ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.10, October 2008, pp.2770−2779
DOI: 10.3724/SP.J.1001.2008.02770
© 2008 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 Markov 决策过程用交叉熵方法优化软件测试*

张德平[1,2+], 聂长海[1], 徐宝文[1]

[1](东南大学 计算机科学与工程学院,江苏 南京 210096)

[2](南京航空航天大学 理学院,江苏 南京 210016)

Cross-Entropy Method Based on Markov Decision Process for Optimal Software Testing

ZHANG De-Ping[1,2+], NIE Chang-Hai[1], XU Bao-Wen[1]

[1](Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

[2](College of Science, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

+ Corresponding author: E-mail: depingzhang@163.com

Zhang DP, Nie CH, Xu BW. Cross-Entropy method based on Markov decision process for optimal software testing. *Journal of Software*, 2008,19(10):2770−2779. http://www.jos.org.cn/1000-9825/19/2770.htm

Abstract: This paper demonstrates an approach to optimize software testing by minimizing the expected cost with given software parameters of concern. Taking software testing process as a Markov decision process, a Markov decision model of software testing is proposed in this paper, and by using a learning strategy based on the cross-entropy method to optimize the software testing, this paper obtains the optimal testing profile. Simulation results show that the testing profile with the learning strategy performs significantly better than the random testing strategy with respect to the expected cost. Moreover, this learning strategy is more feasible and can significantly reduce the number of test cases required to detect and remove a certain number of software defects in comparison with the random testing strategy.

Key words: software testing; Markov decision process; cross-entropy method; optimal testing profile

摘 要: 研究了待测软件某些参数已知的条件下,以最小化平均测试费用为目标的软件测试优化问题.将软件测试过程处理成马尔可夫(Markov)决策过程,给出了软件测试的马尔可夫决策模型,运用交叉熵方法,通过一种学习策略获得软件测试的最优测试剖面,用于优化软件测试.模拟结果表明,学习策略给出的测试剖面要优于随机测试策略,检测和排除相同数目的软件缺陷,学习策略比随机测试能够显著地减少测试用例数,降低测试成本,提高缺陷检测效率.

关键词: 软件测试;马尔可夫决策过程;交叉熵方法;最优测试剖面

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60425206, 60773104, 60633010, 60503033 (国家自然科学基金); the Doctor Subject Fund of Education of Ministry of China under Grant No.20060286020 (国家教育部博士点基金); the National Science Foundation of Jiangsu Province of China under Grant No.BK2006094 (江苏省自然科学基金); the Excellent Talent Foundation on Teaching and Research of Southeast University of China (东南大学优秀青年教师教学科研资助); the Open Foundation of State Key Laboratory of Software Engineering of Wuhan University of China (武汉大学软件工程重点实验室开放基金)

Received 2007-07-30; Accepted 2008-02-25

© 中国科学院软件研究所 http://www.jos.org.cn

软件测试是一种检测软件缺陷、提高可靠性水平的重要手段.证明、检测和预防是软件测试追求的目标,人们可以从不同角度设计软件测试来实现测试目标,例如,如何在有限次测试中尽量多地检测出不同的软件缺陷,或用尽量少的测试用例检测出某些软件故障等.软件测试策略主要包括如何选择测试用例及确定最佳停止测试时间以达到测试目的.软件测试的优化就是优化软件测试策略,关心如何设计软件测试,使得测试中每个行动都是达到测试目的的最优行动[1],即一个最优的软件测试应该在测试中每次都选择最好的(如:对于给定测试目标,测试成本最小、能够最早检测到缺陷或检测到缺陷最多等)测试用例或测试技术,并且在最佳时刻停止测试.测试用例被选择的概率分布称为测试剖面.一个最优测试剖面是指在适当时刻选择合适的测试用例(或测试技术或行为),实现测试目标的最优测试策略.因此,软件测试优化过程也可以看成是最优测试剖面的获取过程.

测试过程中,测试用例的选取不仅依赖于测试策略,而且依赖于已选取的测试用例和已检测到的软件缺陷.它们为下一步的测试用例的选择提供有益的参考.测试用例的生成需要满足充分性与有效性原则,一方面,要求生成足够多的测试用例,充分满足各种测试需求,尽量多地检测出软件缺陷,保证软件质量.另一方面,由于受时间、人力、物力等测试资源的约束,要求不能耗费太多的测试资源,测试成本是测试优化考虑的重要因素.因此如何合理地使用测试资源,充分利用软件测试中收集到的信息、设计科学的测试策略是软件测试优化的关键问题.

1 相关研究

在软件测试优化的研究中,大多数研究者关注的是软件测试策略问题,研究基于某种测试技术或测试准则如何产生测试用例,关注测试过程的一些性质,例如,测试用例的缺陷检测能力、测试数据的复杂性和测试成本等[2−8].同时,与之相反的问题也正逐渐引起人们的兴趣:给定一个测试目标,如可靠性、测试成本等,如何设计一个最优的测试策略来实现这个目标.Tal 等人[9]用统计测试方法,研究了安全攸关系统的软件可靠性演示问题,将测试处理成测试、分析和维修(test,analyse and fix,简称 TAAF)过程,用动态规划方法给出了在事先确定的时间内,以整个过程成功的概率最大为目标,如何选取测试行动的最优测试策略.Sahinoglu 等人[10,11]研究了分支覆盖测试中不同测试策略的选取问题,用复合泊松模型描述不同测试策略产生的分支覆盖数,均衡各种测试策略效率给出了相对应的停止准则.Rajgopal 和 Mazumdar[12]基于马尔可夫(Markov)可靠性模型,用假设检验方法研究了如何设计软件测试计划,满足在整个系统的可靠性低于某个指定值的概率小于某个事先给定的数时,每个模块被测试的用例数最少.Chen 等人[13]基于经典决策理论的思想,研究了划分测试中,在导致错误输入的个数及位置假定是完全未知的条件下,提出了 Laplace 准则、折衷准则以及最小后悔准则来选择测试用例.Cai 等人[1,14,15]首次用控制论的方法将软件测试处理成一个控制问题,对软件测试提出了马尔可夫控制方法,将被测软件作为控制对象,软件测试策略作为相应的控制器,用动态规划方法较好地解决了测试资源受约束时的软件测试的优化,并提出了适应测试策略在线调整软件测试策略.

本文研究了在待测软件某些参数已知的条件下,如何优化软件测试,使得用最少的期望测试成本,检测并排除待测软件中剩余软件缺陷.根据软件测试过程提出一个马尔可夫决策模型,基于交叉熵方法[16−19]给出了一种快速学习算法,获得最优测试剖面,优化整个测试过程.

2 软件测试的马尔可夫决策模型

对于待测软件系统,假定软件最初具有 M 个软件缺陷,我们的目标是检测并排除所有的软件缺陷.在整个软件测试过程中,如果将每个时间点(决策时刻)t(t=0,1,2,…)上软件剩余的缺陷数作为时刻 t 系统所处状态(state),则整个状态空间为 $\boldsymbol{S}=\{s_t,t\geq 0\}=\{M,M-1,...,1,0\}$,将每个时间点 t 按照某种测试策略选取一个测试用例作为一次决策行动(action),各个时刻按照不同测试策略选取测试用例的行动全体组成整个行动空间 $\boldsymbol{A}=\{a_t,t\geq 0\}=\{1,2,...,K\}$ 并且每个时间点 t(t=0,1,2,…)上的状态 s_t 和所采用的行动 a_t 都会影响下一时间点 t+1 的状态 S_{t+1},因此,整个测试过程形成一个马尔可夫决策过程,如图 1 所示.

这里,“行动”还可以有多种理解,如:在划分测试的一个等价类中选取一个输入;软件运行一次;按照某种特定测试技术而形成的一类测试用例.它也可以理解为选择某种测试技术,如:边界测试、路径测试和随机测试.它

© 中国科学院软件研究所 http://www.jos.org.cn

还可以理解为选择一个测试人员.不同的“行动”具有不同的缺陷检测[1].

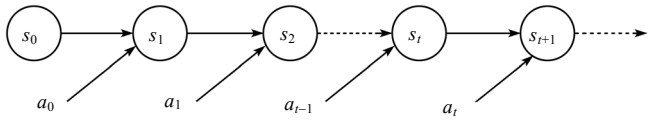

Fig.1 Software testing as a Markov decision process

图 1 作为马尔可夫决策过程的软件测试

为了便于用马尔可夫决策过程来优化软件测试过程,不失一般性,作如下假设:

1) 软件在初始时刻(t=0)包含 M 个缺陷.

2) 在每个决策时刻只采取一个行动,并且这个行动至多只能检测到一个软件缺陷.

3) 若在时刻 t,系统状态为 S_t=j,则行动 a_t 应用到测试中以概率 $j\theta_a$ 检测到一个缺陷,下一时刻 t+1 的状态为 S_{t+1}=j−1;以概率 1−$j\theta_a$ 不能检测到缺陷,下一时刻 t+1 的状态与时刻 t 的状态相同,即 S_{t+1}=S_t,其中,θ_a 可解释为由于行动 a_t 使得每个剩余软件缺陷被检测到的概率,称为缺陷检测率[1,3].

4) 缺陷一旦被检测到,缺陷就被认为已排除,不再产生软件失效[20].

5) S_t=0 是一个吸收状态也是一个目标状态.

6) 在每个时刻 t 采取行动 a_t,则不论是否检测到缺陷均会导致大小为 $C_s(a_t)$的测试费用,在目标状态上采取任何行动的测试费用均记为 0.

7) 在每个时刻 t 总有 K 个可行行动,即行动空间 $\boldsymbol{A}=\{1,2,...,K\}$.

由上面的假设可知,描述软件测试过程的马尔可夫决策过程可定义为一个五元组 $\{\boldsymbol{T},\boldsymbol{S},\boldsymbol{A},\boldsymbol{P},\boldsymbol{C}\}$,其中,$\boldsymbol{T}=\{0,1,2,...\}$ 为决策时刻,$\boldsymbol{S}=\{M,M-1,...,1,0\}$ 为状态空间,$\boldsymbol{A}=\{1,2,...,K\}$ 为行动空间,$\boldsymbol{P}$ 为转移概率矩阵,其元素满足马尔可夫性:

$$P(s_t \mid s_{0:t-1},a_{0:t-1})=P(s_t \mid s_{t-1},a_{t-1}) \tag{1}$$

表示在时刻 t− 1 采取行动 a_{t-1} 时,从状态 S_{t-1} 转移到状态 S_t 的概率,其中记号 $s_{0:t-1}$ 表示 $s_0,s_1,...,s_t$,记号 s_{t-1} 表示在起始时刻之前的状态,记为空集∅.$\boldsymbol{C}$ 中的元素 $C_{s_{t-1}}(a_{t-1})$ 表示在状态 s_{t-1} 采取行动 a_{t-1} 时的测试费用,则从起始时刻 0 到时刻 t 为止,整个测试过程可由此马尔可夫过程的一个历史来描述:

$$H_t=\{s_0,a_0,s_1,a_1,...,s_{t-1},a_{t-1},s_t\} \tag{2}$$

记 $\pi=\{P(a_t \mid H_t);t=0,1,2,...\}$ 为在历史 H_t 的条件下采取行动 $a_t,t=0,1,2,...$ 的概率集,称为一个测试策略.记 $\boldsymbol{\tau}$ 为从系统状态 s_0=M 首次到达状态 0 时所采取测试行动的最小次数,即 $\tau=\min\{t:s_t=0\}$,则从时刻 0 到时刻 $\boldsymbol{\tau}$ 所选取的行动构成了一个测试用例序列 $\boldsymbol{X}=(a_0,a_1,...,a_{\tau-1})$.定义:

$$\varphi(\boldsymbol{X})=E_\pi\left[\sum_{t=0}^{\tau}C_{s_t}(a_t)\right] \tag{3}$$

为期望测试费用,其中 E_π 表示相对于概率测度 π 求期望.

我们的问题是找到一个最优测试策略使得期望测试费用最小,即确定最优测试用例序列,使得:

$$\min\varphi(\boldsymbol{X})=E_\pi\left[\sum_{t=0}^{\tau}C_{s_t}(a_t)\right] \tag{4}$$

由上述可以看到,马尔可夫决策策略(或软件测试策略)在时刻 t 需要决定相对于状态 S_t 将被选择的行动.对于给定的测试目标,一个最优的软件测试策略应该能够设计或决定哪些行动必须采用并且相对测试目标是最优的.这就使得这些行动 $\{a_t,t=0,1,2,...,\tau\}$ 的选取概率相对于给定测试目标形成了一个最优测试剖面.

3 软件测试的学习策略

3.1 交叉熵方法

交叉熵方法最早由 Rubinstein[19]引入用来估计复杂随机网络中稀有事件发生的概率.该方法通过交叉熵的

© 中国科学院软件研究所 http://www.jos.org.cn

简单变形解决稀有事件发生概率的估计问题,是一种使估计的方差达到最小的适应性算法.其主要思想是将“确定性”优化问题转变成一个相关的“随机”优化问题,然后用稀有事件的随机模拟技术来求解此“随机”优化问题.人们已经成功地应用此方法解决了一些很难求解的组合优化问题[16,17],最近一些成功的应用[21,22]再次说明交叉熵方法是解决 NP-hard 问题的一种实用工具.

应用交叉熵方法需要解决两个问题:如何产生随机样本和在每一次迭代中如何修正参数.前者最简单的方法是直接根据软件测试中测试行为的选取过程模拟生成随机样本;后者只需将参数的修正转化成一个与式(4)相关的等价优化问题求解.

令 $A_i(i=1,2,\ldots,M)$表示状态 $S_t=i$ 时选取的行动.记 $\boldsymbol{P}=(p_{ij})_{M\times K}$ 为概率矩阵,其元素 p_{ij} 表示在状态 $s_t=i$ 选取行动 j 的概率,则 $A_i(i=1,2,...,M)$ 在状态 i 分别按概率分布 $p_{i1},p_{i2},\ldots,p_{iK}$ 在 K 个行动中选取一个行动,即 A_i 以概率 p_{i1} 选取行动 1,以概率 p_{i2} 选取行动 2,...,以概率 p_{iK} 选取行动 K,并且 $\sum_{j=1}^{K} p_{ij}=1$.记 $f(\boldsymbol{x},\boldsymbol{u})$为测试用例序列 $\boldsymbol{x}$ 的联合概率分布,则 $f(\boldsymbol{x},\boldsymbol{u})$由概率矩阵 $\boldsymbol{P}$ 唯一确定.定义 $\boldsymbol{\gamma}^*$为采用不同测试策略 π 生成测试用例序列 $\boldsymbol{X}$ 时 $\varphi(\boldsymbol{X})$的最小值,τ 为首次达到吸收状态的时刻(或测试用例数),则式(4)中的最优测试用例序列问题可表示为确定最优测试用例序列问题,使得:

$$\varphi(\boldsymbol{x}^*)=\gamma^*=\min_{\boldsymbol{x}\in\boldsymbol{X}}\varphi(\boldsymbol{x}) \tag{5}$$

定义 $\boldsymbol{I}_{\{\varphi(\boldsymbol{x})\le\gamma\}}$ 为 $\boldsymbol{X}$ 上不同 $\gamma(\in\Re)$ 值的示性函数集合,则式(5)中最优测试用例确定问题可转化为与之相关的概率估计问题来求解:

$$\ell(\gamma)=P_{\boldsymbol{u}}(\varphi(\boldsymbol{X})\le\gamma)=E_{\boldsymbol{u}}[\boldsymbol{I}_{\{\varphi(\boldsymbol{x})\le\gamma\}}]=\sum_{\boldsymbol{x}}\boldsymbol{I}_{\{\varphi(\boldsymbol{x})\le\gamma\}}f(\boldsymbol{x};\boldsymbol{u}) \tag{6}$$

其中,$P_{\boldsymbol{u}}$ 和 $E_{\boldsymbol{u}}$ 分别为相对于概率分布 $f(\cdot;\boldsymbol{u})$ 的概率测度和期望.

当 $\gamma=\gamma^*$ 时,$\ell(\gamma)$ 估计的最直接方法是采用重要抽样方法[16]:根据 $\boldsymbol{A}$ 上的概率分布 g 抽取样本 $\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N$,则 ℓ 的估计为

$$\hat{\ell}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{I}_{\{\varphi(\boldsymbol{x}_i)\le\gamma\}}\frac{f(\boldsymbol{x}_i;\boldsymbol{u})}{g(\boldsymbol{x}_i)} \tag{7}$$

显然,当概率分布 g 取

$$g^*(\boldsymbol{x})=\frac{\boldsymbol{I}_{\{\varphi(\boldsymbol{x})\le\gamma\}}f(\boldsymbol{x};\boldsymbol{u})}{\ell} \tag{8}$$

时,只需抽取一个样本,即 $N=1$ 就可以得到 ℓ 的一个方差为 0 的无偏估计形式:

$$\boldsymbol{I}_{\{\varphi(\boldsymbol{x}_i)\le\gamma\}}\frac{f(\boldsymbol{x};\boldsymbol{u})}{g^*(\boldsymbol{x}_i)}=\ell \tag{9}$$

从式(8)可以看出,g^*依赖于未知参数 ℓ,很难确定.因此,一般采用在概率分布 $f(\cdot;\boldsymbol{u})$ 的概率分布簇 $\{f(\cdot;\boldsymbol{v})\}$($\boldsymbol{v}$ 为参数)中选取概率分布 g 来解决这个问题,即确定推断参数 $\boldsymbol{v}$ 使得 $f(\cdot;\boldsymbol{v})$ 与概率分布 g^*差别达到最小.常用来衡量两个概率分布差别大小的测度是 Kullback-Leibler 距离或交叉熵[16].

这样,最优抽样分布的确定问题就转化为确定推断参数 $\boldsymbol{v}$,使得 $f(\cdot;\boldsymbol{v})$ 与概率分布 g^*的交叉熵最小的问题,即

$$\boldsymbol{v}^*=\arg\min_{\boldsymbol{v}}-\int g^*(\boldsymbol{x})\ln f(\boldsymbol{x};\boldsymbol{v})\mathrm{d}\boldsymbol{x} \tag{10}$$

将式(8)代入式(10)可得如下等价推断参数确定形式:

$$\boldsymbol{v}^*=\arg\max_{\boldsymbol{v}}E_{\boldsymbol{u}}[\boldsymbol{I}_{\{\varphi(\boldsymbol{X})\le\gamma\}}\ln f(\boldsymbol{X};\boldsymbol{v})] \tag{11}$$

由于测试用例序列 $\boldsymbol{x}=(a_0,a_1,\ldots)$是根据概率矩阵 $\boldsymbol{P}$ 生成的,概率矩阵 $\boldsymbol{P}$ 就是推断参数 $\boldsymbol{v}$,而测试用例序列 $\boldsymbol{x}$ 的联合概率分布为

$$f(\boldsymbol{x};\boldsymbol{P})=\prod_i p_{ij},$$

满足

© 中国科学院软件研究所 http://www.jos.org.cn

$$\sum_j p_{ij} = 1,$$

其中 i 为测试用例序列 $\boldsymbol{x}$ 运行中历经的所有状态,j 为测试用例序列 $\boldsymbol{x}$ 中在状态 $s=i$ 时采取的行动.

由拉格朗日乘子法,不同状态下不同行动的最优选择概率为

$$p_{ij} = \frac{E_{\boldsymbol{P}}[\boldsymbol{I}_{\{\varphi(\boldsymbol{X})\le\gamma\}}\sum_i \boldsymbol{I}_{\{\boldsymbol{x}\in\boldsymbol{X}_{ij}\}}]}{E_{\boldsymbol{P}}[\boldsymbol{I}_{\{\varphi(\boldsymbol{X})\le\gamma\}}\sum_i \boldsymbol{I}_{\{\boldsymbol{x}\in\boldsymbol{X}_i\}}]} \tag{12}$$

则最优选择概率的估计为

$$\hat{p}_{ij} = \frac{\sum_{k=1}^{N}\boldsymbol{I}_{\{\varphi(\boldsymbol{x}_k)\le\gamma\}}\sum_i \boldsymbol{I}_{\{\boldsymbol{x}_k\in\boldsymbol{X}_{ij}\}}}{\sum_{k=1}^{N}\boldsymbol{I}_{\{\varphi(\boldsymbol{x}_k)\le\gamma\}}\sum_i \boldsymbol{I}_{\{\boldsymbol{x}_k\in\boldsymbol{X}_i\}}} \tag{13}$$

这样就得到了不同状态下不同行动选择概率的修正公式,式(13)的直观含义为:在状态 i 选择行动 j 的概率为所有成本不大于γ的测试用例序列中,在状态 i 选择行动 j 的次数与测试历经状态 i 的次数之比.由此可知,在每一个状态中参数(不同状态下不同行动的选择概率)的修正都是选用最有利于接近最优目标的"精英"样本来修正.这样,修正后使得下一次生成的测试用例序列更接近于最优测试目标,这种机制会使得每个状态的最优行动很容易找到,最终得到最优测试剖面.

3.2 软件测试的学习策略

应用交叉熵方法于软件测试的马尔可夫决策模型,具体由一个两步迭代过程实现:

1. 适应修正γ_t.对于固定的 $\boldsymbol{v}_{t-1}$,令γ_t 为在参数 $\boldsymbol{v}_{t-1}$ 下的测试成本序列$\varphi(\boldsymbol{X})$的 $\rho\cdot100\%$ -分位数.即γ_t 满足: $P_{\boldsymbol{v}_{t-1}}(\varphi(X)\ge\gamma_t)\ge1-\rho$,并且$P_{\boldsymbol{v}_{t-1}}(\varphi(X)\le\gamma_t)\ge\rho$, 其中 $\boldsymbol{X}\sim f(\cdot;\boldsymbol{v}_{t-1})$.最简单的估计是根据 $f(\cdot;\boldsymbol{v}_{t-1})$ 抽出样本 $\boldsymbol{X}_1,\boldsymbol{X}_2,...,\boldsymbol{X}_N$ 计算出样本测试成本值$\varphi(\boldsymbol{X})$,并将其按从小到大的顺序排列: $\varphi_{(1)}\le\varphi_{(2)}\le...\le\varphi_{(N)}$,最后计算样本测试成本值的 $\rho\cdot100\%$ 来估计γ_t,即 $\hat{\gamma}_t=\varphi_{\lceil\rho\cdot N\rceil}$.

2. 适应修正 v_t.对于固定的γ_t 和 v_{t-1},由式(13)得到修正的 $\tilde{\boldsymbol{v}}_t$,并采用平滑技术得到 v_t 的修正值:

$$\hat{\boldsymbol{v}}_t=\alpha\tilde{\boldsymbol{v}}_t+(1-\alpha)\hat{\boldsymbol{v}}_{t-1},\ \alpha\in(0.4,0.9)\ [16].$$

在步骤 2 中采用平滑技术主要是为了减少 $\hat{\boldsymbol{v}}_t$ 的某些元素 $\hat{\boldsymbol{v}}_{t,i}$ 为 0 的概率,避免在算法的最初阶段一直在局部最优解上搜索.这样,在首次循环中,初始化概率矩阵为均匀矩阵,即均以概率 $1/K$ 进行抽样,由上面的过程可得 $\hat{\gamma}_1$ 和 $\hat{\boldsymbol{v}}_1$,如此循环,可得序列对 $(\hat{\gamma}_t,\hat{\boldsymbol{v}}_t),t=1,2,...$,则整个优化测试剖面生成过程由算法第 3.1 节给出.

算法 3.1. 优化测试剖面的生成.

[1] 初始化 $\hat{\boldsymbol{v}}_0$ 为均匀分布,即 $p_{ij}=1/K(i=1,2,...,K)$,令 t=1.

[2] 由概率分布 $f(\cdot;\hat{\boldsymbol{v}}_{t-1})$ 生成测试轨迹(或测试用例序列) $\boldsymbol{X}_1,\boldsymbol{X}_2,...,\boldsymbol{X}_N$ (由算法第 3.2 节给出),计算各个测试轨迹的测试成本 $\varphi(\boldsymbol{X})$ 并排序 $\varphi_{(1)},\varphi_{(2)},...,\varphi_{(N)}$,找出样本 $\rho\cdot100\%$ 分位数 $\hat{\gamma}_t=\varphi_{\lceil\rho\cdot N\rceil}$.

[3] 用同样的测试轨迹 $\boldsymbol{X}_1,\boldsymbol{X}_2,...,\boldsymbol{X}_N$,由式(13)解出 $\tilde{\boldsymbol{v}}_t$,应用平滑技术得到 $\hat{\boldsymbol{v}}_t$:

$$\hat{\boldsymbol{v}}_t=\alpha\tilde{\boldsymbol{v}}_t+(1-\alpha)\hat{\boldsymbol{v}}_{t-1}.$$

[4] 令 t=t+1,重复步骤 2~步骤 3,直到对于某个给定的 d(如 d=4)有 $\hat{\gamma}_t=\hat{\gamma}_{t-1}=...=\hat{\gamma}_{t-d}$ 为止.

软件测试轨迹可以根据马尔可夫过程的性质运用蒙特卡罗方法来随机产生.考虑到软件测试的目的是检测并排除软件中剩余的缺陷,并且假定软件中初始缺陷数为 M,因此,此马尔可夫过程的初始状态为 $s_0=M$,吸收状态为 s_τ=0,为终止状态(τ为首次到达状态 0 的时刻).在产生轨迹的过程中,如果它的状态转移到了吸收状态,则停止测试轨迹生成过程.这里假定总能在有限步停止.具体的蒙特卡罗模拟过程由算法第 3.2 给出.

算法 3.2. 运用蒙特卡罗模拟生成软件测试轨迹.

Input: $\boldsymbol{P}$//辅助概率矩阵

For (i=1 to N):

© 中国科学院软件研究所 http://www.jos.org.cn

[1] Initial $s_0=M$, set $t=0$.

[2] Repeat until ($s_t=0$)

a. 根据 $p_{s,a}$ 生成行动 a_t.

b. 由生成的行动 a_t 及当前所处的状态 s_t(记为 j)生成下一时刻的状态 s_{t+1},使得 s_{t+1} 以概率 $j\cdot\theta_{a_t}$ 等于 $j-1$,以概率 $1-j\cdot\theta_{a_t}$ 保留为原来状态 j,并观察到测试成本 C_t.

c. Set $t=t+1$.

[3] 得到一个测试轨迹: $H^{(i)}=\{s_0^{(i)},a_0^{(i)},C_0^{(i)},s_1^{(i)},a_1^{(i)},C_1^{(i)},...,a_{t-1}^{(i)},C_{t-1}^{(i)},s_t^{(i)}\}$, 根据轨迹 $H^{(i)}$,计算总的测试成本: $\varphi(\boldsymbol{X}^{(i)})=\sum_{j=0}^{t-1}C_j^{(i)}$.

Output:测试成本φ和轨迹 H.

由算法 3.2 产生的 N 个测试轨迹 $\boldsymbol{X}^{(1)},\boldsymbol{X}^{(2)},...,\boldsymbol{X}^{(N)}$ 及它们分别对应的测试成本, $\varphi(\boldsymbol{X}^{(1)}),\varphi(\boldsymbol{X}^{(2)}),...,\varphi(\boldsymbol{X}^{(N)})$, 通过式(13)修正参数矩阵 $\boldsymbol{P}=(p_{s_i a})_{M\times K}$.算法最终得到修正后的概率矩阵 $\boldsymbol{P}=(p_{s_i a})_{M\times K}$ 为优化测试剖面,按照这种测试剖面测试软件,将会使总的平均测试成本达到最小.我们将这种测试策略简称为“学习策略“.

4 实验分析

为了验证软件测试学习策略的效率,通过随机模拟方法与随机测试策略作比较.在随机测试中,测试用例(或行动)是根据均匀分布随机地在测试用例集中选取,也就是说,每一个可行的测试用例(行动)在测试中都以相同的机会被选取.在软件测试优化问题中,假定待测软件的参数(剩余缺陷数 M,各个行动缺陷检测率 θ_{a_t} 和不同状态下各个行动所需的测试成本 $C_s(a)$已知,在实际应用中,这些参数一般可以通过专家的经验或相关软件工程的方法估计出来.并假定在各个状态下,每种行为均可采用.

实验 4.1. 假定开始测试时软件中剩余缺陷数 M=3,行动空间 $\boldsymbol{A}$={1,2,3,4},相应的缺陷检测率为 $\theta_1=0.5,\theta_2=0.125,\theta_3=0.25,\theta_4=0.075$.进一步假定在不同状态下采用不同行动的测试成本为

$C_3(1)=9,C_3(2)=7,C_3(3)=4,C_3(4)=3,C_2(1)=25,C_2(2)=10,C_2(3)=8,C_2(4)=5,$

$C_1(1)=35,C_1(2)=17,C_1(3)=10,C_1(4)=8,C_0(1)=C_0(2)=C_0(3)=C_0(4)=0.$

表 4.1 给出了 $N=100,\rho=0.1,\alpha=0.5,d=5$ 时用交叉熵方法优化测试策略的学习过程.

Table 4.1 Learning process of testing strategy

表 4.1 测试策略的学习过程

Iteration number k	$\tilde{\gamma}_k$	$\min_i \varphi_k(\boldsymbol{X}^{(i)})$	$E_\pi[\varphi(\boldsymbol{X})]$
1	43.000 0	22.000 0	94.350 0
2	41.000 0	19.000 0	94.400 0
3	37.000 0	22.000 0	81.940 0
4	32.000 0	22.000 0	68.840 0
5	26.000 0	22.000 0	63.620 0
6	27.000 0	22.000 0	64.680 0
7	22.000 0	19.000 0	64.600 0
8	25.000 0	22.000 0	62.060 0
9	26.000 0	18.000 0	65.010 0
10	26.000 0	21.000 0	62.600 0
11	26.000 0	17.000 0	60.540 0
12	26.000 0	22.000 0	60.600 0
13	26.000 0	20.000 0	59.790 0

由学习策略得到的优化测试剖面为

$$\boldsymbol{P}^{opt}=\begin{bmatrix}0.0000 & 0.0000 & 0.9996 & 0.0004\\ 0.0000 & 0.0001 & 0.9951 & 0.0048\\ 0.0002 & 0.0001 & 0.9979 & 0.0019\end{bmatrix}.$$

采用优化测试剖面 $\boldsymbol{P}^{opt}$,对实验 4.1 进行 1 000 次模拟抽样,得到的平均测试成本为 60.959 0.

© 中国科学院软件研究所 http://www.jos.org.cn

从上面的模拟结果来看,每经过一次修正,新抽样出的测试轨迹的平均测试成本都会减少,并且向最优值逼近.由生成的优化测试剖面不难看出:优化剖面形成与比率 $C_s(a)/\theta_a$ 有关,相同状态下优先选择比率较小的行动,因此,这些行动被选择的概率要远远大于同一状态下的其他行动.这与实际情况是完全一致的[14,23].

实验 4.2(与随机测试测试成本的比较). 假定初始缺陷数 $M=7, \boldsymbol{A}=\{1,2,3,4,5,6\}$,并且假定 $\theta_1=0.5$, $\theta_2=0.125, \theta_3=0.25, \theta_4=0.075, \theta_5=0.13, \theta_6=0.15$ 不同状态下采用不同行动的测试成本 $C_s(a)$由表 4.2 给出.

Table 4.2 Test cost of different action a under different state s

表 4.2 不同状态 s 下采用不同行动 a 时的测试成本

State s (remaining software defects)	Action $1a_1$	Action $2a_2$	Action $3a_3$	Action $4a_4$	Action $5a_5$	Action $6a_6$
0	0	0	0	0	0	0
1	25	37	30	49	36	35
2	23	32	27	40	31	30
3	19	29	23	32	27	25
4	11	21	14	27	20	19
5	9	17	12	20	16	15
6	5	10	6	15	9	8
7	3	9	5	12	8	7

在随机测试中,由于测试用例是随机地抽取,即同一状态下,6 种行动等可能地被选取,则随机测试剖面的每个元素均为 1/6.

由学习策略得到的优化测试剖面为

$$\boldsymbol{P}^{opt}=\begin{bmatrix}0.9998 & 0.0000 & 0.0002 & 0.0000 & 0.0000 & 0.0000\\ 0.9996 & 0.0000 & 0.0001 & 0.0000 & 0.0001 & 0.0001\\ 0.9993 & 0.0002 & 0.0002 & 0.0001 & 0.0001 & 0.0002\\ 0.9959 & 0.0003 & 0.0036 & 0.0000 & 0.0001 & 0.0001\\ 0.9965 & 0.0004 & 0.0022 & 0.0003 & 0.0002 & 0.0004\\ 0.9854 & 0.0009 & 0.0109 & 0.0000 & 0.0019 & 0.0009\\ 0.9940 & 0.0008 & 0.0001 & 0.0002 & 0.0013 & 0.0037\end{bmatrix}.$$

采用优化测试剖面 $\boldsymbol{P}^{opt}$ 和随机测试剖面分别模拟运行 10 次,在全部检测并排除 7 个软件缺陷之后,两种测试策略的测试成本如图 2 所示.

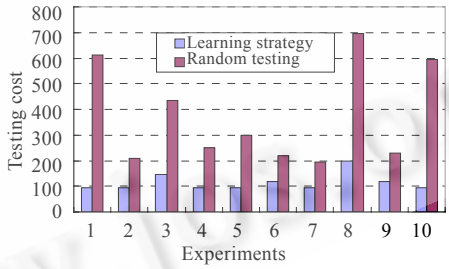

Fig.2 Test cost of two testing strategy

图 2 两种测试策略的测试成本

10 次模拟测试中学习策略的平均测试成本为 115.3,随机测试的平均测试成本为 374.1.模拟结果表明,用优化测试剖面 $\boldsymbol{P}^{opt}$ 测试并不能保证每次测试成本都比随机测试小,但其平均测试成本会远远低于随机测试的平均成本.其原因是,在模拟过程中,每一次行动是否能够检测到缺陷都具有随机性,在与随机测试的比较中应该尽量减少这种随机性的影响,因此只能比较平均测试成本,从这个角度来看,学习测试策略优于随机测试.

实验 4.3(与随机测试测试能力的比较). 假定初始缺陷数 M=35,行动空间 $\boldsymbol{A}$={1,2},两种测试行动的缺陷检测率分别为 $\theta_1=0.025, \theta_2=0.01$,进一步假定在所有状态下采用这两种测试行动的成本都相同,均为 2,即 $C_s(a)=2(s=1,2,...,35, a=1,2)$.用学习策略和随机测试两种方式分别模拟测试 6 次,图 3 给出了 6 次测试过程中

© 中国科学院软件研究所 http://www.jos.org.cn

检测不同缺陷数时所用测试用例数的散点图,图 4 给出 6 个模拟测试中检测不同的缺陷数时所需平均测试用例数的曲线图.

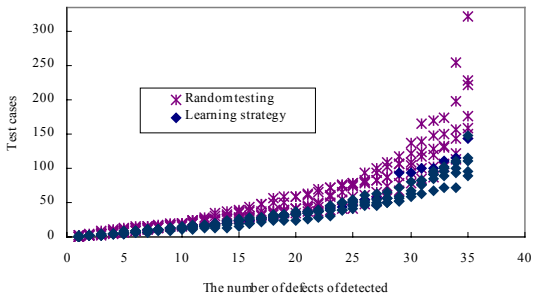

Fig.3 Comparison of the number of test cases used by learning strategy with random testing
图 3 学习策略与随机测试使用的测试用例数比较

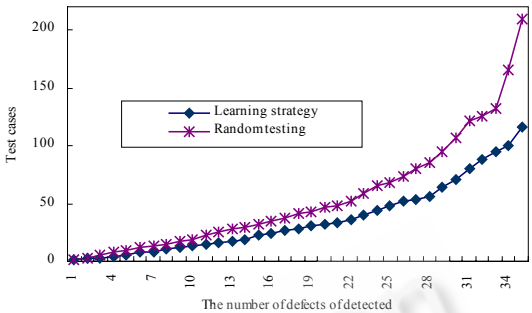

Fig.4 Comparison of the average number of test cases used by learning strategy with random testing
图 4 学习策略与随机测试平均使用的测试用例数比较

模拟结果表明,如果要检测相同个数的软件缺陷,学习策略比随机测试所用的测试用例一般都要少.例如,在第 2 次模拟中,学习策略只需要运行 34 个测试用例就可以检测到 21 个软件缺陷,而随机测试则需要运行 60 个测试用例才能检测到 21 个软件缺陷.另一方面,图 3 也显示出检测并排除某一数目的软件缺陷,学习策略一般比随机测试所用的测试成本都要少.例如,在第 5 次模拟测试中,检测前 17 个软件缺陷学习策略需花费的测试成本为 44,而随机测试则需要花费的测试成本为 60.由图 4 可知,在软件刚开始测试时,学习策略的缺陷检测能力并不明显优于随机测试,例如检测并排除 2 个软件缺陷学习策略平均需要 2.166 7 个测试用例,随机测试平均也只需要 2.666 7 个测试用例.随着软件缺陷的排除,学习策略的缺陷检测能力明显优于随机测试,例如要完全检测出 35 个软件缺陷,学习策略平均需要使用 116.666 7 个测试用例,随机测试则平均需要使用 209.833 3 个测试用例.其原因是软件刚开始测试时,由于待测软件中剩余缺陷数比较大,每运行一个测试用例检测到缺陷的概率都比较大,因此,在检测前面的几个缺陷时学习策略比随机测试的优势并不非常显著.随着测试的进行,待测试软件中所剩余的缺陷数越来越少,每运行一个测试用例检测到缺陷的概率就会变得越来越小,测试难度也越来越大,学习策略检测缺陷的能力要远远地高于随机测试.

5 结论及进一步工作

软件测试的马尔可夫决策模型将测试过程处理成不确定环境下的序列决策问题.本文给出了一个软件测试的马尔可夫决策模型.该模型包括 5 个基本元素:决策时刻、状态空间、行动空间、转移概率矩阵及决策目标.在某些软件参数(如缺陷检测率)已知或已估计出的条件下,以最小化软件测试的平均费用为目标,运用交叉熵的方法,提出了一种学习策略,搜索最优的测试剖面,用于优化软件测试.模拟结果显示出学习策略给出的测试剖面用于指导测试要优于随机测试策略,检测和排除相同数目的软件缺陷,学习策略比随机测试能够显著地减少测试用例数,降低测试成本.

本文给出的学习策略是为实现测试目标而采用的一种带导向性的测试用例选择方法,具有较强的自我学习能力.可用于软件测试的动态仿真运行,特别是对于一些重要系统及高可靠性系统,通过仿真运行,可以减少测试用例数,降低测试成本.更重要的是,可增加重要系统和高可靠性系统的安全性,为软件测试的仿真研究给出了一种新方法,为“基于搜索的软件工程”[24]的研究提供了一种新尝试.由于学习策略采用的交叉熵方法本身具有很强的适应能力,只需处理少量的参数就能保证算法的收敛性,交叉熵方法在求最优解的过程中没有采用梯度的方法,它比其他方法更稳健[25].另外,由于术语“行动”具有多种理解,因此该学习策略可以在软件测试中广泛地应用.

© 中国科学院软件研究所 http://www.jos.org.cn

利用学习策略优化软件测试需要预先确定某些参数值,在实际使用中一般采用经验数据或通过专家经验进行估计,参数估计的精确性直接影响测试策略的优劣,如何利用在测试过程中观察到的数据对这些参数作在线调整还需进一步深入研究.另外,基于马尔可夫决策模型的软件质量度量方法,以及考虑软件可靠性、可信性等多种因素约束时更具实用价值的软件测试优化方法还需作进一步的研究.

致谢 我们衷心感谢不知名的评审人对文章提出的宝贵修改意见,使文章质量得到进一步提高.

References:

[1] Cai KY. Optimal software testing and adaptive software testing in the context of software cybernetics. Information and Software Technology, 2002,44(4):841−855.

[2] Rivers AT, Vouk MA. Resource-Constrained non-operational testing of software. In: Proc. of the 9th Int'l Symp. on Software Reliability Engineering. Paderborn, 1998. 154−163. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=730874

[3] Walton GH, Poore JH. Measuring complexity and coverage of software specifications. Information and Software Technology, 2000,42(12):859−872.

[4] Frankl PG, Hamlet RG, Littlewood B, Strigini L. Evaluating testing methods by delivered reliability. IEEE Trans. on Software Engineering, 1998,24(6):586−601.

[5] Ntafos S. The cost of software failures. In: IEEE Proc. of the Int'l Association of Science and Technology for Development, Int'l Confabulation Software Engineering. IASTED/ACTA Press, 1998. 53−57.

[6] Berman O, Cutler M. Resource allocation during tests for optimally reliable software. Computers and Operations Research, 2004,31(11):1847−1865.

[7] Dai YS, Xie M, Poh KL, Yang B. Optimal testing-resource allocation with genetic algorithm for modular software systems. The Journal of Systems and Software, 2003,66(1):47−55.

[8] Czuchra W. Optimizing budget spendings for software implementation and testing. Computers and Operations Research, 1999,26(7):731−747.

[9] Tal O, McCollin C, Bendell A. An optimal statistical testing policy for software reliability demonstration of safety-critical systems. European Journal of Operational Research, 2002,137(3):544−557.

[10] Sahinoglu M, Mayrhauser VA, Hajjar A, Chen T, Anderson C. On the efficiency of a compound Poisson stooping-rule for mixed strategy testing. In: Proc. of the 4th Int'l High-Assurance Systems Engineering Symposium (HASE). 1999. 249−256. http://ieeexplore.ieee.org/iel5/6411/17136/00790193.pdf?arnumber=790193

[11] Sahinoglu M. An empirical Bayesian stopping rule in testing and verification of behavioral models. IEEE Trans. on Instrumentation and Measurement, 2003,52(5):1428−1442.

[12] Rajgopal J, Mazumdar M. Modular operational test plans for inferences on software reliability based on a Markov model. IEEE Trans. on Software Engineering, 2002,28(4):358−363.

[13] Chen TY, Yu YT. A decision-theoretic approach to the test allocation problem in partition testing. IEEE Trans. on Systems Man and Cybernetics-Part A: Systems and Humans, 2002,32(6):733−745.

[14] Cai KY, Li YC, Ning WY. Optimal software testing in the setting of controlled Markov chains. European Journal of Operational Research, 2005,162(2):552−579.

[15] Cai KY, Chen TY, Li YC, Ning WY, Yu YT. Adaptive testing of software components. In: Haddad H, Liebrock LM, Omicini A, Wainwright RL, eds. Symp. on Applied Computing Proc. of the 2005 ACM Symp. on Applied Computing. ACM, 2005. 1463−1469.

[16] Boer DPT, Kroese DP, Mannor S, Rubinstein RY. A tutorial on the Cross-Entropy method. Annals of Operations Research, 2005,134(1):19−67.

[17] Rubinstein RY. Combinatorial optimization, cross-entropy, ants and rare events. In: Uryasev S, Pardalos PM, eds. Stochastic Optimization: Algorithms and Applications. Kluwer Academic Publishers, 2001. 304−358.

© 中国科学院软件研究所 http://www.jos.org.cn

[18] Rubinstein RY. Combinatorial optimization via cross-entropy. In: Gass S, Harris C, eds. Encyclopedia of Operations Research and Management Sciences. Kluwer Academic Publishers, 2001. 102−106.

[19] Rubinstein RY. The simulated entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability, 1999,2:127−190.

[20] Doerner K, Gutjahr WJ. Extracting test sequences from a Markov software usage model by ACO. LNCS 2724, Springer-Verlag, 2003. 2465−2476.

[21] Ridder A. Importance sampling simulations of Markovian reliability systems using cross-entropy. Annals of Operations Research, 2005,134(1):119−136.

[22] Hui KP, Bean N, Kraetzl M, Kroese DP. The cross-entropy method for network reliability estimation. Annals of Operations Research, 2005,134(1):101−118.

[23] Zhang DP, Nie CH, Xu BW. Optimal allocation of test case in partition testing. Journal of Nanjing University (Natural Sciences), 2005,41(5):553−561 (in Chinese with English abstract).

[24] Harman M, Jones BF. Search-Based software engineering. Information and Software Technology, 2001,43(14):833−839.

[25] Margolin L. On the convergence of the cross-entropy method. Annals of Operations Research, 2005,134(1):201−214.

附中文参考文献:

[23] 张德平,聂长海,徐宝文.划分测试中测试用例最优分配问题研究.南京大学学报,2005,41(5):553−561.

张德平(1973－),男,湖南澧县人,博士生,讲师,主要研究领域为软件测试技术,软件可靠性,数理统计与随机过程.

徐宝文(1961－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为程序设计语言,软件工程,并行与网络软件.

聂长海(1971－),男,博士,副教授,主要研究领域为软件测试技术,模糊信息处理,神经网络.

© 中国科学院软件研究所 http://www.jos.org.cn