运行时验证技术的研究进展

张 硕[1] 贺 飞[1,2,3]

(清华大学软件学院 北京 100084)[1] (信息系统安全教育部重点实验室 北京 100084)[2]

(清华信息科学与技术国家实验室(筹) 北京 100084)[3]

摘 要 运行时验证是一种轻量级的验证方法,通过实时地监测系统的行为,验证系统的正确性,及时发现冲突,并发出警告或作出反应。运行时验证技术已经得到了越来越多的应用,以确保软件系统的正确性。总结了近年来运行时验证技术的研究进展,首先介绍了运行时验证的概念、原理和分类,接着深入分析了现有的几种解决方案,并对该领域中的研究热点进行了深入探讨,最后分析了运行时验证技术面临的主要挑战,并对未来该领域的研究方向进行了展望。

关键词 运行时验证,运行时监控,软件正确性,形式化方法

中图法分类号 TP301.2 **文献标识码** A

Research Advances in Runtime Verification

ZHANG Shuo[1] HE Fei[1,2,3]

(School of Software, Tsinghua University, Beijing 100084, China)[1]

(Key Laboratory for Information System Security, Ministry of Education, Beijing 100084, China)[2]

(Tsinghua National Laboratory for Information Science and Technology(TNList), Beijing 100084, China)[3]

Abstract Runtime verification is a lightweight verification technique which monitors the behaviors of the target systems and checks whether their behaviors satisfy the desired properties. Once a violation is observed, the monitor informs the system to react in time. Runtime verification has been applied to various areas to ensure the correctness of systems. In this paper, the concepts, principles and categories of runtime verification were first introduced. Then several solutions and research hotspots in this area were analyzed. Finally, we discussed the current challenges, and outlined the future research directions of runtime verification.

Keywords Runtime verification, Runtime monitoring, Correctness of software, Formal methods

1 引言

随着计算机科学的发展,越来越多的软件系统渗透到人们的生活之中。年轻人喜欢社交网络,出行的人需要乘坐列车、飞机,银行需要对数以亿记的财产进行管理,生病的人需要到医院进行检查,这些都离不开软件的支持。软件系统关系着人民的信息安全、财产安全乃至生命安全。确保软件的行为在任何时候都与规范相一致,已经成为软件开发者及使用者关注的重要问题。

如果一个软件系统的行为符合规定的行为规范,则称这个系统具有正确性。为了验证系统的正确性,验证技术得到了发展。目前,主要的验证技术有3种:定理证明[1]、模型检测[2]以及测试[3]。定理证明和模型检测都需要对软件进行建模。定理证明使用数学推理的方式证明系统的正确性,而模型检测使用自动验证技术对有限状态系统进行验证,定理证明和模型检测常用来验证系统设计的正确性,不足以保证系统实现的正确性。软件测试通常采用给予系统一定的输入,对比系统的输出与标准输出是否一致的方法,来检查软件的正确性。软件测试可以用来发现软件的漏洞,但有限的测试用例难以覆盖软件运行的所有情形。

运行时验证对运行的软件系统进行实时监控,一旦系统的行为违反某些规则,就立即给予提醒或作出反应。与模型检测、定理证明类似,运行时验证同样采用形式化的方式对系统行为规范进行描述,例如LTL(Linear Temporal Logic),CFG(Context Free Grammars),FSM(Finite State Machines)等。另一方面,运行时验证是针对程序或者程序产生的踪迹进行分析的,其复杂度比模型检测、定理证明要低;与测试方法相比,运行时验证引入了形式化方法,在对属性的描述以及对程序的监控方面更具有灵活性和可扩展性[4]。

由于运行时验证技术的这些特点,它已经被应用到各个领域,如Java程序的运行时验证[5-9]、硬件运行时行为监测[10]、恶意攻击监测[11,12]、网络协议的验证[13,14]、列车运行控制系统的控制[15]等。

本文的目的是介绍运行时监控的研究进展及发展趋势。

本文受973项目(2010CB328003),国家自然科学基金项目(60903030,61272001,91218302),国家科技支撑计划(SQ2012BAJY4052),清华大学自主科研计划、北京市属高等学校高层次人才引进与培养计划项目(YETP0167)资助。

张 硕(1990—),男,硕士生,主要研究方向为运行时验证,E-mail:yeluowusheng@gmail.com 贺 飞(1980—),男,博士,副教授,CCF会员,主要研究方向为形式化验证理论及其在嵌入式系统、软件系统中的应用。

本文第1节给出运行时验证的概念、原理及分类;第2节对运行时验证领域的现有的解决方案进行介绍;第3节深入分析了运行时验证领域里的研究热点,介绍了最新的研究进展;第4节探讨了运行时验证目前所面临的主要挑战及研究展望;最后总结全文。

2 运行时验证的原理及分类

运行时验证是一种检测系统的运行情况,然后对系统是否符合给定的属性或规范进行判定的验证技术。与运行时验证相关的开发及应用的过程也是研究的内容[4]。

对一个系统进行运行时验证,需要将被验证的系统置于监控下。对被测系统进行检测的模块被称为监控器。监控器接收系统的运行踪迹,并根据给定的属性,对系统行为是否满足这些属性做出判定。

如图1所示,典型的运行时验证系统包括两部分:被监控的系统及监控器。系统的运行可以用一个无限长的事件踪迹来表示,其中每个事件是一次变量的赋值、变量的输出或者函数的调用等。监控器接收系统产生的事件踪迹,并对踪迹是否满足属性进行判定,判定结果有助于被检测程序的修正。例如,假设验证的属性为"关闭文件后,不应再对文件进行写操作",该属性可以使用LTL来表示:

$$[\,]\ close \rightarrow \neg write\ \mathrm{U}\ open$$

其中,open,close和write事件分别对应于程序中open()函数,close()函数和write()函数的调用。监控器实时接收被检测程序发送的事件踪迹,如果监控器在接收到close事件后、下一次收到open事件之前收到了write事件,表明被监控程序的行为违反了该属性。

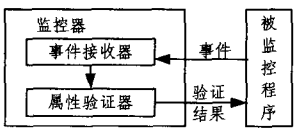

图1 运行时验证系统的典型设计

本质上,模型检测解决的是语言包含问题,而运行时验证解决的则是字符串包含的问题[16],相比而言,其复杂度要小的多。

根据不同的划分标准,运行时验证技术有不同的分类[17]。根据监控器与被检测系统是否同时运行,运行时验证技术分为在线验证和离线验证:在线验证是指监控器与系统同时运行,实时地对系统进行监控;而离线验证,则是对系统的运行记录进行离线的分析。

3 运行时验证的现有解决方案

运行时验证的关键是如何在不影响原系统功能的前提下,收集系统的信息,验证系统的行为。对于软件系统的运行时验证,目前已有一些解决方案。

Jass[18]是对Java程序进行运行时验证的工具,源于契约设计[19]的思想,使用注释及代码生成技术进行运行时验证。Jass使用的方法很直观,但对程序的源代码有很强的依赖性。

JavaMOP[20]是一个针对Java的基于监控的编程(Monitoring-oriented Programming,MOP)的软件开发和分析环境,继承了MOP[5]的架构,它将软件需要满足的规则以及程序的实现结合在一起。JavaMOP最大的特点是支持多种逻辑,例如CFG、LTL、FSM等,使用这些逻辑可以方便地对属性规范进行描述。JavaMOP属性规范的描述采用mop语法,生成的监控器代码是AspectJ源文件,采用abc编译器[21]将AspectJ源文件与Java文件进行混合编译即可对程序进行运行时验证。使用JavaMOP,用户还可以实现特定的接口,定义自己的属性描述语言,具有可插拔性和灵活性。

MaC(Monitoring and Checking)体系为程序的正确运行提供了保证,可以对目标程序进行运行时监控[22]。Java-MaC是MaC体系的一个原型实现,可以应用于Java程序的运行时监控。Java-MaC最大的特点是较低层次的、与实现相关的程序行为的监控和较高层次的、与编程语言无关的、使用形式化描述的规范的分离。该设计使JavaMaC具有很强的灵活性和可扩展性。

Jass和JavaMOP都使用了切面编程(Aspect-oriented Programming,AOP)[23]技术及代码生成技术,而Java-MaC采用了更为精细的字节码插入技术。这3种解决方案涵盖了目前运行时验证领域里的常用技术。

除这3种工具外,还有很多其他的研究成果,列举如下:Java PathExplorer[24],LogScope[25],TraceMatches[26],TraceContract[27],EAGLE[28],RulerR[29],LOLA[30],Larva[31],Hawk[32],J-LO[33],Temporal Rover[34],Conspec[35],MOPBox[36]。这些成果的原理与上述的3种方案是类似的,其中加下划线的工具与本节介绍的3个工具都可以在其网站上获得源代码或者可执行文件。

4 运行时验证的研究现状

4.1 属性的表达

与模型检测类似,运行时验证中的正确性属性也常常使用某种逻辑语言进行表述。一般情况下,运行时验证只针对安全性属性[37]。

软件系统中很多需要监控的属性都属于离散时间系统的正则属性,这种属性经常使用LTL[38]进行描述。在标准的LTL语义中,每个变量的取值有两个:T(真)和F(假)。LTL3扩展了LTL的标准语义,增添了一个表示"不确定"的取值"?"。在一些信息不完善的情况下,使用LTL3可以使表达的意思更加明确[39,40]。此外,还有一些其他针对LTL的扩展。MITL(Metric Interval Temporal Logic)可以对实时属性进行描述[41]。RV-LTL的语义为四值语义,会对处于不确定状态的LTL属性做出可能为真或可能为假的推测[42]。

除了LTL外,近年来也出现了许多采用其他逻辑语言描述监控属性的研究成果。J-LO对LTL增加了对参数化命题的支持[33],TraceMatches使用正则表达式来描述属性[26],LARVA可以对使用动态交互自动机描述的属性生成监控器[31],LOLA基于μ演算进行运行时监控[30],Eagle[28]和RulerR[29]使用基于规则的逻辑进行属性描述,具有很强的表达能力。

属性的表达方式对监控器的生成有很大的影响,这些形式化方法的提出使运行时验证有了更多的选择。由于运行时验证的应用领域不断扩展,参数化的、数据绑定的、实时的、使用方便的形式化语言将成为主流的监控属性描述语言。

4.2 验证算法的研究

运行时验证的过程实际上是判断一个有限串是否属于一个给定的语言的过程。因此,很多运行时验证的算法都采用自动机或者公式重写的方法来实现。

MOP[5]支持基于自动机的验证算法。在监控器生成的初期，MOP将所有的非自动机的逻辑描述转化为自动机，然后生成一个监控器抽象表示，之后再生成监控器代码。LARVA采用时间自动机描述属性，并生成AspectJ监控器[31]。Java-MaC与TraceMatches也采用了类似的方法[22,26]。

基于公式重写的验证算法被面向规则的运行时验证技术[28,29]所采用，它们一般使用公式重写的方法对有限踪迹是否满足给定的属性进行判定。LOLA，Java PathExplorer在对LTL属性进行监控时也采用了这样的方法[24,30]。

4.3 运行时验证的开销控制

运行时验证的方法需要对原程序的行为进行监控，并对监测到的行为进行检查，当检查到冲突时，还需要做出相应的反应。运行时监控带来的开销有时是巨大的，因而很多运行时监控都只能采用离线分析日志的方式进行。为了实现在线的运行时验证，减少运行时开销成为最重要的研究问题。

减少需要监控的属性，可以有效地减少运行时验证的开销。在对一个产品线的属性进行运行时监控时，也可以使用静态分析的方法对组合之后的产品特性进行检查，删减掉不会发生冲突的属性，从而达到减少开销的目的[43]。Clara工具通过减少需要检测的事件来减少运行时开销[44]，可以对使用JavaMOP、TraceMatches、J-LO等工具生成的AspectJ监控器进行优化。减少开销的一个有效方法是控制事件采集的频率，在尽可能少的采样频率下获得足够所需要的事件，就可以控制运行时验证的开销。

4.4 监控数据的不完整性

由于系统设计之初考虑不周、软件错误，导致系统的监控数据常常出现不完整或不一致的情况。此外，为了减少系统的负担，常常需要对监控数据进行采样，这也会造成数据的不完整。为解决这一问题，近年来基于不完整数据的离线运行时验证吸引了很多研究者的关注。

在遇到数据不完整或不一致的情况下，可以采用多值语义进行判定[45,46]。多值语义虽不能完全解决数据不完整或不一致的现象，但却可以使验证的结果变得更精确。

在使用采样方法记录系统行为之后，得到的事件踪迹是间断的，因而无法确定地知道某个属性是否被满足。文献[47]采用HMM(Hidden Markov Model)模型，对采样区段之间的时间段进行事件猜测并填充，并使用状态估计技术估计属性被满足的概率。

4.5 时间相关属性的验证

实时属性是指与系统运行时间相关的属性。实时属性在很多嵌入式系统中都有重要的应用。对实时属性进行监控要比对其他类型的属性进行监控难得多。对系统进行运行时监控需要在系统中插入若干指令，以获取系统的运行信息。然而插入的指令本身也会引入时间和内存的开销，这种额外的时间开销有可能导致原本满足时间约束的系统行为变为不满足[48-50]。

对于一些实时系统的实时属性验证，在获取时间戳后，可以采用Jass、JavaMOP等工具进行处理，这与其他属性的验证并无区别。但对于实时要求很高的实时系统，这种方法是不准确的，因为代码的插入会改变系统的行为。Fischmeister S等人提出了一种基于对实时系统进行代码插入的思想：在插入代码之后，程序最长执行路径的运行时间不超过未产生冲突时的最长时间。以这种思想为基础，他们提出了基于静态分析、代码优化的代码插入方法[48-50]。

5 主要挑战及研究展望

目前，运行时验证技术正处于快速发展的时期，已经出现了多种工具，如JavaMOP、J-LO、TraceMatches等，在多个领域都有所应用，如Java程序的运行时验证[5-9]、硬件运行时监控[10]、恶意攻击及病毒检测[11,12]、网络协议的验证[13-14]、列车运行控制系统的控制[15]等。

运行时验证技术的发展趋势是在线的、自适应的、可自我学习和自我修复的系统。Bartocci E等人在这个方向进行了初步研究，提出了自适应地、动态地对运行时开销进行控制的系统[51]。此外，对被监控系统的系统干预和系统恢复也成为人们研究的热点[52-54]。就运行的系统而言，仅仅进行属性验证是不够的，还要根据验证的结果对系统的运行进行调整；如果系统崩溃，需要在系统崩溃之后进行系统的恢复。现有的运行时验证的工具在系统恢复、系统干预方面都还不健全。

实现在线监控面临的主要挑战是如何控制运行时的开销。将所有事件都记录下来会产生巨大的系统开销[48]，而使用采样的方法又面临数据不完整的问题[47]，如何根据待验证的属性在这两个方向上进行折中是控制监控开销的关键。此外，为实现在线的运行时验证，对验证算法的效率也提出了很高的要求。

高效的验证算法、系统干预和恢复的机制、实时属性的验证将成为未来运行时验证领域研究的重点，只有将这些技术结合起来，运行时验证才能够发挥它的巨大价值。

结束语 运行时验证技术已成为一种重要的验证技术，已经被应用到多个领域。本文对运行时验证技术的基本原理进行了阐述，对比分析了目前该领域里常见的解决方案，对该领域里的研究热点进行了深入的阐述，并对该领域的研究进行了预测。运行时监控技术的原理及方法非常适合安全性监测，随着该技术的发展，必将在航空航天、医疗器材等领域发挥其应有的价值。

参 考 文 献

[1] Bertot Y, Castéran P. Interactive theorem proving and program development: Coq' Art: the calculus of inductive constructions [M]. Springer, 2004

[2] Clarke E M, Grumberg O, Peled D A. Model checking[M]. MIT press, 1999

[3] Myers G J, Sandler C, Badgett T. The art of software testing [M]. John Wiley & Sons, 2011

[4] Leucker M, Schallhart C. A brief account of runtime verification [J]. Journal of Logic and Algebraic Programming, 2009, 78(5): 293-303

[5] Meredith P O N, Jin D, Griffith D, et al. An overview of the MOP runtime verification framework[J]. International Journal on Software Tools for Technology Transfer, 2012, 14(3): 249-289

[6] Bodden E. MOPBox: a library approach to runtime verification [C]// Runtime Verification. Springer Berlin Heidelberg, 2012: 365-369

[7] Havelund K, Roşu G. Monitoring java programs with java pathexplorer[J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 200-217

[8] Kim M, Kannan S, Lee I, et al. Java-MaC: a run-time assurance

tool for Java programs[J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 218-235

[9] 张献, 董威, 齐治昌. 基于 AOP 的运行时验证中的冲突检测[J]. 软件学报, 2011, 22(6): 1224-1235

[10] Malik S. Runtime verification: a computer architecture perspective[C] // Runtime Verification. Springer Berlin Heidelberg, 2012: 49-62

[11] Beaucamps P, Gnaedig I, Marion J Y. Behavior abstraction in malware analysis [C] // Runtime Verification. Springer Berlin Heidelberg, 2010: 168-182

[12] Milea N A, Khoo S C, Lo D, et al. Nort: Runtime anomaly-based monitoring of malicious behavior for windows [C] // Runtime Verification. Springer Berlin Heidelberg, 2012: 115-130

[13] Bucur D. Temporal Monitors for TinyOS[C] // Runtime Verification. Springer Berlin Heidelberg, 2013: 96-109

[14] Basu A, Bensalem S, Bozga M, et al. Verification of an AFDX Infrastructure using Simulations and Probabilities[C] // Runtime Verification. Springer Berlin Heidelberg, 2010: 330-344

[15] 张林, 唐涛, 徐田华, 等. 运行时验证及其在列车运行控制中的应用[J]. 铁道学报, 2011, 33(12): 65-71

[16] Sistla A P, Clarke E M. The complexity of propositional linear temporal logics[J]. Journal of the ACM(JACM), 1985, 32(3): 733-749

[17] Leucker M. Teaching runtime verification[C] // Runtime Verification. Springer Berlin Heidelberg, 2012: 34-48

[18] Bartetzko D, Fischer C, Möller M, et al. Jass—Java with assertions [J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 103-117

[19] Leavens G T, Cheon Y. Design by Contract with JML[OL]. http://www.bowdoin.edu/~allen/courses/cs260/readings/leavens.pdf

[20] Chen F, Roşu G. Java-MOP: A monitoring oriented programming environment for Java[C] // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2005: 546-550

[21] Avgustinov P, Christensen A S, Hendren L, et al. abc: An extensible AspectJ compiler[C] // Transactions on Aspect-Oriented Software Development I. Springer Berlin Heidelberg, 2006: 293-334

[22] Kim M, Kannan S, Lee I, et al. Java-MaC: a run-time assurance tool for Java programs[J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 218-235

[23] Irwin J, Kickzales G, Lamping J, et al. Aspect-oriented programming[C] // Proceedings of ECOOP. IEEE, Finland, 1997: 220-242

[24] Havelund K, Roşu G. Monitoring java programs with java pathexplorer[J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 200-217

[25] Barringer H, Groce A, Havelund K, et al. An entry point for formal methods: Specification and analysis of event logs[J]. arXiv preprint arXiv: 1003. 1682, 2010

[26] Avgustinov P, Tibble J, Bodden E, et al. Efficient trace monitoring[C] // Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. ACM, 2006: 685-686

[27] Barringer H, Havelund K. TraceContract: A Scala DSL for trace analysis[M]. Springer Berlin Heidelberg, 2011

[28] Barringer H, Goldberg A, Havelund K, et al. Rule-based runtime verification[C] // Verification, Model Checking, and Interpretation. Springer Berlin Heidelberg, 2004: 44-57

[29] Barringer H, Rydeheard D, Havelund K. Rule systems for runtime monitoring: from Eagle to RuleR[J]. Journal of Logic and Computation, 2010, 20(3): 675-706

[30] d'Angelo B, Sankaranarayanan S, Sanchez C, et al. Lola: Runtime monitoring of synchronous systems[C] // 12th International Symposium on Temporal Representation and Reasoning, 2005 (TIME 2005). IEEE, 2005: 166-174

[31] Colombo C, Pace G J, Schneider G. Larva-safer monitoring of real-time java programs(tool paper)[C] // 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods. IEEE, 2009: 33-37

[32] d'Amorim M, Havelund K. Event-based runtime verification of Java programs [J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(4): 1-7

[33] Stolz V, Bodden E. Temporal assertions using AspectJ[J]. Electronic Notes in Theoretical Computer Science, 2006, 144(4): 109-124

[34] Drusinsky D. The temporal rover and the ATG rover[M] // SPIN Model Checking and Software Verification. Springer Berlin Heidelberg, 2000: 323-330

[35] Aktug I, Naliuka K. ConSpec-a formal language for policy specification[J]. Electronic Notes in Theoretical Computer Science, 2008, 197(1): 45-58

[36] Bodden E. MOPBox: a library approach to runtime verification [C] // Runtime Verification. Springer Berlin Heidelberg, 2012: 365-369

[37] Havelund K, Roşu G. Synthesizing monitors for safety properties [M] // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2002: 342-356

[38] Pnueli A. The temporal logic of programs [C] // 18th Annual Symposium on Foundations of Computer Science, 1977. IEEE, 1977: 46-57

[39] Bauer A, Leucker M, Schallhart C. Monitoring of real-time properties[C] // FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science. Springer Berlin Heidelberg, 2006: 260-272

[40] Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL[J]. ACM Transactions on Software Engineering and Methodology(TOSEM), 2011, 20(4): 14

[41] Maler O, Nickovic D, Pnueli A. From MITL to timed automata [C] // Formal Modeling and Analysis of Timed Systems. Springer Berlin Heidelberg, 2006: 274-289

[42] Bauer A, Leucker M, Schallhart C. The good, the bad, and the ugly, but how ugly is ugly? [C] // Runtime Verification. Springer Berlin Heidelberg, 2007: 126-138

[43] Kim C H P, Bodden E, Batory D, et al. Reducing configurations to monitor in a software product line[C] // Runtime Verification. Springer Berlin Heidelberg, 2010: 285-299

[44] Bodden E, Lam P. Clara: Partially Evaluating Runtime Monitors at Compile Time [C] // Runtime Verification. Springer Berlin Heidelberg, 2010: 74-88

[45] Ehlers R, Finkbeiner B. Monitoring realizability[C] // Runtime Verification. Springer Berlin Heidelberg, 2012: 427-441

[46] Basin D, Klaedtke F, Marinovic S, et al. Monitoring compliance policies over incomplete and disagreeing logs [C] // Runtime

Verification. Springer Berlin Heidelberg, 2013: 151-167

[47] Stoller S D, Bartocci E, Seyster J, et al. Runtime verification with state estimation[C]// Runtime Verification. Springer Berlin Heidelberg, 2012: 193-207

[48] Bonakdarpour B, Fischmeister S. Runtime monitoring of time-sensitive systems[C]// Runtime Verification. Springer Berlin Heidelberg, 2012: 19-33

[49] Fischmeister S, Lam P. Time-aware instrumentation of embedded software[J]. IEEE Transactions on Industrial Informatics, 2010, 6(4): 652-663

[50] Fischmeister S, Lam P. On time-aware instrumentation of programs[C]// 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009 (RTAS 2009). IEEE, 2009: 305-314

[51] Bartocci E, Grosu R, Karmarkar A, et al. Adaptive runtime verification[C]// Runtime Verification. Springer Berlin Heidelberg, 2013: 168-182

[52] Colombo C, Pace G J, Abela P. Compensation-aware runtime monitoring[C]// Runtime Verification. Springer Berlin Heidelberg, 2010: 214-228

[53] Demsky B, Zhou J, Montaz W. Recovery tasks: an automated approach to failure recovery[C]// Runtime Verification. Springer Berlin Heidelberg, 2010: 229-244

[54] Colombo C, Pace G J. Fast-Forward Runtime Monitoring—An Industrial Case Study[C]// Runtime Verification. Springer Berlin Heidelberg, 2013: 214-228

(上接第 339 页)

纲对数据的影响，先对样本数据进行了归一化处理，并根据 3.4 节给出的方法计算了阈值 δ，实验运行结果如表 2 所列。

表 2 约简后属性集合

数据集	δ	FHARA 算法	B_FHARA 算法
Iris	0.0547	0,2	0,2
Wine	0.0310	1,7,10	1,7,10
Ionosphere	0.0275	4,12,21	4,12,21
Wdbc	0.0340	2,24,26	2,24,26
pima-indians-diabetes	0.026	0,1,5,7	0,1,5,7

由表 2 可知，FHARA 算法与 B_FHARA 算法在阈值取值相同的情况下，运行结果是一致的。所以结果的属性依赖度也是一致的。但是在运行时间上，B_FHARA 算法则要快很多，如表 3 所列。

表 3 运行时间(单位：s)

数据集	FHARA 算法	B_FHARA 算法
Iris	0.016	0.011
Wine	0.061	0.053
Ionosphere	0.678	0.526
Wdbc	1.541	0.921
pima-indians-diabetes	0.880	0.503

折线图如图 3 所示。

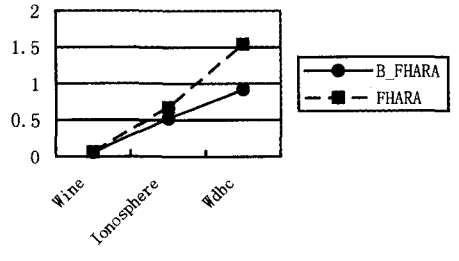

图 3 两种算法在不同数据集上运行时间的折线图

通过对比运行时间，可以发现随着样本属性的增多及样本数量的增长，B_FHARA 算法的增长速度相对较平缓。

为去除程序差异造成的计算时间差别，实验统计了样本的比较次数，如表 4 所列。

表 4 比较次数

数据集	FHARA 算法	B_FHARA 算法
Iris	15968	11221
Wine	58217	22849
Ionosphere	477927	150994
Wdbc	1106403	540604
pima-indians-diabetes	692879	131517

柱状图如图 4 所示。

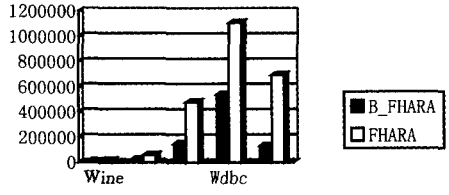

图 4 两种算法在不同数据集上比较次数的比较

通过比较次数的对比，我们发现 B_FHARA 算法能明显地减少约简过程中的比较次数。

结束语 本文研究了当前邻域粗糙集模型中邻域计算的研究现状，针对当前算法的不足，根据样本空间的分布，提出了块集的概念及其与样本的映射算法，并进一步证明了每个样本的邻域只存在于其相邻的块集中，在此基础上，提出了一种基于块集的属性快速约简算法。

经过多个 UCI 标准数据集的实验验证，在运算结果相同的情况下，该算法能有效地减少比较次数，提高计算效率。

参 考 文 献

[1] Yong L, Wenliang H, Yunliang J, et al. Quick attribute reduct algorithm for neighborhood rough set model[J]. Information Sciences, 2014, 271: 65-81

[2] Pawlak Z. Rough Sets—Theoretical Aspects of Reasoning about Data[M]. Dordrecht: Kluwer Academic, 1991

[3] 曾宇. 高效能计算机若干关键技术的研究与实现[D]. 北京：中国科学院，2009

[4] 胡清华，于达仁，谢宗霞. 基于邻域粒化和粗糙逼近的数值属性约简 [J]. 软件学报，2008，19(3)：640-649

[5] 胡清华，赵辉，于达仁. 基于粗糙集的符号与数值属性的快速约简算法[C]// 第七届中国 Rough 集与软计算学术会议. 太原，2007：640-649

[6] 刘遵仁，吴耿锋. 基于邻域粗糙模型的高维数据集快速约简算法[J]. 计算机科学，2012，39(10)：268-271

[7] Hu Q, Yu D, Liu J, et al. Neighborhood rough set based heterogeneous feature subset selection[J]. Inform. Sci., 2008, 178(18): 3577-3594

[8] 张冬雯，王鹏，仇计清. 基于邻域粗糙集和蚁群优化的属性约简算法[J]. 河北科技大学学报，2011，32(5)：403-408

[9] 王丽娟，吴陈，杨习贝，等. 邻域系统粗糙集和覆盖粗糙集[J]. 计算机科学，2013，40(1)：221-224