基于形式概念分析的依赖簇检测方法研究

尚 颖 程 克 李 征
(北京化工大学信息科学与技术学院 北京 100029)

摘 要 依赖簇是相互依赖的程序组件的最大集合,大尺寸依赖簇已被证实在程序中普遍存在。依赖簇中任意一点产生变动都会引起其他组件的连锁反应,进而对整个系统造成潜在的影响,这将会阻碍软件理解、测试、维护等方面的工作。检测出依赖簇是消除不良影响的前提,目前通过单调切片尺寸图近似检测依赖簇的方法的准确度较低,会出现漏报和误报。提出了一种基于形式概念分析的依赖簇检测方法,通过概念包含度选取的大型概念来检测大尺寸依赖簇,并进一步提出轻量化策略以有针对性地选取大型概念,降低计算开销。在12个不同规模和领域的开源程序上,将所提方法与单调切片尺寸图法进行对比实验,结果表明所提方法及其轻量化策略能够有效地检测大尺寸依赖簇,可以提高依赖簇检测的准确度和效率。

关键词 形式概念分析,概念格,依赖簇
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.031

Research on FCA Based Dependence Cluster Detection

SHANG Ying CHENG Ke LI Zheng
(College of Information Science & Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract Dependence cluster is a maximal set of program components that all depend upon each other. The current view is that large dependence cluster is extremely universal, which widely exists in all kinds of program source code. The existence of large dependence cluster may lead to the significant ripple effect, i.e., a change to a certain point of the cluster will cause potential adverse effects on the rest of the cluster, which might affect the whole system. It has a negative impact on many different software engineering activities, including comprehension, testing and maintenance. Dependence cluster detection is a prerequisite for solving adverse effects caused by large dependence cluster. Previous researchers have proposed monotonic slice-size graph (MSG) based method to detect the same slice size of dependence cluster. However, the proposed method is of conservative approximation, which will cause false positives and false negatives. This paper proposed a technique to detect dependence clusters using formal concept analysis (FCA), in which concept inclusiveness is defined to select large concepts. Furthermore, a light-weight computing strategy for FCA was proposed to compute large concept to decrease the computation cost significantly. Based on 12 open source subjects, the empirical comparison showes that the proposed technique can increase the detection accuracy of large dependence clusters with higher efficiency.

Keywords Formal concept analysis, Concept lattice, Dependence cluster

随着软件规模的增大,软件维护和修改的难度也越来越大。原因之一是软件的程序组件之间广泛存在着不可避免的相互依赖关系,大量的相互依赖关系将组成依赖簇,依赖关系的传递性使得依赖簇中任意细小的修改都可能引起程序组件的"连锁反应"。因此分析包含依赖簇尤其是大尺寸依赖簇的软件已成为软件理解、测试和维护等领域的重大挑战。通常的解决方法是通过重构来降低组件之间的依赖关系,所以检测程序中依赖簇的组成是需首要解决的问题。

David Binkley 和 Mark Harman 提出一种通过单调切片尺寸图(Monotone Slice-size Graph, MSG)检测依赖簇的方法[1],即具有相同切片尺寸的组件属于同一依赖簇。这是一种近似估算的方法,因为两个具有相同尺寸的切片内容可能会有所不同,同时可发现具有不同切片尺寸的程序组件可能是相互依赖的(见表1)。MSG方法虽具有较高的效率,但检测准确度较低,会出现误报和漏报。

形式概念分析(FCA)是一种精确的细粒度数学分析方法,目前在软件工程、数据挖掘、知识工程、信息检索、本体研究等领域有着广泛应用。本文针对MSG检测依赖簇时在准

到稿日期:2015-11-30 返修日期:2016-03-02 本文受国家自然科学基金(61170082,61472025,61672085),教育部新世纪优秀人才计划(NCET-12-0757)资助。

尚 颖(1976—),女,博士,讲师,主要研究领域为常识推理、优化算法、软件测试;程 克(1989—),男,硕士生,主要研究领域为源代码分析;李 征(1974—),男,博士,教授,主要研究领域为基于搜索的软件工程和软件测试、程序依赖性分析及程序切片技术、基于状态模型的依赖性分析和切片技术等,E-mail:lizheng@mail.buct.edu.cn。

确度方面的不足，探讨并研究了如何将从源代码中获取的依赖关系抽象和整理成形式背景，应用 FCA 方法生成概念格，并在概念格中定义大型概念，通过大型概念来检测依赖簇。主要研究内容如下：

1)提出了基于 FCA 的依赖簇检测方法，包括依赖关系的抽象整理、概念的生成、大型概念的定义与选取以及通过大型概念进行依赖簇分析检测。

2)提出了轻量化检测策略，设计了剪枝的 Fast CloseBy-One(FCBO)算法，排除不满足条件的概念的生成，节省了依赖簇检测的计算开销。

3)通过实验验证了本文方法及其轻量化检测策略的有效性。

本文第 1 节介绍依赖簇概念、MSG 检测方法及本文方法应用的 FCA 技术；第 2 节介绍基于 FCA 的依赖簇检测方法；第 3 节进行实验分析；最后总结全文。

1　相关背景理论与技术

1.1　依赖簇及 MSG 检测方法

依赖分析在很多源代码分析技术中扮演着重要的角色，近几十年，一大批国内外科研工作者致力于这方面的研究，例如程序理解、影响分析等。David Binkley 和 Mark Harman 在 2005 年将相互依赖的程序组件的最大集合正式定义为依赖簇（Dependence Cluster，DC）[1]。Acharya 和 Robinson、Beszédes 和 Szegedi 等通过大量的实例研究，发现依赖簇广泛存在于各种 C 基准程序中[2-4]。Beszédes 和 Szegedi 等发现大尺寸依赖簇也存在于 C++和 JAVA 系统中。Hajnal A 于 2012 年发现 Legacy Cobol Systems 中同样存在大尺寸依赖簇[5]。当程序中包含大尺寸依赖簇时，程序的修改将会产生明显的连锁效应，即依赖簇中某一点的变动都会对簇中其他部分造成潜在的影响。大尺寸依赖簇是一种依赖反模式[6]，对软件维护、软件测试、软件理解和软件重构等过程均有不良影响[7-13]。

定义 1(基于程序切片的相互依赖集/依赖簇，Slice-based MDS/Cluster[10])　一个基于程序切片的相互依赖集是一系列元素的集合 S，且 $\forall x,y\in S: x\in BSlice(y)$。那么，一个基于程序切片的相互依赖簇是一个基于程序切片的相互依赖集，且不包含其他基于程序切片的相互依赖集。

MSG 检测方法通过切片技术，将依赖簇定义为具有相同切片的最大集合。该方法通过计算所有被测程序系统依赖图(System Dependence Graph，SDG)中程序节点的切片，检查其切片的尺寸是否相同，来近似判断程序组件组成的集合是否为依赖簇。依赖簇在 MSG 图中表现为一个陡峭的悬崖面，而后伴随着长且稳定的平面。通过 MSG 中陡峭的悬崖面顶端的平面长度来判断依赖簇的大小，通过对比陡峭崖面与周围曲线的陡峭程度来判断依赖描述中的连续性。

1.2　形式概念分析

德国达姆施塔特科技大学的数学家 Rudolf Wille 教授于 1982 年基于 Brikhoff 的格理论研究提出 FCA 这种经典数学理论，也称为概念格理论[14]。

定义 2　三元组 $K:=(G,M,I)$ 组成的二维交叉表称为形式背景(Formal Context)。其中，G 是由形式对象组成的对象集，M 是由形式属性组成的属性集。G 和 M 均是非空有限集合。I 表示对象集 G 与属性集 M 之间的二元关系，$I\subseteq G\times M$。某一形式对象 $g\in G$ 具有某一形式属性 $j\in M$，可以表示为 $g\in m$ 或者 $(g,m)\in I$。在形式背景中，用 1 表示对象 g 具有 m 属性，即 $(g,m)\in I$；0 表示对象 g 不具有 m 属性，即 $(g,m)\notin I$。

定义 3　对于形式背景 $K:=(G,M,K)$，对象集 G 的任一子集 A 与属性集 M 的任一子集 B 存在对偶算子 $f(A)$，$g(B)$，其中，$f(A)$ 称为 A 中所有对象的共同属性的集合，$g(B)$ 称为 B 中所有属性的共同对象的集合。如果存在二元组 (A,B)，根据形式背景 $K=(G,M,K)$ 使得 $A=g(B)$ 且 $B=f(A)$，那么二元组 (A,B) 则被称为形式概念(Formal Concept)，简称概念(Concept)，其中 A,B 分别是概念的外延和内涵。

$$f(A)=\{m\in M|\ \forall g\in A,(g,m)\in I\} \tag{1}$$

$$g(B)=\{g\in G|\ \forall m\in B,(g,m)\in I\} \tag{2}$$

定义 4　形式背景 $K=(G,M,I)$ 生成两个概念 $C_1=(A_1,B_1)$ 和 $C_2=(A_2,B_2)$。若两个概念的外延满足 $A_1\in A_2$（等价于概念内涵满足 $B_2\in B_1$），则称 (A_1,B_1) 和 (A_2,B_2) 之间存在偏序关系，C_1 是 C_2 的子概念，C_2 是 C_1 的父概念，运用符号“$\leqslant$”来表示概念间的偏序关系，将这种偏序关系标记为 $C_1\leqslant C_2$。

定义 5　形式背景 $K=(G,M,K)$ 中所产生的偏序集 $CS(K)$ 是一个完全格，称之为形式背景 K 的概念格，记为 $L(K)$。

FCA 方法的输入为形式背景，包含对象集、属性集及它们之间对应的二元关系；输出为概念格，即概念及其偏序关系。其中计算概念的过程即为聚类的过程，能够很好地发掘对象和属性之间的关系，可以得到哪些对象拥有哪些共同的属性；概念格的构造拥有成熟算法，且概念格中的每个概念都是“最大”的。本文基于上述特征，应用 FCA 方法研究依赖的簇的检测方法。

2　基于形式概念分析的依赖簇检测方法

依赖簇是相互依赖的程序节点组成的最大集合，相互依赖的节点在彼此的后向切片中，依赖簇检测的作用是检测哪些节点共同依赖于哪些节点。大型依赖簇检测的本质是在系统依赖图中寻找拥有尽可能大的影响集且由尽可能多的节点组成的集合，这与 FCA 中的概念相一致。本节主要介绍基于 FCA 的依赖簇检测方法的流程与关键步骤。

2.1　方法概述

通过定量理论来解决定性问题是一种常用的数学方法。基于形式概念分析的依赖簇检测方法通过计算形式背景中概念的对象数和属性数所占的比例来反映概念的对象和属性的量化关系，进而选取大型概念，再利用大型概念来检测大尺寸依赖簇。

方框图如图 1 所示。

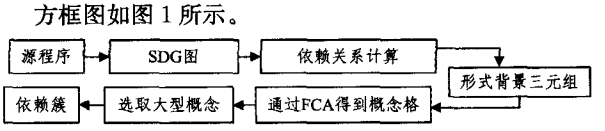

图 1　基于形式概念分析的依赖簇检测方法框架

如图 1 所示，该方法以被测程序的源代码为输入，生成被测程序的系统依赖图 SDG，从中选取反映源代码的程序节点，计算所有被选节点的后向切片（包括程序中所有与该节点具有依赖关系的其他节点）。其中将切片准则作为对象，切片中的节点作为属性，切片（即依赖关系）作为对象和属性之间的二元关系（1 表示某节点在某切片准则的后向切片中，0 表示不在其后向切片中）。将此三元组抽象为形式背景，应用 FCA 生成概念格。在此基础上提出概念包含度的度量准则，应用概念包含度在概念格中选取大型概念以检测依赖簇。其中，概念对象包含度和属性包含度的定义如下。

定义 6 对于形式背景 $K:=(G,M,I)$，$L(K)$ 为通过 K 生成的所有概念，$C(A,B)$ 是 $L(K)$ 中任一概念，那么概念 $C(A,B)$ 在形式背景 $L(K)$ 上的对象包含度 GI 和属性包含度 MI 为：

$$GI=\frac{|A|}{|G|},GI\in[0,1] \tag{3}$$

$$MI=\frac{|B|}{|M|},MI\in[0,1] \tag{4}$$

定义 7 对于形式背景 $K:=(G,M,I)$，任一概念 $C(A,B)$ 在形式背景 K 上的整体包含度 CI 为：

$$CI=GI\times MI,CI\in[0,1] \tag{5}$$

定义 8 对于由形式背景 $K:=(G,M,I)$ 生成的所有概念 $C_0,C_1,C_2,\cdots,C_n\in L(K)$，令容差为 ε，其最大整体包含度 $MaxCI=\mathrm{MAX}(CI(C_0),CI(C_1),CI(C_2),\cdots,CI(C_n))$，那么称具有 $[MaxCI-\varepsilon,MaxCI]$ 整体包含度的概念为大型概念 LC。一般取 $\varepsilon=0.01$。

2.2 依赖簇检测

依赖簇检测的核心目标是检测被测程序中的大尺寸依赖簇，即拥有尽可能大的影响集且由尽可能多的节点组成的依赖簇。本文采用 10%的阈值[1]，将规模超过 10%的依赖簇称为大尺寸依赖簇。以下通过示例来介绍如何利用大型概念检测大尺寸依赖簇。

设有大型概念 LC，对象集 A 为切片准则{1,2,3,4,5}，属性集 B 为拥有的共同影响集{1,2,3,5}，每一个切片准则及其切片如表 1 所列。其中 1,2,3 这 3 个节点在彼此的后向切片中，具有相同的切片{1,2,3,5}，它们是相互依赖的，则它们必然存在于同一依赖簇中。准则 4 的切片与准则 1、准则 2、准则 3 不同，但同样包含{1,2,3,5}，1,2,3 在 4 的后向切片中，但是 4 不在 1,2,3 的后向切片中，因此 4 与 1,2,3 并不是相互依赖的。准则 5 与准则 1、准则 2、准则 3 对应的切片不同，但 1,2,3 在 5 的后向切片中，且 5 也在 1,2,3 的后向切片中，所以也是相互依赖的。可以得出结论：拥有相同影响集但切片尺寸不同的节点仍可能存在相互依赖关系，这些节点的特点是其切片准则节点也在其影响集中。通过以上分析，将大型概念 $LC(A,B)$ 对象集 A 中的节点分为 3 类：1）A_0，这些节点的后向切片即为属性集 B，如上例中的节点 1,2,3，它们是相互依赖的；2）为 A_1，属性集 B 为 A_1 中节点后向切片的子集，但 A_1 与 A_0 中节点存在相互依赖关系，如节点 5；3）为 A_2，属性集 B 为 A_2 中节点后向切片的子集，但 A_2 与 A_0 中节点不存在相互依赖关系。

表 1 切片示例

切片准则	切片内容
1	{1,2,3,5}
2	{1,2,3,5}
3	{1,2,3,5}
4	{1,2,3,4,5}
5	{1,2,3,4,5,6,7}

基于 FCA 的依赖簇检测方法最终得到的依赖簇为：$A_0\cup A_1$。这不但可以得到通过 MSG 方法排除误报节点后的依赖簇 A_0，还补充了拥有相同属性集但切片不同而与节点存在相互依赖关系的漏报节点 A_1。第 3 节将通过实验来对比验证本文方法的有效性。

2.3 轻量化检测策略

应用本文方法检测依赖簇的前提是获取大型概念，故需通过概念格构造算法生成所有的概念，并计算所有概念的整体包含度 CI。FCA 计算的开销较大，会严重影响依赖簇的检测效率。本文在 FCBO 概念格构造算法的基础上设计了基于对象包含度 GI 的剪枝法，不但减少了生成概念的时间开销，还约减了大型概念的搜索域。

概念格是所有概念的偏序关系的累加，所有概念中，对象包含度最大的为顶概念 $Ctop$，它的对象包含度 GI 为 1，内涵可能为空集。由于概念本身具有最大扩展性，任一概念都不可能与它的直接父概念具有相同的对象包含度，因此可以得出以下推论。

推论 1 概念 C 为形式背景 $K:=(G,M,I)$ 概念格中的概念，$Parents(C)$ 为概念 C 的父概念组成的集合，$Children(C)$ 为概念 C 的子概念的集合，如果 $GI(C)\geqslant\lambda$，那么任意 $C_1\in Parents(C),GI(C_1)>\lambda$；如果 $GI(C)\leqslant\lambda$，那么 $C_2\in Children(C),GI(C_2)<\lambda$。

根据推论 1，在概念格的生成过程中，如果某个概念 C 的对象包含度 $GI<\lambda$，则必然存在概念 C 的所有子概念的 $GI<\lambda$，为了得到大型概念，概念 C 及其子节点都应该被剪去。本文称这种方法为基于概念对象包含度的剪枝法。在概念格生成过程中，运用基于概念对象包含度的剪枝法可以使得生成的所有概念的 GI 均大于 λ，从而避免生成和计算不能形成大尺寸依赖簇的概念。剪枝的 FCBO 算法伪码如下。

```
Pruned_FastGenerateFrom(Concept(A,B),m,{Nm|m∈M})
输入：形式背景 K:=(G,M,I)，头概念 Concept_top(A,B)，首个要处理
      的属性 m，属性子集 Nm=∅
输出：所有满足 GI≥λ 的概念
Print Concept (A,B);
// 概念(A,B)已经被处理过，例如存储或输出；
if B=Morm>n then
  return//判断为尾概念或所有超过字典序
endif //最后一位则跳出本次调用；
For i from m to n do//开始处理字典序位于 i 后的所有属性；
  set Pi→Ni;// Pi 是指向 Ni 的指针
  if i∉B and Ni⋂Mi B⋂Mi then
      //提前检测将要生成的概念是否有效；
  setComputeClosure(Concept(A,B),i)→Concept(C,D);
```

```
    //加入属性 i 生成子概念；
  if B∩Mi=D∩Mi then//检测生成的概念是否有效；
  if|A|/|G|≥λ//判断概念是否满足对象包含度条件
    In (Concept(C,D),i+1) to Queue;
      //将新生的概念加入队列；
endif
    else
  set Pi→D;//更新 implied 信息，指针 Pi 指向 D;
    endif
endif
endfor
WhileOut(Concept(C,D),i) from Queue do//递归调用 Pruned_Fast-
                                        GenerateFrom
Pruned_FastGenerateFrom(Concept(C,D),i+1,{Pm|m∈M});
endwhile
return
```

3　实验分析

3.1　实验目的

为了验证基于 FCA 的依赖簇检测方法和轻量化检测策略的有效性，对两个问题进行实验：

1)新方法能否有效检测大尺寸依赖簇，与 MSG 方法相比依赖簇检测结果如何？

2)轻量化检测策略能否提高大尺寸依赖簇的检测效率？

3.2　实验对象与评价指标

选取 Binkley 研究实验中 12 个开源被测程序，以便进行对比。代码的规模为 600LOC～18kLOC，这些程序涉及多个领域的应用，如实用工具、操作系统代码等。被测程序的具体信息如表 2 所列。

表 2　被测程序的基本信息

被测程序	有效语句	节点	边	切片
acct-6.3.2	2384	2140	4349	678
barcode	3975	13424	35919	4822
bc	7342	18791	48806	7538
copia	1112	43975	128116	4686
dc	6352	9694	21438	4934
ed	9046	19791	58470	5688
EPWIC-1	4232	19545	38068	6492
findutils	11843	38033	174162	14445
indent	5424	20776	107446	6222
replace	512	3406	11177	2514
userv-client	1920	11856	82649	2681
which	3618	5247	12015	1163

实验在 PC 上进行，操作系统为 64 位 Windows 7，处理器为 Intel Xeon E3-1230 v2，时钟频率为 3.30GHz，内存为 16GB。采用 Codesurfer 计算系统依赖图(SDG)并对选择节点进行后向切片。基于 C 语言在 DEV Cpp 上开发形式背景抽象整理、形式概念分析、概念格构造算法等。

为了将本文方法与相同切片尺寸法的依赖簇检测结果进行对比，采用以下评价指标对准确度进行分析，其中 DC_1 和 DC_2 分别表示相同切片尺寸法与本文方法得到的依赖簇节点集合。

重合数：若存在节点集 $S_1=\{n_0,n_1,\cdots,n_i,\cdots,n_m\}(m,i$ 均为正整数)，对于集合 S 中的任意 n_i 均有 $n_i\in DC_1\cap DC_2$，那么集合 S_1 称为依赖簇的重合集(Coincidence Sets，CS)，$|S_1|$ 称为两个依赖簇节点的重合数。

相对漏报数(Relative False Negative)：若存在节点集 $S_2=\{n_0,n_1,\cdots,n_i,\cdots,n_m\}(m,i$ 均为正整数)，对于集合 S_2 中任意 n_i 均有 $n_i\in DC_1$ 且 $n_i\notin DC_2$，并且 n_i 与重合集 CS 中的节点都是相互依赖的，那么称 S_2 为 DC_2 相对于 DC_1 的漏报集，$|S_2|$ 为 DC_2 的相对漏报数。

误报数(False Positive)：若存在节点集 $S_3=\{n_0,n_1,\cdots,n_i,\cdots,n_m\}(m,i$ 均为正整数)，对于集合 S 中任意 n_i 均有 $n_i\in DC_1$ 且 $n_i\notin DC_2$，并且 n_i 与重合集 CS 中的节点不存在相互依赖关系，那么称 S_3 为误报集，$|S_3|$ 为 DC_1 的漏报数。检测优化率 *Improvement* 用来衡量准确度的提升；概念约减率 *Reduction* 为约减的概念数与完整概念格概念数的比值。

$$Improvement=\frac{|false\ positive|+|relative\ false\ negative|}{|same\_slice\_size\ dependence\ cluster|}\times 100\% \tag{6}$$

$$Reduction=\frac{|reduce\ concepts|}{|L(K)|}\times 100\% \tag{7}$$

3.3　结果分析

一般称节点数和对应影响集规模均大于 50%的依赖簇为巨大尺寸依赖簇。按规模可将依赖簇分为 3 类：小型 S(<10%)，大型 L(10% ～ 50%)，巨大型 E(>50%)。表 3 列出应用两种方法对 12 个被测程序进行依赖簇检测的结果。结果表明，FCA 检测方法可以有效检测依赖簇，同时能有效避免漏报和误报，提高了检测准确度，平均优化率为 25.74%。

表 3　两种方法的准确度分析

被测程序	MSG 检测		FCA 检测		结果
	相对漏报	误报	误报漏报	优化率/%	
acct-6.3.2	0	25	0	53.19	S
EPWIC-1	42	0	0	8.53	S
barcode	16	0	0	0.57	L
copia	0	83	0	1.97	L
findutils	697	4	0	24.57	L
replace	325	20	0	71.88	L
userv-client	85	0	0	22.08	L
which	121	2	0	28.02	L
bc	1388	0	0	33.57	E
dc	1366	4	0	61.35	E
ed	37	0	0	0.97	E
indent	73	0	0	2.15	E

为了验证本文提出的轻量化检测策略的执行效果以及轻量化策略是否选取了与完整概念格方法相同的大型概念，分别构造了 12 个被测程序的完整概念格和应用轻量化检测策略生成的概念格，并记录两种方法生成的概念数和概念生成时间，以此来计算概念数约减率和时间缩减幅度，进而评价本文提出的轻量化策略的效果。

从表 4 可以看出，在 12 个程序中，除了 epwic 以外，通过轻量化策略可以达到与生成完整概念格相同的效果，选取了相同的大量概念，但是生成概念平均数却从 1865 减少到 592，平均约简率为 69.68%。在概念格构造时间方面，从 3400s 降低到 1781s，效率提升约为 47.65%。由此得出，本文

(下转第 176 页)

Computing, 2005. Santa Fe, 2005: 1499-1504.

[4] ARAFEEN M J, DO H. Test Case Prioritization Using Requirements-Based Clustering[C]// 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013. Luembourg: IEEE Conference Publications, 2013: 312-321.

[5] HONG M, DAN H, Z L Mm, et al. A static approach to prioritizing JUnit test case[J]. IEEE Transactions on Software Engineering, 2012, 38(6): 1258-1275.

[6] KE Z, BO J, W K C. Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study[J]. IEEE Transactions on Software Engineering, 2014, 7(1): 54-67.

[7] ALESSANDRO M, MAHFUZUL I, WASEEM A, et al. A Multi-Objective Technique to Prioritize Test Cases[J]. IEEE Transactions on Software Engineering, 2015, 99: 1-22.

(上接第 147 页)

提出的轻量化检测策略在保证检测到相同大型概念的情况下，无论在概念数还是构造时间的性能方面均有大幅度提升，能够有效缩减概念数，降低时间开销。

表 4 轻量化策略生成时间与效率提升

被测程序	完整概念格生成时间/s	轻量化策略生成时间/s	效率提升/%
acct-6.3.2	45.9642	18.2294	60.34
EPWIC-1	3263.1785	2244.0879	31.23
barcode	3759.1561	2054.5517	45.35
copia	1837.3601	848.1254	53.84
findutils	4216.6482	2856.3575	32.26
replace	2487.2165	1287.1022	48.25
userv-client	4332.8168	3060.0865	29.37
which	1897.4587	1305.6717	31.19
bc	5674.0319	1975.6185	65.18
dc	4218.1547	1309.8762	68.95
ed	3824.9426	1409.6520	63.15
indent	5246.4660	3009.2785	42.64
平均	3400.2829	1781.5531	47.65

结束语 针对 MSG 在依赖簇检测准确度方面的不足，本文提出基于 FCA 的依赖簇检测新方法。首先，探讨了如何将源代码中获取的依赖关系抽象和整理成形式背景，搭建了形式概念分析和依赖簇检测的桥梁。从理论层面探讨了概念格构造算法生成所有概念后如何从中选取大型概念，分析和给出了利用大型概念检测依赖簇的方法，并设计了基于形式概念分析的依赖簇检测方法的流程。其次，为提高效率，更有针对性地选取大型概念，提出了基于对象包含度的 FCBO 算法剪枝改进策略，并从理论层面证明了其有效性，同时设计了轻量化策略的流程。最后，为验证新方法及其轻量化检测策略的有效性，选取了 12 个不同规模、不同领域的被测程序进行实验研究。结果表明，本文提出的方法有效地提高了依赖簇检测的准确度，轻量化检测策略不但可以选取与完整概念格相同的大型概念，还能大幅度约减概念数，缩小大型概念的搜索域，降低构造概念格的时间开销。

参 考 文 献

[1] BINKLEY D, HARMAN M. Locating dependence clusters and dependence pollution[C]// Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005 (ICSM'05). IEEE, 2005: 177-186.

[2] ACHARYA M, ROBINSON B. Practical change impact analysis based on static program slicing for industrial software systems [C]// Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 746-755.

[3] BESZÉDES A, GERGELY T, JÁSZ J, et al. Computation of tatic execute after relation with applications to software maintenance[C]// IEEE International Conference on Software Maintenance, 2007 (ICSM 2007). IEEE, 2007: 295-304.

[4] SZEGEDI A, GERGELY T, BESZEDES A, et al. Verifying the concept of union slices on Java programs[C]// 11th European Conference on Software Maintenance and Reengineering, 2007 (CSMR'07). IEEE, 2007: 233-242.

[5] HAJNAL A, FORGÁCS I. A demand-driven approach to slicing legacy COBOL systems[J]. Journal of Software: Evolution and Process, 2012, 24(1): 67-82.

[6] BINKLEY D, GOLD N, HARMAN M, et al. Dependence anti patterns[C]// 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2008: 25-34.

[7] ACHARYA M, ROBINSON B. Practical change impact analysis based on static program slicing for industrial software systems [C]// Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 746-755.

[8] BINKLEY D, HARMAN M. Identifying' Linchpin Vertices' That Cause Large Dependence Clusters[C]// Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, 2009 (SCAM'09). IEEE, 2009: 89-98.

[9] BINKLEY D, HARMAN M, HASSOUN Y, et al. Assessing the impact of global variables on program dependence and dependence clusters[J]. Journal of Systems and Software, 2010, 83 (1): 96-107.

[10] HARMAN M, BINKLEY D, GALLAGHER K, et al. Dependence clusters in source code[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2009, 32(1): 1-33.

[11] BLACK S, COUNSELL S, HALL T, et al. Fault analysis in OSS based on program slicing metrics[C]// 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009 (SEAA'09). IEEE, 2009: 3-10.

[12] BLACK S, COUNSELL S, HALL T, et al. Using program slicing to identify faults in software[M]// Dagstuhl, Seminar N°. 2005.

[13] ISIAM S S, KRINKE J, BINKLEY D. Dependence cluster visualization[C]// Proceedings of the 5th International Symposium on Software Visualization. ACM, 2010: 93-102.

[14] WILLE R. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts[M]// Ordered Sets. Springer Netherlands, 1982: 445-470.