基于PSO的多目标测试用例预优化

陈云飞 李 征 赵瑞莲
(北京化工大学计算机系 北京 100029)

摘 要 随着软件规模的增大,在软件回归测试中,重复执行庞大的全部测试用例集已不再现实。在这种情况下,对测试用例集进行预处理就尤为重要。测试用例预优化是寻找最佳测试用例执行序列的一种技术。在实际的软件回归测试中,基于多目标的测试用例优化技术已逐步取代了单目标优化;应用进化算法解决多目标测试用例预优化是当前研究的热点。但由于进化算法主要是基于种群进行遗传迭代,种群间的交互机制相对复杂,算法的执行效率会随着种群及测试用例集规模的增大而显著下降。针对上述情况,提出了一种基于粒子群优化算法(PSO)的测试用例预优化方法,设计了粒子的表示和状态更新方式,研究了不同粒子更新方式和迭代次数及粒子群大小对多目标测试用例预优化结果的影响。实验结果显示,同基于NSGA-Ⅱ的方法相比,所提方法的执行效率显著提高,可以解决实际回归测试中的多目标测试用例预优化问题。

关键词 回归测试,测试用例预优化,多目标,PSO
中图法分类号 TP311 **文献标识码** A

Applying PSO to Multi-objective Test Cases Prioritization

CHEN Yun-fei LI Zheng ZHAO Rui-lian
(Department of Computer Science, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract It may be impossible to re-execute the whole test suite in regression testing with the increasing size of software. In such a case, prioritizing test cases is vital to software regression testing. Test cases prioritization is a technique to search the best sequence of test cases execution. In regression testing for real projects, single-objective test cases prioritization has been gradually replaced by multi-objective prioritization, where the application of Evolution Algorithms to address multi-objective optimization problems is a hot spot in current research. However EAs are based on population genetic iterations in which the mechanism of exchanging information among populations is relatively complex, and consequently the efficiency of test suite prioritization based on EAs is sharply declining with the increase in the scale of the population. To address this problem, the paper presented a test suite prioritization technique based on Particle Swarm Optimization, proposed the corresponding particle representation, position and speed updated methods. Empirical studies were conducted to study the effect caused by different types of particle updating methods and size of particle swarm. Compared to the NSGA-Ⅱ based test case prioritization, the proposed technique is more efficient in test suite prioritization in real programs.

Keywords Regression testing, Test suite prioritization, Multi-objective optimization, PSO

1 引言

在软件尤其是大规模软件的开发过程中,软件回归测试是保证软件质量的有效手段[1]。随着软件规模的增大,其测试用例集也会逐渐增大,软件回归测试已成为软件开发极其耗时的主要部分。对测试用例集进行优化处理是当今软件测试领域关注的热点,主要优化技术包括3种:测试用例集最小化、测试用例选择和测试用例预优化,其中测试用例集最小化和测试用例选择都需要对原有测试用例集进行约减,这可能会引入不可预知的问题。测试用例预优化是寻找最佳的测试用例执行序列以优化回归测试的一种技术,不对测试用例集自身产生任何简化[2],因而不存在引入未知风险的问题,受到了越来越多的研究者关注。Yoo和Harman对这3种技术进行了系统的研究与总结,指出关于软件回归测试中测试用例集优化技术的研究仍处于初级阶段[3]。

基于搜索的软件工程[4]是把软件工程问题转化为优化问题处理,应用进化算法解决测试用例预优化问题是其中一个研究热点,特别是针对预优化中的多目标优化问题,以NSGA-Ⅱ为代表的多目标进化算法以其特有的种群进化特性成为解决这类优化问题的首选方案,并取得了较好的成果。

到稿日期:2013-09-17 返修日期:2013-11-21 本文受国家自然科学基金(61170082,61073035),教育部新世纪优秀人才支持计划(NCET-12-0757),教育部留学回国人员科研启动基金(LXJJ201303)资助。
陈云飞(1990—),男,硕士生,主要研究方向为软件测试,E-mail:yunfeitianshang@163.com;李 征(1974—),男,博士,教授,CCF会员,主要研究方向为软件测试、程序分析;赵瑞莲(1964—),女,博士,教授,CCF会员,主要研究方向为软件测试、软件可靠性。

但由于进化算法是基于种群迭代遗传的，种群内部信息交互机制相对复杂，在处理较大规模问题时其效率会显著下降。虽然粒子群优化算法（Particle Swarm Optimization，PSO）是一种基于种群的全局随机优化算法，但同以 GA 为代表的进化算法相比，PSO 算法中粒子群内部信息交互机制简单，需要调整的参数较少。另外与其它进化算法相比，粒子群算法易于实现，收敛速度较快，全局搜索能力较强，在处理多目标优化问题上有着明显的优势[8]。

本文提出了一种基于 PSO 算法的多目标测试用例预优化方法。针对测试用例预优化这类离散问题，设计了 PSO 算法中粒子位置及速度的表示方法和两种粒子状态更新方法；同时在不同规模被测程序测试用例预优化实验的基础上，研究了迭代次数和粒子群大小对优化结果的影响；最后与采用 NSGA-Ⅱ算法的测试用例预优化方法进行了实验比较分析，结果显示在同等实验的条件下，本文方法的执行效率明显优于采用 NSGA-Ⅱ算法的测试用例预优化方法。

2 PSO 算法及多目标测试用例预优化

PSO 算法是一种基于种群的全局随机优化方法，是由 Kennedy 及 Eberhart 在 1995 年受社群行为如鸟类及鱼群聚集行为启发而提出[7]的。在 PSO 算法中，每个粒子的位置代表解空间的一个解；粒子根据自身速度、自身最优位置及全局最优位置迭代更新自身的位置。PSO 算法简化了进化算法中一些繁复的操作，因而性能也有了很大的提升。

假设搜索空间的维度为 D，粒子群的大小为 m，则粒子可用一个 D 维的向量表示为 $X_i, i=1,2,3,\cdots,m$，第 i 个粒子在解空间内的位置为 $X_i=(x_{i1},x_{i2},\cdots,x_{iD})$，粒子的速度可用 D 维向量 V_i 表示，粒子的自身最优位置定义为粒子所经过的所有位置中的最优值，表示为 $P_i=(p_{i1},p_{i2},\cdots,p_{iD})$，粒子群全局最优则是所有粒子经历过的所有位置中的最优值，表示为 $P_g=(p_{g1},p_{g2},\cdots,p_{gD})$。在每次迭代中，计算粒子的适应度值，并按照式(1)和式(2)进行速度及位置的更新[7]。

$$v_{id}(k+1)=v_{id}(k)+c_1r_1(p_{id}(k)-x_{id}(k))+c_2r_2(p_{gd}(k)-x_{id}(k)) \quad (1)$$

$$x_{id}(k+1)=x_{id}(k)+v_{id}(k+1) \quad (2)$$

由式(1)可以看出，粒子的速度由 3 部分组成：第一部分表示粒子上次移动的速度；第二部分代表粒子自身最优对速度的影响，又称为认知部分，其中 c_1 被称为学习因子，r_1 则是 0～1 之间的随机数，用来平衡自身认知对移动速度的影响；第三部分代表全局最优对速度的影响，又称为社会部分，其中 c_2 等同于 c_1，r_2 也是 0～1 之间的随机数，用来平衡社会部分对速度的影响。由式(2)可以看出，粒子的位置由 2 部分组成：第一部分表示粒子的原有位置；第二部分则是粒子的移动速度。k 表示迭代次数。

软件测试用例预优化是寻找最佳测试用例执行序列的一种技术。在实际软件测试过程中，通常需要同时满足多个测试准则，也就是需要根据多个目标，对软件测试用例执行序列进行优化，其形式化定义如下。

定义 1（多目标测试用例预优化） 给定一个测试用例集 T，T 中测试用例的全排列集合 PT 及一个目标函数向量 $\vec{F}=[f_1(p),f_2(p),\cdots,f_i(p),\cdots,f_M(p)]$，$f_i$ 表示第 i 个优化目标的目标函数 $f_i:PT\to R$，其中 $p\in PT$，$1\leqslant i\leqslant M$。

目的：找到一个 $PT'\subset PT$，使得 $\forall p'\in PT'\wedge\vec{F}(p')$ 达到 Pareto 最优。

在上面的定义中，目标函数向量 $\vec{F}$ 是对测试用例预优化目标的数学抽象表示，其中分量 f_i 是第 i 个优化目标的数学抽象，它表示一个从测试用例全排列空间到实数空间的映射。

Pareto 最优的具体含义是指测试用例集的任意两个排序方案 $p_A,p_B\in PT'$，$\vec{F_A}$ 和 $\vec{F_B}$ 分别是 p_A 和 p_B 的目标函数向量，若存在至少一个 $f_i(p_A)$ 不如 $f_i(p_B)$，且同时存在至少一个 $f_j(p_B)$ 不如 $f_j(p_A)$，其中 $1\leqslant i,j\leqslant M$，则说明 p_A、p_B 互为非支配解。多目标测试用例预优化的目的就是在测试用例的全排列集合中，寻找最优非支配解集，也就是 Pareto 最优集。

在实际的回归测试中，测试人员会根据个人偏好、测试计划的要求等原因从 Pareto 最优集中选取某一个特定的测试用例排序方案来实施真正的回归测试。因此，多目标测试用例预优化技术实质是为测试人员提供一组可选择的最优测试用例排序方案。

多目标粒子群优化算法如图 1 所示，其中第 2 步评价粒子在每个目标函数上的适应度值，第 3 步则找出当前粒子群中的非支配粒子并将其加入到 archive 中，archive 即为上述形式化定义中的 Pareto 最优集。第 4 步中，condition 一般是指有限的迭代次数或有限的时间。

```
Input：swarm
Output：the archive
1. Initializing the swarm
2. Evaluate the particles in the swarm
3. Insert non-dominated particles to the archive
4. While(condition)
5. {
6.    Update the particles
7.    Evaluate particles and update pbest
8.    Update the archive and gbest
9. }
```

图 1 多目标粒子群算法

3 基于 PSO 的测试用例预优化方法

测试用例预优化是寻找最佳的测试用例执行序列，其本质是在离散空间进行搜索的优化过程。测试用例预优化问题的解空间是测试用例集中所有测试用例的全排列集合。应用 PSO 算法需要定义粒子的编码方式和粒子位置速度的更新方式。本文中，每个粒子表示一个测试用例排列方案，称为测试用例执行序列，对应的编码方式采用序列编码，即对于一个包含 N 个测试用例的测试用例集，粒子的编码为一个长度为 N 的整数数组，数组中第 i 项的值代表第 i 个测试用例在测试用例执行序列中的位置。针对这样的粒子编码方式，本文设计了 PSO 算法中粒子的位置与速度更新方式，具体如下。

3.1 粒子位置及速度更新

PSO 算法中粒子的速度及位置更新方式取决于粒子的编码方式。针对序列编码方式，由于所解决的问题不同，采用的方法也不尽相同。在测试用例预优化问题中，为保证粒子在迭代过程中的多样性，粒子速度及位置的更新方法参照了 GA 中的交叉操作，实验中选用了单点交叉和顺序交叉两种方法。

单点交叉法是一种常见且仿生的交叉方式，适用于二进

制编码，但在序列编码的情况下会产生非法个体，所以 Antoniol 等人针对序列编码的特点对单点交叉方法进行了修改，其具体交叉过程如下：

1)随机选取交叉位置 k，$k\in[0,L]$，L 为个体的编码长度；

2)1 号父个体的前 k 个基因直接作为 1 号子个体的前 k 个基因；

3)将 2 号父个体中与 1 号个体的前 k 个基因中相同的基因去掉，得到一个长度为 $L-k$ 的新基因片段，并将其作为 1 号子个体的后 $L-k$ 个基因；

4)2 号子个体的产生过程与 1 号子个体类似。

顺序交叉法注重序列的特性，即基因的顺序比基因本身更重要。因此，该方法既保留一部分原有的基因排列片段，又融合其他个体的基因排列片段。在本文的测试用例预优化问题中，顺序交叉具体过程如下：

1)随机产生两个交叉点 k_1 和 k_2，$k_1,k_2\in[0,L]$，L 为个体的编码长度；

2)两个交叉点之间的基因片段直接作为子个体的对应位置的基因片段；

3)从第二个交叉点开始分别循环遍历两个父个体的编码序列直至回到第二个交叉点，对应产生两个新基因序列 O_1 和 O_2；

4)从 O_1 和 O_2 中分别去掉两个子个体中已有的基因，从而得到新基因片段 O_1' 和 O_2'；

5)分别从两个子个体的第二个交叉点开始依次放入 O_1' 和 O_2' 对应的基因，最终形成两个编码完整的子个体。

粒子通过单点交叉和顺序交叉进行位置及速度的更新，其更新公式如式(3)－式(5)所示。其中，k 表示迭代次数，$\oplus$ 表示一种交叉操作。通过对上一次迭代产生的个体最优及全局最优进行交叉得到一个速度增量 $v_i'(k+1)$，之后 $v_i'(k+1)$ 与上一次迭代个体的速度交叉得到新的速度 $v_i(k+1)$，再用粒子的位置与新的速度 $v_i(k+1)$ 进行交叉得到粒子新的位置。

$$v_i'(k+1)=p_i(k)\oplus p_g(k) \tag{3}$$

$$v_i(k+1)=v_i'(k+1)\oplus v_i(k) \tag{4}$$

$$x_i(k+1)=x_i(k)\oplus v_i(k+1) \tag{5}$$

3.2 *archive* 更新

基于 PSO 的多目标测试用例预优化方法中，*archive* 中存储着粒子群在迭代过程中产生的所有 Pareto 非支配粒子。在每次迭代时，需要将当前粒子群的 Pareto 非支配粒子加入到 *archive* 中。新加入的粒子可能被 *archive* 中的其它粒子支配或支配 *archive* 中的粒子，所以加入新的粒子后需重新计算 Pareto 非支配粒子集并更新 *archive*。

3.3 全局最优更新

全局最优是指初始粒子和在迭代过程中生成的所有粒子中最优的粒子。由式(3)可以看出，全局最优对粒子的速度及位置的变化有一定的指导作用。*archive* 中存储的非支配解集中，每个粒子相对于其它粒子而言都是最优的，而在计算过程中只需要一个粒子，所以需要从 *archive* 中选取一个粒子。一般情况下，可以根据 *archive* 中粒子间距选择全局最优，但这样会引入大量的计算，同时也不利于全局最优的离散性。在本文的方法中，从 *archive* 中随机选择粒子作为全局最优，既可增加全局最优的离散性，又能平衡算法的收敛速度。

4 实验及结果分析

4.1 研究问题

在基于 PSO 的测试用例预优化方法中，对结果有显著影响的因素主要有粒子的更新方式和迭代次数及粒子群大小，所以本文从以下 3 个方面开展实验研究：

1)粒子更新方式对结果的影响；

2)迭代次数及粒子群大小对结果的影响；

3)同等实验条件下本文方法与基于 NSGA-Ⅱ算法的测试用例预优化方法的比较。

4.2 实验设计

为回答上述问题，本文选取测试用例序列的平均语句覆盖率(APSC)和测试用例序列的有效执行时间(EXT)作为优化目标。APSC 是对测试用例序列更快达到更高语句覆盖率的度量，EXT 表示测试用例序列首次达到最大语句覆盖率时所执行的测试用例的总时间。两个优化目标的计算公式如式(6)和式(7)所示。

$$APSC=1-\frac{TS_1+TS_2+\cdots+TS_M}{NM}+\frac{1}{2N} \tag{6}$$

$$EXT=\sum_{i=1}^{N'}ET_i \tag{7}$$

本文实验从 Software-artifact Infrastructure Repository (SIR)[1] 中选取了 8 个被测程序，规模及测试用例池大小如表 1 所列。

表 1 被测程序的统计信息

被测程序	有效语句数	用例池规模
printtokens	209	4130
printtokens2	198	4115
replace	273	5542
schedule	129	2650
schedule2	135	2710
tcas	73	1608
flex	3406	566
space	3827	13550

其中，前 6 个较小规模的被测程序由西门子研究实验室编写；后 2 个规模较大的程序中，flex 是一个 Unix 词法分析器，space 程序是由欧洲航天局开发的一个 ADL 语言解释器。8 个被测程序的测试用例池中存在大量冗余的测试用例，在实验准备阶段先从用例池中选取出冗余度较小的目标测试用例集，选取准则是使目标测试用例集的语句覆盖率达到用例池的最大语句覆盖率。

分别针对不同的测试用例集规模、不同的粒子群大小，采用本文方法及基于 NSGA-Ⅱ的方法进行测试用例预优化实验。实验相关参数设置如下：

1)PSO 算法的粒子群大小分别为 30，50，80，100，150；NSGA-Ⅱ算法的种群大小为 128；

2)两种算法最大迭代次数都设置为 200；

3)实验重复 30 次。

本文算法在 DEV C++4.9.9.2 平台上采用 C 语言实现。实验运行在 Windows 7 Ultimate 64 位操作系统上，使用

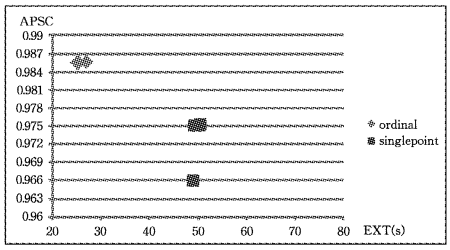
[1] http://sir.unl.edu/portal/index.php

Intel Core i5-2300 CPU，该 CPU 具有 2.8GHz 的时钟频率和 4GB 的内存空间。

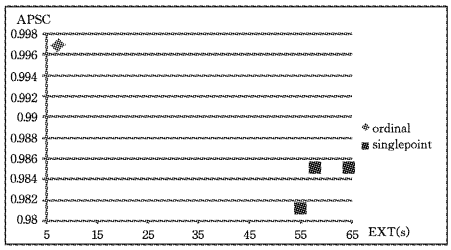
4.3 结果分析

1)粒子更新方式对测试用例预优化结果的影响

实验采用 3.1 节中介绍的两种粒子更新方式，其中粒子群规模设定为 100，迭代次数为 100(实验得到的经验值)。实验结果表明，对于较小规模的被测程序，顺序交叉和单点交叉两种粒子更新方式对最优解集没有显著影响；而对于较大规模的被测程序，顺序交叉优于单点交叉的更新方式。图 2(a)、(b)分别给出了两种粒子更新方式下 space 程序和 flex 程序最优解集的分布。图中横坐标表示测试用例序列的有效执行时间(EXT)，纵坐标表示测试用例序列的平均语句覆盖率(APSC)，图中的点表示一个最优非支配解。从图 2 可以看出，使用顺序交叉的粒子更新方式得到的最优解集优于使用单点交叉的粒子更新方式。原因主要是单点交叉方式下，新粒子只能保留原粒子编码的前半段，后半段的编码会重新排列，在计算 APSC 及 EXT 时也主要依赖粒子编码的前半段；而顺序交叉方式下，保留下来的是整个编码中随机选择的一段，这样粒子对整个解空间的搜索能力就强一些。



(a) space

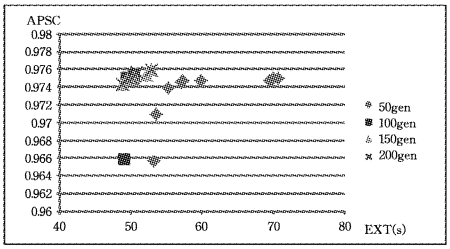


(b) flex

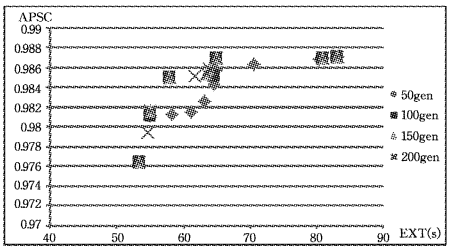
图 2 不同粒子更新方式下最优解的分布

2)迭代次数及粒子群大小对测试用例预优化结果的影响

本文首先比较了一定粒子群大小下不同迭代次数对测试用例预优化结果的影响。实验结果表明，针对较小规模的被测程序，结果收敛较快，迭代次数在 50 以上就对最优解集没有显著影响了。对于较大规模的被测程序，图 3(a)、(b)分别给出了在粒子群规模为 100 的前提下不同迭代次数时 space 和 flex 程序最优解集的分布。从图 3 可以看出，随着迭代次数的增加，本文方法得到的非支配解集不断优化；而在迭代次数达到 100 后，非支配解集不再出现显著变化。所以对于一定规模的被测程序而言，得到稳定非支配解集所需要的迭代次数会保持在一个相对稳定的数值。这是因为粒子群内部的交互机制只涉及个体最优及全局最优，并且只有在一次迭代完成之后，粒子群算法才会对全局最优进行更新，这样信息在粒子群内的传递量和迭代次数之间就存在相关性；而一定规模的被测程序达到稳定非支配解集所需的信息传递量也是一定的。



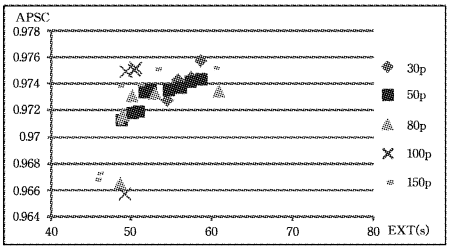
(a) space



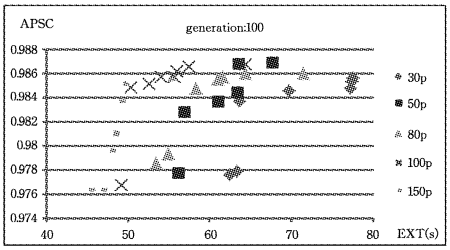
(b) flex

图 3 不同迭代次数下最优解在解空间的分布

其次，本文比较了一定迭代次数下不同粒子群大小对最优解集的影响。对于较小规模的程序，粒子群规模达到 30 以上就对结果没有显著影响了。针对较大规模的程序，图 4(a)、(b)分别给出了在迭代次数为 100 次的前提下不同粒子群规模时 space 和 flex 最优解集的分布。从图 4 可以看出，在相同的迭代次数下，粒子群规模的增大能够在一定程度上优化非支配解集。这是因为粒子群中粒子的初始化是随机的，粒子群规模的增大提升了粒子的多样性，使得非支配解集在解空间的分布更加离散。



(a) space



(b) flex

图 4 不同粒子群规模下最优解在解空间的分布

3)本文方法与基于 NSGA-Ⅱ算法的方法在执行效率上的比较

由图 3 可以看出，针对一定规模的程序，本文方法在 100 次迭代时就达到了稳定非支配解集，而实验中发现基于 NSGA-Ⅱ的方法在 300 次迭代后得到的非支配解集仍未稳定。图 5 给出了 PSO 算法迭代 100 次与 NSGA-Ⅱ算法迭代 300 次得到的最优解集的分布，从中可以看出 PSO 算法得到的非支配解集优于 NSGA-Ⅱ算法。所以，本文方法相比NSGA-Ⅱ能够在较少的迭代次数得到较优的非支配解集。

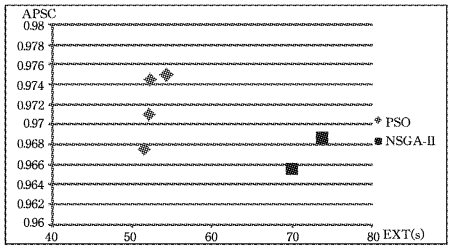


图 5　PSO 算法与 NSGA-Ⅱ算法的 Pareto 集合

为了进一步比较两种算法的执行效率，在实验中分别统计了两种算法在同一种群规模下迭代 100 次的平均执行时间，如表 2 所列，可以看出，本文方法的执行时间明显少于 NSGA-Ⅱ。其主要原因是：(1)NSGA-Ⅱ涉及了更多的操作如变异、精英选择；(2)NSGA-Ⅱ种群内的交互机制要更为复杂，在一次迭代中个体之间相互影响。这些都会对算法的性能产生影响。

表 2　PSO 方法与 NSGA-Ⅱ方法的平均执行时间

方法名称	平均有效执行时间(s/100generation)
PSO	65
NSGA-Ⅱ	139

5　相关工作研究

基于搜索的软件回归测试用例预优化近年来受到广泛研究。Wong 等人最早提出测试用例预优化的概念及方法[9]，Harrold 和 Rothermel 进一步发展了这一技术，并对其进行了评估[10]。在基于进化算法的多目标回归测试优化的研究中，Yoo 研究了基于 NSGA-Ⅱ的多目标测试用例选择技术[11]，程俊研究了基于 NSGA-Ⅱ的测试用例预优化技术及其并行化[12]。除此之外，还有研究者结合其它技术进行多目标测试用例预优化。文献[13]在测试用例预优化技术中引入了 Software Agents 及 Fuzzy Logic 的概念，即把每个软件模块及每个测试用例都视作 Software Agent，通过计算得到每个测试用例的重要程度并以之为标准对测试用例进行排序。文献[14]在测试用例预优化技术中引入了机器学习的概念，设计实现了一种基于测试用例推理的优化方法。然而这些技术只在特定的问题上有着较好的表现。还有一种研究则是针对优化目标的，文献[15]就是考虑回归测试中检测到的错误之间的依赖关系，给出了新的算法，并通过实验对新算法进行了分析及评估。

另一方面，PSO 算法由于其简单有效得到了许多研究者的青睐。很多是针对 PSO 算法本身的研究，比如各种参数的设定、种群信息交互机制的改进以及改进算法搜索过程的有效性等等。Kusum 等人为平衡 PSO 算法初始化过程中随机解的分布及算法易陷入局部最优问题，引入了混乱(Chaos)机制，虽解决了问题，但同时也使得算法产生大量的重复中间解[16]。廖子贞等在粒子群优化算法中引入自适应惯性权重并对粒子群进行分组，使得算法的收敛速度及精度都有所提升，此外还使用了多线程技术对算法进行了并行处理[17]。此外，PSO 算法已成功应用到解决工作流分配、旅行商问题等离散问题上，但在测试用例预优化方面并没有太多的研究。Lian 等人在处理工作流分配问题上结合 GA 算法设计了新的位置及速度的更新公式，对如何使用 PSO 算法处理离散问题具有一定的指导意义[18]。Khin 等人应用 PSO 算法解决嵌入式实时系统中测试用例预优化问题，主要根据实时系统的变化来快速衡量测试用例的重要性，但文中只对较小规模的被测程序进行了实验验证[19]。沈恺涛与胡德敏基于改进 PSO 算法设计了一种云计算环境下的任务调度方法，通过粒子群间的协同寻优获取最优解[20]。

结束语　本文提出了一种基于 PSO 的测试用例预优化方法，其能够有效处理实际软件回归测试中同时满足多个测试准则的问题。同采用 NSGA-Ⅱ方法的测试用例预优化方法相比，本文方法可以在较少的时间内得到较优的最优解集，提高了多目标测试用例预优化的效率。

同时实验结果表明，在采用 PSO 的测试用例预优化方法中，粒子状态的更新方式对结果有显著的影响，应根据实际问题选取适当的粒子更新方式；另一方面，粒子群越大，最优非支配解集越好，但粒子群的增大使得每一次迭代的计算量增大，而针对一定规模的被测程序，得到稳定最优解集的迭代次数也相对稳定。因此在实际中，也需要根据问题的规模，选取合适的粒子群大小及迭代次数。

本文下一步的工作主要包括：

1)针对测试用例预优化问题设计更为有效的群间交互策略，本文使用 GA 算法中的交叉操作作为群间交互手段，虽然可以获得较好的结果，但也引入了大量的计算。设计更加有效的交互策略可以进一步提升方法的效率。

2)本文基于 PSO 的测试用例预优化方法在执行效率上要优于现有的 NSGA-Ⅱ方法，但由于 PSO 算法也是采用种群迭代的，随着被测程序规模及测试用例集规模的增加，效率也在降低，因此后续的研究拟采用并行方法进一步提高测试用例预优化的效率。

参 考 文 献

[1] Harrold M J. Testing: a roadmap[C]// Proceedings of the conference on the future of software engineering. ACM, 2000: 61-72

[2] Li Z, Harman M, Hierons R M. Search algorithms for regression test case prioritization[J]. IEEE Transactions on Software Engineering, 2007, 33(4): 225-237

[3] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey[J]. Software Testing, Verification and Reliability: 2012, 22(2): 67-120

[4] Harman M, Jones B F. Search-based software engineering[J]. Information and Software Technology, 2001, 43(14): 833-839

[5] Zhang T, Brorsen B W. Particle swarm optimization algorithm for agent-based artificial markets[J]. Computational Economics, 2009, 34(4): 399-417

[6] Zhu H, Wang Y, Wang K, et al. Particle Swarm Optimization (PSO) for the constrained portfolio optimization problem[J]. Expert Systems with Applications, 2011, 38(8): 10161-10169

[7] Kennedy J, Eberhart R. Particle swarm optimization[C]// IEEE International Conference on Neural Networks, 1995. IEEE, 1995, 4: 1942-1948

[8] Coello Coello C A, Lechuga M S. MOPSO: A proposal for multiple objective particle swarm optimization[C]// Proceedings of the 2002 Congress on Evolutionary Computation, 2002 (CEC'02). IEEE, 2002, 2: 1051-1056

[9] Leung H K N, White L. Insights into regression testing [software testing][C]// Conference on Software Maintenance, 1989. IEEE, 1989: 60-69

[10] Harrold M J. Testing evolving software[J]. Journal of Systems

and Software,1999,47(2):173-181
[11] Yoo S,Harman M. Pareto efficient multi-objective test case selection[C]//Proceedings of the 2007 international symposium on Software testing and analysis. ACM,2007:140-150
[12] 程俊,李征,赵瑞莲. CPU+GPU 异构并行多目标测试用例预优化技术[C]//第七届中国测试学术会议(CTC2012). 2012,6:116-121
[13] Malz C,Jazdi N,Gohner P. Prioritization of Test Cases Using Software Agents and Fuzzy Logic[C]//2012 IEEE Fifth International Conference on Software Testing,Verification and Validation (ICST). IEEE,2012:483-486
[14] Tonella P, Avesani P, Susi A. Using the case-based ranking methodology for test case prioritization[C]//22nd IEEE International Conference on Software Maintenance,2006(ICSM'06). IEEE,2006:123-133
[15] Kayes M I. Test case prioritization for regression testing based on fault dependency[C]//2011 3rd International Conference on Electronics Computer Technology (ICECT). IEEE,2011,5:48-52
[16] Deep K,Chauhan P,Pant M. Totally disturbed chaotic Particle Swarm Optimization[C]//2012 IEEE Congress on Evolutionary Computation (CEC). IEEE,2012:1-8
[17] 廖子贞,罗可,周飞红,等. 一种自适应惯性权重的并行粒子群聚类算法[J]. 计算机工程与应用,2007,43(28):166-168
[18] Lian Z,Gu X,Jiao B. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan[J]. Applied Mathematics and Computation,2006,175(1):773-785
[19] Hla K H S,Choi Y S,Park J S. Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting[C]//IEEE 8th International Conference on Computer and Information Technology Workshops,2008(CIT Workshops 2008). IEEE,2008:527-532
[20] 沈恺涛,胡德敏. 基于云计算和改进离散粒子群的任务调度研究[J]. 计算机测量与控制,2012,20(011):3070-3072

(上接第 67 页)

可靠性增长幅度,N 表示监控点数目,λ_m 表示监控频率之和。其次,我们假设系统的可靠性约束不变,为 0.95,t=10h,采用了 10 组随机选择的组件可靠性参数数据。同理,单目标算法只能获得一组解,并且被包含于 MOMA 方法获得的非支配集中。其中 4 组数据获得的详细结果如图 5(b)所示。根据图 5 可以看出,单目标的解被包含在多目标的非支配集中,并且多目标方法提供了更多种监控配置方式选择的方案。虽然部分情况下单目标方法看似比 MOMA 的解更好,但在此情况下单目标方法的解达不到可靠性约束条件要求。

结束语 由于分布式系统中的组件来自于网络,不易再通过传统的测试手段来维护和提高其可靠性,因此监控被作为一种常用手段用于系统可靠性的保障中。除了组件的冗余,监控也将对系统的可靠性产生一定影响,同时也将给系统带来额外的开销,然而服务监控在可靠性分配中的作用却很少得到研究。本文在传统面向可靠性优化设计的资源分配问题的基础上加入了监控机制的影响,通过分配监控资源对系统可靠性进行优化。首先分析了分布式冗余系统中基于监控的组件可靠性模型,建立了基于不同监控点的整体系统可靠性模型。在分析监控代价的基础上,改进了传统的测试资源模型,形成了监控资源分配模型,提出在可靠性约束条件下将监控点的数目和监控频率之和作为系统可靠性优化设计的两大资源优化目标。然后采用遗传算法解决了该问题。为了分析监控资源分配对系统可靠性的影响,本文分别对单目标和多目标监控资源分配进行了实验,实验结果表明:①随着可靠性约束的提高,需要增加系统中被监控的组件数目或者提高组件监控频率。②与单目标分配相比,多目标分配方法能够提供更好的资源优化方案。

参 考 文 献

[1] Ben H R,Emna F,Khalil D,et al. A large-scale monitoring and measurement campaign for Web Services-based applications [J]. Concurrency Computation Practice and Experience,2010,22(10):1207-1222
[2] He Pan,Wu Kai-gui,Wen Jun-hao, et al. Monitoring resources allocation for service composition under different monitoring mechanisms[C]//Proceedings of 5th International Conference on Complex,Intelligent and Software Intensive Systems,2011. Washington,D C:IEEE Computer Society,2011:263-270
[3] Dai Yu, Yang Lei, Zhang Bin. QoS-driven self-healing Web service composition based on performance prediction [J]. Journal of Computer Science and Technology,2009,24(2):250-261
[4] Yu Tao, Lin K-J. Adaptive algorithms for finding replacement services in autonomic distributed business processes[C]//Proceedings of International Symposium on Autonomous Decentralized Systems,2005. Washington,D C:IEEE Computer Society,2005:427-434
[5] Girish C,Koustuv D,Arun K,et al. Adaptation in Web Service composition and execution[C]//Proceedings of IEEE International Conference on Web Services, 2006. Washington, D C: IEEE Computer Society,2006:549-557
[6] Gerardo C, Di P M, Raffaele E, et al. QoS-aware replanning of composite Web services[C]//Proceedings of IEEE International Conference on Web Services, 2005. Washington, D C: IEEE Computer Society,2005:121-129
[7] Rüdiger R. Reliability analysis—a review and some perspectives [J]. Structural Safety,2001,23(4):365-395
[8] Gokhale S S, Trivedi K S. Analytical models for architecture-based software reliability prediction: A unification framework [J]. IEEE Transactions on Reliability,2006,55(4):578-590
[9] Gokhale S S. Architecture-based software reliability analysis: Overview and limitations [J]. IEEE Transactions on Dependable and Secure Computing,2007,4(1):32-40
[10] Wang Li-jun,Bai Xiao-ying,Zhou Li-zhu,et al. A hierarchical reliability model of service-based software system[C]//Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference, 2009. Washington, D C: IEEE Computer Society,2009:199-208
[11] Wang Zai,Tang Ke,Yao Xin. Multi-objective approaches to optimal testing resource allocation in modular software systems [J]. IEEE Transactions on Reliability,2010,59(3):563-575
[12] Xia Yun-ni,Wang Hanp,Huang Y,et al. A stochastic model for workflow QoS evaluation [J]. Scientific Programming,2006,14(3/4):251-265
[13] Directory of Public Soap Web Services[EB/OL]. http://www.service-repository.com