·软件技术与数据库· 文章编号:1000-3428(2016)03-0084-05 文献标识码:A 中图分类号:TP309

基于三目标搜索的测试用例集约简方法

张 妍,傅秀芬

(广东工业大学计算机学院,广州 510006)

摘 要: 现有测试用例集约简方法基本只考量优化集的规模和错误覆盖率,很少考虑其测试效率,导致优化集的测试运行时间较长,且运行代价较大。针对该问题,提出一种三目标搜索算法。对测试用例覆盖度和运行代价的双目标优先级模型进行细化,给出每个目标的定量细化方法,根据此模型计算出各个用例的优先级,运用到改进蚁群算法中作为各节点的初始信息素值进行优化遍历搜索,将错误检测率作为蚂蚁挑选下一个节点的影响因子,并制定信息素更新规则,使其能够尽快找到最优解。实验结果表明,与贪心算法、基本蚁群算法等相比,该算法得到的测试用例最小集规模较小,具有较高的错误检测率。

关键词: 软件测试;双目标模型;优先级;错误检测率;改进蚁群算法

中文引用格式: 张 妍,傅秀芬. 基于三目标搜索的测试用例集约简方法[J]. 计算机工程,2016,42(3):84-88.

英文引用格式: Zhang Yan, Fu Xiufen. Test Suite Reduction Method Based on Triple Target Search[J]. Computer Engineering,2016,42(3):84-88.

Test Suite Reduction Method Based on Triple Target Search

ZHANG Yan, FU Xiufen

(School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China)

【Abstract】 Existing test case reduction methods basically only take the size and the error detection rate of the optimization set into consideration, rarely paying attention to the test efficiency, so that the test of the opitimization set takes long, and the test cost is high. For this problem, this paper proposes a triple target search algorithm. It details the double target priority model based on the coverage and the test cost of the test case, refines each target's quantitative method, then calculates the priority of each test case according to the model, uses the priority as the initial pheromone value in the improved ant colony algorithm for ergodic optimization, at last regards the error detection rate as the impact factor for the ant to pick the next node, and formulates the pheromone update rule to find the optimal solution as soon as possible. Experimental results show that compared with the greedy algorithm, basic ant colony algorithm and other algorithms, this algorithm can get a smaller optimization test suite, higher error detection rate.

【Key words】 software test; double target model; priority; error detection rate; improved ant colony algorithm

DOI: 10.3969/j.issn.1000-3428.2016.03.015

1 概述

软件测试是软件产品开发中的不可忽视的阶段,规模越大的软件产品的测试需求越大,需要的测试用例就越多。为了缩减测试成本,同时提高测试效率,需要对测试用例集进行优化,约简不必要的测试用例,用尽可能少的测试代价对软件产品进行测试。如何使用更少的测试用例及测试代价达到测试目标一直是软件测试的核心问题。不少学者及研究机构对此进行研究,提出了主流算法,如贪心算法[1]、HGS(High Greedy Strategy)算法[2]、GRE(Greedy Redundance Essential)算法[3]、整数规划方法[4]和 concolic 算法[5]等。这些算法互有优劣,也都具有一定改进空间。为此,本文先对各测试用例求其优先级,然后将其用于改进的蚁群算法,以进行测试用例集的优化。

2 研究现状

1993 年,Harrold M J 等[1]提出了约简软件测试用例集的概念,并给出了一个启发式方法,根据测试用例的重要性进行选择。文献[6]进一步对软件测试

基金项目: 广东省自然科学基金资助项目"对象监控机及其权限提升攻击检测研究"(9151009001000007)。

作者简介: 张 妍(1990-),女,硕士,主研方向为软件测试、协同计算;傅秀芬,教授。

收稿日期: 2015-03-05 **修回日期:** 2015-04-24 **E-mail:** 1010551576@qq.com

用例集约简技术中的一些术语给出了通用定义,加入了 1-1 冗余测试用例的考虑。文献[7-8]给出了测试用例排序选择的想法,在此基础上,文献[9-10]提出了基于优先级选择测试用例的方法。文献[11-12]提出了对测试需求进行优化的方法,文献[3]将启发式算法与贪心算法融合,提出了 HGS、GRE 等算法,文献[4]借助 LINDO 等软件包将测试用例集优化难题使用 0-1 规划思想来求解,以保证得到的是最优测试用例代表集。以上描述的都是基于启发式算法演化而来的方法,除此之外还有基于图、遗传算法、蚁群算法等遍历搜索优化技术[13-15],和基于形式化方法的测试用例研究,例如基于统一建模语言(Unified Modeling Language,UML)模型的测试用例生成方法[16]和文献[17]提出的 concolic 方法。其中,concolic 方法将具体执行与符号执行方法相混合,对于程序原始代码使用具体执行,而插入函数则在不干扰正常执行的情况下使用符号执行技术,2 种方法并行使用,从对方那里得到结果反馈。这种方法自动化性能更优越,但使用范围还有待于进一步推广。

上述方法在选择测试用例时基本都只以测试用例对测试需求的覆盖率为依据进行考虑,但实际上还有一些重要的因素同样也应该纳入考虑范围,比如错误检测率和测试代价[18]。以多个因素为影响因子来选择测试用例更符合现实中的情况,能得到效率更高、代价更小的优化集[19]。

3　双目标优先级模型

文献[20]提出同时考虑测试用例对需求的覆盖度和其运行代价的测试用例集约简方法,但只是简单地将其运用于贪心算法和遗传算法中进行选择,并没有进行深入研究。本文对双目标优先级模型[21]进行优化,提出每个目标的细化方法,并将其结合起来计算各测试用例的优先级,用于后续测试用例集优化操作。

3.1　测试用例覆盖度

初始测试用例集能够满足整个需求,每个用例都满足其中一部分。本节用非空有限集 T 和 R 分别代表测试用例集和需求集,且令 $|T|=m$,$|R|=n$。测试用例对需求的覆盖度可以用 Cov 来表示:

$$Cov_i = \sum_{j=1}^{n} \sum_{x=\text{true}}^{\text{false}} \frac{1}{N_{j_x}} \tag{1}$$

其中,Cov_i 表示测试用例 t_i 的覆盖度,$t_i \in T$,且 $1 \leqslant i \leqslant m$。

设 $T_{j_x}(T_{j_x} \in T)$ 是满足条件 $C_j(x)$ 的测试用例集。如果 $t_i \in T_{j_x}$,则 $|N_{j_x}| = |T_{j_x}|$;否则 $N_{j_x}=0$,且规定 $\frac{1}{N_{j_x}}=0$。$C_j \in C$,且 $1 \leqslant j \leqslant n$。$x \in \{\text{true},\text{false}\}$。

由此可见,一个用例满足的需求越广,能和其相比的用例就越稀少[22]。

3.2　测试用例运行代价

在软件测试中,测试代价是测试质量的限制因素之一。在测试时间及成本有限的情况下,测试用例集中用例不可能每个都测试到,所以,需要对用例进行选择,从具有相同需求覆盖率的用例中挑选出那些运行代价比较小的进行测试。测试代价用下式表示:

$$Cost_i = (1-\lambda) \cdot price(t_i) + \lambda \cdot time(t_i) \tag{2}$$

其中,$Cost_i$ 表示测试用例 t_i 的执行代价,$t_i \in T$,且 $1 \leqslant i \leqslant m$;$price(t_i)$ 是用例 t_i 的运行开销;$time(t_i)$ 是用例 t_i 的运行时间;λ 是一个偏好系数,$0 \leqslant \lambda \leqslant 1$,当 $\lambda=0$ 时,表示本次测试的约束条件是用例的运行开销,与运行时间无关,当 $\lambda=1$ 时,则表示约束条件是用例的运行时间,与运行开销无关,当 $0<\lambda<1$ 时,λ 越大则说明测试用例的运行时间越重要。

3.3　双目标模型的构造

根据上述分析,本节给出的双目标模型如下:

$$F(t_i)=\alpha \cdot \frac{1}{\lambda_{\text{cov}}} cov(t_i) + (1-\alpha) \cdot \frac{1}{\lambda_{\text{cost}}} cost(t_i) \tag{3}$$

其中,μ 是偏好系数,反映了测试人员对 2 个因素的侧重程度,且 $0 \leqslant \mu \leqslant 1$;$\lambda_{\text{cov}}$ 和 λ_{cost} 分别为 2 个因素的权系数,取:

$$\lambda_{\text{cov}} = \max Cov(t)$$

$$\lambda_{\text{cost}} = \max Cost(t)$$

且若 $\lambda_{\text{cost}}=0$,则定义 $\frac{1}{\lambda_{\text{cost}}}=0$。

在双目标模型中,$F(t)$ 值越大说明该测试用例的性能越强,可以根据这个模型按各测试用例的 $F(t)$ 值来设定一个优先级,为下一步优化测试用例做准备。

4　改进的蚁群算法

有学者通过观察蚁群觅食规则,提出了基于蚁群遍历搜索的测试用例集优化技术[23],但这种方法同样存在一些缺陷:(1)基本蚁群算法对一个规模较大的测试用例集进行优化时搜索速度较慢、成本较高;(2)基本蚁群算法的随机性过大;(3)蚁群迭代次数过大时容易造成时间和成本的浪费,若迭代次数太小又无法保证找到最优解。

针对以上缺点,本文将双目标优先级模型结合在蚁群算法中进行改进,即对初始测试用例集求出每个用例的优先级值,然后按其大小来定义各节点的初始信息素浓度,使改进蚁群算法在遍历时更有目的性,且收敛更快。再以测试用例的错误检测率作为蚂蚁挑选下一个节点的影响因子,减小

算法随机程度,尽快找到最优解,这种算法对大型系统操作更有优势。实验证明,改进蚁群算法得到的测试用例优化集能达到较小的用例规模和运行代价。

4.1 测试用例的错误检测度

测试用例发现问题的能力可能由多种影响因素构成[14],本节将某测试用例上次测试中发现的问题数量当做影响该用例发现问题能力的关键。设用例 t_i 在第 s 次测试中错误检测度为 $[Fau_i]_s$,则:

$$[Fau_i]_s = [Fau_i]_{s-1} + f_i \tag{4}$$

其中,f_i 是 t_i 在第 s 次测试中发现的错误数目,且 $[Fau_i]_0 = 0$。

可见测试用例的错误检测度在回归测试过程中会不停增大,若某用例的错误检测度持续增大,则说明该用例发现的问题一直没有被彻底修复。

4.2 定义和术语

设 n 表示节点个数;m 为蚂蚁的总数量;这里令开始时每个节点上都有一只蚂蚁,即 $m = n$;$\tau_i(t)$ 代表 t 时刻节点 i 的信息素值;$\eta_i(t)$ 是启发函数,代表蚂蚁选择节点 i 的期望,大小是 $[Fau_i]_s$,代表错误检测率越高,该用例被挑选的可能性就越高;$\Delta\tau_i(t)$ 表示这次遍历过程中蚂蚁在节点 i 上留下的信息素值,开始时 $\Delta\tau_i(0) = F(i)$;$p_{ij}^k(t)$ 代表蚂蚁 k 在 t 时刻由节点 i 转移到 j 的可能性,计算公式如下:

$$p_i^k = \begin{cases} \dfrac{\tau_i^\alpha \cdot \eta_i^\beta}{\sum\limits_{s \subset allowed_k} \tau_s^\alpha \cdot \eta_s^\beta} & j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

其中,禁忌表 $tabu_k(k = 1,2,\cdots,m)$ 用来放置蚂蚁 k 已走过的节点,并在整个迭代过程中不停进行调整;$allowed_k = \{T - tabu_k\}$ 代表蚂蚁 k 还没有访问过的节点;$\alpha(\alpha > 0)$ 为信息启发因子,代表某节点上的信息素总量对蚂蚁选择的影响;$\beta(\beta \geqslant 0)$ 是期望启发因子,越大则表示启发信息对蚂蚁挑选的影响越大,节点越容易被选中。

4.3 信息素更新规则

为防止各节点上的启发信息被遗留过多的信息素所掩盖,在一次遍历后要更新节点上的信息素含量。本文除了要更新每个曾被选中节点上的信息素,还要另外更新最优解集中节点的信息素。若节点 i 被蚂蚁选中,将 i 添加到解集里,并从 R 里删除对应的需求,直到 R 变成 Φ,则本轮遍历搜索结束,持续到每只蚂蚁都找到解集就停止迭代,然后按以下规则对各节点信息素进行更新:

$$\tau_j(nc_{i+1}) = (1-\rho)\tau_j(nc_i) + \Delta\tau_j(nc_i, nc_{i+1}) + \Delta\tau_j^{best}(nc_i, nc_{i+1}) \tag{6}$$

其中,ρ 是挥发系数,$1-\rho$ 反映了信息素的遗留情况;nc_i 代表第 i 次遍历;$\Delta\tau_j(nc_i, nc_{i+1})$ 表示节点 j 在第 i 次迭代中新增的信息素含量:

$$\Delta\tau_j(nc_i, nc_{i+1}) = \begin{cases} \dfrac{Q}{C_k(nc_i)} & \text{蚂蚁 } k \text{ 在第 } i \text{ 轮中经过节点 } j \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$\Delta\tau_j^{best}(nc_i, nc_{i+1}) = \begin{cases} \dfrac{Q}{C^{best}(nc_i)} & \text{节点 } j \text{ 在第 } i \text{ 次迭代的最优解集中} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

其中,$C_k(nc_i)$ 代表第 i 次遍历时蚂蚁 k 构造的解集代价;$C^{best}(nc_i)$ 代表第 i 次遍历得到的最小用例集的代价。当第 i 次遍历时,按式(7)更新所有被选中的节点的信息素浓度,再按式(8)另外增强第 i 次遍历的最优解集中各节点的信息素,也就是说在第 $i+1$ 次遍历前先用式(6)计算得出所有节点的信息素含量。

5 实验研究

5.1 三目标搜索算法

三目标搜索算法如下:

```
Begin:
  构造关系矩阵 T×R
  for i = 1 to n
    计算 F(t_i),令 Δτ_i(0) = F(t_i)
  End for
  While(i = i to 100 or 2 次迭代的最优解差值不小于 0.01)
    for k = 1 to n
      tabu_k = φ      //蚂蚁 k 走过的节点
    End for
    Repeat until 每只蚂蚁都有一子集且能覆盖所有需求
      在 allowed_k 中挑选 p_ij^k(t)最大的一个节点,将其加到 tabu_k 中
    End Repeat
    for k = 1 to n
      选出代价最小的 tabu_k
    End for
    依据信息素更新规则更新每个节点上的信息素含量
  End While
  输出最后一轮得到的 min{tabu_k}
End
```

5.2 模拟实验

为了验证算法效率,设计一系列仿真对比实验,将改进蚁群算法与贪心算法、GRE 算法、基本蚁群算法和 UML 模型算法进行比较,这 4 种算法分别代表基本测试用例集优化算法、启发式算法、基本蚁群算法和形式化算法。将这些算法对同一段程序的测试用例集进行优化,并比较约简效果、运行时间、测试代价和错误检测率 4 项,可以充分说明实验结果具

有普遍性。

实验设置了不同规模的测试用例-需求集关系矩阵,每个测试用例的测试代价由式(2)计算得到,各算法参数不变进行 20 次实验然后取结果的平均值,比较结果如图 1 所示。

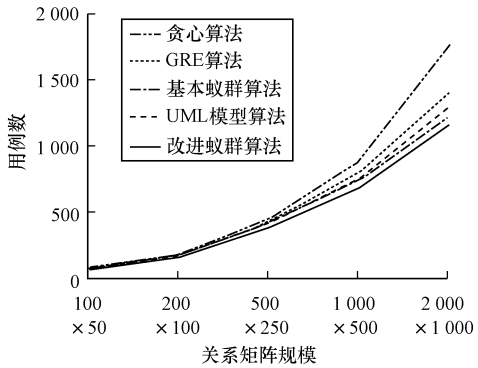

图 1　最优解规模比较

由图 1 可以直观地看出 5 种算法的优化效果:贪心算法规模最大,GRE 算法和 UML 模型算法次之,基本蚁群算法再次,改进蚁群算法最小,优化效果最好。并且当关系矩阵规模较小时,各算法结果相差不大,但随着关系矩阵规模变大,改进蚁群算法的解集相比于其他算法的优势越来越明显,测试用例集的约简效果越来越好。

通过图 2 可以看出,基本蚁群算法所需运行时间最长,改进蚁群算法次之。这是因为蚁群算法的运行时间与迭代次数密切相关,次数越多,耗费时间也就越多,改进过的蚁群算法迭代次数远比基本蚁群算法少,所以所用时间也较少。其次是 UML 模型算法、GRE 算法,贪心算法所需执行时间最短。

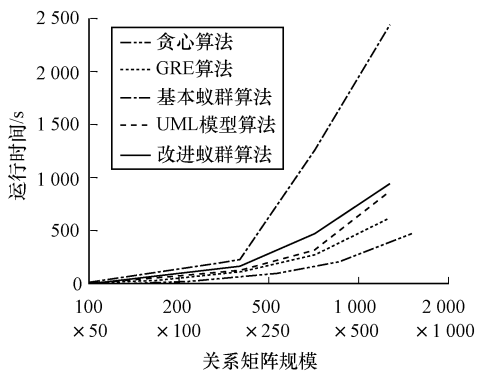

图 2　运行时间比较

从图 3 可以看出,随着关系矩阵规模的增大,改进蚁群算法所得测试用例优化代表集的测试代价始终小于其他 4 种算法的测试代价,且此优势随着规模的增大而愈发明显。这是因为改进蚁群算法是这些算法中唯一考虑了测试用例的测试代价的算法,在这方面进行了有意识的控制,所以优势比较明显。

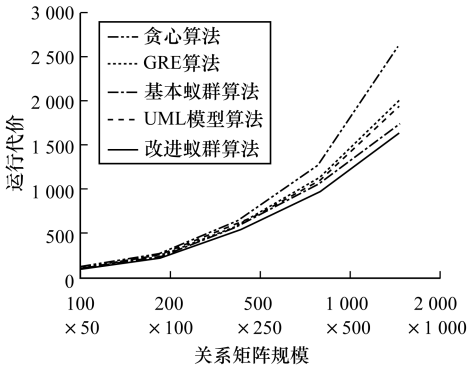

图 3　运行代价比较

在这一项实验中,分别用 5 种算法的最优解集对已知错误总数的 4 段程序进行测试,统计每种算法各自发现的错误数量,然后除以错误总数计算其错误检测率。由于只有改进蚁群算法考虑了测试用例的错误检测率,因此该算法最优解的错误检测率一直高于其他 4 种算法,且随代码规模增大,错误数量增多有差距变大的趋势。结果如图 4所示。

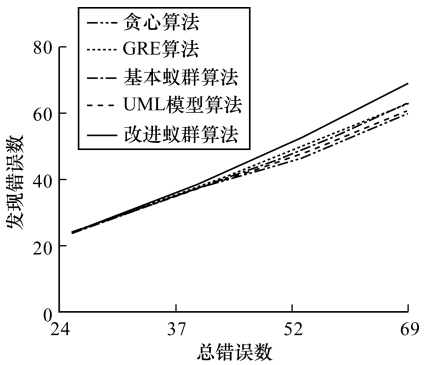

图 4　5 种算法发现错误数和总错误数的关系

综上所述,无论测试用例集-测试需求集关系矩阵规模如何变化,三目标搜索算法最优解集的运行代价和错误检测率都比其他 4 种算法要好,并且随着初始测试用例和需求规模的增大,改进蚁群算法的优化效果也越突出,但其运行时间略多于启发式算法及形式化算法,需要继续改进。

6　结束语

本文分析典型测试用例集优化算法的优劣,并提出一种基于测试覆盖度、运行代价和错误检测率 3 个目标的搜索算法,从多维度优化测试用例集。实验结果表明,在同等初始测试用例集规模下,该算法得到的测试用例最小集规模和运行代价都小于其他算法,错误检测率较优,且初始集规模越大,优化效

果越明显。下一步将引入局部最优搜索策略对改进蚁群算法中的选路算法进行优化,从而减少蚂蚁选路次数,降低算法时间复杂度。

参考文献

[1] Harrold M J, Gupta R, Soffa M L. A Methodology for Controlling the Size of a Test Suite[J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285.

[2] Xu Shenwei, Miao Huaikou, Gao Honghao. Test Suite Reduction Using Weighted Set Covering Techniques[C]// Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Washington D. C., USA: IEEE Press, 2012: 307-312.

[3] 万松松,薛锦云,谢武平. 最小测试用例集生成方法改进及应用[J]. 计算机技术与发展,2008,18(10):181-183.

[4] Lee J G, Chuang C G. An Optimal Representative Set Selection Method[J]. Information and Software Technology, 2000, 42(1): 17-25.

[5] Sen K. Concolic Testing[C]//Proceedings of International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2007: 571-572.

[6] Chen T Y, Lau M F. On the Divide-and-conquer Approach Towards Test Suite Reduction[J]. Information Sciences, 2003, 152(1): 89-119.

[7] 朱海燕. 软件测试用例集缩减的一个算法[J]. 微电子学与计算机,2007,24(1):204-206.

[8] 朱海燕,范 辉,谢青松,等. 测试用例排序的研究[J]. 计算机工程与科学,2008,30(1):79-81.

[9] 孙继荣,李志蜀,倪建成,等. 回归测试用例集优化策略[J]. 吉林大学学报:工学版,2008,(S2):184-190.

[10] 彭园园,熊盛武. 测试用例集的优化技术分析与改进[J]. 现代电子技术,2009,32(6):77-80.

[11] 章晓芳,徐宝文,聂长海,等. 一种基于测试需求约简的测试用例集优化方法[J]. 软件学报,2007,18(4):821-831.

[12] 章晓芳,陈 林,徐宝文,等. 测试用例集约简问题研究及其进展[J]. 计算机科学与探索,2008,2(3):236-247.

[13] 罗文兵,赵 亮,赵洪宇. 基于图分析的测试用例集优化[J]. 计算机工程,2010,36(15):92-93,96.

[14] 丁革建,郑燕妮,张 璐. 基于蚁群算法的测试用例集最小化研究[J]. 计算机工程,2009,35(6):213-218.

[15] 申利民,高 洁. 基于遗传蚁群融合算法的测试用例最小化研究[J]. 计算机工程,2012,38(16):57-61.

[16] 任洪丽. 基于形式化方法的软件测试技术研究——基于 UML 模型蚁群算法的软件测试用例研究[D]. 无锡:江南大学,2011.

[17] Sen K, Marinov D, Agha G. Cute: A Concolic Unit Testing Engine for C[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 263-272.

[18] Hareton K N, Leung L W. A Cost Model to Compare Regression Test Strategies[C]//Proceedings of Conference on Software Maintenance. Washington D. C., USA: IEEE Press, 1991: 201-208.

[19] 王子元,聂长海,徐宝文,等. 相邻因素组合测试用例集的最优生成方法[J]. 计算机学报,2007,30(2):200-211.

[20] 马雪芳,盛斌奎. 测试用例最小化研究[J]. 计算机应用研究,2007,24(7):35-39.

[21] 潘丽丽. 软件测试用例集简化及其构建方法研究[D]. 长沙:湖南大学,2008.

[22] Gu Qing, Tang Bao, Chen Daoxu. Optimal Regression Testing Based on Selective Coverage of Test Requirements[C]//Proceedings of International Symposium on Parallel and Distributed Processing with Applications. Washington D. C., USA: IEEE Press, 2010: 419-426.

[23] 张 瑞. 基于改进蚁群算法的测试用例集约简技术研究[D]. 广州:华南理工大学,2012.

编辑 刘 冰

(上接第 83 页)

[4] Bernaille L, Teixeira R, Salamatian K. Early Application Identification[C]//Proceedings of the 9th International Conference on Emerging Networking Experiment and Technologies. New York, USA: ACM Press, 2006: 1-12.

[5] 邓 河,阳爱民,刘永宝. 一种基于 SVM 的 P2P 网络流量分类方法[J]. 计算机工程与应用,2009,44(14):122-126.

[6] 张宏莉,鲁 刚. 分类不平衡协议流的机器学习算法评估与比较[J]. 软件学报,2012,23(6):1500-1516.

[7] Wang Yu, Xiang Yang, Zhang Jun, et al. Internet Traffic Clustering with Side Information[J]. Journal of Computer and System Sciences, 2014, 80(5): 1021-1036.

[8] Zhang Jun, Chen Chao, Xiang Yang, et al. Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions[J]. IEEE Transactions on Information Forensics and Security, 2013, 8(1): 5-15.

[9] Isabelle G, Elisseeff A. An Introduction to Variable and Feature Selection[J]. Journal of Machine Learning Research, 2003, 3: 1157-1182.

[10] Duda R O, Peter E H, David G S. Pattern Classification[M]. 2nd ed. [S.l.]: John Wiley & Sons, 2000.

[11] Josef K, Hatef M, Duin R P W, et al. On Combining Classifiers[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998, 20(3): 226-239.

[12] Lee S, Kim H, Barman D, et al. Netramark: A Network Traffic Classification Benchmark[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(1): 22-30.

[13] Hyunchul K, Claffy K C, Fomenkov M, et al. Internet Traffic Classification Demystified: Myths, Caveats and the Best Practises[C]//Proceedings of 2008 ACM CoNEXT Conference. New York, USA: ACM Press, 2008: 9-12.

编辑 金胡考