

本章首先介绍了相关适应度函数选取的研究方法和解决办法，随后给出目标问题转化为符合问题求解的适应度函数的两种方法，并提出两种解决算法以及给出算法伪代码和讲解。

4.1 适应度函数的选取问题研究及解决

在粒子群优化算法中，目标函数便是待求解的问题。在优化问题中，我们将目标函数作为适应度函数来计算每个粒子的适应值，并以此为依据对粒子的优劣进行判断。因此，只有将目标函数转化为待求解问题相对应的适应度函数才能合理有效的对粒子进行优劣评估。

我们在基于遗传算法的两两组合配对算法基础上[24]。对其进行了改进，并针对该文章中所给出的适应度计算方法进行了适当调整，使其在粒子群优化算法中起到同样的效果。

下面我们会给出两种适应度的计算方法，并给出相应的流程图和讲解。并会在随后给出两种算法的相应步骤和流程图。

4.1.1 算法 1 基于粒子群优化算法的组合测试数据生成

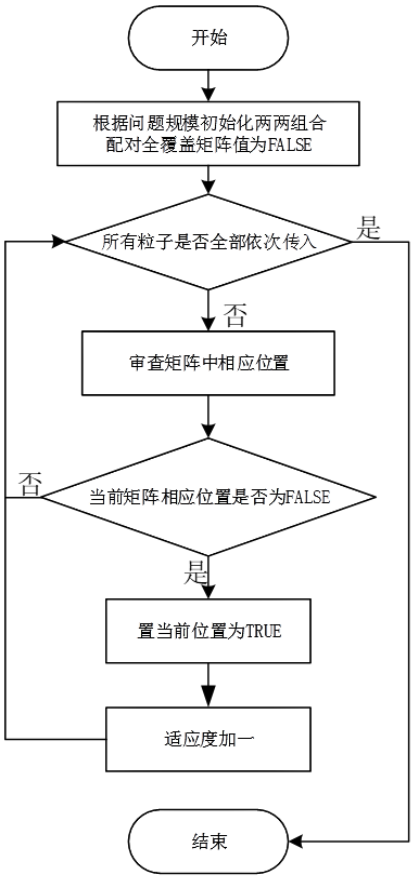

图 4.1 算法 1 分支循环图

(1). 根据前文的定义，问题的规模为k，v。据此首先初始化适应度数组和两两组合配对全覆盖矩阵，该矩阵可看作由k^2数值个$v*v$组成的二维分块矩阵所组成，将该矩所有值阵置为 FALSE。

(2). 将粒子种群中，所有粒子依次传入进行计算，此时每个粒子的每一个位置参数代表一条测试用例的参数个数。每计算完一个粒子，后移到下一个粒子的位置，直到所有粒子计算完毕。

(3). 对传入粒子的所有位置参数进行两两配对组合，每组合配对一次，在两两组合配对全覆盖矩阵中对相应的位置进行审查。如果该位置的值为 FALSE，则置 TRUE，该粒子当前的适应度加一。直到该粒子的所有内部参数全部配对并审查完毕。

(4). 返回存储所有粒子适应度的数组。当前一代每个粒子的优劣性评估完毕。

4.1.2 算法 2 基于 one test at a time 策略的组合测试数据生成

(1). 和算法 1 第一步类似。我们首先初始化适应度数组，和两两组合配对全覆盖矩阵，按照算法 1 的方法，将该矩阵看为分块矩阵。按照右上三角矩阵定义规则，但除过对角线，将其所有两两组合配对对应位置的值置为 TRUE，剩余部分为 FALSE。

(2). 传入每一个粒子进行计算，计算方式与算法 1 类似。但每次计算完全部粒子之后选取适应度最高的任意一个粒子，将其拿出。并放入最终的测试集合中。

(3). 根据拿出的粒子，修改矩阵相应位置的值，TRUE 置为 FALSE。

(4). 重置适应度数组，准备下一次计算。

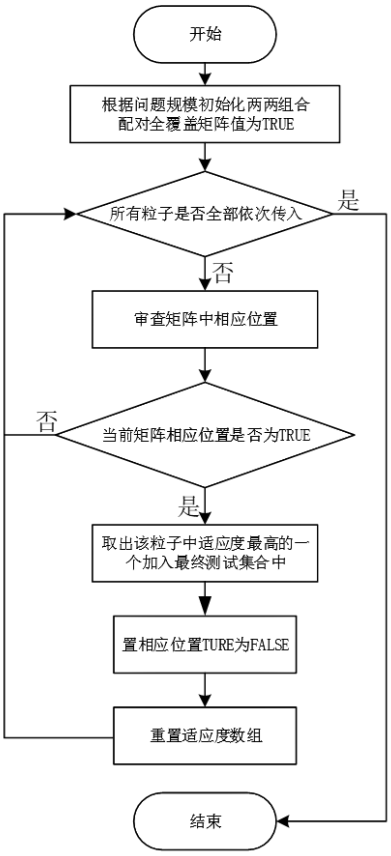

图 4.2 算法 2 分支循环图

此处需要注意的是，算法 1 中每次传入参数后，需要计算的数据为输入最终得到的测试用例组数，例如输入 15 组，则每个粒子中包含 15 组，每组k个参数的测试用例，也就是一条测试用例中包括$15*k$个参数。因此在迭代终止后，最终得到的就是一个粒子，也是一整条测试用例。而算法 2 每次传进去参数后的区别在于，只需要计算当前种群中适应度最大的的一个粒子，该粒子仅包含 k 个参数，并将其添加到最终的测试用例集合中，因此每次迭代都选取适应度最大的测试用例添加至集合中直到达到迭代次数。

4.2 两种基于粒子群算法的提出解决

4.2.1 基于粒子群优化算法的测试用例生成

本算法基于 PSO 来进行迭代计算，并以此为依据生成了最终的测试集合。在初始化时，所需要的测试用例数由人为给出，因此我们在运行该算法时更多地去关注在已给定的测试用例条数时，能达到怎样的适应度数值和覆盖矩阵中的剩余程度。最终我们输出的是在给定测试用例数条件下所对应的一个粒子，即一整条测试用例。

① $InitializePopilation()$

② $for\ i\ in\ range(\max\_itera)$:

③ $update\ veolcity$

④ $update\ position$

⑤ $caculate\ fitness\ and\ update\ pbest$

⑥ $update\ gbest$

⑦ end

⑧ $Output\ one\ particle\ in\ population$

4.2.2 基于 one test at a time 的粒子群优化算法测试用例生成

本算法在基于 PSO 迭代的基础上加入了 one test at a time 策略，该方法同样可以给出最终的数据测试集合。在最开始时初始化粒子种群和最终给出的测试集合P为空集。在经过每次运算之后，会选出一条最优的测试用例加入到测试集合P中，并在相应的矩阵中去掉已被覆盖测试过的用例。直到达到迭代次数。最终我们输出的是测试集合P。

① $InitializePopilation()$

② $P = InitializeFinalTestCaseArray()$

③ $CaculateFitness$， $find\ \max FitnessValue$

④ $extend\ corrspending\ testcase\ into\ set\ P$

⑤ $for\ i\ in\ range(\max\_itera)$:

⑥ $CaculateFitness$，$update\ pbest$

⑦ $update\ gbest$

⑧ $find\ \max FitnessValue$

⑨ $extend\ corrspending\ testcase\ into\ set\ P$

⑩ $update\ veolcity$

⑪ $update\ position$

⑫ end

⑬$Output\ set\ P$

5 实验设计与分析

本章根据上文所给出的相关解决算法，首先对两种算法的正确性进行对比，再给出合适的实验分析方案。对问题进行两种规模量级的测试，以及和其他算法相应优劣程度的对比，给出相关实验数据和统计图线。综合各方面因素来衡量两算法的差异性和优劣性。

5.1 分析论证实验正确性

我们基于小规模数据对两算法正确性进行验证，选择规模为3^2的数据集进行代码测试和手动运算两种方式对比，并分析相关正确性。随后设计相关实验，对得出数据分析类比，得出结论。此处我们给定粒子群的种群大小$particle\ size$为 50 个粒子。最大迭代次数$max\ itera$ 为 500 代，惯性权重和学习因子数值的选取前文已经给出。

首先我们进行手动运算，$k=2$，$v=3$。则代表每一个测试数据中含有 2 个参数，每个参数具有 3 个候选值。则共有 6 个参数。我们以 0 为下标初始化两两配对组合矩阵。其中自身无法进行配对，重复配对之计算一组，其余置 0。如下所示：

下标	0	1	2	3	4	5
0				1	1	1
1				1	1	1
2				1	1	1
3	0	0	0			
4	0	0	0			
5	0	0	0			

图 5.1 两两组合覆盖矩阵

可以看出，所有 pairwise 组合共有 9 种情况，即 9 种组合便可使得测试用例集达到全覆盖。所以满适应度为 9。测试用例分别是：

表 5.1 手动运算两两组合配对

规模：3^2	配对组合
1	0,3
2	0,4
3	0,5
4	1,3
5	1,4
6	1,5
7	2,3
8	2,4
9	2,5

我们随即使用算法 2 和算法 1 运行对比，其中算法 1 给定一个粒子中所包含的测试组数为 9 组。根据算法运行，得出下表：

表 5.2 算法 2 两两组合配对

算法 2	测试数据
1	2,4
2	1,4
3	0,3
4	2,3
5	0,5
6	1,5
7	2,5
8	0,4
9	1,3

表 5.3 算法 1 两两组合配对

算法 1	测试数据
1	1,3
2	1,5
3	0,3
4	2,3
5	2,5
6	1,3
7	2,4
8	2,4
9	0,5

由上两表所见，虽然生成的测试数据集合顺序为乱序。但都对 pairwise 矩阵进行了全覆盖，因此该算法正确性在一定程度上得到了论证。在后面的试验中，我们将会在每次计算小，中，大三种规模数据集合时，先计算总适应度值。再计算当前结果下的适应度在该矩阵中的占比。为了避免在三种规模下发生不确定性因素，我们保存了两种算法所生成的相应规模的测试数据集合，方便后续需要验证时进行使用。

5.2 两种算法小型规模测试数据计算对比

为了检验粒子群优化算法（PSO）算法在两两配对组合数据生成方面的效果，我们在以上所提出的两种方法的基础上，进行了进一步的实验。本章将对实验数据进行分析，并以以下参数符号来表示具体的 SUT：V^K来表示 SUT 中有K个参数，其中每个参数具有V个取值。例如：3^4则代表 SUT 有 4 个参数，其中每个参数具有 3 个取值。在实验中，我们给定粒子群的种群大小$particle\ size$为 50 个粒子。最大迭代次数$max\ itera$ 为 500 代，惯性权重和学习因子数值的选取前文已经给出。

由于 pairwise 问题是 NP 的，且 PSO 算法是启发式算法，因此不能保证在任何情况下都能生成最小两两组合覆盖的测试数据。为了检验其测试的效果，我们将从以下几个方面考虑：首先是小型，中小型数据规模对比，将上文提出的两种算法在算法 2 生成已定测试集合用例数的情况下，在算法 1 中输入同样的测试集合用例数，分析对比两者的运行时间，适应度大小，对两两组合矩阵的覆盖占比情况。其次按照同等原理对中等，较大型规模数据进行对比分析并绘制出相应的数据统计图线，以此为依据得出两者中综合性能较优的算法，最后将该算法在最后与其他测试系统所给出的数据进行分析对比。以数据生成规模和运行时间为依据，更进一步分析算法的优劣性。

将算法 1 和算法 2 的两种方法对其在相同问题规模，输出同等测试数据集合数的情况下进行对比，分析其生成最终测试数据所用的运算时间，适应度大小，矩阵覆盖情况。我们根据以下两种测试规模对两种算法进行测试：2^3， 3^4，并进行相关实验分析。

表 5.4 小规模测试数据

规模：2^3	测试数据个数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	4	29.1	12（100%）
	4	30.7	12（100%）
	4	29.9	12（100%）
	5	30.6	12（100%）
	4	29.6	12（100%）
算法 1	4	36.9	12（100%）
	4	33.5	12（100%）
	4	35.2	12（100%）
	5	42.9	12（100%）
	4	34.0	12（100%）

由表 5.4 可知，在测试小规模数据时。两算法在预设数据下，生成等量数据集时间相差不大，都可生成全覆盖的测试数据集合。

表 5.5 小规模测试数据

规模：3^4	测试数据集数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	13	35.5	54（100%）
	13	34.9	54（100%）
	12	37.4	54（100%）
	12	37.6	54（100%）
	11	35.5	54（100%）
算法 1	13	130.8	51（94.4%）
	13	129.0	50（92.5%）
	12	109.9	50（92.5%）
	12	111.5	50（92.5%）
	11	109.8	48（88.8%）

由表 5.5 可知，在测试小规模数据时。两算法在预设数据下，算法 2 生成等量数据集所需的时间较小于算法 1，并且在该种情况下算法 2 依旧拥有 100%适应度。算法 1 运行时间随数据集规模增长有较为明显的增长，并且适应度也有一定下降，但能够保持在 92%左右的矩阵覆盖占比。

5.3 两种算法在不同规模下的测试数据计算对比

同理，将算法 1 和算法 2 的两种方法对其在相同问题规模，输出同等测试数据集合数的情况下进行对比，分析其生成最终测试数据所用的运算时间，适应度大小，矩阵覆盖情况。我们根据以下四种测试规模对两种算法进行测试 ：3^{13}，3^{20}，10^{20}，10^{30}，并进行相关实验分析。

由表 5.6 可知，在测试中型规模数据时。两算法在预设数据下，算法 2 生成等量数据集所需的时间明显小于算法 1。随问题规模增加，该算法的运行时间也有一定程度增加。在该种情况下算法 2 对两两组合矩阵依然可以全覆盖。算法 1 运行时间随规模增长显著增长，并且适应度一度维持在 664 左右，对两两组合矩阵的覆盖情况也相应下降，但稳定在 95%左右。

表 5.6 中规模测试数据

规模：3^{13}	测试数据集数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	26	93.4	702（100%）
	26	103.1	702（100%）
	25	96.4	702（100%）
	24	104.0	702（100%）
	27	102.1	702（100%）
算法 1	26	769.1	664（94.5%）
	26	862.2	670（95.4%）
	25	793.9	670（95.4%）
	24	715.4	662（94.3%）
	27	792.3	674（96.0%）

表 5.7 中规模测试数据

规模：3^{20}	测试数据集数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	30	142.4	1710（100%）
	30	142.2	1710（100%）
	31	138.1	1710（100%）
	30	133.5	1710（100%）
	30	139.9	1710（100%）
算法 1	30	1635.1	1628（95.2%）
	30	1356.6	1634（95.5%）
	31	1374.8	1637（95.7%）
	30	1389.4	1641（95.9%）
	30	1647.0	1634（95.5%）

由表 5.7 可知，在测试中型规模数据时。两算法在预设数据下，算法 2 生成等量数据集所需的时间显著小于算法 1。随问题规模增加，算法 2 的运行时间相应增长。在该种情况下算法 2 对两两组合矩阵依然可以全覆盖。算法 1 运行时间相比之前有明显增长，并且适应度一度维持在 1634 左右，对两两组合矩阵的覆盖情况也相应下降，但依然稳定在 95%左右。

表 5.8 中大规模测试数

规模：10^{20}	测试数据集数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	452	144.4	18999（99.9%）
	440	143.6	18997（99.9%）
	442	142.4	18897（99.9%）
	454	137.7	18998（99.9%）
	445	137.6	18999（99.9%）
算法 1	452	15556.8	16816（88.5%）
	440	15531.1	16830（88.5%）
	442	15120.0	16781（88.3%）
	454	16136.6	16839（88.6%）
	445	16599.8	16809（88.4%）

由表 5.8 可知，在测试中大型规模数据时。两算法在预设数据下，算法 2 生成等量数据集所需的时间远远小于算法 1。随问题规模增加，算法 2 的运行时间较之前无太大变化。在该种情况下算法 2 对两两组合矩阵无法做到全覆盖，但都保持在 99.9%，根据前几种情况分析，应为迭代次数相

比于数据规模较小而导致此种原因产生。算法 1 运行时间相较之前增长 10 倍，并且适应度一度维持在 16816 左右，对两两组合矩阵的覆盖情况也下降较多，稳定在 88.5%左右。

表 5.9 大规模测试数据

规模：10^{30}	测试数据集数(条)	运行时间(s)	适应度(矩阵覆盖占比)
算法 2	501	211.2	43465（99.9%）
	501	217.3	43456（99.8%）
	500	214.9	43446（99.8%）
	501	209.2	43464（99.9%）
	500	205.1	43458（99.8%）
算法 1	501	29091.6	38981（89.6%）
	501	28754.2	38901（89.4%）
	500	31097.8	38959（89.5%）
	501	28724.5	38860（89.3%）
	500	30076.1	38941（89.5%）

由表 5.9 可知，在测试大型规模数据时。两算法在预设数据下，算法 2 生成等量数据集所需的时间远远小于算法 1。随问题规模量级增加变化，算法 2 的运行时间相比之前也有相应的增加但并不明显。在该种情况下算法 2 对两两组合矩阵无法做到全覆盖，但都保持在 99.8%左右，根据前几种情况分析，应为迭代次数相比于数据规模较小而导致此种原因产生。算法 1 运行时间相较之前增加巨大，约为之前的 1.8 倍，并且适应度维持在 3890 左右，对两两组合矩阵的覆盖情况也下降严重，稳定在 89.5%左右

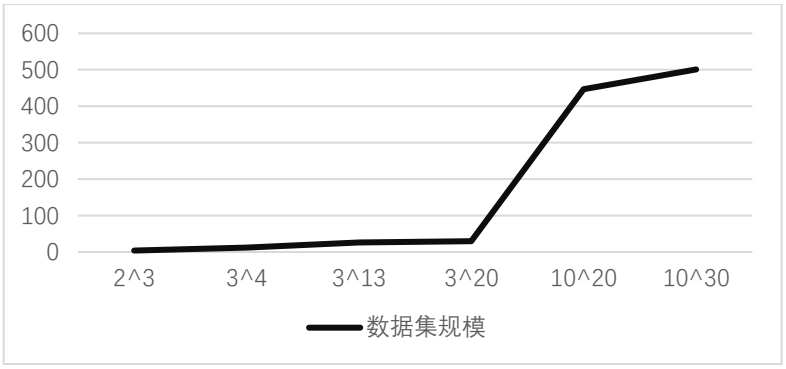

图 5.2 问题规模与生成的测试集合规模

根据图 5.2 所示，随着问题规模增长，生成的最终测试集合规模也随之上涨，但变化并非线性。随着k即指数的增长，最终生成的测试用例集合规模增长明显。由此可见该问题为 NP 问题。

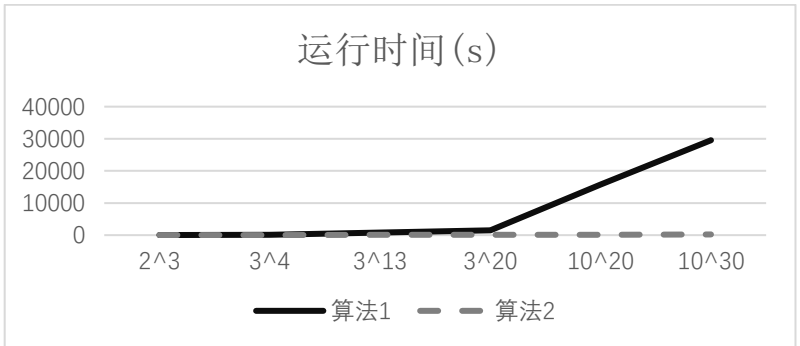

图 5.3 运行时间(s)

根据图 5.3 所示，随着问题规模增长，算法 1 运行时间在k，v发生明显较大增长变化时，运行时间显著增长。而算法 2 时间增长变化趋势低，在大规模问题的情况下还能保持较小的运行时间。

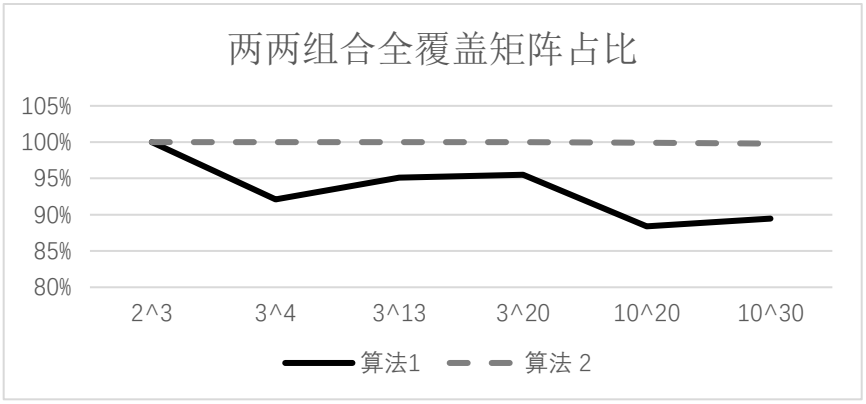

图 5.3.3 两两组合全覆盖矩阵占比

根据图 5.3.3 所示，随着问题规模量级的增长，算法 1 全覆盖矩阵占比逐渐下降，在数据集合规模较大时，算法 1 适应度占比下降较为严重，在10^{20}时低于 90%，根据其曲线的发展趋势来看，适应度占比将会随着问题规模的增大而逐渐降低。算法 2 随问题规模量级增大，其全覆盖矩阵占比情况略微下降，但依旧保持在 99%左右，适当增加其迭代次数将会回到 100%占比。

根据上两种针对小型，中型，大型数据规模的测试实验，以及绘制的相应数据统计图线，进行数据分析对比。在规模较小时，两种算法并无明显的优劣体现。在处理问题的时间花费，适应度方面都有较好的表现。随着问题规模量级增长至中型规模数据集，算法 1 运行时间明显增长，并且对两两组合矩阵的覆盖情况也有所下降。无法做到最后生成的测试集完全覆盖两两组合矩阵，但依旧有较高的覆盖率，稳定在 90%以上。算法 2 运行时间小，能达到两两组合矩阵全覆盖。对于较大型数据规模，随规数据规模大幅度增加，算法 2 由于迭代次数限制，适应度和相应的两两组合矩阵覆盖比也有略微下降，但无论在运行时间，适应度和两两组合矩阵覆盖比上均远优于算法 1。

在只考虑运行时间，适应度和两两组合矩阵覆盖比条件下，用算法 2 生成固定规模的测试数据集合后。使用算法 1 定义生成等量的测试集合数据并进行对比。两种算法中，算法 2 较为优秀。综上所述，得出下表。

表 5.3.5 运行时间

运行时间	小规模数据	中规模数据	大规模数据
算法 2	相当	较优	优
算法 1	相当	较劣	劣

表 5.3.6 矩阵覆盖占比

适应度（矩阵覆盖占比）	小规模数据	中规模数据	大规模数据
算法 2	相当	较优	优
算法 1	相当	普通	普通

5.4 相关算法生成数据集个数及时间对比

将根据上文实验所选出的算法 2 和与其它同类方法在生成规模上进行比较.实验中比较的对象有 AETG 系统、PAIRTEST 系统、基于网络图组合测试工具 NetWork [13]， 基于解空间树的组合测试工具[13] 、CE（交叉熵）方法[26]。其中同类方法数据皆来自于已公开发表的文献中。

表 5.4.1 测试数据集规模比较

算法	测试数据个数			
	3^4	7^8	8^9	11^{10}
AETG [13]	9	80	116	207
PAIRWISE [13]	9	85	132	236
NetWork	9	49	140	121
PSST [13]	9	49	64	121
CE [26]	9	74	106	210
算法 2	12	123	179	369

根据上两种实验中选出的算法 2 与相关的其他常见演化算法在时间上进行比较，采用 SA（模拟退火)方法、GA（遗传算法)方法、ACA（蚁群算法)、CE（交叉熵）方法在时间上给与比较，其中同类方法数据皆来自于已公开发表的文献中。

表 5.4.2 运行时间比较

算法	运行时间（s）	
	10^{10}	10^{20}
SA[25]①	NA	10833
GA[25]②	886	6365
ACA[25]②	1180	7083
CE[26]③	985	4378
算法 2④	73	141

注 1：NA 表示文献中未给出相应有效的数据

注 2:①C++, Linux, INTEL Pentium IV 1.8GHz;②C, Windows XP, INTEL Pentium IV 2.26GHz;③MATLAB, Windows XP, INTEL Core(TM) 2 2.66G Hz . ④Python, Windows, Intel(R)Core(TM) i5-7200U CPU@ 2.50GHz 2.71 GHz

由表 5.4.1，表 5.4.2 可知，算法 2 在生成最终的数据测试集时，相比于其他算法略差一些，当规模在指数级上增长时，算法 2 最终生成的测试集合数相较于其他系统生成的测试集较大。在相应测试集的生成时间方面，算法 2 速度比以上均快。一方面由于其他算法测试时系统较为陈旧，其次与算法 2 使用了 one test at a time 策略，并与进行了模块封装，算法优化也有一定的关系。因此，在生成最终所需的测试集合时，本算法也不失为一种较为优秀的方法，相比于更多的测试数据来说，极快的生成速度是本算法的最大优点。

5.5 实验总结

我们首先在可接受的一定范围内论证了该方法的可行性，正确性。随后设计了实验对算法进行相关测试，得出相应的数据并进行了保存。

基于以上的测试数据来看，算法 2 无论是在运行时间或测试覆盖矩阵的覆盖占比上面，均优于算法 1。但算法 2 也会由于数据量规模增大，迭代次数不够的原因而无法产生 100%覆盖的测试数据集合。

我们根据所得出的测试数据，结合相关代码进行分析。由于算法 2 只关注传入每个粒子所包含的k组中哪一组适应度值最大，而算法 1 在传入每个粒子时，需要计算当前一整条测试数据中所有测

试组总和的适应度值。因此在数据规模越大时，每个粒子的位置参数随之变多，算法 1 因此也需要更多地时间去计算每一个粒子的所有两两配对组合的累加适应度值，并以此为依据在迭代次数达到阈值时，输出一个在当前情况下最优秀的粒子。而算法 2 在每次计算完所有粒子当前适应度后，应用 one test at a time 策略从传入的k组中只需选出适应度最大的一个并将其添加到最终的测试集合中。因此算法 2 在策略上优于算法 1。故而直接区别的体现就是算法运行时间大小的不同。

但算法 1 可自己选择生成多少条测试用例数，所以对于一个给定规模需要生成两两组合配对测试数据的SUT来说，算法 1 可根据需求来对最终生成的测试条数进行规定，而算法 2 由于策略限制，不具有该项功能。

综上所述，算法 1 具有生成给定测试数据规模个数的能力，但需要花费更大的时间代价，适应度也会相应下降。算法 2 由于策略原因，在迭代次数足够时，总能生成 100%覆盖的两两配对组合测试数据，并且生成时间较短，但无法规定相应的测试数据规模。

6 结论

本文提出了以仿生学为基础的粒子群优化算法的两两组合测试配对数据生成方法，给出了相应两种方法的算法伪代码，对算法的可行性和正确性在一定规模下进行了论证。进行了在一定量数据规模下的实验，用实验数据和图表分析了相关算法的不足之处和优点，指出了当前工作的完成度。使两两配对测试集生成问题在一定程度上得到解决。

关于组合测试的研究虽然至今已有相当丰富的成果，但依旧还存在许多问题：N 维度（$N \geq 2$）下组合测试覆盖矩阵以及多维度的组合覆盖矩阵构造由于是 NP-hard 问题，至今还没有很好的解决办法，现有的方法中大多数都是以启发式算法为基本方法的近似优化算法。除此之外，算法的性能往往无法满足需求，运行时间，覆盖程度等多数因素在维度高时难以解决。而此方面的组合覆盖研究相对较少。在很多情况下为了提高软件的可靠性和适用性，需要采取$N>2$维度的组合覆盖测试进行相关研究测试，因此多维组合覆盖测试还需要进一步进行相关研究。

参考文献

[1] Dunietz I S, Ehrlich W K, Szablak B D, Mallows C L, Lannino A. Applying design of experiments to software testing: Experience report// Proceedings of the 19th International Conference on Software Engineering. Boston, Massachusetts, USA, 1997 :205-215.

[2] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing// Proceedings of the 27th NASA/ IEEE Software Engineering Workshop. NASA Goddard Space Flight Center, 2002 :91-95.

[3] Lei Y, Tai K C. In Parameter Oder: A test generation strategy for pairwise testing. Department of Computer Science, North Carolina State University, Raleigh, North Carolina: Technical Report T R-2001-03, 2001.

[4] Mandl R. Orthogonal Lat in squares: An application of experimental design in compiler testing. Communications of the ACM, 1985, 28(10) :1054-1058.

[5] Kobayashi N, Tsuchiya T, Kikuno T. A new method for constructing pair-wise covering designs for software testing. Information Processing Letters, 2002, 81(2) :85-91.

[6] Williams A W. Software component interaction testing: Coverage measurement and generation of configurations [Ph. D. dissertation]. Ottawa-Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, Canada, 2002.

[7] Cohen D M, Dalai S R, Fredman M L, Patton G C. The AETG system: An app roach to testing based on combinatorial design. IEEE Transactions on Software Engineering,1997, 23(7) :437-444.

[8] Tung Y W, Aldi wan W S. Automating test case generation for the new generation mission software system// Proceedings of the IEEE Aerospace Conference. Big Sky, M T, USA,2000 :431-437.

[9] Yan Jun, Zhang Jian. Backtracking algorithms and search heuristics to generate test suites f or combinatorial testing//Proceedings of the 30th Annual International Conference on Computer Software and Applications (COMPSAC06). Chicago, I L, 2006, 1 :385-394.

[10] Cheng Xian g, Gu Qing, Li Ang, Cheng Daoxu. Variable strength interaction testing with an ant colony system approach//Proceedings of the 16th Asia-Pacific Software Engineering Conference. Penang, 2009 :160-167.

[11]聂长海,徐宝文.基于接口参数的黑箱测试用例自动生成算法[J].计算机学报,2004(03):382-388.

[12] Nie Chang-Hai, Xu Bao-Wen, Shi Liang, Dong Guo-Wei. Automatic test generation for N-way combinatorial testing//Proceedings of the 2nd International Workshop on Software Quality (SOQUA 2005). Fair and Convention Center, Erfurt, Germany,2005:203-211.

[13]史亮,聂长海,徐宝文.基于解空间树的组合测试数据生成[J].计算机学报,2006(06):849-857.

[14]聂长海,徐宝文,史亮.一种新的二水平多因素系统两两组合覆盖测试数据生成算法[J].计算机学报，2006（06）:841-848.

[15]王子元,聂长海,徐宝文,史亮.相邻因素组合测试用例集的最优生成方法[J].计算机学报，2007(02)：200-211.

[16] Kennedy J , Eberhart R. Particle swarm optimization[C]// Proceedings of the IEEE Conference on Neural Networks. Perth, Australia, 1995, 4 :1942-1948.

[17] Kennedy J , Eberhart R. Particle swarm optimization[C]// Proceedings of the IEEE Conference on Neural Networks. Perth, Australia, 1995, 4 :1942-1948.

[18] Kyler M , Kennedy J .The particle swarm-Explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computer, 2002, 6(1) :58-73.

[19] 陈翔，顾庆，王新平，等. 组合测试研究进展 [J]. 计算机科学，2001，37(3): 1-5.

[20] 王子元，徐宝文，聂长海. 组合测试用例生成技术 [J]. 计算机科学与探索，2008，2(6):571-588.

[21] 严俊，张健. 组合测试:原理与方法 [J]. 软件学报，2009，20(6):1393-1405.

[22]王建峰，孙超，姜守达.基于粒子群优化的组合测试数据生成算法[J].哈尔滨工程大学学报，2013，34(4)：477-482.

[23] HOVLAND H J. Three-dimensional slope stability analysis method [J] .Journal of the Geotechnical Engineering Division，1977，103(9) : 971－986.

[24] IEEE International Conference on Computer Science and Automation Engineering - PWiseGen: Generating test cases for pairwise testing using genetic algorithms., (), 747752.doi: 10.1109/CSAE.2011.5952610.

[25] Shiba T , Tsuchiya T,Kikuno T .Using artificial life techniques to generate testcases for combinatorial testing//Proceedings of the 28th Annual International Computer Software and Application s Conference (COMPSAC’04). Hong Kong, China, 2004, 1 :72-77.

[26]查日军，张德平，聂长海，等. 组合测试数据生成的交叉熵与粒子群算法及比较[J].计算机学报，2010，33(10): 1896-1908.