



## 动态可靠性约束的多阶段测试资源分配研究

占德志<sup>1</sup>, 张国富<sup>1,2,3</sup>, 苏兆品<sup>1,2,3</sup>, 岳 峰<sup>1,2</sup>

(1. 合肥工业大学 计算机与信息学院, 合肥 230601; 2. 合肥工业大学 工业安全与应急技术安徽省重点实验室, 合肥 230601;  
3. 安全关键工业测控技术教育部工程研究中心, 合肥 230601)

**摘 要:** 为满足测试资源分配过程中用户对软件可靠性的需求, 构建一种动态可靠性约束的多阶段多目标测试资源分配模型 DRC-MSMOTRA。从理论上分析不同阶段满足可靠性约束的测试时间下限并设计相应的种群初始化策略, 结合参数估计、加权归一化方法和多目标差分进化, 提出一种动态可靠性约束的多阶段多目标测试资源分配算法 MS-DRC-GDE3。实验结果表明, 与 MSMOTRA 模型相比, DRC-MSMOTRA 模型在 2 种不同规模的软件系统上所获解的覆盖值分别提高约 62 和 59 个百分点, 与 MS-GDE3 算法相比, MS-DRC-GDE3 算法在 2 种软件系统上所获解的覆盖值分别提高约 69 和 80 个百分点, 即所提模型和算法能够根据用户对可靠性的需求来为用户提供更多更优的测试资源分配方案。

**关键词:** 软件可靠性; 测试资源分配; 动态可靠性约束; 加权归一化; 多目标差分进化

开放科学(资源服务)标志码(OSID):



**中文引用格式:** 占德志, 张国富, 苏兆品, 等. 动态可靠性约束的多阶段测试资源分配研究[J]. 计算机工程, 2021, 47(2): 246-253, 260.  
**英文引用格式:** ZHAN Dezhi, ZHANG Guofu, SU Zhaopin, et al. Research on multi-stage testing resource allocation with dynamic reliability constraints[J]. Computer Engineering, 2021, 47(2): 246-253, 260.

## Research on Multi-Stage Testing Resource Allocation with Dynamic Reliability Constraints

ZHAN Dezhi<sup>1</sup>, ZHANG Guofu<sup>1,2,3</sup>, SU Zhaopin<sup>1,2,3</sup>, YUE Feng<sup>1,2</sup>

(1. School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China;

2. Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei University of Technology, Hefei 230601, China;

3. Engineering Research Center of Safety Critical Industry Measure and Control Technology, Ministry of Education, Hefei 230601, China)

**[Abstract]** To meet the user's requirements for software reliability in testing resource allocation, this paper constructs a multi-stage multi-objective testing resource allocation model called DRC-MSMOTRA with dynamic reliability constraints. The lower bounds of the testing time to meet the reliability constraints in different stages are theoretically analyzed and the corresponding population re-initialization strategy is designed. Then a multi-stage multi-objective testing resource allocation algorithm with dynamic reliability constraints, MS-DRC-GDE3, is developed according to parameter estimation, weighted normalization and multi-objective differential evolution. The experimental results show that compared with the MSMOTRA model, the proposed DRC-MSMOTRA model increases the coverage value of the obtained solution on two different scales of software systems by 62 and 59 percentage points respectively. Compared with the MS-GDE3 algorithm, the proposed MS-DRC-GDE3 algorithm increases the coverage value of the obtained solution on two software systems by 69 and 80 percentage points respectively, which demonstrates that the proposed model and algorithm can provide more optimized testing resource allocation schemes for users based on their requirements for reliability.

**[Key words]** software reliability; testing resource allocation; dynamic reliability constraint; weighted normalization; multi-objective differential evolution

DOI: 10.19678/j.issn.1000-3428.0057688

**基金项目:** 国家自然科学基金(61573125); 教育部人文社会科学研究青年基金项目(19YJC870021, 18YJC870025); 中国工程院咨询研究重点项目(2020-XZ-3); 中央高校基本科研业务费专项资金(PA2019GDQT0008, PA2019GDPK0072)。

**作者简介:** 占德志(1993—), 男, 硕士研究生, 主研方向为软件工程; 张国富, 教授、博士; 苏兆品, 副教授、博士; 岳 峰, 副研究员、博士。

**收稿日期:** 2020-03-12 **修回日期:** 2020-04-14 **E-mail:** 876821246@qq.com

## 0 概述

软件测试的目的是检测软件故障、发现和纠正软件缺陷从而提高软件的可靠性, 其对软件项目开发至关重要<sup>[1-3]</sup>。对于一个软件开发项目而言, 有接近一半的软件开发资源耗费于软件测试<sup>[4]</sup>。实际可分配的测试资源往往有限, 因此, 在软件测试中, 软件项目经理的首要任务是制定合理且高效的测试资源分配方案, 实现以最小代价(指测试成本和测试资源消耗)获得最大软件可靠性<sup>[5-7]</sup>的目的。

近年来, 为了在软件可靠性、测试成本和测试资源之间实现一个理想的均衡, 测试资源分配被描述为一个多目标优化问题, 以最大程度地提高软件可靠性并尽可能地降低测试成本和测试资源消耗<sup>[8-10]</sup>。部分多目标进化算法(Multi-Objective Evolutionary Algorithms, MOEAs)被用于解决多目标测试资源分配问题, 如非支配排序遗传算法<sup>[11-12]</sup>和加权归一化多目标差分进化算法<sup>[13]</sup>等。与单目标优化算法只输出单个解相比, MOEAs可以得到一个 Pareto 最优解集, 从而为用户提供多种选择, 这些不同的选择分别对应可靠性、成本和测试时间之间不同的折中方案, 因此, 可以帮助用户对整个测试周期进行更加合理的安排<sup>[14]</sup>。

但是, 上述研究均针对静态优化环境, 都是简单地假设软件测试只有一个完整的静态阶段。在实际的软件测试过程中, 整个测试周期往往会被划分成若干个测试阶段, 在不同测试阶段用户可能有不同的需求, 这就需要根据每个阶段的测试需求动态制定不同的测试资源分配方案。为此, LEUNG<sup>[15]</sup>利用拉格朗日乘法最小化每个阶段的平均错误数, 该方法属于单目标多阶段测试资源分配方案。在多阶段多目标测试资源分配(Multi-Stage Multi-Objective Testing Resource Allocation, MSMOTRA)方面, 陆阳等人<sup>[16]</sup>利用多目标差分进化算法, 在每个阶段根据不同的测试时间总量来实现最大化可靠性和最小化测试成本的目的, 但是, 该方法忽略了一个重要细节, 即软件中的模块在经历了上一阶段的测试后其关键模块参数(如剩余错误数、错误检测率等)都已经发生变化, 如果还是按照初始参数进行优化, 将缺少前一测试阶段的测试结果反馈, 从而导致算法不能达到更深层的收敛, 给出的解往往严重偏离实际最优解。为了解决这一问题, 牛福强<sup>[17]</sup>首先根据上一阶段的测试资源分配方案和测试结果对模块参数进行重新估计, 然后利用新的模块参数和第三代广义多目标差分进化(Generalized Differential Evolution 3, GDE3)算法<sup>[18]</sup>在当前阶段进行优化。但是, 该方法在选择每个阶段的最优方案时偏好性太强, 只考虑最优非支配解集中可靠性最高的解, 而没有考虑测试成本和测试时间, 不能充分体现多目标优化的优势。

虽然已有研究可以很好地解决多目标测试资源分配问题, 但是通常只关注测试时间总量约束, 没有考虑用户对软件的可靠性需求, 导致在解集中存在许多可靠性非常低的测试资源分配方案, 这些无用解带来了巨大的计算开销和信息冗余, 也背离了研究多目标测试资源分配问题的初衷。

本文从用户对软件可靠性需求的角度出发, 构建一种动态可靠性约束的多阶段多目标测试资源分配模型, 基于每个阶段不同的可靠性约束来为各个模块分配一个测试时间下限, 以降低搜索空间。基于 GDE3、参数估计、种群重新初始化和最优方案选择, 提出一种多阶段多目标测试资源分配算法, 以在满足用户对软件可靠性需求的同时为用户提供满意的解集。

## 1 问题描述

与惯例相同<sup>[11-13]</sup>, 本文基于经典的并串联模块软件系统研究测试资源分配问题。如图 1 所示, 软件系统包括  $m \in \mathbb{N}$  个子系统, 每个子系统  $S_j$  ( $j \in (1, 2, \dots, m)$ ) 包含  $n_j \in \mathbb{N}$  个模块。测试资源分配的目的是给系统中的每个模块  $M_{jk}$  ( $k \in (1, 2, \dots, n_j)$ ) 分配合适的测试资源, 以提高系统的整体可靠性。

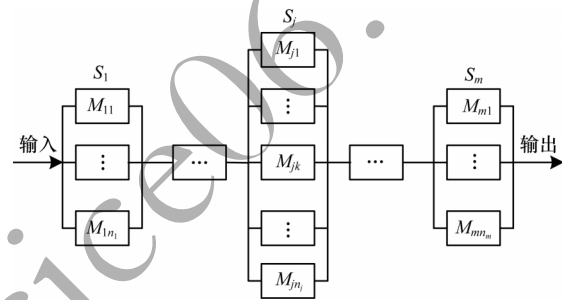


图1 并串联模块软件系统结构

Fig.1 Structure of parallel-series modular software system

在软件测试过程中, 主流的测试资源分配问题均是针对测试时间分配<sup>[11-13]</sup>, 考虑如何合理分配软件测试人员的总工作时间(即软件测试人员数乘以每个测试人员的工作时间, 通常以小时为单位)。与多数已有研究相同<sup>[11-13]</sup>, 本文也依据测试时间来分析测试资源分配问题。不失一般性, 本文采用文献[17]中测试阶段的划分方式, 假设总的测试时间  $T^*$  被划分成  $p \in \mathbb{N}$  个连续的测试阶段, 其中, 第  $i \in (1, 2, \dots, p)$  个阶段的测试时间预分配量为  $T_i^* \geq 0$ , 满足  $\sum_{i=1}^p T_i^* = T^*$ 。设模块  $M_{jk}$  在阶段  $i$  分配到的测试时间为  $t_{jk}^i$ , 则阶段  $i$  的实际分配总量  $T^i$  应该小于前  $i$  个阶段的预分配量之和减去前  $i-1$  个阶段的实际分配量之和<sup>[17]</sup>, 计算如下:

$$T^i = \sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^i \leq \sum_{l=1}^i T_l^* - \sum_{l=1}^{i-1} \sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^l \quad (1)$$

在前面每个阶段选取的分配方案不一定恰好用完预分配量, 因此, 可能在每个阶段都有少量的剩余测试时间可供下一阶段使用。

本文使用软件可靠性增长模型<sup>[6]</sup>来描述模块可靠性和测试时间的关系,即模块 $M_{jk}$ 在阶段 $i$ 可达的可靠性 $r_{jk}^i$ 满足<sup>[17]</sup>:

$$r_{jk}^i = \exp\left[-\lambda \cdot a_{jk}^i \cdot b_{jk}^i \cdot \exp(-b_{jk}^i \cdot t_{jk}^i)\right] \quad (2)$$

其中, $\lambda$ 表示系统的估算寿命,即最大可靠服务时间(以小时为单位), $a_{jk}^i \in \mathbb{R}$ 与 $b_{jk}^i \in \mathbb{R}$ 分别表示模块 $M_{jk}$ 在阶段 $i$ 的剩余错误总数和错误检测率。

子系统 $S_j$ 是由 $n_j$ 个子模块并联组成的,当子模块都不工作时,该子系统寿命结束,可以根据 $M_{jk}$ 模块的可靠性求出子系统 $S_j$ 在第 $i$ 个测试阶段的可靠性<sup>[17]</sup>,计算如下:

$$R_j^i = 1 - \prod_{k=1}^{n_j} (1 - r_{jk}^i) \quad (3)$$

根据文献[17]中对串并联模块软件系统的可靠性定义,第 $i$ 个测试阶段的系统可靠性 $R^i$ 可定义如下:

$$R^i = \prod_{j=1}^m \left[ 1 - \prod_{k=1}^{n_j} (1 - r_{jk}^i) \right] \geq R_i^* \quad (4)$$

其中, $R_i^*$ 为用户在测试阶段 $i$ 的可靠性要求,满足 $0 \leq R_1^* \leq R_2^* \leq \dots \leq R_i^* \leq \dots \leq R_p^* \leq 1$ 。

模块 $M_{jk}$ 在阶段 $i$ 的测试成本 $C_{jk}^i$ 可以根据其达到的可靠性 $r_{jk}^i$ 进行估计,一般认为模块的可靠性越高,需要消耗的测试成本越大<sup>[17]</sup>,计算如下:

$$C_{jk}^i = c_1^{jk} \cdot \exp(c_2^{jk} \cdot r_{jk}^i - c_3^{jk})$$

其中, $c_1^{jk}, c_2^{jk}, c_3^{jk} \in \mathbb{R}$ 为成本估算参数,它们控制成本的增长幅度<sup>[12]</sup>。据此,测试阶段 $i$ 的系统总成本为<sup>[17]</sup>:

$$C^i = \sum_{j=1}^m \sum_{k=1}^{n_j} C_{jk}^i \quad (5)$$

综上,本文所研究的动态可靠性约束的多阶段多目标测试资源分配问题(DRC-MSMOTRA)可以形式化表示为:

$$\begin{cases} \min f_1 = 1 - R^i \\ \min f_2 = C^i \\ \min f_3 = T^i \\ \text{s.t.} \\ R^i \geq R_i^*, i=1, 2, \dots, p \\ T^i \leq \sum_{l=1}^i T_l^* - \sum_{l=1}^{i-1} \sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^l, i=1, 2, \dots, p \\ 0 \leq t_{jk}^i \leq T^i, i=1, 2, \dots, p, j=1, 2, \dots, m, k=1, 2, \dots, n_j \end{cases} \quad (6)$$

在每个测试阶段需要满足不同的可靠性约束和测试时间约束,因此,式(6)是一个典型的动态约束多目标优化问题,解决该问题的难点是每个阶段的解空间都大不相同,传统的优化方法很难适应搜索空间的动态变化,因此,本文采用适应性更高的多目标进化算法进行求解。

## 2 多阶段多目标测试资源分配算法设计

本文选择广义多目标差分进化算法GDE3<sup>[18]</sup>作为DRC-MSMOTRA问题的基本求解框架,这是因为

GDE3在针对3个目标优化问题时相比非支配排序遗传算法<sup>[11-12]</sup>速度更快,需要设置的参数更少,算法收敛性更好。关于GDE3的详细介绍可参考文献[18],本文结合基本GDE3,提出多阶段动态可靠性约束处理算法MS-DRC-GDE3。MS-DRC-GDE3算法的每个个体采用一维实数编码,编码中的每一个基因位代表一个模块被投入的测试时间。每个个体的优劣用式(6)中的 $f_1, f_2, f_3$  3个函数同时进行评估。

MS-DRC-GDE3算法流程如图2所示,具体步骤为:

**步骤1** 判断 $i$ 是否已达到最大阶段数 $p$ ,如果已达到,则结束算法;否则,执行步骤2。

**步骤2** 判断当前阶段是否为第1个阶段,如果不是,则根据前面阶段的测试结果对所有模块重新进行参数估计,然后根据新的模块参数对种群重新进行初始化以生成父代种群。

**步骤3** 对种群执行选择、差分变异和交叉等进化操作以生成子代种群,再将父代和子代种群一起进行非支配排序和拥挤度计算,选择最优的个体组成新的父代种群。

**步骤4** 判断算法是否达到本阶段的最大迭代次数,如果未达到,则执行步骤3;否则,对最后的最优解集执行加权归一化处理,选择适应度最小的解作为阶段 $i$ 的最优分配方案,当阶段 $i$ 测试完毕,进入下一阶段, $i=i+1$ ,执行步骤1。

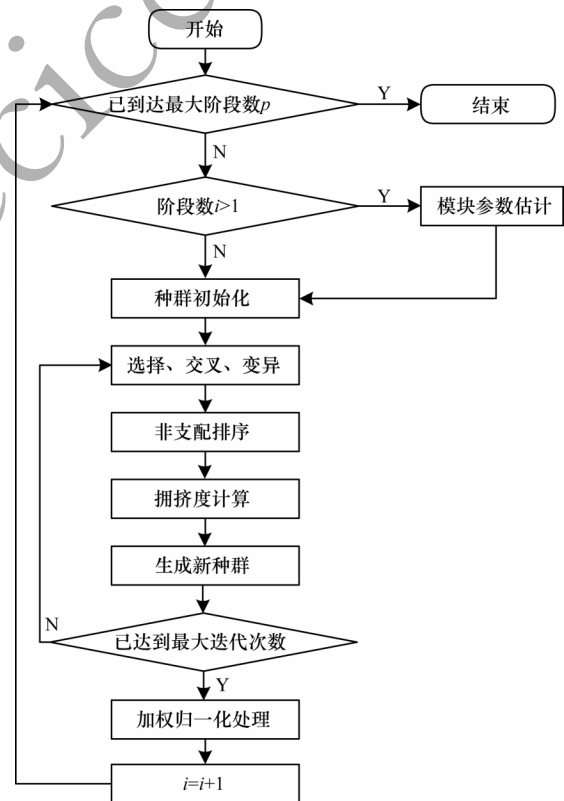


图2 MS-DRC-GDE3算法流程

Fig.2 Procedure of MS-DRC-GDE3 algorithm



## 2.1 模块参数估计

在多阶段测试资源分配中, 前面阶段的测试结果会直接影响模块的关键参数, 包括剩余错误总数  $a_{jk}^i$  与错误检测率  $b_{jk}^i$ 。因此, 在一个测试阶段  $i$  执行完毕后, 需要对  $i+1$  阶段的模块参数重新进行估计, 否则会误导算法的进化。在软件测试中, 通常的做法是收集在每个模块上消耗的测试时间和检测出的错误数, 然后利用这些数据对模块的参数进行极大似然估计<sup>[17]</sup>, 计算如下:

$$\begin{cases} a_{jk}^{i+1} = a_{jk}^i - \sum_{l=1}^i E_{jk}^l \\ b_{jk}^{i+1} = -\frac{1}{t_{jk}^i} \ln \left( 1 - \frac{E_{jk}^i}{a_{jk}^{i+1} + \sum_{l=1}^i E_{jk}^l} \right) \end{cases} \quad (7)$$

其中,  $E_{jk}^i$  为模块  $M_{jk}$  在阶段  $i$  检测出的错误总数。通过求解上述二元一次方程组的通解, 可以估计出新的参数  $a_{jk}^{i+1}$  和  $b_{jk}^{i+1}$ 。

## 2.2 种群初始化

在 DRC-MSMOTRA 问题中, 每个阶段的可靠性约束和测试时间约束均不相同, 因此, 在种群进化前需要对种群按照新的约束空间重新进行初始化。鉴于每个阶段的迭代次数不可能无限大, 因此, 如何在较少的迭代次数内迅速搜索到较好的解是首先需要解决的问题。一个最直接的做法就是在种群初始化时尽量让种群靠近最优解集区域, 为此, 本文通过模型分析来降低搜索空间。以第  $i$  个测试阶段为例, 因为可靠性的取值为  $[0, 1]$  之间的实数, 由式(2)的连乘积可以得到如下的必要条件:

$$1 - \prod_{k=1}^{n_j} (1 - r_{jk}^i) \geq R_i^*, \quad \forall j = 1, 2, \dots, m$$

即每个子系统  $S_j$  的可靠性至少要达到  $R_i^*$ , 否则式(5)不可能成立。同理, 在每个子系统  $S_j$  中, 至少要存在一个模块  $k$  满足:

$$r_{jk}^i \geq 1 - \sqrt[n_j]{1 - R_i^*} \quad (8)$$

根据式(3)和式(8)可得:

$$t_{jk}^i \geq \frac{\ln \left( \frac{-\lambda \cdot a_{jk}^i \cdot b_{jk}^i}{\ln \left( 1 - \sqrt[n_j]{1 - R_i^*} \right)} \right)}{b_{jk}^i}, \quad \exists k \in (1, 2, \dots, n_j)$$

在子系统  $S_j$  中, 至少要有有一个模块分配的测试时间满足上式, 本文将该测试时间下限记为:

$$\tau_j^i = \frac{\ln \left( \frac{-\lambda \cdot a_{jk}^i \cdot b_{jk}^i}{\ln \left( 1 - \sqrt[n_j]{1 - R_i^*} \right)} \right)}{b_{jk}^i}$$

由式(3)可知, 模块的可靠性会随着投入的测试

时间  $t_{jk}^i$  的增大而呈非线性递增趋势, 对于不同的  $a_{jk}^i$ 、 $b_{jk}^i$ , 增长趋势不同, 增长快意味着只需消耗较少的测试时间就能达到较高的可靠性。本文从子系统  $S_j$  中选择增长最快的模块(对应可靠性达到 1 时消耗的测试时间最少), 设该模块为  $k_j^*$ , 则给子系统  $S_j$  中每个模块  $M_{jk}$  重新设定时间下限  $\pi_{jk}^i$  如下:

$$\pi_{jk}^i = \begin{cases} \tau_j^i, & k = k_j^* \\ 0, & k \neq k_j^* \end{cases}$$

则在第  $i$  个测试阶段, 整个系统的测试时间下限为:

$$\Pi^i = \sum_{j=1}^m \sum_{k=1}^{n_j} \pi_{jk}^i$$

本文在种群初始化时利用上述条件尽量让种群靠近最优解集区域。对于第 1 个测试阶段, 个体编码中每个模块对应的基因位随机初始化为:

$$t_{jk}^1 = \text{rand}(\pi_{jk}^1, T_1^*) \quad (9)$$

这种随机分配可能会导致所有基因位的测试时

间之和超过了上界, 即  $\sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^1 > T_1^*$ , 此时可以按照如下方式进行缩放以满足约束条件:

$$\vec{t}_{jk}^1 = \pi_{jk}^1 + (t_{jk}^1 - \pi_{jk}^1) \cdot \frac{T_1^* - \sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^1 - \Pi^1}{\sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^1 - \Pi^1} \quad (10)$$

对于第  $i > 1$  个阶段, 由于前一阶段的分配方案对于本阶段具有一定指导作用, 比如可以显示哪些模块比较耗时。基于此, 本文根据前一阶段方案的分配比例来进行初始化, 如下:

$$\begin{cases} \text{prop}^i = \frac{\left( \sum_{l=1}^i T_l^* - \sum_{l=1}^{i-1} \sum_{j=1}^m \sum_{k=1}^{n_j} t_{jk}^l \right) - \Pi^i}{T_{i-1}^* - \Pi^i} \\ t_{jk}^i = \pi_{jk}^i + (t_{jk}^{i-1} - \pi_{jk}^i) \cdot \text{prop}^i \end{cases} \quad (11)$$

容易验证通过式(9)和式(10)或式(11)初始化的个体都满足测试时间的上下界约束。需要指出的是, 上述初始化过程并不能保证每个个体都满足可靠性约束下界, 因为式(8)只是一个必要条件。本文目的是尽可能地让个体靠近满足可靠性需求的区域, 从而让个体迅速逼近最优解集, 加快算法的收敛。

## 2.3 加权归一化处理

在每一个测试阶段, 当种群进化到最大迭代次数时, MS-DRC-GDE3 算法会输出一个最优解集, 用户需要从解集中挑选一个解作为当前阶段的测试时间分配方案。传统的做法是选择解集中可靠性目标值最大的解, 但这种选择方式偏好性太强, 忽略了测试成本和测试时间消耗的平衡。本文采用文献[13]中的加权归一化方法来选取最优解。加权归一化方法可以简单快速地衡量多目标解的优劣, 该方法在

衡量解的分布性方面有较大优势,同时可兼顾收敛性,已在多目标优化问题的求解中得到广泛应用<sup>[19-20]</sup>。

加权归一化方法首先对解集中的每一个目标进行 min-max 归一化,以消除 3 个目标之间的统计误差,其次利用复合目标适应度函数将多目标整合成复合单目标: $f(x)=\sum_{y \in N} w_y \cdot \vec{f}_y$ ,其中, $w_y$  为目标  $y$  的加权值, $f(x)$  表示第  $x$  个多目标解的适应度值。由于本文的目的是权衡可靠性、测试成本和测试时间 3 个目标,因此适应度函数表示为:

$$f(x)=w_1 \cdot \vec{f}_1+w_2 \cdot \vec{f}_2+w_3 \cdot \vec{f}_3$$

其中, $w_1、w_2、w_3 \in [0,1]$  表示项目经理分配给 3 个目标的加权值,满足  $w_1+w_2+w_3=1$ , $\vec{f}_1、\vec{f}_2、\vec{f}_3$  分别为 min-max 归一化后的目标值。

最后,比较每个解的适应度值,选取适应度值最小的解作为当前阶段的最优测试资源分配方案。

2.4 时间复杂度分析

时间复杂度直接影响算法的收敛速度以及运行时间。MS-DRC-GDE3 算法的基本流程如图 2 所示,假设种群的规模为  $N$ ,优化的目标个数为  $M$ ,在种群初始化以及选择、交叉和变异阶段只处理最高等级的  $N$  个个体,该操作的时间复杂度为  $O(N)$ 。在非支配排序和拥挤度计算阶段算法的时间复杂度与原有的 GDE3 算法保持一致,为  $O(MN \cdot \lg N)$ <sup>[18]</sup>。加权归一化处理阶段也仅处理最高等级的  $N$  个个体,同理,其复杂度也是  $O(N)$ 。另外,本文所提算法为多阶段算法,其总阶段数为  $P$ ,每个阶段的迭代次数为  $G_{\max}$ 。综上,MS-DRC-GDE3 算法的总时间复杂度为  $O(P \cdot G_{\max} \cdot MN \cdot \lg N)$ 。

3 实验结果与分析

3.1 实验参数和评价指标

为了验证本文所提模型和算法的有效性,模拟 2 个并串联模块软件系统进行测试。第 1 个系统共有 11 个子系统、30 个模块,称为复杂系统,第 2 个系统共有 16 个子系统、50 个模块,称为大型系统。系统模型参数信息如表 1 所示,系统中的初始模块参数范围如表 2 所示,测试阶段相关参数如表 3 所示。对于每个系统,根据表 2 所给的范围随机生成 10 组初始模块参数,与表 3 所给的约束参数一起构成相应的实验测试实例,且每个实例在 Intel Core i7CPU、10 GB RAM 个人计算机上独立运行 30 次。

表 1 系统模型参数信息

Table 1 System model parameters information

系统	$\lambda$	$T^*$	$m$	$n_j$
复杂系统	200	150 000	11	{1,2,3,3,4,4,4,3,3,2,1}
大型系统	200	230 000	16	{1,2,3,3,3,4,4,5,5,4,4,3,3,3,2,1}

表 2 初始模块参数范围

Table 2 Initial parameters range of modules

模块	$a_{jk}^1$	$b_{jk}^1$	$c_{jk}^1$	$c_{jk}^2$	$c_{jk}^3$
串联	[30,35]	[5.8E-3,6.2E-3]	[3.4,3.55]	[6,6.2]	[4.0,4.1]
并联	[200,350]	[3E-4,9E-4]	[3.4,3.55]	[6,6.2]	[4.9,5.1]

表 3 测试阶段相关参数

Table 3 Related parameters of testing stage

系统	$p$	$T_i^*$	$R_i^*$	最大迭代次数
复杂系统	3	{8E+4,3E+4,4E+4}	{0.2,0.45,0.75}	500/ $p$
大型系统	3	{1.5E+5,4E+4,4E+4}	{0.2,0.45,0.75}	1 000/ $p$

为了验证本文所提 MS-DRC-GDE3 算法的有效性,将其与文献[17]中的多阶段测试资源分配算法 MS-GDE3 进行对比分析。为了对比的公平性,加权归一化过程中的  $w_1、w_2、w_3$  采用文献[13]中的推荐设置 {0.1,0.4,0.5},MS-DRC-GDE3 算法的其他参数与 MS-GDE3 算法保持一致,种群规模为 100,交叉概率为 0.9,变异概率为 0.7。

为了对比不同算法所得分配方案的优劣,本文采用经典的覆盖值(Coverage Value,CV)<sup>[21]</sup>指标来评估不同算法的收敛性。覆盖值提供了一种最直接的比较方式,假设 A 和 B 分别是 2 种不同的算法所获得的解集,A 中一个解的所有目标值都不比 B 中另一个解的所有目标值差,则认为前者覆盖了后者。CV(A,B) 表示 B 被 A 中解集所覆盖的百分比,即 B 中被 A 覆盖的解的个数与 B 中解的总数的比值。若 CV(A,B) 大于 CV(B,A),则意味着 A 中的解相比 B 更优。在计算时,为了使算法在每个实例中获得的解集更具统计学意义,首先将每个算法对应每个实例运行 30 次的解组合在一起,然后去掉重复的解,最后计算每个算法组合解集之间的 CV 值<sup>[22]</sup>。此外,为了衡量算法的整体性能,本文采用较流行的超体积值指标<sup>[11,13]</sup>。超体积表示非支配解集和参考点之间的空间体积大小,其可以从整体上衡量解集的收敛性和多样性,超体积值越高表示解集的整体质量越高。在计算超体积时,参考点的选择至关重要。对于每个实例,本文合并所有算法 30 次运行所获得的解集,然后去掉重复解并根据非支配关系对所有解进行排序,去掉所有被支配的解,最后将剩余非支配解集中每个目标的最大值略微放大作为最终的参考点<sup>[22]</sup>。

3.2 不同模型的对比

与文献[17]中的多阶段模型 MSMOTRA 不同,本文 DRC-MSMOTRA 模型为了满足用户的可靠性需求,在每个阶段都有不同的可靠性约束,本次实验将分析上述 2 种模型的优劣。根据表 1 的 2 个模拟系统各随机生成 10 个不同的实例,为了对比的公平性,基于与 MSMOTRA 模型契合的 MS-GDE3 算法进行测试。

2 种模型所得解集的覆盖值结果如表 4 所示,其中,A 和 B 分别表示 DRC-MSMOTRA 模型和 MSMOTRA 模型,较优的覆盖值用加粗字体表示。从表 4 可以看出,在复杂系统中,与 MSMOTRA 模

型获得的解相比,DRC-MSMOTRA模型在3个阶段中解的覆盖值分别提高约97个、42个和48个百分点,平均提高了约62个百分点。对于大型系统,与MSMOTRA模型获得的解相比,DRC-MSMOTRA模型在3个阶段中解的覆盖值分别提高约90个、61个和26个百分点,平均提高了约59个百分点。上述实验结果表明,在不同软件系统不同参数的条件下,与MSMOTRA模型相比,DRC-MSMOTRA模型可以获得更优的解,更能满足用户的可靠性需求。

表4  2种模型的覆盖值对比结果

Table 4  Comparison results of coverage values of two models

实例	复杂系统中的CV值/%						大型系统中的CV值/%					
	第1阶段		第2阶段		第3阶段		第1阶段		第2阶段		第3阶段	
	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)
1	93.65	0.00	54.95	22.18	65.52	20.56	95.76	0.61	74.90	9.20	68.03	9.03
2	100.00	0.00	69.38	11.45	80.46	7.94	94.24	0.30	66.19	16.55	41.62	33.92
3	96.53	0.50	70.67	9.52	76.86	19.15	68.82	9.65	67.00	15.19	54.20	33.65
4	99.29	0.00	51.63	17.55	71.95	17.52	84.62	0.92	66.52	9.51	53.06	25.98
5	100.00	0.00	59.07	17.77	60.23	27.31	96.25	0.29	56.25	19.03	56.78	29.72
6	97.12	0.22	57.67	9.42	48.65	24.33	98.69	0.00	64.85	7.92	52.86	32.40
7	98.10	0.00	48.25	16.25	65.90	20.00	100.00	0.00	66.18	3.32	65.71	22.18
8	98.85	0.00	60.11	13.99	55.91	30.40	98.40	0.00	96.57	0.55	64.58	25.97
9	95.81	0.79	75.44	5.73	77.27	17.78	87.34	1.18	82.71	7.17	68.00	17.86
10	100.00	0.00	39.64	27.08	71.76	10.66	87.21	1.32	68.69	11.14	64.00	21.58

需要指出的是,在第2阶段和第3阶段,在极个别实例上会出现CV(A,B)与CV(B,A)相差不是很大的情况,尤其是在大型系统的第3阶段,这说明MS-GDE3算法在DRC-MSMOTRA模型上有时也能获得理想的解集,因此,有必要进一步测试分析MS-GDE3算法在DRC-MSMOTRA模型上的综合表现。

3.3 不同算法的对比

本文根据表1的2个模拟系统随机生成10个不同的实例,基于本文DRC-MSMOTRA模型对比分析MS-DRC-GDE3算法和MS-GDE3算法的性能。2种

算法得到的覆盖值结果如表5所示,其中,A和B分别表示MS-DRC-GDE3算法和MS-GDE3算法。从表5可以看出,在复杂系统中,与MS-GDE3算法相比,MS-DRC-GDE3算法在3个阶段中解的覆盖值分别提高约73个、66个和67个百分点,平均提高了约69个百分点。对于大型系统,与MS-GDE3算法相比,MS-DRC-GDE3算法在3个阶段中解的覆盖值分别提高约76个、75个和89个百分点,平均提高了约80个百分点。上述实验结果表明,在不同软件系统不同参数的条件下,MS-DRC-GDE3算法在每个阶段所获解的质量均高于MS-GDE3算法。

表5  2种算法的覆盖值对比结果

Table 5  Comparison results of coverage values of two algorithms

实例	复杂系统中的CV值/%						大型系统中的CV值/%					
	第1阶段		第2阶段		第3阶段		第1阶段		第2阶段		第3阶段	
	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)	CV(A,B)	CV(B,A)
1	59.33	11.29	94.07	0.27	83.21	0.48	87.20	1.41	82.21	4.56	84.72	6.28
2	88.81	1.06	85.50	3.09	74.64	5.58	74.24	4.68	81.03	7.04	95.45	0.67
3	89.91	1.04	53.87	26.69	64.11	13.55	96.55	0.18	73.98	3.85	88.57	1.41
4	96.08	0.29	73.15	15.27	68.26	11.11	82.56	4.70	85.98	2.58	96.52	0.00
5	67.55	9.33	85.76	5.88	79.94	3.15	49.12	13.09	77.85	2.24	81.73	5.26
6	72.22	9.31	71.57	12.71	70.38	9.71	79.41	4.58	92.44	3.52	90.63	3.65
7	63.31	7.23	91.06	1.72	78.93	10.23	76.29	3.70	67.90	6.15	89.16	0.73
8	79.29	6.29	84.38	3.79	67.92	7.05	81.98	3.22	73.70	7.21	94.48	2.33
9	85.17	4.23	39.36	29.02	75.60	3.70	81.34	2.32	83.75	2.32	87.88	0.00
10	79.33	3.60	85.71	3.51	77.90	3.75	90.26	0.98	75.30	3.90	98.53	0.00

为了进一步分析2种算法的整体性能,对比2种算法所得解集的超体积的均值和标准差,结果如表6、表7所示,括号内为标准差。从表6、表7可以看出,MS-DRC-GDE3算法的超体积明显优于MS-GDE3算法,这说明

MS-DRC-GDE3算法的解集整体质量更高。此外,标准差能够体现数据的波动幅度,在第1阶段,MS-DRC-GDE3算法的超体积标准差比MS-GDE3算法小一个数量级,在第2、第3阶段,虽然MS-DRC-GDE3算法的



超体积标准差比 MS-GDE3 算法略大,但是结合均值和标准差明显可以看出,MS-DRC-GDE3 算法的超体积实际波动都比 MS-GDE3 算法小,这说明 MS-DRC-GDE3 算法比 MS-GDE3 算法更加稳定,可以在收敛性和多样性之间实现更好的平衡。造成上述结果的原因是 MS-DRC-GDE3 算法根据每个阶段中用户的可靠性需求推

导出每个模块的测试时间下限,根据该最低时间要求对种群进行初始化,从而使得整个种群迅速地向用户满意的解区域逼近。此外,MS-DRC-GDE3 算法采用的加权归一化方法,可以取得可靠性、成本和测试时间三者之间的均衡,从而在每个阶段使种群有更多的空间去探索更好的解。

表 6 2 种算法在复杂系统中的超体积结果对比

Table 6 Comparison of super volume results of two algorithms in complex system

实例	超体积值					
	第 1 阶段		第 2 阶段		第 3 阶段	
	A	B	A	B	A	B
1	3.091E+5	2.375E+5	3.921E+6	2.914E+6	7.237E+6	4.883E+6
	(9.367E+3)	(6.424E+4)	(2.935E+5)	(1.866E+5)	(6.908E+5)	(1.712E+5)
2	7.895E+4	4.715E+4	3.231E+6	2.743E+6	5.222E+6	3.538E+6
	(2.727E+3)	(2.211E+4)	(2.622E+5)	(3.498E+5)	(5.399E+5)	(1.566E+5)
3	8.423E+4	3.986E+4	4.270E+6	3.560E+6	6.666E+6	4.604E+6
	(3.476E+3)	(2.143E+4)	(3.513E+5)	(5.962E+5)	(4.673E+5)	(2.873E+5)
4	1.227E+5	7.995E+4	3.077E+6	2.395E+6	4.345E+6	2.996E+6
	(5.360E+3)	(2.365E+4)	(3.523E+5)	(5.812E+5)	(4.050E+5)	(1.789E+5)
5	2.704E+5	1.924E+5	2.771E+6	2.178E+6	5.771E+6	3.889E+6
	(8.718E+3)	(3.919E+4)	(2.318E+5)	(3.218E+5)	(6.158E+5)	(2.546E+5)
6	2.068E+5	1.629E+5	3.910E+6	2.962E+6	8.016E+6	5.504E+6
	(6.235E+3)	(3.972E+4)	(3.898E+5)	(4.012E+5)	(8.704E+5)	(2.605E+5)
7	2.022E+5	1.488E+5	3.676E+6	2.989E+6	4.964E+6	3.416E+6
	(7.854E+3)	(4.565E+4)	(3.912E+5)	(3.221E+5)	(4.624E+5)	(2.007E+5)
8	6.730E+4	4.848E+4	3.832E+6	3.234E+6	6.141E+6	4.151E+6
	(2.397E+3)	(1.529E+4)	(3.091E+5)	(2.953E+5)	(5.678E+5)	(1.995E+5)
9	3.772E+5	3.086E+5	4.213E+6	3.023E+6	6.122E+6	4.116E+6
	(8.247E+3)	(4.511E+4)	(4.005E+5)	(5.739E+5)	(5.887E+5)	(1.978E+5)
10	7.690E+4	4.373E+4	4.040E+6	3.287E+6	4.827E+6	3.301E+6
	(2.469E+3)	(2.160E+4)	(3.525E+5)	(5.341E+5)	(3.977E+5)	(1.521E+5)

表 7 2 种算法在大型系统中的超体积结果对比

Table 7 Comparison of super volume results of two algorithms in large scale system

实例	超体积值					
	第 1 阶段		第 2 阶段		第 3 阶段	
	A	B	A	B	A	B
1	1.626E+6	1.329E+6	7.144E+6	4.371E+6	8.741E+6	5.183E+6
	(6.414E+4)	(1.977E+5)	(7.573E+5)	(8.174E+5)	(1.163E+6)	(1.937E+5)
2	1.440E+6	1.146E+6	7.236E+6	4.146E+6	7.751E+6	4.474E+6
	(7.606E+4)	(1.510E+5)	(6.288E+5)	(9.222E+5)	(1.014E+6)	(2.752E+5)
3	1.479E+6	1.171E+6	7.911E+6	5.108E+6	8.789E+6	5.208E+6
	(5.797E+4)	(1.477E+5)	(6.710E+5)	(3.686E+5)	(7.262E+5)	(3.278E+5)
4	2.281E+6	1.942E+6	7.026E+6	3.983E+6	7.795E+6	4.733E+6
	(7.138E+4)	(2.396E+5)	(8.063E+5)	(5.987E+5)	(1.022E+6)	(2.780E+5)
5	1.950E+6	1.497E+6	7.624E+6	4.468E+6	1.030E+7	5.997E+6
	(8.709E+4)	(2.099E+5)	(7.075E+5)	(6.99E+5)	(1.259E+6)	(3.193E+5)
6	1.362E+6	1.110E+6	9.007E+6	5.503E+6	6.922E+6	4.061E+6
	(5.502E+4)	(1.406E+5)	(7.053E+5)	(6.034E+5)	(7.278E+5)	(2.054E+5)
7	1.280E+6	1.008E+6	7.338E+6	4.617E+6	8.679E+6	4.941E+6
	(6.148E+4)	(1.571E+5)	(5.814E+5)	(7.024E+5)	(9.307E+5)	(2.645E+5)
8	9.949E+5	4.681E+5	7.181E+6	3.434E+6	7.085E+6	2.941E+6
	(4.267E+4)	(3.567E+5)	(6.464E+5)	(2.364E+6)	(7.647E+5)	(1.968E+6)
9	1.771E+6	1.352E+6	6.972E+6	3.941E+6	8.487E+6	4.355E+6
	(6.832E+4)	(1.593E+5)	(8.194E+5)	(6.831E+5)	(1.056E+6)	(2.781E+5)
10	1.558E+6	1.015E+6	7.029E+6	3.568E+6	8.698E+6	4.342E+6
	(5.706E+4)	(4.284E+5)	(6.396E+5)	(1.649E+6)	(8.638E+5)	(1.761E+6)

为了更加直观地解释超体积指标代表的含义,图3给出2种算法分别在复杂系统实例6、大型系统实例5上最后阶段的解集的分布,2种对比算法在这2种实例上的超体积值差异较大,可以更加清晰地展现超体积值的特点。从图3可以看出,与MS-GDE3算法相比,MS-DRC-GDE3算法得到的解在目标空间的分布更广,具有更好的多样性。此外,MS-DRC-GDE3算法的收敛性比MS-GDE3算法更好,MS-DRC-GDE3算法解集中有相当多的分配方案明显优于MS-GDE3算法,即前者具有更高的可靠性、更低的测试成本和更短的测试时间。

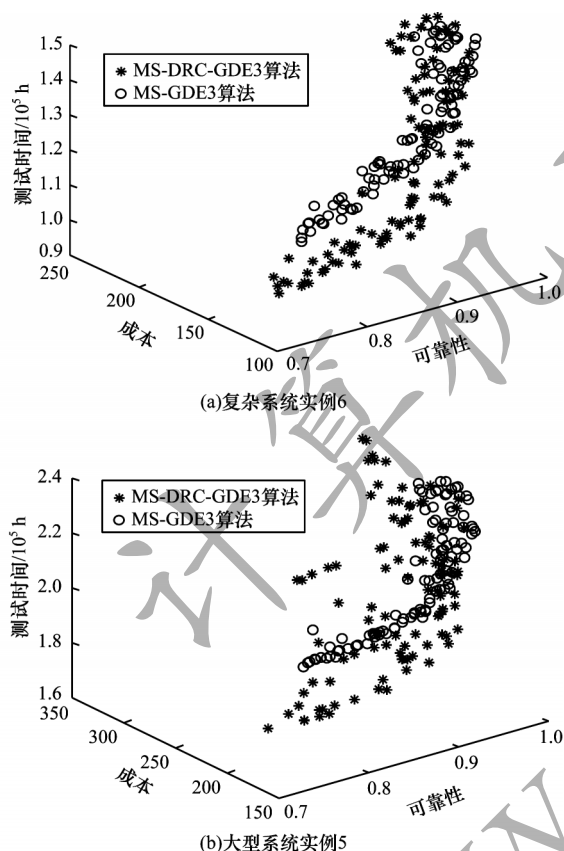


图3 2种算法所获解的分布

Fig.3 Distribution of solutions obtained by two algorithms

#### 4 结束语

基于多目标进化算法的多目标测试资源分配可以给用户在可靠性、成本和测试时间上提供较多折中的方案,从而使用户有更多的体验,但已有测试资源分配方案大多只考虑测试时间约束而忽略了用户的可靠性需求,导致解集中存在大量可靠性很低的方案,降低了用户满意度。本文针对多阶段测试资源分配问题,从用户对软件的可靠性需求角度出发,构建一种动态可靠性约束的多阶段多目标测试资源分配模型,基于模块参数估计、考虑时间下限的种群初始化、加权归一化、多目标差分进化算法,设计一

种动态可靠性约束的多阶段多目标测试资源分配算法。实验结果表明,本文所提模型和算法具有有效性且能够取得良好的资源分配效果。下一步将考虑成本约束并尝试基于成本约束来分析和推演每个模块的测试时间上限,此外,还将改进算法中的拥挤度计算等环节,从而进一步提升算法的整体性能。

#### 参考文献

- [1] BIANCHI F A, MARGARA A, PEZZE M. A survey of recent trends in testing concurrent software systems[J]. IEEE Transactions on Software Engineering, 2018, 44(8): 747-783.
- [2] RAMIREZ A, ROMERO J R, SIMONS C. A systematic review of interaction in search-based software engineering[J]. IEEE Transactions on Software Engineering, 2019, 45(8): 760-781.
- [3] CHEN T, THOMAS S W, HEMMATI H, et al. An empirical study on the effect of testing on code quality using topic models; a case study on software development systems[J]. IEEE Transactions on Reliability, 2017, 66(3): 806-824.
- [4] LYU M R, RANGARAJAN S, VAN MOORSEL A P A. Optimal allocation of test resources for software reliability growth modeling in software development[J]. IEEE Transactions on Reliability, 2002, 51(2): 183-192.
- [5] HUANG C Y, LYU M R. Optimal testing resource allocation, and sensitivity analysis in software development[J]. IEEE Transactions on Reliability, 2005, 54(4): 592-603.
- [6] HUANG CHIN-YU, LO J H. Optimal resource allocation for cost and reliability of modular software systems in the testing phase[J]. Journal of Systems and Software, 2006, 79(5): 653-664.
- [7] PIETRANTUONO R, RUSSO S, TRIVEDI K S. Software reliability and testing time allocation; an architecture-based approach[J]. IEEE Transactions on Software Engineering, 2010, 36(3): 323-337.
- [8] WANG Z, TANG K, YAO X. A multi-objective approach to testing resource allocation in modular software systems[C]// Proceedings of 2008 IEEE Congress on Evolutionary Computation. Washington D. C., USA: IEEE Press, 2008: 1148-1153.
- [9] YU S S, FEI D, BIN L. Optimal testing resource allocation for modular software systems based on multi-objective evolutionary algorithms with effective local search strategy[C]// Proceedings of 2013 IEEE Workshop on Memetic Computing. Washington D. C., USA: IEEE Press, 2013: 1-8.
- [10] PIETRANTUONO R, POTENA P, PECCHIA A, et al. Multiobjective testing resource allocation under uncertainty[J]. IEEE Transactions on Evolutionary Computation, 2018, 22(3): 347-362.
- [11] WANG Z, TANG K, YAO X. Multi-objective approaches to optimal testing resource allocation in modular software systems[J]. IEEE Transactions on Reliability, 2010, 59(3): 563-575.

(下转第260页)



(上接第 253 页)

- [12] ZHANG Guofu, SU Zhaopin, LI Miqing, et al. Constraint handling in NSGA-II for solving optimal testing resource allocation problems[J]. IEEE Transactions on Reliability, 2017, 66(4): 1193-1212.
- [13] YANG B, HU Y M, HUANG C Y. An architecture-based multi-objective optimization approach to testing resource allocation[J]. IEEE Transactions on Reliability, 2015, 64(1): 497-515.
- [14] PIETRANTUONO R. On the testing resource allocation problem: research trends and perspectives [EB/OL]. [2020-02-10]. <http://wpage.unina.it/roberto.pietrantuono/papers/JSS2019.pdf>.
- [15] LEUNG Y W. Dynamic resource-allocation for software-module testing[J]. Journal of Systems and Software, 1997, 37(2): 129-139.
- [16] LU Yang, YUE Feng, ZHANG Guofu, et al. Model and solution to testing resource dynamic allocation for series-parallel software systems[J]. Journal of Software, 2016, 27(8): 1964-1977. (in Chinese)  
陆阳, 岳峰, 张国富, 等. 串并行软件系统测试资源动态分配建模与求解[J]. 软件学报, 2016, 27(8): 1964-1977.
- [17] NIU Fuqiang. Research on dynamic multi-objective testing resource allocation problem [D]. Hefei: Hefei University of Technology, 2019. (in Chinese)  
牛福强. 动态多目标测试资源分配问题研究[D]. 合肥: 合肥工业大学, 2019.
- [18] KUKKONEN S, LAMPINEN J. GDE3; the 3rd evolution step of generalized differential evolution [C]//Proceedings of IEEE Congress on Evolutionary Computation. Washington D. C., USA: IEEE Press, 2005: 443-450.
- [19] JAKOB W, BLUME C. Pareto optimization or cascaded weighted sum; a comparison of concepts[J]. Algorithms, 2014, 7(1): 166-185.
- [20] KADDANI S, VANDERPOOTEN D, VANPEPERSTRAETE J M, et al. Weighted sum model with partial preference information; application to multi-objective optimization[J]. European Journal of Operational Research, 2017, 260(2): 665-679.
- [21] ZITZLER E, THIELE L. Multiobjective evolutionary algorithms; a comparative case study and the strength Pareto approach[J]. IEEE Transactions on Evolutionary Computation, 1999, 3(4): 257-271.
- [22] LI Miqing, YAO Xin. Quality evaluation of solution sets in multiobjective optimisation[J]. ACM Computing Surveys, 2019, 52(2): 1-38.

编辑 吴云芳