基于 Spark 的并行化组合测试用例集生成方法

戚荣志[1)] 王志坚[1)] 黄宜华[2)] 李水艳[3)]

[1)](河海大学计算机与信息学院 南京 211106)

[2)](南京大学计算机软件新技术国家重点实验室 南京 210046)

[3)](河海大学理学院 南京 211106)

摘 要 软件系统的正常运行受很多因素影响,各种因素及其相互作用可能引发软件故障,需要设计测试用例检测这些故障.如果因素数量较多且取值情况较复杂,则所需测试用例的数量将非常庞大.如何设计规模较小的用例集是测试用例生成研究的一个关键问题.组合测试能够从待测软件的大规模组合空间中,生成小规模的用例集,实现对各因素取值组合的充分覆盖.已有研究表明,组合测试的最小测试用例集生成问题是一个 NP 完全问题.目前已有一些研究尝试使用启发式搜索算法生成尽可能小的用例集.启发式搜索算法将组合测试用例集生成问题转化为搜索问题,并使用元启发式算法生成用例集.启发式搜索算法通常能够生成较小规模的用例集,但需要较长的计算时间.为了解决这个问题,文中提出了一种基于 Spark 的岛模型并行化遗传算法,利用 Hadoop 分布式文件系统实现了 Spark 运行节点间交换信息的方法,进而实现个体在子种群间的迁移.该算法首先从初始种群创建 Spark 的弹性分布式数据集;然后,将该数据集划分为多个子种群分布到集群的多个节点中;接着,各个子种群在各自的节点上计算适应度函数值和独立进化,并每隔一定的进化代数选择一些个体在各个子种群间迁移,提高了种群的多样性以及搜索最优解的有效性和性能;最后,算法返回满足覆盖准则的最优测试用例集.这种基于 Spark 的并行化遗传算法是大规模并行化在组合测试用例集生成方面的一个有效尝试.在实验部分,首先对文中提出的并行化算法进行系统的参数调整,给出适合组合测试用例集生成的推荐参数配置;接着将文中所提算法与串行遗传算法和独立运行遗传算法进行比较.实验结果表明,文中所提算法在生成用例集规模和消耗时间上均显著优于这两个算法.在运行所选实例时,该算法比串行算法加速约 4 至 30 倍,比独立运行遗传算法加速约 2 至 3 倍.相对于已有的组合测试用例集生成方法,文中所提算法在生成用例集规模上也具备显著优势.

关键词 组合测试;测试用例集生成;并行化遗传算法;岛模型;Spark

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2018.01284

Generating Combinatorial Test Suite with Spark Based Parallel Approach

QI Rong-Zhi[1)] WANG Zhi-Jian[1)] HUANG Yi-Hua[2)] LI Shui-Yan[3)]

[1)](*College of Computer and Information, Hohai University, Nanjing* 211106)

[2)](*National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046)

[3)](*College of Science, Hohai University, Nanjing* 211106)

Abstract The normal operation of computer system is influenced by many factors. Failures may be triggered by various factors and interactions of these factors in software. In order to detect these interaction failures, the researchers try to design suitable test cases. If the quantity of factors is larger and the values of these factors are more complex, the number of test cases may become very huge. How to design a small number of test cases, which can achieve full coverage of combinations of factor values of the software, is a key issue of the research of test case generation.

收稿日期:2015-12-07;在线出版日期:2016-08-29.本课题得到国家重点研发计划(2016YFC0400910)、国家科技支撑计划(2013BAB06B04)和中央高校基本科研业务费项目(2015B22214,2013B07514)资助.**戚荣志**,男,1980 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为基于搜索的组合测试.E-mail:rzqi@hhu.edu.cn.**王志坚**,男,1958 年生,博士,教授,主要研究领域为领域软件工程和网络安全.**黄宜华**,男,1962 年生,博士,教授,主要研究领域为大数据并行处理技术和并行计算.**李水艳**,女,1980 年生,博士研究生,主要研究方向为进化计算和数据挖掘

Combinatorial testing can generate small test suite from large scale combinatorial space of the software under test. The generated test suite can cover combinations of factor values of the software fully. It has been proven that generating minimum test suite is an NP-complete problem, so, heuristic search algorithms have been used for generating smaller test suite by many researchers in recent years. Heuristic search algorithms formulate combinatorial test generation problem as a search problem, and apply metaheuristic algorithms to generate combinatorial test suite. Heuristic search algorithms can often produce smaller test suite than other approaches, but require a longer computation. To solve this problem, in this paper, we propose a Spark based parallel genetic algorithm based on the island model. This algorithm uses Hadoop distributed file system to implement the method of exchanging information among the running nodes of Spark, and implement the migration of individuals among various subpopulations. The Spark resilient distributed dataset is first generated from initial population. Then the resilient distributed dataset is split into several subpopulations. These subpopulations are then distributed into the nodes of the Spark cluster. Each subpopulation calculates the fitness function on its own node, and evolves separately. During the process of evolution, some individuals will be selected according to the migration strategy every other generation. Then these selected individuals will migrate among various subpopulations on the nodes of the Spark cluster. The migration of individuals improves the diversity of the whole population, and improves the effectiveness and performance on searching the optimal results. Finally, the algorithm returns the optimal combinatorial test suite that satisfies the desired coverage criterion. The Spark based parallel genetic algorithm is an effective attempt in applying massive parallelization to combinatorial test suite generation. In the experimental section, parameter tuning is first conducted systematically to find recommended settings for combinatorial test suite generation. Then the proposed parallel genetic algorithm is evaluated against the sequential genetic algorithm and the isolated running genetic algorithm using 20 benchmarks. Experiment results show that the proposed parallel genetic algorithm outperforms the sequential genetic algorithm and the isolated running genetic algorithm in both generated test suite sizes and computational effort. It can achieve about 4× to 30× speedup than the sequential genetic algorithm, and about 2× to 3× than the isolated running genetic algorithm for the selected benchmarks. The proposed parallel genetic algorithm also has significant advantages on the sizes of generated test suite when compared with other existing approaches.

Keywords　combinatorial testing; test suite generation; parallel genetic algorithm; island model; Spark

1　引　言

软件测试是人们为了发现软件错误而运行软件系统的活动.在一些测试场景中,许多影响软件系统正常运行的故障是由系统的多个因素(功能选项和输入参数等)间的相互作用引起的,这类故障被称为交互故障[1].为了确保软件在这些因素的相互作用下正常运行,需要设计相应的测试用例来检测出这些交互故障.同时,如今的软件系统越来越复杂,可能有多个参数,每个参数又可能有多种取值,因此,参数取值的组合数量会变得非常大,穷尽测试需要的用例集规模也会非常大.比如,一个软件有 30 个参数,每个参数有 6 种取值,穷尽测试这些参数间的交互需要进行 6^{30} 次,在有限的测试时间和测试成本下,这几乎是不可能实现的.因此,有必要设计一种生成尽可能小规模测试用例集的方法,同时该方法能够覆盖尽可能多的因素交互组合.组合测试就是这样一种方法,该方法可以从待测软件面临的庞大组合空间中,选取少量的测试用例,高效、科学和系统地检测软件系统中各种因素相互作用可能促发的故障[2].Kuhn 等人[1,3]通过对一些具体软件系统中

各种错误的分析,给出了组合测试的交互准则:绝大多数的软件故障是由单个因素或者两个因素的交互引起的,随着交互的因素数量的增加(通常不超过 6),所发现的软件故障越来越少.因此,对于 t-way($2\leq t\leq 6$)组合测试用例(集)生成方法的研究一直是组合测试领域的研究热点.

由于组合测试的最小测试用例集生成问题是一个 NP 完全问题[4],因此,人们研究了多种方法来生成尽可能小规模的用例集.Nie 和 Leung[5] 给出了四类主要的方法:贪心算法、启发式搜索算法、数学方法和随机方法.其中,贪心算法是使用最广泛的组合测试用例生成方法.与启发式搜索算法相比,贪心算法通常速度比较快,但生成的测试用例集规模较大[6].而启发式搜索算法将组合测试用例生成问题转化为搜索问题,并使用模拟退火(Simulated Annealing,SA)、蚁群算法(Ant Colony Algorithm,ACA)、粒子群优化算法(Particle Swarm Optimization,PSO)和遗传算法(Genetic Algorithm,GA)等元启发式算法来生成组合测试用例.与贪心算法相比,启发式搜索算法通常能够生成更小的测试用例集,但是需要更长的计算时间[5].梁亚澜等人[7] 选取了多个实例,通过实验详细分析了遗传算法生成覆盖表的性能.其中一个实例有 11 个参数,每个参数有 10 种取值,生成大小为 205 的覆盖表耗时约 3 小时.

为了解决上述启发式算法带来的计算性能问题,本文提出了一种基于 Spark[8] 的并行化遗传算法,用并行化方法快速生成组合测试用例集.研究表明,并行化是提高遗传算法性能的一种有效方法[9].遗传算法用于生成组合测试用例集时,需要解决的关键问题是如何在尽量短的时间内找到规模尽可能小的测试用例集.遗传算法中,个体表示测试用例集,多个个体组成一个种群.种群规模越大代表问题的搜索空间越大,找到最小规模测试用例集的可能性就更大,但运行时间越长.Spark 可以将大规模的种群分割为多个小规模的子种群,分发到集群中的多个节点上并行进化,在扩大搜索空间的同时提高了算法的运行速度.

Spark 的这种运算方式非常适合并行化遗传算法的实现,因此,本文使用 Spark 研究实现并行化遗传算法,并用于生成 t-way($t=2,3$)组合测试用例集.据我们所知,使用基于 Spark 的并行化遗传算法生成组合测试用例集,这方面目前在公开文献中尚未见到相关的研究报道,因此,基于 Spark 大数据并行计算技术和平台,本文所进行的并行化组合测试用例集生成方法研究,是在软件测试领域的一个全新的尝试.

本文所提并行化组合测试用例集生成方法的基本思路是,首先从由测试用例集构成的种群创建 Spark 的弹性分布式数据集(Resilient Distributed Dataset,RDD),并将 RDD 划分为多个分区分布到集群的多个节点中,每个分区对应一个子种群;然后各个子种群在各自的节点上进行适应度函数值的计算和独立进化,每隔一定的代数选择一些个体在各个子种群间进行迁移,整个进化过程一直持续到发现了满足覆盖准则的测试用例集或者达到最大进化代数为止.

本文的主要研究工作和贡献点如下:

(1)针对遗传算法在组合测试用例集生成方面所存在的计算性能问题,本文基于 Spark 提出了运行节点间交换信息的方法.

(2)在此基础上,提出了基于 Spark 的岛模型并行化遗传算法,用于生成组合测试用例集.

(3)实现了基于 Spark 的岛模型并行化遗传算法,通过实验对该算法进行参数调整,并将该算法与串行遗传算法、独立运行遗传算法以及其它组合测试用例生成方法在性能和有效性方面进行了比较和分析.

本文第 2 节介绍基础背景知识;第 3 节描述岛模型并行化遗传算法;第 4 节给出实验及结果分析;第 5 节介绍相关工作;第 6 节是总结和展望.

2 基础背景

2.1 覆盖表

本节沿用聂长海[2] 以及 Nie 和 Leung[5] 中的相关定义给出测试用例和覆盖表的定义,再给出一个覆盖表的例子.

定义 1. 假设待测软件系统(Software Under Test,SUT)有 k 个参数 $c_i(1\leq i\leq k)$,每个参数有 v_i 个取值,形成集合 V_i.k 元组 $(v_1,v_2,\cdots,v_k)$ 称为 SUT 的一个测试用例($v_i\in V_i$).

由所有可用的测试用例组成的集合称为测试用例集,组合测试方法使用覆盖表作为测试用例集.

定义 2. 覆盖表 $CA(N;t,k,v)$ 是一个 $N\times k$ 数组,满足如下两个属性:

(1)每一列 $i(1\leq i\leq k)$ 仅包含集合 V_i 中的元素 $v_i=|V_i|$;

(2)任意 t 个参数形成的子数组 $N\times t$ 至少覆盖

一次该 t 个参数所有取值的可能组合.

其中,覆盖表的每一行对应一条测试用例,N 是测试用例的个数,t 为组合测试的强度,当 $t=2$ 时,它称为 2-way 覆盖表. 覆盖表 $CA(N;t,k,v)$ 也可表示为:$CA(N;t,v^k)$. 当一个覆盖表包含最小可能的行数时,它就是最优的,它的规模被定义为 $CAN(t,k,v)$.

覆盖表假设所有参数的取值数目都相同,然而,软件系统通常包含取值数目不同的参数,于是人们提出了混合覆盖表 $MCA(N;t,k,v_1v_2\cdots v_k)$,其中,$v_1v_2\cdots v_k$ 为每个参数 $c_i(1\leq i\leq k)$ 的取值数目. 当 $v_1=v_2=\cdots=v_k=v$ 时,混合覆盖表就转变为覆盖表.

例如,假设 SUT 有 3 个参数 p_1、p_2、p_3,每个参数的取值分别为 $\{a_0,a_1\}$、$\{b_0,b_1\}$、$\{c_0,c_1\}$,参数值的所有可能组合数目为 2^3:(a_0,b_0,c_0)、(a_0,b_0,c_1)、(a_0,b_1,c_0)、(a_0,b_1,c_1)、(a_1,b_0,c_0)、(a_1,b_0,c_1)、(a_1,b_1,c_0)、(a_1,b_1,c_1). 当进行两两组合测试时,所生成的测试用例将覆盖所有参数值的两两组合,共有 12 个两两组合:(a_0,b_0)、(a_0,b_1)、(a_0,c_0)、(a_0,c_1)、(a_1,b_0)、(a_1,b_1)、(a_1,c_0)、(a_1,c_1)、(b_0,c_0)、(b_0,c_1)、(b_1,c_0)、(b_1,c_1). 在这个例子中,下面 4 个测试用例集可以覆盖这 12 个两两组合:(a_0,b_0,c_1)、(a_0,b_1,c_0)、(a_1,b_0,c_0)、(a_1,b_1,c_1). 表 1 给出了该 SUT 的 2-way 覆盖表 $CA(4;2,3,2)$.

表 1　$CA(4;2,3,2)$

p_1	p_2	p_3
a_0	b_0	c_1
a_0	b_1	c_0
a_1	b_0	c_0
a_1	b_1	c_1

从上面这个例子可以看出,2-way 组合测试能够将测试用例数量从 8 个减少为 4 个,即有了 50% 的降低. 随着问题规模的增加,测试用例数量的减少幅度会更大.

2.2　Spark

Spark 是一种快速且通用的并行计算框架. Spark 基于内存的计算模型和有向无环图(Directed Acyclic Graph,DAG)的执行模型能够加快机器学习和元启发式搜索等迭代式算法的运行速度.

Spark 的核心是一种称为 RDD 的分布式内存抽象[8,10]. RDD 是只读的记录分区的集合,Spark 创建 RDD 后,并对 RDD 进行并行切片,然后分发到集群中的多个节点上完成相应的变换(transformation)操作,最后由行动(action)操作触发所有的运算. Spark 通过血统关系图(lineage graph)记录下不同 RDD 之间相互转换的依赖关系,以便恢复丢失的数据.

Spark 根据内存中的数据或外部数据来创建 RDD,*parallelize*()函数将内存中的数据集转换为 RDD,*textFile*()函数将外部数据集转换为 RDD. RDD 支持变换和行动两种类型的操作,其中,变换用于对数据进行操作并将 RDD 转换为新的 RDD,行动操作触发 Spark 提交作业. 变换操作和行动操作的主要函数说明分别如表 2、表 3 所示.

表 2　变换操作函数

函数名称	函数功能
map(f)	RDD 中的每个元素通过自定义的函数 f 转换为一个新的元素
mapPartitions(f)	获取每个分区的迭代器,并通过迭代器对整个分区的元素进行转换
mapPartitionsWithIndex(f)	功能同 *mapPartitions* 函数,增加了每个分区的索引

表 3　行动操作函数

函数名称	函数功能
collect()	返回分布式 RDD,并将其转换为一个单机数组
foreach(f)	对 RDD 中的每个元素应用函数 f,不返回 RDD

2.3　岛模型

并行化遗传算法主要包括主从模型、分布式模型和细胞模型等 3 种模型[9,11].

主从模型将适应度函数分布到从节点上进行计算,主节点负责遗传算法的进化过程,整个遗传算法还是针对一个种群进行进化操作,因此,当种群规模较大时,运算时间较长.

针对这种问题,分布式模型将整个种群切分为相互独立的多个子种群,这些子种群在集群的多个节点上独立执行进化操作. 根据各个子种群间是否存在个体的迁移,分布式模型又可分为独立运行模型和岛模型. 其中,独立运行模型是多个遗传算法同时独立地运行在集群的多个节点上,相互之间没有个体的交换. 我们在以往的工作中[12]研究实现了基于 Spark 的独立运行模型,并用于两两组合测试用例集的生成. 岛模型在各个子种群独立进化的过程中,每隔一定的进化代数选择一些个体在子种群间进行迁移,实现了子种群间的信息交换,增强了种群的多样性,从而带来了遗传算法性能上的提升[13]. 岛模型与独立运行模型相比,所要解决的特有问题是如何实现个体在各个子种群间的迁移. 本文借助于 Hadoop 分布式文件系统(HDFS)首次实现了

Spark 运行节点间交换信息的方法，进而实现个体在 Spark 集群中各个子种群间的迁移.

细胞模型中一个个体即为一个子种群，每个子种群只与其直接相连的邻居子种群交换信息，会带来子种群间过于频繁的通信开销.

综上，我们选择岛模型遗传算法用于生成组合测试用例集. 岛模型的一个重要特性是岛间的个体迁移，好的迁移策略不仅可以将较好的个体在岛间传递，而且还能够维持分割子种群所带来的种群多样性. 在岛模型的迁移策略中，以下 4 个参数对算法有着重要的影响：迁移间隔(Migration Interval)、迁移个体的数量(Migration Size)、迁移个体的选择和替换方法(Selection/Replacement of Migrants)和迁移拓扑结构(Migration Topology)[13-15].

(1) 迁移间隔是子种群在进化过程中两次成功个体迁移之间的进化代数. 过分频繁的迁移会导致较高的选择压，从而使算法易于陷入局部最优[14-16]. 同时，频繁的迁移会给整个算法带来额外的时间开销.

(2) 迁移个体的数量由子种群的大小确定[14]，一般设定为所占子种群大小的百分比. 如果迁移的个体数量很少，则迁移带来的影响就可以忽略不计，反之，太多的迁移个体就会快速占据目标子种群，导致全局多样性的降低.

(3) 个体在不同子种群间的迁移，涉及到如何选择个体，以及如何替换目标子种群上的个体. 选择个体的方法包括选择最好的个体和随机选择个体，替换个体的方法包括替换最差的个体和随机替换个体[13-14]. 因此，共有 4 种迁移个体的选择和替换方法：选择最好的个体，替换最差的个体；随机选择个体，替换最差的个体；选择最好的个体，随机替换个体；随机选择个体，随机替换个体.

(4) 迁移拓扑结构是岛间的组织结构，常用的迁移拓扑结构包括环形、圆环面以及随机结构等. 拓扑结构可以用图来表示，图中的节点表示子种群，节点间的箭头表示个体迁移的方向，图中任意两个节点之间的最大距离称为直径. Tomassini[16] 指出环形的直径为 $O(n)$，圆环面的直径为 $O(\sqrt{n})$，随机结构的直径为 $O(\log(n))$.

3 岛模型并行化遗传算法

本节我们给出岛模型并行化策略，提出 Spark 运行节点间交换信息的方法，设计基于 Spark 的岛模型迁移机制和迁移策略，并在此基础之上提出用于 t-way 组合测试用例集生成的岛模型并行化遗传算法(Island Parallel Genetic Algorithm based on Spark，IPGAS).

3.1 并行化策略

IPGAS 的并行化策略为：首先从初始种群创建 Spark 的 RDD，并将 RDD 划分为多个分区分布到集群的多个节点中，每个分区对应一个子种群；然后各个子种群在各自的节点上计算适应度函数值和独立进化，每隔一定的进化代数，按照一定的策略选择一些个体在各个子种群间进行迁移，实现子种群间的信息交换.

IPGAS 的并行化过程如图 1 所示，并行化过程分为 3 个阶段：

(1) 阶段 1 随机生成由 n 个个体组成的初始种群 P：$P=\{Individual_i \mid 1\leq i\leq n\}$.

(2) 阶段 2 将初始种群 P 转换为 RDD，并将 RDD 划分为 n 个分区，每个分区由多个个体组成，比如，分区 1 包含个体 1 和个体 3，分区 2 包含个体 2 和个体 4. 划分的分区数量和每个分区包含的个体数量由 Spark 自动分配.

(3) 阶段 3 将 n 个分区分布到集群的 n 个节点

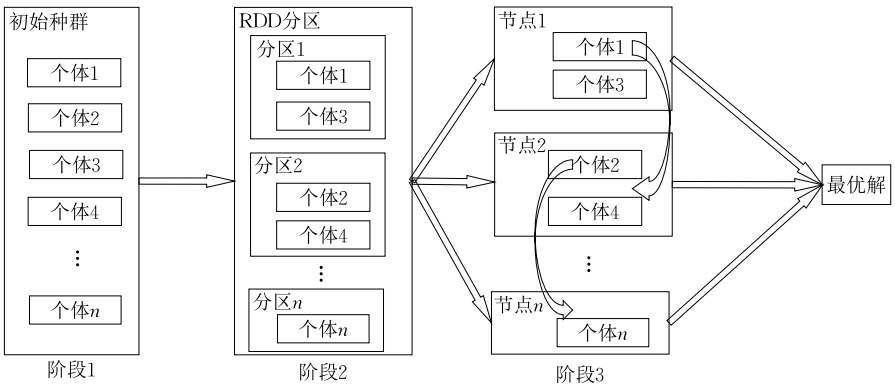

图 1 并行化过程图

上计算适应度函数值，并执行进化操作. 在进化过程中，每隔一定的进化代数(迁移间隔)，选择一些个体在节点间迁移，比如，节点 1 的个体 1 迁移到节点 2 中，替换其中的某个节点，节点 2 的个体 2 迁移到节点 n 中，替换其中的某个节点. 迁移间隔的大小、迁移的源节点和目标节点以及选择和替换个体的方式，由岛模型的参数决定.

IPGAS 根据 Spark 的计算模型，对 RDD 执行一系列的变换操作，整个 RDD 的变换过程可以用如图 2 所示的血统关系图表示. 首先，初始种群(population)通过 Spark 的 *parallelize*()算子变换为种群 RDD(populationRDD)，再通过 Spark 的 *mapPartitions*(*assessFitness*())算子将种群 RDD 转换为适应度值 RDD(fitnessRDD)，该 RDD 包含键值对〈*key*，*value*〉，其中 *key* 是一个个体，*value* 是该个体的适应度值. 函数 *assessFitness*()用于计算个体的适应度值，它被分布到集群中不同的节点上并行计算. 然后，适应度值 RDD 通过 Spark 的 *mapPartitionsWithIndex*(*evolution*())算子变换为进化 RDD(evolutionRDD)，其中函数 *evolution*()执行遗传算法的进化过程：选择、交叉、变异以及个体的迁移. 最后，*foreach*()算子触发 Spark 的运算流程，完成前面一系列的转换，并输出本次搜索的最优结果.

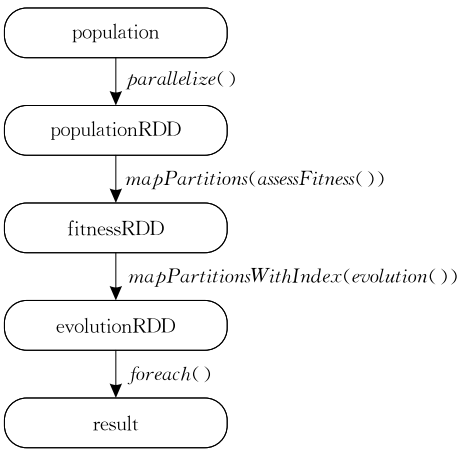

图 2　血统关系图

3.2　岛模型设计

岛模型的设计需要解决以下两个问题：迁移机制的设计和迁移策略的选择.

3.2.1　迁移机制

Spark 是一种单指令多数据流的数据并行系统，这种并行系统的特点是高效，但不够灵活. 同时，Spark 通过血统的方式构造 RDD 依赖关系，并在出错时通过回溯 RDD 依赖关系进行重算而实现容错. 如果运行过程中的任务间能够随意地通信、同步，那么这个关系就会很复杂，比较难以维护. 因此，标准的 Spark 不支持集群中各个节点间的信息交换，不能直接进行个体的迁移. 本文在 Spark 上研究实现了岛模型的迁移机制：(1) 在某个节点上搜索到最优解后停止 Spark 的执行过程；(2) 节点间的信息交换.

(1)Spark 通过驱动程序(Driver)控制整个应用的流程. Driver 对数据集进行分块，以任务的形式分发到集群中的各个节点上并发执行，各个任务执行完成后，由 Driver 收回结果. 在使用遗传算法生成组合测试用例集的应用场景中，初始种群分割为多个子种群，分发到集群中的各个节点上计算适应度函数值和执行进化操作. 进化操作的终止条件为搜索到满足一定覆盖准则的最小规模的测试用例集，或者是达到了预先设定的最大进化代数. 由于各个子种群在不同的节点上进化的速度不一样，可能出现某个节点因为搜索到了最优测试用例集而停止进化，但是，其它节点还没有到达终止条件，需要继续进化. 这种情况下，标准 Spark 的执行流程需要等到所有的节点都停止进化才输出结果.

针对这个问题，本文借助于 HDFS 实现当某个节点搜索到最优测试用例集后停止 Spark 的执行过程：对于每个任务，当其搜索到最优测试用例集后，就将结果输出到 HDFS 中，同时创建一个空文件 *_success*. 另一方面，Driver 在触发各节点并发计算之后，不断地监测 HDFS 中是否存在 *_success* 文件，如果发现该文件，则直接停止 Spark 的执行.

(2)对于节点间的信息交换，本文借助于 HDFS 实现个体在各个子种群间的迁移. 在进化过程中，子种群每隔一定的进化代数选择个体进行迁移. 这里我们引入迁移轮数的概念，所谓迁移轮数是指在整个进化过程中迁移发生的次数，可以由最大进化代数除以迁移间隔得到. 比如，最大进化代数为 10 000，迁移间隔为 100，则迁移轮数为 100 轮.

子种群间个体迁移的过程如图 3 所示. 依据 3.1 节给出的并行化策略，初始种群被分割为 n 个子种群，并分布到集群中的 n 个节点上并行进化，当进入每一轮迁移时，首先将 n 个节点中的 n 个子种群写入 HDFS 中对应每一轮目录下的 n 个文件中；再将这 n 个文件从 HDFS 读出到某个节点中(比如图中的节点 n)；然后按照一定的迁移策略在各个子

种群间迁移个体;接着将各个子种群再次分发到 n 个节点中进行下一轮进化,再进入下一轮迁移,整个过程持续到算法满足终止条件为止.

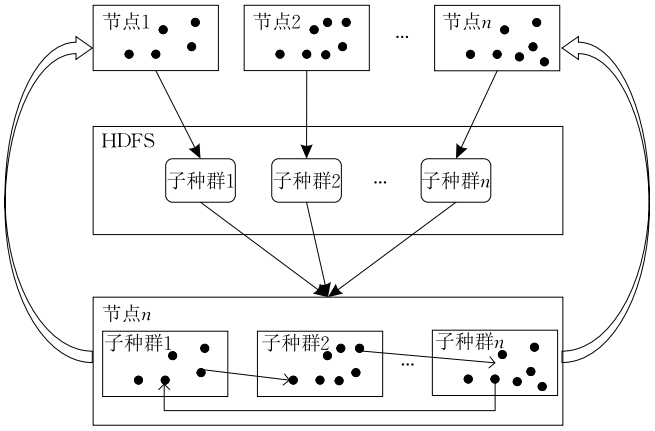

图 3 个体迁移过程图

该方法借助于共享的 HDFS 实现个体的迁移,涉及到子种群的读写操作,中间数据量的大小主要取决于子种群规模的大小和组合测试用例集生成问题的规模.本文主要目标是生成 t-way(t=2,3)组合测试用例集,中间数据量不是很大,因此,该方法对整个算法的性能影响不大.如果组合测试用例集生成问题的规模变大,比如 t=4,5,6 时,中间读写的数据量会增加,为了不影响算法的性能,需要借助于其它方法实现个体的迁移,比如使用分布式的基于内存的文件系统代替 HDFS,或者修改 Spark 的通信机制部分的源码,这些将是我们今后的研究工作.

3.2.2 迁移策略

基于岛模型求解 t-way 组合测试用例集生成问题时,需要确定迁移策略.迁移策略主要包括以下几个参数:迁移间隔、迁移个体的数量、迁移个体的选择和替换方法以及迁移拓扑结构.Skolicki 在他的博士论文中[13]对岛模型进行了详细的分析,并通过实验给出了一个典型的参数配置:多个小规模的子种群、较长的迁移间隔、少量的迁移个体和随机交换个体.但是,该参数配置主要用于求解连续组合优化问题,而 t-way 组合测试用例集生成问题是一个离散优化问题,因此,该参数配置并不一定适用.为了得到合适的参数配置,我们使用 Skolicki 给出的典型参数配置作为基本参数配置,通过 4.2 节的参数调整实验,给出 IPGAS 的推荐参数配置.

3.3 并行化遗传算法描述

IPGAS 的并行化算法如算法 1 所示.算法 1 的输入包括参数-值文本文件、参数个数、每个参数的取值、测试用例集大小和种群大小,输出是生成的测试用例集.首先计算需要覆盖的所有参数值的 t 元组的个数 AT(Line1),如果是两两组合测试,则计算所有二元组的个数;如果是三三组合测试,则计算所有三元组的个数.函数 *initializePop*()随机生成初始种群 *population*(Line2),该种群包含 n 个个体,每个个体由 m 个测试用例组成.然后,算法进入如图 2 所示的一系列 RDD 变换过程(Lines3~6),在此过程中并发执行遗传算法的进化过程.其中,进化过程使用 Spark 的 *mapPartitionsWithIndex*()算子,将每个岛(Spark 的分区)的索引值传入进化过程,这样,在并发进化过程中就能够知道每个岛上运行的是哪个子种群,便于个体在各个子种群间迁移机制的实现(Line5).*foreach*()算法只负责触发 Spark 的运算过程,不回收所有岛上所有个体进化的结果,这样可以减少通信开销(Line6).在触发 Spark 的运算后,算法开始监测 HDFS 中是否存在搜索成功的标志文件*_success*,如存在,则停止 Spark 的运行,并返回生成的测试用例集(Lines7~9),这里的标志文件*_success* 将在每个工作节点进化过程中找到最优解时创建.

算法 1. 岛模型并行化遗传算法(IPGAS).

输入:参数-值文本文件 *pv.txt*,参数个数 k,每个参数的取值 v_i,测试用例集大小 m,种群大小 n

输出:测试用例集

```
1. AT ← numOfAllTuples(k, v_i)
2. population ← initializePop(m, n, "pv.txt")
3. populationRDD ← parallelize(population)
4. fitnessRDD ←
       populationRDD.mapPartitions(assessFitness())
5. evolutionRDD ←
       fitnessRDD.mapPartitionsWithIndex(evolution())
6. evolutionRDD.foreach()
7. IF fileExists("_success")
8.     RETURN getFile("TestSuite")
9. END IF
```

在算法 1 中,两个函数传入 Spark 的驱动程序,并被分发到集群中并发执行:*assessFitness*()和 *evolution*()(Lines4~5).其中,*assessFitness*()在每个节点上并行计算每个个体的适应度函数值,适应度函数为一个染色体中所有测试用例所覆盖的不同的 t 元组的总数.*evolution*()在每个节点上并发执行进化操作,进化操作包括选择、交叉和变异等基本遗传操作,以及个体的迁移.带迁移的进化过程如算法 2 所示.

算法 2.　带迁移的进化过程.

输入：子种群 P，最大进化代数 $maxGeneration$，子种群大小 n，迁移间隔 $migrationInterval$

输出：最优测试用例集

```
1.  it←1
2.  WHILE (it≤maxGeneration && !solutionFound())
3.     FOR n/2 times
4.        Parent P1←Selection(P)
5.        Parent P2←Selection(P)
6.        Children C1,C2←Crossover(Copy(P1),
                              Copy(P2))
7.        Mutate(C1,C2)
8.        ReplaceWorstIndividual(P)
9.     END FOR
10.    Mutate(bestIndividual)
11.    IF (it % migrationInterval==0)
12.       Migrate(P)
13.    END IF
14.    it←it+1
15. END WHILE
16. IF (solutionFound())
17.       createFile("_success")
18.       createFile(bestIndividual)
19. END IF
```

算法 2 的输入包括子种群、最大进化代数、子种群大小和迁移间隔，输出是最优测试用例集. WHILE 循环(Lines2～15)是完整的带迁移的进化过程，其终止条件为进化到了最大代数或搜索到了最优测试用例集. 在 WHILE 循环的每次迭代过程中，首先，算法进入 FOR 循环执行选择、交叉、变异和替换等遗传操作(Lines3～9). FOR 循环结束后，算法接着对最优个体进行变异(Line10). 当到达迁移间隔时，算法进行个体的迁移(Lines11～13). WHILE 循环结束后，如果算法搜索到了最优测试用例集，则在 HDFS 中创建成功标志文件 *_success*，以及最优测试用例集(Lines16～19)，算法 1 监测到 *_success* 存在时就停止 Spark 的运行，返回最优测试用例集.

算法 2 中的 *Migrate*()函数使用 3.2.1 节中的迁移机制进行个体的迁移，迁移过程如算法 3 所示.

算法 3.　个体迁移过程.

输入：子种群 P

输出：完成迁移后的子种群集合

```
1. createFile(P)
2. populationList←loadFile(partitionIndex)
3. SelectMigrateStrategy()
4. MigrateStrategy.migrate(populationList)
5. RETURN populationList
```

算法 3 的输入为子种群，输出为完成迁移后的子种群的集合. 首先为每个子种群在 HDFS 中分别创建一个文件(Line1)，假定有 n 个子种群并发运行在 Spark 的 n 个工作节点上，则创建 n 个文件. 然后，根据算法 1 中的 $mapPartitionsWithIndex(evolution())$ 函数传入 Spark 中的分区索引值，将 n 个文件加载到该索引值对应的分区上(Line2). 然后算法选择迁移策略(Line3)，包括迁移拓扑结构和迁移个体的选择和替换方法，其中，拓扑结构为环形和随机结构. 选择和替换方法包括选择最好的个体，替换最差的个体；随机选择个体，替换最差的个体；选择最好的个体，随机替换个体；随机选择个体，随机替换个体. 然后，算法进行所选个体的迁移(Line4)，为了保持子种群规模不变，只迁移所选个体的拷贝，个体本身还保留在原子种群上. 最后，算法将迁移后的子种群集合返回给算法 2(Line5).

4　实　验

为了检验本文所提出方法的效果，我们基于第 3 节的算法设计实现了 IPGAS，用以生成 t-way (t=2,3)组合测试用例集. 本节我们将通过实验来分析 IPGAS 的性能和有效性，首先给出相关实验设计；接着探讨 IPGAS 用于 t-way 组合测试用例集生成的优化的参数设置；然后设计两组实验评价 IPGAS：与我们以前研究工作中提出的并行化遗传算法(Parallel Genetic Algorithm based on Spark，PGAS)[12] 和串行遗传算法(Sequential Genetic Algorithm，SGA)之间的比较，与其它已有方法的比较；最后分析实验结果.

4.1　实验设计

我们使用 Java，基于 Spark 实现了 IPGAS. PGAS 和 IPGAS 运行在一个由 9 个节点组成的集群上，其中 1 个节点为 Hadoop 的 Namenode，其余 8 个节点为 Datanodes. 每个节点的硬件环境为：1 个 2.66GHz i5 750 Quad-Core 处理器和 8GB 内存，软件环境为：Ubuntu 14.10、Java 1.7、Hadoop 2.4 和 Spark 1.3.0. SGA 运行在集群中的一个节点上.

在实验中，我们从文献[17-18]中选取了 10 个 2-way 和 5 个 3-way 的合成实例，以及 5 个 2-way 真实软件实例，分别如表 4 和表 5 所示，表中 $4^1 3^{39} 2^{35}$ 表示该实例有 1+39+35=75 个参数，其中 1 个参数有 4 个取值，39 个参数有 3 个取值，35 个参数有 2 个取值.

表 4 合成实例

编号	模型	编号	模型
S2-1	3^4	S3-1	3^6
S2-2	$5^1 3^8 2^2$	S3-2	4^6
S2-3	3^{13}	S3-3	$3^2 4^2 5^2$
S2-4	$4^1 3^{39} 2^{35}$	S3-4	5^6
S2-5	$5^1 4^4 3^{11} 2^5$	S3-5	5^7
S2-6	$4^{15} 3^{17} 2^{29}$		
S2-7	$6^1 5^1 4^6 3^8 2^3$		
S2-8	$7^1 6^1 5^1 4^5 3^8 2^3$		
S2-9	4^{100}		
S2-10	6^{16}		

表 5 真实软件实例

名称	模型
SPIN-S	$2^{13} 4^5$
Bugzilla	$2^{49} 3^1 4^2$
SPIN-V	$2^{42} 3^2 4^{11}$
Apache	$2^{158} 3^8 4^4 5^1 6^1$
GCC	$2^{189} 3^{10}$

4.2 参数调整

用 IPGAS 生成 t-way 组合测试用例集时既要考虑基本遗传算法的参数又要考虑并行遗传算法的参数. 基本遗传算法主要有 4 个控制参数:种群规模 N、进化代数 M、交叉概率 P_c 和变异概率 P_m. IPGAS 中涉及到基本遗传算法参数配置时,我们使用 PGAS 的参数配置[12],涉及到并行遗传算法参数配置时,我们重点考虑岛模型的 5 个参数:子种群规模 L、迁移间隔 i、迁移个体的数量 α、迁移个体的选择和替换方法 P 以及迁移拓扑结构 T. 依据 3.2.2 节的迁移策略以及我们的实验经验,我们给出如表 6 所示的基本参数配置.

表 6 基本参数配置

参数名称	参数设置
子种群规模	100
迁移间隔	500
最大代数	5000
迁移个体的数量	10%
选择和替换方法	随机,迁移个体的拷贝
迁移拓扑结构	随机结构

为了研究岛模型参数对 IPGAS 性能的影响,同时给出 IPGAS 在生成 t-way 组合测试用例集时的推荐参数配置,我们在基本参数配置的基础上,依据集群规模设置子种群数量为 32,然后,依次改变每个参数的值来生成多个参数配置,在改变某个参数值的时候,其余参数的值保持不变.

在这一组实验中,我们从表 4 和表 5 中选取了 4 个实例:S2-2、S2-8、S3-1 和 SPIN-S. 为了获得统计显著性结果,我们对每个实例都执行 30 次,分别给出测试用例集规模和消耗时间的平均值.

(1) 子种群规模 L

种群规模影响算法的搜索空间,大规模种群能够提高多样性,带来较好的搜索效果,但同时也增加了搜索时长. 种群规模可以由子种群规模乘以工作节点数求得. 实验中 L 的取值集合为 10、50、100、300 和 500,分别对应 5 种不同的参数配置. 集群的最大工作节点数为 32,因此,实验中最大种群规模为 16000.

表 7 和表 8 分别给出了不同 L 下各实例生成的平均测试用例集规模和消耗时间,表 7 和表 8 以及后续表格中最小测试用例集和最短消耗时间在表中用加粗标记. 从表中可以看出,L 取 10 时,消耗时间最少,但是生成规模较大. 随着 L 从 10 增加到 100,生成规模变小,而消耗时间增多. 随着 L 从 100 增加到 500,生成规模趋于平稳,消耗时间持续增多. 因此,L 取 100 是生成较小规模用例集和消耗较少时间的一个好的选择.

表 7 不同 L 下测试用例集的规模

实例	子种群规模				
	10	50	100	300	500
S2-2	17.8	16.2	**15.0**	15.6	15.2
S2-8	48.4	44.4	**42.4**	43.0	42.6
S3-1	42.2	40.2	33.8	**33.4**	34.0
SPIN-S	20.4	19.6	17.4	17.8	**17.2**

表 8 不同 L 下测试用例集的消耗时间(单位:s)

实例	子种群规模				
	10	50	100	300	500
S2-2	**5.2**	10.2	9.6	15.0	15.6
S2-8	**6.6**	9.4	25.2	33.6	52.4
S3-1	**6.8**	11.0	28.4	39.6	57.8
SPIN-S	**7.8**	10.2	20.0	18.2	29.8

(2) 迁移间隔 i

迁移间隔是子种群在进化过程中两次成功的个体迁移之间的进化代数,过分频繁的迁移会导致较高的选择压,从而使得算法易于陷入局部最优. 实验中 i 的取值集合为 50、100、500、1000 和 2000,分别对应 5 种不同的参数配置.

表 9 和表 10 给出了不同 i 下各实例生成的平均测试用例集规模和消耗时间. 从表中可以看出,随着 i 从 50 增加到 1000,生成规模变小,所消耗的时间也变少. 在 S2-2 和 S2-8 中,i 从 1000 增加到 2000 时,前者消耗的时间要多于后者,但是生成规模要小于后者,综合考虑,i 取 1000 是生成较小规模用例集和消耗较少时间的一个好的选择.

表 9　不同 i 下测试用例集的规模

实例	迁移间隔				
	50	100	500	1000	2000
S2-2	18.4	16.9	**15.0**	**15.0**	16.2
S2-8	46.7	44.3	42.4	**42.2**	46.5
S3-1	39.8	37.2	33.8	**33.2**	38.6
SPIN-S	21.5	20.3	17.4	**17.1**	20.8

表 10　不同 i 下测试用例集的消耗时间（单位：s）

实例	迁移间隔				
	50	100	500	1000	2000
S2-2	44.6	27.5	9.6	8.7	**8.3**
S2-8	56.3	45.8	25.2	22.9	**22.4**
S3-1	60.5	56.6	28.4	**24.8**	25.2
SPIN-S	53.7	41.3	20.0	**16.8**	17.1

(3) 迁移个体的数量 α

迁移个体的数量为所占子种群规模的百分比. 实验中 α 的取值集合为：0.1、0.2、0.4、0.6 和 0.8，分别对应 5 种不同的参数配置. 表 11 和表 12 给出了不同 α 下各实例生成的平均测试用例集规模和消耗时间，从表中可以看出，随着 α 的增加，生成规模变大，所消耗的时间也增多，因此，我们选取 $\alpha=0.1$.

表 11　不同 α 下测试用例集的规模

实例	迁移个体的数量				
	0.1	0.2	0.4	0.6	0.8
S2-2	**15.0**	**15.0**	15.8	16.3	17.8
S2-8	**42.4**	42.8	43.2	43.7	44.5
S3-1	**33.8**	**33.8**	34.7	35.2	37.6
SPIN-S	**17.4**	17.9	18.3	19.2	19.8

表 12　不同 α 下测试用例集的消耗时间（单位：s）

实例	迁移个体的数量				
	0.1	0.2	0.4	0.6	0.8
S2-2	**9.6**	9.7	10.2	23.4	28.6
S2-8	**25.2**	25.8	29.3	38.5	43.6
S3-1	**28.4**	29.7	40.6	53.7	63.5
SPIN-S	**20.0**	20.7	27.4	37.8	45.3

(4) 迁移个体的选择和替换方法 P

实验中 P 有 4 种情况：选择最好的个体，替换最差的个体(best-worst)；随机选择个体，替换最差的个体(random-worst)；选择最好的个体，随机替换个体(best-random)；随机选择个体，随机替换个体(random-random). 这 4 种方法分别对应 4 个不同的参数配置.

表 13 和表 14 给出了不同 P 下各实例生成的平均测试用例集规模和消耗时间，其中，随机选择个体，随机替换个体方法生成规模最小，消耗时间最少.

表 13　不同 P 下测试用例集的规模

实例	选择迁移个体的选择和替换方法			
	best-worst	random-worst	best-random	random-random
S2-2	15.4	17.8	18.3	**15.0**
S2-8	43.5	45.3	44.6	**42.4**
S3-1	34.6	37.8	39.4	**33.8**
SPIN-S	17.8	19.6	19.3	**17.4**

表 14　不同 P 下测试用例集的消耗时间（单位：s）

实例	选择迁移个体的选择和替换方法			
	best-worst	random-worst	best-random	random-random
S2-2	10.2	10.4	11.2	**9.6**
S2-8	26.7	30.3	39.6	**25.2**
S3-1	29.5	35.4	41.2	**28.4**
SPIN-S	21.2	29.3	37.6	**20.0**

(5) 迁移拓扑结构 T

迁移拓扑结构选择环形和随机结构两种，分别对应 2 种不同的参数配置，在这 2 种结构下各实例生成的平均测试用例集规模和消耗时间如表 15 和表 16 所示. 从表中可以看出，随机结构要优于环形结构.

表 15　不同迁移拓扑结构下测试用例集的规模

实例	迁移拓扑结构	
	环形	随机
S2-2	**15.0**	**15.0**
S2-8	44.5	**42.4**
S3-1	36.5	**33.8**
SPIN-S	19.3	**17.4**

表 16　不同迁移拓扑结构下测试用例集的消耗时间（单位：s）

实例	迁移拓扑结构	
	环形	随机
S2-2	10.4	**9.6**
S2-8	29.1	**25.2**
S3-1	35.3	**28.4**
SPIN-S	28.7	**20.0**

综上所述，实验得到的推荐参数配置如表 17 所示，与 Skolicki 给出的典型参数配置相比，该参数配置更适合生成组合测试用例集. 在实际情况下，很难找到一个适合所有实例的最优的参数配置，我们经过多次尝试后给出的推荐参数配置可以在较短的时间内生成较小规模的组合测试用例集，该配置将作为本文后续实验工作的参数配置.

表 17　推荐参数配置

参数名称	参数设置
子种群规模	100
迁移间隔	1000
最大代数	5000
迁移个体的数量	10%
选择和替换方法	随机，迁移个体的拷贝
迁移拓扑结构	随机结构

4.3 IPGAS 和 PGAS、SGA 之间的比较

在实验中,我们分别比较 IPGAS 和 PGAS、SGA 的有效性和性能.对于有效性,我们比较三者生成用例集的规模和消耗时间.我们使用加速比(speedup)[9]来评价 PGAS 和 IPGAS 的性能.消耗时间是算法搜索到最优解的时长,加速比是在搜索到最优解的情形下,SGA 消耗时间与 IPGAS(或 PGAS)消耗时间的比值.通过加速比我们可以比较 IPGAS(或 PGAS)相对于 SGA 性能上的提高.为了获得统计显著性结果,我们对每个实例都执行 30 次,分别给出测试用例集规模和消耗时间的最优值和平均值.

(1)为了比较有效性,我们运行了表 4 和表 5 中的 20 个实例,表 18 和表 19 分别给出了测试用例集规模和消耗时间的最优值和平均值.表 18 中数据显示,除了 S2-1 以外,在所有实例中,PGAS 和 IPGAS 生成的测试用例集规模都小于 SGA.IPGAS 生成的测试用例集规模等于或小于 PGAS.表 19 中数据显示,对于 S2-1,SGA 的消耗时间要小于 IPGAS,我们分析这是由于 Spark 的驱动程序和运行节点间存在一定的通信开销,同时 S2-1 的问题规模非常小,随着问题规模的增大,在其它所有实例中,IPGAS 的消耗时间都最少.

表 18 SGA、PGAS、IPGAS 生成测试用例集的规模

编号	算法						编号	算法					
	SGA		PGAS		IPGAS			SGA		PGAS		IPGAS	
	最优	平均	最优	平均	最优	平均		最优	平均	最优	平均	最优	平均
S2-1	**9**	9.0	**9**	9.0	**9**	9.0	S3-1	33	34.6	**33**	34.1	**33**	33.8
S2-2	17	17.6	**15**	15.7	**15**	15.0	S3-2	66	69.2	**64**	67.5	**64**	67.2
S2-3	15	15.9	**15**	15.7	**15**	15.3	S3-3	108	112.4	102	106.7	**100**	104.6
S2-4	26	28.2	23	24.8	**22**	23.4	S3-4	150	155.6	**125**	130.6	**125**	130.2
S2-5	26	28.7	24	26.7	**23**	25.2	S3-5	218	225.7	210	217.5	**206**	213.5
S2-6	34	27.7	32	35.5	**31**	34.3	SPIN-S	19	21.2	**17**	19.4	**17**	17.4
S2-7	34	37.4	**31**	34.6	**31**	34.7	Bugzilla	18	20.7	**16**	18.3	**16**	17.3
S2-8	44	47.7	**42**	45.6	**42**	42.4	SPIN-V	29	31.2	**27**	29.4	**27**	29.1
S2-9	57	61.5	48	55.6	**47**	51.9	Apache	34	36.4	**32**	34.7	**32**	34.3
S2-10	72	76.8	68	73.4	**67**	71.5	GCC	20	22.8	**17**	19.7	**17**	19.3

表 19 SGA、PGAS、IPGAS 生成测试用例集的消耗时间 (单位:s)

编号	算法						编号	算法					
	SGA		PGAS		IPGAS			SGA		PGAS		IPGAS	
	最优	平均	最优	平均	最优	平均		最优	平均	最优	平均	最优	平均
S2-1	**0.6**	0.6	7	7.0	5	5.0	S3-1	58	60.7	75	77.8	**28**	28.4
S2-2	158	160.3	10	10.8	**9**	9.6	S3-2	160	163.4	113	116.8	**34**	34.8
S2-3	146	148.4	18	19.3	**14**	15.3	S3-3	368	371.6	198	201.6	**126**	128.6
S2-4	614	618.6	283	287.6	**168**	172.4	S3-4	482	486.7	268	272.3	**168**	171.4
S2-5	300	304.5	69	72.4	**48**	50.6	S3-5	671	675.6	324	328.4	**212**	216.6
S2-6	653	656.7	392	395.4	**206**	209.5	SPIN-S	389	392.6	75	77.8	**20**	21.6
S2-7	114	117.4	92	95.6	**42**	44.5	Bugzilla	435	438.5	26	27.3	**17**	17.4
S2-8	69	71.3	38	40.3	**23**	25.2	SPIN-V	662	667.2	67	69.5	**46**	47.6
S2-9	2234	2239.3	1568	1572.6	**1092**	1096.4	Apache	4862	4868.6	1053	1058.4	**965**	968.6
S2-10	387	391.5	98	100.5	**84**	86.1	GCC	6138	6145.6	1629	1634.6	**1214**	1218.6

我们还对三者生成的平均测试用例集规模和平均消耗时间分别进行 T 检验(t-test)分析,显著性水平值设为 0.05,分析所得 p 值如表 20 所示.从表中可以看出,3 种算法两两之间的 p 值都小于 0.05,表示它们之间有显著性差异.由此我们得出结论:PGAS 在生成的测试用例集规模和消耗时间上都要优于 SGA,IPGAS 要优于 PGAS.

表 20 T 检验结果表

	测试用例集规模	消耗时间
SGA 与 PGAS 之间 p 值	0.0232	0.0345
SGA 与 IPGAS 之间 p 值	0.0040	0.0227
PGAS 与 IPGAS 之间 p 值	0.0002	0.0040

(2)为了比较性能,我们选取实例 S2-9 和 Apache.表 18 中数据显示,在 S2-9 和 Apache 中,SGA 生成测试用例集规模的最优值大于 PGAS 和 IPGAS 的平均值,因此,SGA 不能生成与 PGAS 和 IPGAS 规模相同的最优测试用例集.为了获得可比较的数据,我们以 SGA 生成的规模的最优值(分别为 57 和 34)为算法寻优目标,再次执行实例 30 次来获取三者各自的平均消耗时间.

实验中我们分别得到 SGA 的平均消耗时间,以及 PGAS 和 IPGAS 在子种群数量为 4、8、16 和 32 时的平均消耗时间,由于 SGA 是串行遗传算法,不分割子种群,因此 SGA 的平均消耗时间只有 1 条数

据.实验所得结果如表 21 和表 22 所示,表中“—”表示无相关实验数据,α 和 β 分别表示 PGAS 和 IPGAS 的加速比,即 α=SGA 平均消耗时间/PGAS 平均消耗时间,β=SGA 平均消耗时间/IPGAS 平均消耗时间.

表 21　PGAS、IPGAS 的加速比(S2-9)

子种群数量	SGA 平均消耗时间/s	PGAS 平均消耗时间/s	IPGAS 平均消耗时间/s	α	β
1	2245.6	—	—	—	—
4	—	839.4	615.6	2.68	3.65
8	—	465.9	249.7	4.81	8.99
16	—	286.5	143.8	7.83	15.62
32	—	236.8	76.4	9.48	29.39

表 22　PGAS、IPGAS 的加速比(Apache)

子种群数量	SGA 平均消耗时间/s	PGAS 平均消耗时间/s	IPGAS 平均消耗时间/s	α	β
1	4870.3	—	—	—	—
4	—	1656.60	1323.50	2.94	3.68
8	—	849.96	555.97	5.73	8.76
16	—	508.38	317.08	9.58	15.36
32	—	387.76	171.49	12.56	28.40

从表中可以看出,随着子种群数量的增加,PGAS 和 IPGAS 的加速比都逐渐增加,IPGAS 的加速比更接近于子种群数量,因此,IPGAS 比 PGAS 具有更好的可伸缩性.同时,IPGAS 比 PGAS 加速约 2 至 3 倍.由此我们得出结论:IPGAS 的性能优于 PGAS,PGAS 的性能优于 SGA.

4.4　IPGAS 与已有方法的比较

在这一组实验中我们将 IPGAS 与已有的组合测试用例集生成方法做比较,主要比较生成的最优测试用例集规模,已有方法的最优值取自于在相关文献中公开的数据.比较的方法包括非并行化方法和并行化方法.在所比较的方法中,有些方法没有在相关文献中公开生成测试用例集的消耗时间,有些方法虽然公开了消耗时间,但由于运行环境的不同,很难做到公平的比较,因此,本组实验主要比较生成测试用例集的规模.我们对每个实例都执行 30 次,分别给出测试用例集规模和消耗时间的最优值.

(1)比较的非并行化方法包括贪心算法(IPO[4]、mTCG[19-20]和 mAETG[19-20])和启发式搜索算法(ACA[21]、SA[19]、CASA[17]、HHSA[18]、PSO[22]、GA[21]、GAPTS[23]和 PWiseGen[24]).表 23 和表 24 分别给出了 2-way 和 3-way 合成实例的测试用例集规模.对于 2-way 真实软件实例,比较的方法为 CASA[17],表 25 给出了 2-way 真实软件实例的测试用例集规模.

表 23　2-way 合成实例测试用例集的规模

编号	贪心算法			ACA	模拟退火			PSO	遗传算法			IPGAS	
	IPO	mTCG	mAETG		SA	CASA	HHSA		GA	GAPTS	PWiseGen	规模	时间/s
S2-1	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	5
S2-2	—	18	20	16	**15**	**15**	**15**	17	**15**	—	—	**15**	9
S2-3	19	17	17	17	16	**15**	**15**	18	17	**15**	**15**	**15**	14
S2-4	—	26	27	27	**21**	22	**21**	27	27	27	26	22	168
S2-5	36	28	28	25	**21**	23	**21**	27	26	—	—	23	48
S2-6	—	34	37	37	30	30	**29**	38	37	35	34	31	206
S2-7	—	35	35	32	**30**	**30**	**30**	35	33	—	—	31	42
S2-8	—	**42**	44	**42**	**42**	**42**	**42**	43	**42**	—	—	**42**	23
S2-9	—	56	56	—	**45**	46	**45**	—	—	—	—	47	1092
S2-10	—	—	70	—	**62**	64	63	—	—	—	—	67	84

表 24　3-way 合成实例测试用例集的规模

编号	mAETG	GA	模拟退火			ACA	IPGAS	
			SA	CASA	HHSA		规模	时间/s
S3-1	38	**33**	**33**	**33**	**33**	**33**	**33**	28
S3-2	77	**64**	**64**	96	**64**	**64**	**64**	34
S3-3	114	108	**100**	**100**	**100**	106	**100**	126
S3-4	194	**125**	152	185	**125**	**125**	**125**	168
S3-5	218	218	**201**	213	202	218	206	212

表 25　2-way 真实软件实例测试用例集的规模

名称	CASA	IPGAS	
		规模	时间/s
SPIN-S	**16.4**	**17**	20
Bugzilla	**16.0**	**16**	17
SPIN-V	**26.4**	**27**	46
Apache	**32.0**	**32**	965
GCC	**17.0**	**17**	1214

表 23 中数据显示,在生成 2-way 组合测试用例集方面,和贪心算法相比,IPGAS 能够生成更小规模的用例集.和启发式搜索方法相比,IPGAS 优于 ACA、PSO 和串行遗传算法.在 S2-1、S2-2、S2-3 和 S2-8 等 4 个实例中,IPGAS 能够生成和模拟退火相等大小的用例集,其余实例中,模拟退火生成的用例集要小于 IPGAS.我们认为,模拟退火优于 IPGAS 的原因在于它的搜索策略[17-19].如何进一步提高遗传算法搜索最优测试用例集的能力将是我们今后的研究工作.

表 24 中数据显示,在生成 3-way 组合测试用例集方面,和贪心算法相比,IPGAS 能够生成更小规模的用例集.和启发式搜索方法相比,IPGAS 优于

ACA 和串行遗传算法. 在 S3-5 实例中,模拟退火生成的用例集要小于 IPGAS,其余 4 个实例中,两者生成的用例集大小相等.

表 25 中 CASA 的结果是运行 5 次后得到的平均值,从表中可以看出,在 5 个真实软件实例中,IPGAS 能够生成和 CASA 规模相同的用例集.

(2)比较的并行化方法为并行化模拟退火 CSA[25],我们选取了文献[25]中的 8 个典型的 3-way 实例,表 26 给出了测试用例集规模和消耗时间的最优值. 从表中可以看出,IPGAS 生成的测试用例集规模等于或略高于 CSA.

表 26 IPGAS 与 CSA 比较($t=3$)

实例	CSA	IPGAS	
		规模	时间/s
3^6	**33**	**33**	28
3^8	**42**	**42**	54
3^{10}	**45**	**45**	78
3^{13}	**49**	50	158
3^{16}	**59**	**59**	286
3^{21}	**67**	**67**	765
3^{25}	**72**	73	964
3^{27}	**79**	81	1196

4.5 实验结果分析与结论

通过 4.3 节和 4.4 节的实验可以看出,基于 4.2 节给出的推荐参数配置,IPGAS 在 2-way 和 3-way 组合测试用例集生成中具有比较好的效果. 4.3 节的实验结果表明:IPGAS 在生成用例集的规模和消耗时间上都要优于 SGA 和 PGAS,同时 IPGAS 比 PGAS 具有更好的加速比. 4.4 节的实验结果表明:IPGAS 在生成用例集的规模上要优于贪心算法、GA、ACA 和 PSO,与模拟退火及并行化模拟退火相比,在一些实例中,IPGAS 能够生成和它们相同规模的测试用例集,在另外一些实例中,IPGAS 生成的测试用例集的规模略大于它们. 总之,IPGAS 能够在合理的时间内生成较小规模的组合测试用例集,是一种较优的选择.

5 相关工作

目前,启发式搜索算法已被应用于组合测试生成领域. Ghazi 等人[26]最先提出使用遗传算法生成两两组合测试配置,并设计了简单的实验表明其可行性. Shiba 等人[21]使用遗传算法和蚁群算法生成 2-way 和 3-way 测试用例集,并通过实验验证了方法的有效性. McCaffrey[23]设计了一种用于两两组合测试用例生成的算法,称为 GAPTS. GAPTS 使用整数对染色体进行编码,将适应度函数定义为一个个体所覆盖的所有不同两两组合的个数. 与其它方法相比,GAPTS 能够生成更小规模的测试用例集,但是,需要更长的运算时间. Flores 和 Yoonsik[24]也使用遗传算法生成两两组合测试用例,他们设计并实现了一个开源工具,称为 PWiseGen,并通过实验验证了该工具的有效性、可配置性、可扩展性和复用性. Nie 等人[27]设计了遗传算法、粒子群优化算法和蚁群算法及其变种算法,并通过实验比较了这些算法在生成 2-way 覆盖表方面的有效性和性能. 梁亚澜等人[7]通过实验对遗传算法的种群规模、进化机制、交叉概率、变异概率及其变种算法这 5 个因素进行取值组合,设计了 3 条不同的实验路线,以生成覆盖表规模和消耗时间为分析依据寻找出最佳配置. Cohen 等人[19]使用爬山法和模拟退火生成可变力度覆盖表,与贪心算法相比,这两种方法能够生成更小规模的可变力度覆盖表. Cohen 等人[28]提出混合代数方法和模拟退火方法来构建 3-way 覆盖表,由于结合了两种方法的优点,该方法能够生成一些较小规模的 3-way 覆盖表. Garvin 等人[17]改进了模拟退火方法来生成带约束条件的覆盖表,实验表明该方法在求解带约束条件和不带约束条件问题时具有生成覆盖表规模较小、消耗时间较短等优点. Jia 等人[18]首次使用超启发式算法生成组合测试用例集,他们通过实验求解带约束条件和不带约束条件问题,实验结果表明了该方法的有效性和效率. Chen 等人[22]使用粒子群优化算法生成两两组合测试用例,使用 one-test-at-a-time 策略和参数序(IPO)策略构造用例集,并通过实验验证了该方法的有效性和效率. Wu 等人[29]扩展了一种基于集合的离散粒子群优化算法用于生成覆盖表,并提出了两种辅助策略来提高性能,实验结果表明,与其它粒子群优化算法和进化算法相比,该方法能够生成规模更小的覆盖表.

上述算法研究主要还是集中在常规的串行化算法设计. 当利用这些串行化算法进行测试用例生成处理时,其时间开销仍然是难以忍受的. 为此,除了研究寻找良好的算法本身以外,另一个重要的问题是,需要研究解决并行化的测试用例生成算法和处理方法. 并行化是提高启发式搜索算法的有效方法,这方面已经取得了一些研究成果[9,11]. 然而,将并行元启发式搜索算法应用于软件测试领域的研究并不多. Geronimo 等人[30]提出了一种基于 Hadoop MapReduce 的并行化遗传算法用于生成 JUnit 单

元测试用例集，该方法实现了主从模型，并通过实验与串行遗传算法进行比较，得到性能上的提升. Ferrucci 等人[31]使用 MapReduce 实现并行化遗传算法用于测试数据生成，并实现了到云环境下的迁移. 他们描述了三种并行化模型，并基于 Google 的应用引擎框架(App Engine framework)实现了全局模型和粗粒度模型. 上述两项工作都是使用基于 MapReduce 的并行化遗传算法生成白盒测试数据. Younis 等人[32]基于多核架构提出了一种改进的参数序方法，称为 MC-MIPOG，该方法能够生成 $t>6$ 的测试用例集. 实验结果表明，该方法在生成测试用例集的规模和性能上明显优于已有的单机方法. Lopez-Herrejon 等人[33]提出一种并行化遗传算法用于生成软件产品线的优化的两两组合测试用例集，该方法基于主从模型实现了适应度函数计算的并行化. Avila-George 等人[25]提出了 3 种并行模拟退火算法用于覆盖表生成问题：独立搜索、半独立搜索和协同搜索. 他们使用 C 语言和消息传递接口(Message Passing Interface，MPI)实现了这 3 种并行方法，并通过实验比较了 3 种方法的性能和有效性，实验结果表明独立搜索方法性能最差，可扩展性也不好；半独立搜索方法提供了可接受的消耗时间. 和独立搜索方法相比，当问题规模变大时，半独立搜索方法的通信开销会增加；协同搜索方法是三者中效果最好的方法，通过引入异步信息交换，该方法大大缩短了搜索时间，同时，该方法生成的覆盖表的规模等于或优于当前文献中报道的最好结果.

与 Younis 等人、Lopez-Herrejon 等人和 Avila-George 等人提出的并行方法相比，本文基于主流的并行计算框架 Spark，提出一种岛模型并行化遗传算法，用于生成 t-way 组合测试用例集. 该方法支持大规模种群下的快速寻优，能够快速生成组合测试用例集.

6　总结和展望

针对遗传算法在组合测试用例集生成时存在的计算性能问题，本文研究实现了基于 Spark 的岛模型并行化遗传算法，用于快速生成组合测试用例集. 在具体的并行化算法设计实现方法上，本文主要进行了 Spark 运行节点间交换信息、基于 Spark 的岛模型迁移机制和迁移策略等方面的优化.

我们通过实验探索 IPGAS 的参数配置对生成组合测试用例集规模和消耗时间的影响，给出了适合组合测试用例集生成的推荐参数配置. 在此基础之上，我们设计了两组实验：与串行遗传算法和独立运行模型遗传算法比较；与已有的测试用例集生成方法比较. 实验结果表明，IPGAS 在生成测试用例集规模和消耗时间上均优于 SGA 和 PGAS. 和已有方法比较时，IPGAS 在运行 28 个实例中的 18 个实例时，能够在合理的时间内生成和当前最好结果相同规模的用例集. 因此，IPGAS 在生成组合测试用例集方面具有一定的竞争力，是遗传算法在组合测试用例集生成方面的一个有效的改进.

本文中，IPGAS 主要用于生成 2-way 和 3-way 组合测试用例集，如果组合测试强度变大，比如 $t=4,5,6$ 时，中间读写的数据量会增加，将会影响算法的性能. 在今后的研究工作中，我们将考虑使用性能更好的方法来实现个体的迁移，比如使用分布式的基于内存的文件系统代替 HDFS，或者修改 Spark 的通信机制部分的源码.

IPGAS 在各个节点上运行的都是相同的遗传算法，参数配置也相同，将来我们计划在不同的节点上运行不同的遗传算法和不同的参数配置，实现异构的并行遗传算法，进一步提高遗传算法在生成组合测试用例集方面的性能.

实际软件系统中各种因素间通常会存在约束关系，约束处理和组合测试用例生成一样都是计算密集型任务，如何在并行化遗传算法中增加高效的约束处理机制将是我们今后的研究工作.

参　考　文　献

[1] Kuhn D R, Kacker R N, Lei Yu. Introduction to Combinatorial Testing. Boca Raton, USA: CRC Press, 2013

[2] Nie Chang-Hai. Combinatorial Testing. Beijing: Science Press, 2015(in Chinese)
(聂长海. 组合测试. 北京：科学出版社，2015)

[3] Kuhn D R, Wallace D R, Gallo A M. Software fault interactions and implications for software testing. IEEE Transactions on Software Engineering, 2004, 30(6): 418-421

[4] Lei Yu, Tai K C. In-parameter-order: A test generation strategy for pairwise testing//Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium. Washington, USA, 1998: 254-261

[5] Nie Chang-Hai, Leung H. A survey of combinatorial testing. ACM Computing Surveys, 2011, 43(2): 1-29

[6] Khalsa S K, Labiche Y. An orchestrated survey of available algorithms and tools for combinatorial testing//Proceedings of the 25th International Symposium on Software Reliability Engineering(ISSRE). Naples, Italy, 2014: 323-334

[7] Liang Ya-Lan, Nie Chang-Hai. The optimization of configurable genetic algorithm for covering arrays generation. Chinese Journal of Computers, 2012, 35(7): 1522-1538(in Chinese)
(梁亚澜，聂长海. 覆盖表生成的遗传算法配置参数优化. 计算机学报，2012，35(7)：1522-1538)

[8] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. San Jose, USA, 2012: 15-28

[9] Luque G, Alba E. Parallel Genetic Algorithms Theory and Real World Applications. Berlin: Springer, 2011

[10] Zaharia M, Chowdhury M, Franklin M, et al. Spark: Cluster computing with working sets. EECS Department, University of California, Berkeley: Technical Report UCB/EECS-2010-53, 2010

[11] Alba E, Luque G, Nesmachnow S. Parallel metaheuristics: Recent advances and new trends. International Transactions in Operational Research, 2013, 20(1): 1-48

[12] Qi Rong-Zhi, Wang Zhi-Jian, Li Shui-Yan. A parallel genetic algorithm based on spark for pairwise test suite generation. Journal of Computer Science and Technology, 2016, 31(2): 417-427

[13] Skolicki Z. An Analysis of Island Models in Evolutionary Computation [Ph. D. dissertation]. Department of Computer Science, George Mason University, Fairfax, USA, 2007

[14] Cantú-Paz E. Migration policies, selection pressure, and parallel evolutionary algorithms. Journal of Heuristics, 2001, 7(4): 311-334

[15] Skolicki Z, De Jong K. The influence of migration sizes and intervals on island models//Proceedings of the 2005 ACM Genetic and Evolutionary Computation Conference (GECCO). Washington, USA, 2005: 1295-1302

[16] Tomassini M. Spatially Structured Evolutionary Algorithms Artificial Evolution in Space and Time. Secaucus, USA: Springer-Verlag New York Inc., 2005

[17] Garvin B J, Cohen M B, Dwyer M B. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Engineering, 2011, 16(1): 61-102

[18] Jia Yue, Cohen M B, Harman M, et al. Learning combinatorial interaction test generation strategies using hyperheuristic search//Proceedings of the 37th International Conference on Software Engineering (ICSE). Florence, Italy, 2015: 540-550

[19] Cohen M B, Gibbons P B, Mugridge W B, et al. Constructing test suites for interaction testing//Proceedings of the 25th International Conference on Software Engineering (ICSE). Portland, USA, 2003: 38-48

[20] Cohen M B. Designing Test Suites for Software Interaction Testing [Ph. D. dissertation]. The University of Auckland, Auckland, New Zealand, 2004

[21] Shiba T, Tsuchiya T, Kikuno T. Using artificial life techniques to generate test cases for combinatorial testing//Proceedings of the 28th Annual Computer Software and Applications Conference (COMPSAC). Hong Kong, China, 2004: 72-77

[22] Chen Xiang, Gu Qing, Qi Jing-Xian, et al. Applying particle swarm optimization to pairwise testing//Proceedings of the 34th Annual Computer Software and Applications Conference (COMPSAC). Seoul, Korea, 2010: 107-116

[23] Mccaffrey J D. An empirical study of pairwise test set generation using a genetic algorithm//Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG). Las Vegas, USA, 2010: 992-997

[24] Flores P, Yoonsik C. PWiseGen: Generating test cases for pairwise testing using genetic algorithms//Proceedings of the IEEE International Conference on Computer Science and Automation Engineering (CSAE). Shanghai, China, 2011: 747-752

[25] Avila-George H, Torres-Jimenez J, Hernádez V. New bounds for ternary covering arrays using a parallel simulated annealing. Mathematical Problems in Engineering, 2012, 2012(4): 245-255

[26] Ghazi S A, Ahmed M A. Pair-wise test coverage using genetic algorithms//Proceedings of the 2003 Congress on Evolutionary Computation. Canberra, Australia, 2003: 1420-1424

[27] Nie Chang-Hai, Wu Hua-Yao, Liang Ya-Lan, et al. Search based combinatorial testing//Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC). Hong Kong, China, 2012: 778-783

[28] Cohen M B, Colbourn C J, Ling A C H. Augmenting simulated annealing to build interaction test suites//Proceedings of the 14th International Symposium on Software Reliability Engineering. Denver, USA, 2003: 394-405

[29] Wu Hua-Yao, Nie Chang-Hai, Kuo Fei-Ching, et al. A discrete particle swarm optimization for covering array generation. IEEE Transactions on Evolutionary Computation, 2015, 19(4): 575-591

[30] Geronimo L D, Ferrucci F, Murolo A, et al. A parallel genetic algorithm based on hadoop MapReduce for the automatic generation of JUnit test suites//Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation. Montreal, Canada, 2012: 785-793

[31] Martino D S, Ferrucci F, Maggio V, et al. Towards migrating genetic algorithms for test data generation to the cloud//Tilley S, Parveen T eds. Software Testing in the Cloud: Perspectives on an Emerging Discipline. Hershey, USA: IGI Global, 2012: 113-135

[32] Younis M, Zamli K. MC-MIPOG: A parallel t-way test generation strategy for multicore systems. ETRI Journal, 2010, 32(1): 73-83

[33] Lopez-Herrejon R E, Ferrer J, Chicano F, et al. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines//Proceedings of the 2014 ACM Genetic and Evolutionary Computation Conference(GECCO). Vancouver, Canada, 2014: 1255-1262

QI Rong-Zhi, born in 1980, Ph.D., lecturer. His research interest is search based combinatorial testing.

WANG Zhi-Jian, born in 1958, Ph.D., professor. His research interests include domain software engineering and network security.

HUANG Yi-Hua, born in 1962, Ph.D., professor. His research interests include big data parallel processing and parallel computing.

LI Shui-Yan, born in 1980, Ph.D. candidate. Her research interests include evolution computing and data mining.

Background

Software testing is the activity of executing a system to detect failures. In some testing scenarios, many failures are triggered by interactions among parameters of the software. Combinatorial testing can detect these failures effectively. It is based on the observation that most interaction failures are caused by interactions of at most six factors. To date, test suite generation is the most active area of combinatorial testing research in the world. As generating minimum test suite for combinatorial testing is an NP-complete problem. Many methods have been proposed to generate combinatorial test suite, including greedy algorithms, heuristic search algorithms, mathematic methods, and random methods. Among these methods, the greedy algorithms are the most widely used for combinatorial test suite generation. They are usually faster than the metaheuristic algorithms but do not always produce the smallest test suite. The heuristic search algorithms can often produce smaller test suite than other approaches, but typically require a longer computation.

To solve this problem, this paper presents a parallel genetic algorithm based on island model to generate combinatorial test suite. Our approach uses Spark, a fast and general parallel computing platform, to implement the island model. The key issue of island model is how to migrate individuals among subpopulations, which is achieved by means of Hadoop Distributed File System (HDFS). Parameters of island model are tuned systematically to find recommended settings for combinatorial test suite generation. We evaluate performance of our approach against sequential genetic algorithm, isolated running genetic algorithm and other existing approaches in both generated test suite sizes and computational effort. Experiment results show that our approach is a promising improvement of genetic algorithm for combinatorial test suite generation.

This work is supported by the National Key Research and Development Program of China (2016YFC0400910), the National Key Technology Research and Development Program of the Ministry of Science and Technology of China (2013BAB06B04), and the Fundamental Research Funds for the Central Universities (2015B22214, 2013B07514).