计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200200067

软件演化历史的逆向工程生成方法研究

钟林辉[1] 扶丽娟[1] 叶海涛[1,2] 齐 杰[1] 徐 静[3]
1 江西师范大学计算机信息工程学院 南昌 330022
2 南昌大学第一附属医院 南昌 330000
3 江西师范大学文学院 南昌 330022

摘 要 为了更好地管理软件的演化,越来越多的软件演化管理模型被提出,然而现存的软件演化管理模型或版本管理系统中存储的软件大多是以文件或者项目为单位的,而这些模型中又缺乏软件体系结构及组成构件的演化历史信息,这就致使软件演化管理人员无法直观有效地了解以及管理软件体系结构及其构件的演化。为此,文中通过定义软件演化二叉树这一概念,表示一个软件及其组成构件的演化历史,并提出了一种基于软件体系结构逆向技术逆向出软件体系结构及其组成构件演化二叉树的方法,即利用软件源代码以及体系结构逆向技术逆向出软件系统的(原子)构件与软件体系结构(文中将软件体系结构看作一种特殊的复合构件),并度量相应的原子构件以及复合构件的多维属性,再利用这些属性通过提出的演化二叉树构造算法构造出软件的演化历史。最后设计了两组实验,用以分析演化二叉树构造的影响因素,分别利用 Bunch 以及 ACDC 体系结构逆向工具,在不同属性相似度阈值下生成演化二叉树和不同属性组合下生成演化二叉树。通过对 8 个开源软件(Cassandra,Hbase,Hive,OpenJpa,Zookeeper,RxJava,Groovy,Sqoop)的实验,可以发现属性相似度阈值以及构件属性对演化二叉树构造的影响,也可以看出逆向出的复合构件演化二叉树与真实的演化二叉树十分相似,并且可以得出使用体系结构逆向工具 ACDC 恢复演化二叉树的准确度更高。由此可见,提出的方法能够有效地逆向出这些开源软件及其组成构件的演化历史。

关键词:软件演化;演化历史逆向;软件体系结构;演化二叉树
中图法分类号 TP311

Study on Reverse Engineering Generation Method of Software Evolution History

ZHONG Lin-hui[1],FU Li-juan[1],YE Hai-tao[1,2],QI Jie[1] and XU Jing[3]
1 College of Computer and Information Engineering,Jiangxi Normal University,Nanchang 330022,China
2 The First Affiliated Hospital of Nanchang University,Nanchang 330000,China
3 School of Chinese Language and Literature,Jiangxi Normal University,Nanchang 330022,China

Abstract For the better management of software evolution,more and more software evolution management models have been proposed. However,most of the software are stored in the models in the unit of files or projects,and the models lack of the evolution history information of those software components,which make the evolution process hard to be intuitively and effectively understood and managed. In this paper,software evolution binary tree is defined to express the evolution history of software and its components. And a method to recover the evolution binary tree of software and its components by software architecture reverse technology is proposed as well. First of all,the (atomic) components of software system and software architecture (taken as a special composite component here) are recovered by software source codes and architecture reverse technology,and the multi-dimensional attributes of the corresponding atomic components and that of the composite components are measured,based on which the software evolution histories are constructed by the evolution binary tree construction algorithm. Finally,after analyzing the main factors that affect the construction of evolution binary trees according to two groups of experiment,some evolution binary trees are generated according to similarity thresholds with different attributes and some are generated in the basis of the combinations of different attributes by Bunch and ACDC (architecture reverse tools) respectively. Through the experiments of four open source software (Cassandra,Hbase,Hive,Openjpa,Zookeeper,RxJava,Groovy,Sqoop),the best similarity thresholds affecting the construction of evolution binary trees and the best attribute combinations suitable for the software are realized. And it's also can be seen that the evolution binary trees of composite components recovered are extremely similar to their corresponding real trees in framework. And using the architecture reverse tool ACDC to restore the evolutionary binary tree has higher accuracy. Consequently,the proposed method is effective to recover the evolution histories of these open source software and that of their components.

Keywords Software evolution,Evolution history reverse,Software architecture,Evolution binary tree

---

本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。

基金项目:国家自然科学基金(61462040,61662032,61662035,61966017,61762049,61262015);江西省教育厅科学技术项目(GJJ170207)
This work was supported by the National Natural Science Foundation of China(61462040,61662032,61662035,61966017,61762049,61262015) and Science and Technology Project of Jiangxi Provincial Education Department (GJJ170207).

通信作者:钟林辉(shiningto@jxnu.edu.cn)

1 引言

软件演化信息是一类重要的软件资产，能辅助软件开发者对软件开发过程和演化过程进行度量和分析，进而实现对软件过程的理解和复用。而软件体系结构同样作为一类重要的软件资产支持在较高的层次对软件的结构进行描述和分析，亦起到了从软件分析设计到软件实现的桥梁作用；而软件配置管理则支持对整个软件生命周期中软件资产的演化管理。为了有效支持以软件体系结构为中心，基于构件的软件开发方法，许多研究者提出了将软件体系结构和软件配置管理相结合的思想[1-2]以及以软件配置管理为基础的软件演化管理模型，将软件配置管理有关概念纳入软件体系结构、构件等元素中[3-4]。

虽然这些软件演化管理模型存储着软件的演化历史信息，但这些演化历史信息大多是以文件或者项目为单位，其中并未存储软件构件及软件体系结构的演化历史。所以通过这些软件演化管理模型无法追踪管理软件构件及软件体系结构的演化历史，而且对于那些游离于软件配置管理系统之外的遗留软件的演化也没有很好的支持。

因此，本文提出了一种软件演化历史逆向方法将软件演化历史逆向为一棵演化二叉树的形式。该方法首先使用体系结构逆向算法将软件系统逆向为其对应的类的关系依赖图；再结合软件源代码逆向生成原子构件和复合构件；然后使用本文提出的演化二叉树的概念及生成规则自动地构造原子构件和复合构件的演化二叉树，并生成原子构件以及复合构件之间的版本关系。为了分析影响演化二叉树构造的因素，我们设计了两组实验，分别为使用不同的体系结构逆向工具(ACDC[5]和 Bunch[6])在不同相似度阈值下生成演化二叉树和不同属性组合下生成演化二叉树。通过这两组实验可以得出属性相似度阈值以及构件属性是如何影响演化二叉树生成的。

本文第 2 节介绍相关研究；第 3 节介绍如何基于类的关系依赖图和系统源代码来逆向生成原子构件和复合构件；第 4 节主要介绍本文对演化历史的定义以及演化二叉树的构造规则；第 5 节在 8 个开源软件系统(Cassandra，Hbase，Hive，OpenJpa，Zookeeper，RxJava，Groovy，Sqoop)上进行了演化历史逆向影响因素分析实验；最后总结全文。

2 相关研究

为了管理软件的演化历史信息，目前主要利用软件配置管理模型及相应的工具对软件元素的变化以及不同软件元素之间的变化进行跟踪和管理。例如，文献[7-8]设计实现了一个支持多层、基于构件的软件体系结构的演化管理模型，可以在规约、实现和部署 3 个层次进行变化的追踪。Tudor 等提出的 Hismo 模型是一种以软件变化历史为中心的软件演化元模型，该模型能刻画不同软件实体及其组成部分的变化，并为软件演化的可视化和度量提供基础[9-10]。在本文的早期研究中，也将版本信息的概念同构件模型、软件配置管理模型相结合，提出了支持构件化软件演化的构件模型，从而支持构件化软件的演化管理[11]，在后续的研究中提出了一种能够自动检测不同软件版本中的多重重构操作的方法[12]。文献[13]介绍了一个名为 Historage 的工具，它可以提供 Java 中细粒度实体的完整演化历史，如方法、构造函数、字段等。该工具的一个特点是能够跟踪实体的演化历史，包括重命名变化。文献[14]提出了一个名为 Design Observer 的框架用于在软件演化过程中自动监控和跟踪设计变化。文献[15]提出了一种多视图多层级的演化管理模型，以支持软件体系结构的演化。

对于没有利用软件配置管理系统进行存储的软件系统，许多研究者提出了利用静态分析技术、文本分析和可视化等技术进行软件元素间变化关系的追踪。例如，文献[16]提出了一种结合关键字检索和启发式规则，能在多个层次恢复软件演化信息追踪关系的逆向方法。文献[17]通过对代码变化进行分组和聚类以发现变化的轨迹模式。文献[18]提出了一种多粒度的软件网络模型，从包、类、方法 3 个粒度级别表示多版本软件系统的拓扑结构，以更好地理解不同维度的软件演化。文献[19-20]从代码行演化的角度提出了将模糊历史图作为细粒度代码历史的模型，以表示不同代码行演化到其他代码行的程度，并提出了将模糊历史切片作为一种自动代码历史分析技术，该技术利用模糊历史图来提高代码历史分析的准确性。文献[21]提出了一种基于演化切片的变更影响分析技术，以分析代码级别的大型软件演化数据。文献[22]提出了一种动态模型切片方法，以解决遗留系统的演化研究问题。文献[23]采用可视化技术从多个角度展示软件在演化过程中版本以及关联关系的变化。文献[24]介绍了名为 CuboidMatrix 的可视化工具，该工具提供简单易用的导航操作探索了在大量软件修订中质量规则被打破的原因。文献[25]提出了一种名为 EVA 的可视化工具，该工具可以可视化和探索不断发展的软件体系结构，使用户能够评估架构设计决策的质量和架构变化对系统整体稳定性的影响。文献[26]提出了一种基于软件可视化和软件度量的组合方法来可视化软件系统的演化。文献[27]将动态波及效应分析方法和基于文本的波及效应分析方法相结合，能确定软件资产变化的潜在影响范围。

不同于上述方法，本文方法主要是在软件体系结构层级进行软件演化历史的逆向，通过度量原子构件以及复合构件的多维属性来逆向出软件的演化历史，以一棵演化二叉树的形式表示，从而协助软件构件以及软件体系结构演化的管理。

3 软件体系结构逆向

在构件化的软件开发中，一个大的软件系统是由若干个软件构件所构成的，我们可以把系统内部的构件看作原子构件，而把系统看作为一个由原子构件及其相互之间的依赖关系所组成的复合构件。

现有的软件体系结构逆向的方法较多，本文通过使用经典的软件体系结构逆向工具 Bunch 工具以及 ACDC 算法进行体系结构逆向，把系统版本的体系结构逆向成由包、簇、模块等表示的中间结果，这些中间结果可以表示为类的关系依赖图，如图 1 所示。

图 1 所示的类的关系依赖图中，右边的椭圆表示 JDK 中的类的集合，左边的矩形表示组成该系统版本的类集合，矩形中的 A，B，C 3 个圆形表示经过软件体系结构逆向得到的 3 个类簇，C_1，C_2，C_3，…，C_{12} 表示 12 个类，这 12 个类分别位于簇 A，B，C 和 JDK 中，类之间的有向边表示两个类之间存在依赖关系。

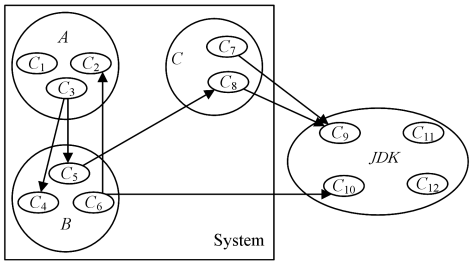


图 1 类的关系依赖图示例

Fig. 1 Example of class relational dependency graph

显而易见，从类的关系依赖图中只能获取到系统版本中所包含的类，以及这些类之间的功能需求关系和类所组成的簇等信息，而无法获取到对应的系统版本的版本信息和不同的系统版本之间的演化信息。

3.1 原子构件逆向生成

每个系统版本都可以通过逆向得到一个类的关系依赖图，而类的关系依赖图是由若干个簇以及簇中的类之间的依赖关系组成。在使用某个系统版本的类的关系依赖图以及该系统版本的源代码逆向生成所有的原子构件时，首先，把类的关系依赖图中的每个簇都逆向成为一个原子构件，令构件名为对应的簇名，并将簇中所有的类都添加到生成的原子构件中；然后，根据簇中的类之间的依赖关系构造原子构件之间的 Provides 关系和 Requires 关系，并分别保存到原子构件的 Provides 集合和 Requires 集合中；最后，根据系统源代码获取每个原子构件的多维属性值信息，并保存到对应的原子构件中，以便后期根据多维属性信息构造原子构件的演化树。

原子构件代表系统版本中的一个独立的功能单元，随着功能需求的变化，原子构件可能会发生变化。在对同一系统的多个系统版本逆向生成原子构件时，对于逆向出来的每一个原子构件，都有可能生成相同的原子构件版本，也可能生成不同的原子构件版本。

3.2 复合构件逆向生成

原子构件代表系统版本中的一个独立的功能单元，而无法代表软件系统的体系结构，因此，一个系统版本中所有的原子构件都逆向生成出来之后，再使用这些原子构件逆向生成一个能够代表系统版本的体系结构的复合构件。每个系统版本都生成一个复合构件，令复合构件名为对应的系统名，并把每个系统版本逆向出来的所有原子构件都作为复合构件的成员构件。同时，根据所有原子构件的多维属性值信息计算复合构件的多维属性值信息，并保存到对应的复合构件中，以便后期根据多维属性信息构造复合构件的演化树。

复合构件代表一个系统版本的体系结构，系统演化过程中，构成复合构件的原子构件集合可能会发生变化，即系统版本会发生变化，因此，认为同一个系统的多个版本逆向生成的多个复合构件是同一种复合构件的不同版本，而不同的系统逆向生成的是不同的复合构件。

4 演化历史逆向

软件系统版本之间的演化关系可以用演化树来表示，为了便于表示和处理，本文采用演化二叉树表示软件系统版本之间的演化关系。

4.1 演化树及演化二叉树

(1)演化树和演化二叉树的定义

定义 1 演化树是一棵树 $T=(root,F)$，如 4.2 节图 2(a)所示。其中，$root$ 是树根结点，F 是 $m(m\geqslant 0)$棵子树构成的森林，即 $F=(T_1,T_2,\cdots,T_i,\cdots,T_m)$，$T_i$ 称为根 $root$ 的第 i 棵子树。

在软件演化的语境下，若从版本演化二叉树的非叶子结点 N_m 到某一个叶子结点 N_n 存在一条路径，则表示存在一个从 N_m 到 N_n 的演化分支，显然可能存在从 N_m 出发的多条演化分支。

定义 2 演化二叉树是一种特殊语义的演化树，即 $T=(root,L,R)$，如 4.2 节图 2(b)所示。其中 L,R 分别是根结点 $root$ 的左右孩子结点。

1)本文规定从版本演化二叉树中任意一个非叶子结点 N_m 出发，一直沿着左孩子结点访问到达叶子结点 N_n 构成的演化分支称为主演化分支；从内部结点 N_m 出发到达以 N_m 右孩子结点为根的子二叉树中任意叶结点 N_k 的演化分支称为侧演化分支。

2)规定任意结点 N，若存在右孩子结点 $N.R$，则 $N.R$ 是 N 的一个拷贝，即 $N.R=Clone(N)$，且称拷贝结点为虚结点，非拷贝结点为实结点。

(2)演化树和演化二叉树的转换

假设演化树的根 $root$ 存在若干个孩子结点 $N_1,N_2,\cdots,N_m$，则将其转换化演化二叉树时需要作如下处理：

1)将 N_1 作为左孩子挂在 $root$ 上。

2)克隆一个新的结点 $Clone(root)$作为 $root$ 的右孩子，将 N_2 作为 $Clone(root)$的左孩子；克隆一个新的结点 $Clone(Clone(root))$作为 $Clone(root)$的右孩子，将 N_3 作为 $Clone(Clone(root))$的左孩子；以此类推，将 $root$ 的 $m-1$ 个结点(从 N_2 到 N_m)作为右孩子挂在对应的克隆结点上。

3)对 $root$ 的每个演化子树 $T(N_1),T(N_2),\cdots,T(N_m)$按照上述步骤重复处理。

按照上述处理后，演化树中任意一个结点 N 的第一个分支中的所有结点将落在演化二叉树中 N 的主演化分支中；N 的其他分支中的结点将落在相应的某个 N 的侧演化分支中。

假设演化二叉树的根 $root$ 存在左孩子 $root.L$ 和右孩子 $root.R$，则将其转化成演化树需要作如下处理：

1)将 $root.L$ 作为 $root$ 的第 1 个孩子结点。

2)若 $root.R$ 存在，则将$(root.R).L$ 作为 $root$ 的第 2 孩子结点；若$(root.R).R$ 存在，则将$((root.R).R).L$ 作为 $root$ 的第 3 个孩子结点；以此类推，若 $root$ 的第 $n-1$ 个侧演化分支存在，则将第 $n-1$ 个侧演化分支的根结点的左孩子作为 $root$ 的第 n 个孩子结点。

3)对 $root$ 的每个非克隆结点按照上述步骤重复处理。

4.2 演化二叉树的逆向生成规则

针对构件的一组版本，演化二叉树逆向生成规则利用构件的多维属性信息可以构造构件版本之间的演化关系，即演化二叉树。演化二叉树中存在实结点和虚结点这两种类型的结点，实结点中保存了该构件版本所对应的系统版本的版本基线、该构件的构件名、多维属性数组以及为该构件生成的版本号，虚结点中只保存了其父结点的构件名和为该虚结点生成的版本号。在构造演化二叉树时，每种构件单独构造一棵演化二叉树，而某种构件的每个构件版本都对应演化二叉树中的一个实结点。

构造某种构件的演化二叉树，主要是通过将该构件的多个构件版本按照它们逆向生成的先后顺序依次生成实结点并插入到演化二叉树中来实现的。在把由某个构件版本生成的实结点 Q 插入到演化二叉树 *tree* 中时，其主要步骤是：先计算演化二叉树中所有的实结点与 Q 之间的相似度，然后按相似度从高到低的顺序依次选择演化二叉树中的实结点作为待插入位置结点 P，然后根据 Q 与 P 之间相似度的大小进行判断，并根据判断的结果来选择是将 Q 结点作为主演化分支插入到 P 的左孩子上还是作为侧演化分支插入到 P 的右孩子上，直到将 Q 插入到演化二叉树中为止。在插入时有如下的几种情形。

(1)演化二叉树为空，则将 Q 作为演化二叉树的根结点，令 Q 的版本号为初始版本号。

(2)演化二叉树不为空，则分为如下几种情形：

1)Q 和 P 为同一个构件版本，则将 Q 和 P 这两个结点合并，只需将 Q 的版本基线添加到 P 的版本基线序列当中，令 Q 的版本号与 P 的版本号相同；

2)Q 和 P 为不同的构件版本，且相似度较高，且 P 没有左孩子结点，则将 Q 作为 P 的左孩子结点，令 P 的版本号的最末位加 1 作为 Q 的版本号；

3) Q 和 P 为不同的构件版本，且相似度较高，但 P 有左孩子结点，则重新选择一个次相似的实结点作为新的待插入位置结点 P，并重新判断；

4)Q 和 P 为不同的构件版本，且相似度较低，且 P 没有右孩子结点，则构造一个虚结点 *clone*(P)，并将 *clone*(P)作为 P 的右孩子结点，然后将 Q 作为 *clone*(P)的左孩子结点，令 P 的版本号后面加上“.0”作为 *clone*(P)的版本号，令 *clone*(P)的版本号的最末位加 1 作为 Q 的版本号；

5)Q 和 P 为不同的构件版本，且相似度较低，但 P 有右孩子结点，则找到 P 的最右侧的侧演化分支的根结点 R，构造一个虚结点 *clone*(R)，并将 *clone*(R)作为 R 的右孩子结点，然后将 Q 作为 *clone*(R)的左孩子结点，令 R 的版本号后面加上“.0”作为 *clone*(R)的版本号，令 *clone*(R)的版本号的最末位加 1 作为 Q 的版本号。

在对某种构件的一组版本构造演化二叉树时，为了把由某个构件版本生成的实结点插入到演化二叉树中的适当位置，需要在多个构件版本之间进行相似性度量。相似性度量指的是针对给定的多维属性变化向量(例如文件数目、构件大小等)，度量其在给定的两个构件版本之间的相似度。本文采用三角余弦公式作为相似性度量的计算公式，具体如下：

$$Sim_p(S_1,S_2)=\frac{\overrightarrow{S_1}*\overrightarrow{S_2}}{\|S_1\|*\|S_2\|}=\frac{\sum_{i=1}^{n}a_i*b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2}*\sqrt{\sum_{i=1}^{n}(b_i)^2}}$$

其中，S_1 和 S_2 分别表示构件版本 S_1 与 S_2 在多维属性组 p 上的属性变化向量，即 S_1 对应的属性变化向量为$\langle a_1,a_2,a_3,\cdots,a_i,\cdots,a_{n-1},a_n\rangle$，$S_2$ 对应的属性变化向量为$\langle b_1,b_2,b_3,\cdots,b_i,\cdots,b_{n-1},b_n\rangle$，则两个构件版本的相似度可以表示为：$similarity=Sim_p(S_1,S_2)$。显然，相似度的值位于$(0,1]$之间，数值越大代表两个构件的相似度越大，反之，则相似度越小。此外，为了区分构件之间相似性的程度，人为设定了一个相似度阈值，当两个构件之间的相似度不小于阈值时，则认为这两个构件之间的相似度较高，当相似度小于阈值时，则认为这两个构件之间的相似度较低。

原子构件的相似性度量所使用的多维属性是：原子构件中类的个数、原子构件中类文件的个数、原子构件中类文件的大小总和。

复合构件的相似性度量所使用的多维属性是：复合构件中原子构件的个数、复合构件中原子构件的大小总和、复合构件的软件体系结构大小、有效代码行数以及 java 文件数。其中，采用图编辑距离[28]的方式对复合构件的体系结构大小进行量化，即：

$$ValueSA(V_i)=GraphEditDistance(g(V_i)-g(V_0))$$

其中，$i>0$，$g(V_0)=\varnothing$。

图 2(b)是按照以上规则构造出来的一棵演化二叉树，其对应的演化树如图 2(a)所示。其中，B 表示该构件版本对应的系统版本的版本基线，P 表示该构件版本的多维属性值数组信息，V 表示该构件版本的版本号，L 和 R 分别表示左孩子结点和右孩子结点。从图 2(b)中可以看出，该演化二叉树是由 8 个构件版本构成的，其中第 7 个构件版本与第 5 个构件版本是相同的，因此该演化二叉树中只有 7 个实结点；此外，版本基线为 1 的实结点拷贝生成了 2 个虚结点，版本基线为 2 的实结点拷贝生成了 1 个虚结点。

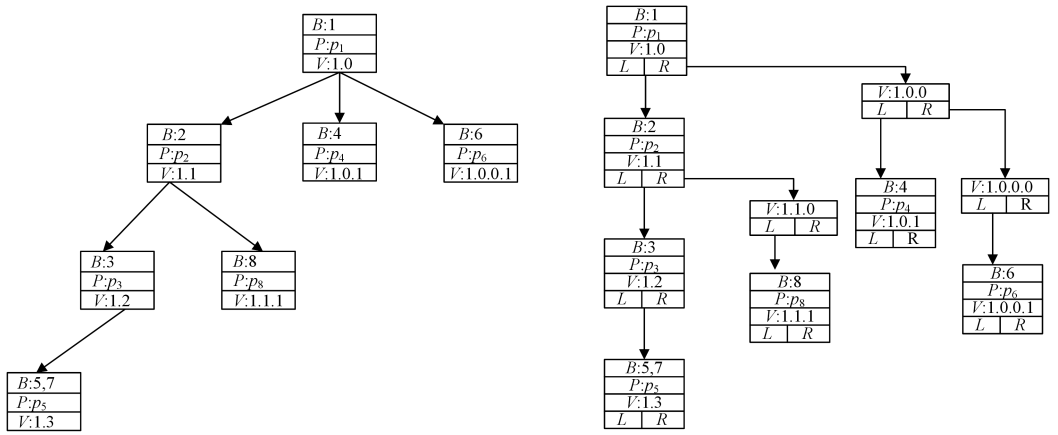


图 2 构件版本演化树和演化二叉树示例

Fig. 2 Example of component version evolution tree and evolution binary tree

5 案例分析

本节主要展示使用本文所提出的软件演化二叉树构造算法对一些开源软件系统的实验，并展示所恢复出的构件演化二叉树，其中包括原子构件演化二叉树以及复合构件演化二叉树。

5.1 构件演化二叉树示例

本节选用 4 个系统(Cassandra，Hbase，Hive，OpenJpa)展示所恢复出的构件演化二叉树。这 4 个系统的具体版本信息以及逆向生成的原子构件的个数如表 1 所列。其中，系统版本号是 github 中设定的版本号，编制的版本基线是按照提取的开源系统版本重新按序编排(从 1 开始)；除了表 1 中所展示的每个系统版本各自逆向生成的原子构件的个数之外，表 1 还统计出 Hive 系统共逆向生成了 58 棵原子构件演化二叉树，Cassandra 系统共逆向生成了 37 棵原子构件演化二叉树，Openjpa 系统共逆向生成了 43 棵原子构件演化二叉树，Hbase 系统共逆向生成了 64 棵原子构件演化二叉树。此外，每个系统都逆向生成了一棵复合构件版本演化二叉树。

对表 1 所选取的这 4 个系统的 40 个系统版本进行逆向，一共生成了 4 棵复合构件演化二叉树和 202 棵原子构件演化二叉树。其中，当 Hive，Cassandra，Openjpa 和 Hbase 这 4 个开源系统在相似度阈值分别取值为 0.999 85，0.999 50，0.999 90 和 0.999 50 时，各自生成的复合构件演化二叉树如图 3 所示。

表 1 系统版本信息及各版本所逆向生成的原子构件的个数

Table 1 System versions information and number of atomic components generated by each version

版本基线	Hive		Cassandra		Openjpa		Hbase	
	版本号	原子构件数	版本号	原子构件数	版本号	原子构件数	版本号	原子构件数
1	0.3.0	19	0.3.0	19	1.0.1	31	0.1.0	13
2	0.4.1	34	0.4.1	14	1.0.3	31	0.1.3	12
3	0.5.0	37	0.5.1	16	1.1.0	33	0.18.0	15
4	0.6.0	40	0.6.2	23	1.2.0	33	0.19.0	16
5	0.7.0	46	0.6.5	22	1.2.1	33	0.19.3RC1	17
6	0.7.1	46	0.7.0	28	1.2.2	33	0.20.2	22
7	0.8.1	54	0.7.5	29	2.0.0	40	0.89.20100621	23
8	0.9.0	55	0.7.8	29	2.0.0-M3	39	0.89.20100924RC1	26
9					2.0.1	40	0.90.2	35
10					2.1.0	41	0.90.4	35
11					2.1.1	41	0.92.0	40
12					2.2.0	42	0.94.0	48

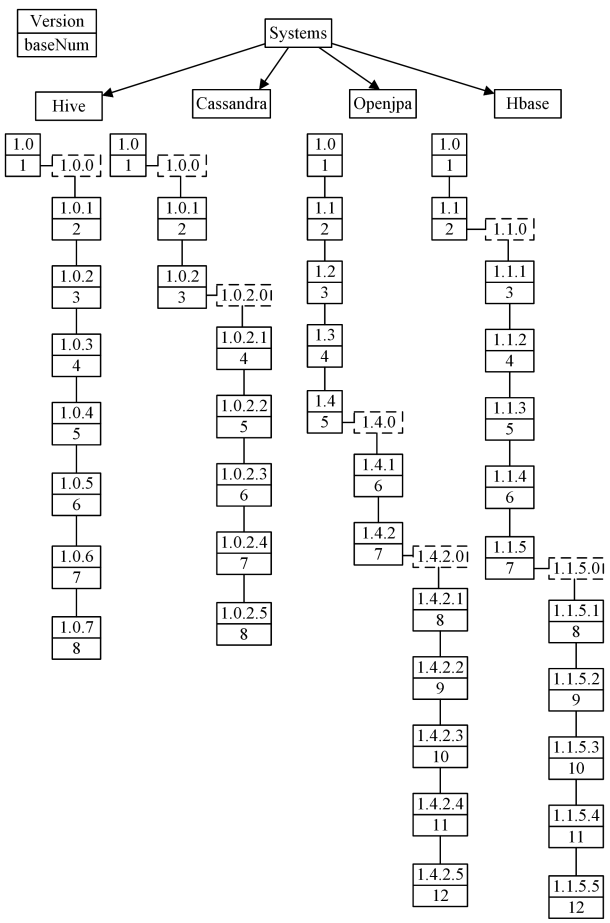


图 3 4 个系统生成的复合构件演化二叉树

Fig. 3 Evolution binary trees of composite components generated from 4 systems

5.2 构件演化二叉树有效性实验

本实验使用了 8 个开源软件系统(Cassandra，Hbase，Hive，OpenJpa，Zookeeper，RxJava，Groovy，Sqoop)的源代码作为实验基础，以分析以下两个问题：

RQ1 相似度阈值对演化二叉树恢复准确度的影响，在最佳的属性相似度阈值比较利用不同体系结构逆向工具(Bunch)恢复构件演化二叉树的准确度。

RQ2 属性组合对演化二叉树恢复准确度的影响，在最佳的属性组合情况下比较利用不同体系结构逆向工具(Bunch)恢复构件演化二叉树的准确度。

由于软件演化过程中并未保存真实的原子构件演化二叉树，无法将恢复出的原子构件演化二叉树与真实的演化二叉树进行对比。因此，本实验主要用于验证恢复的复合构件(软件系统)演化历史的准确度，并采用树编辑距离算法[28]计算逆向生成的复合构件演化二叉树转化为真实的软件系统演化二叉树所需的最小代价，该代价作为两棵树的相似性度量。同时规定实验准确度的度量公式如下：

$$P=\frac{K}{S}$$

其中，P 表示构件演化二叉树恢复实验的准确度，S 表示总实验次数，K 表示实验中准确恢复的实验次数。本文将与真实演化二叉树树编辑距离小于或等于 3 的实验视为准确恢复。

针对 RQ1，图 4 展示了使用不同的体系结构逆向工具(Bunch)在不同相似度阈值取值情况下最小代价的变化情况。

图 5 展示使用 Bunch 不同属性组合所对应的树编辑距离的变化情况。

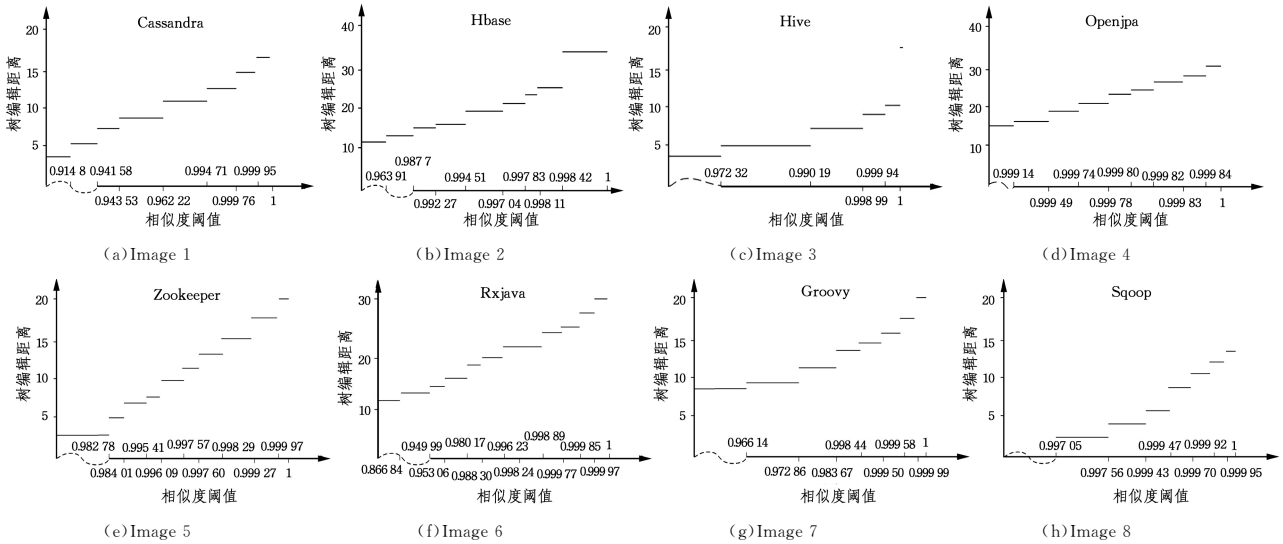


图 4 使用 Bunch 不同相似度阈值所对应的树编辑距离的变化

Fig. 4 Change of tree edit distance corresponding to different similarity thresholds by using Bunch

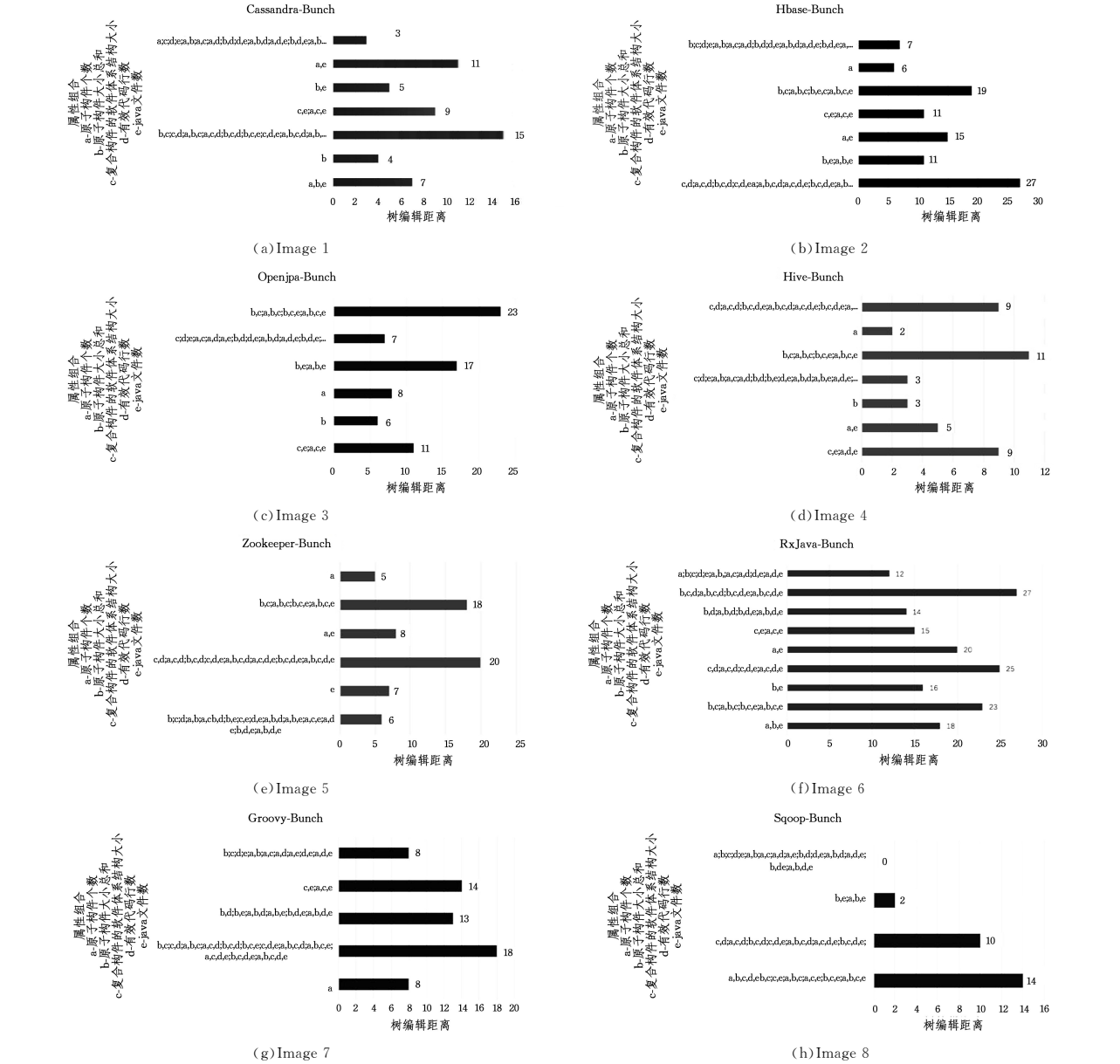


图 5 使用 Bunch 不同属性组合所对应的树编辑距离的变化

Fig. 5 Change of tree edit distance corresponding to different attribute combinations by using Bunch

从图4可以看出，相似度阈值取值在一些区域时，逆向生成的复合构件演化二叉树与真实的系统版本演化二叉树之间的编辑距离是相同的，即取这些区域内的相似度阈值时，逆向生成的复合构件演化二叉树是相同的；此外，还可以看出，随着相似度阈值取值逐渐增大，逆向生成的复合构件演化二叉树与真实的系统版本演化二叉树之间的编辑距离是呈台阶式的增长趋势的，并当相似度阈值取值为1.0时，编辑距离达到最大。

根据上述使用Bunch体系结构逆向工具的实验可知，在8次实验中树编辑距离小于等于3的有3次，可得其准确度$P_{\text{Bunch}}=37.5\%$。

从图5可以看出，在Cassadndra实验中，用Bunch逆向出的演化二叉树与真实演化二叉树的树编辑距离最小为3，选用这些属性生成的演化二叉树与真实的演化二叉树最相似。在Hbase实验中可以看出，其逆向效果比较好的属性影响为原子构件的大小，在此属性下生成的演化二叉树与系统真实演化二叉树的树编辑距离最小为6。在Hive的实验中可以看出，使用Bunch逆向Hive系统的演化二叉树时，属性影响为原子构件个数，逆向出的演化二叉树与真实系统的演化二叉树的树编辑距离最小为2。在Openjpa实验中可以看出，使用Bunch逆向得到的演化二叉树较为合适的属性影响都为原子构件的大小总和，在此属性下生成的演化二叉树与系统真实演化二叉树的树编辑距离最小都为6。

从实验数据可知，使用体系结构逆向工具Bunch恢复的软件演化历史的准确度为$P_{\text{Bunch}}=37.5\%$。可以看出使用不同的软件体系结构逆向工具，以及针对不同软件系统采用不同的属性组合对构造出来的演化二叉树有较大的影响。因此，不同的软件系统应对有针对性地选择合适的属性来逆向其构件演化二叉树。

结束语 为弥补现有的软件演化管理模型未能很好地支持、管理软件构件以及软件体系结构的演化，本文将软件历史定义为一棵演化二叉树，并提出了演化二叉树的构造算法，然后通过度量出的原子构件以及复合构件的多维属性逆向出软件的演化历史并用演化二叉树的形式加以表示，从而弥补了软件演化管理模型的不足。同时，基于上述方法，对8个开源软件系统进行了实验，得到了这些开源软件系统的组成构件及系统体系结构的演化历史，通过实验分析了演化二叉树生成时受相似度阈值和构件属性的影响。从实验的结果可以看出，通过本方法逆向出的软件演化历史与真实软件演化历史是十分相似的，也就是说本文提出的方法可以有效地逆向出软件的演化历史，即通过使用本文方法可为构件化软件的开发和管理提供帮助。

参 考 文 献

[1] WESTFECHTEL B，CONRADI R. Software architecture and software configuration management[M]. Software Configuration Management. Springer，Berlin，Heidelberg，2003：24-39.

[2] VAN DER HOEK A，HEIMBIGNER D，WOLF A L. Software Architecture，Configuration Management，and Configurable Distributed Systems：A Ménage a Trois. Technical Report CU-CS-862-98[R]. University of Colorado，1998.

[3] MOKNI A，HUCHARD M，URTADO C，et al. An evolution management model for multi-level component-based software architectures[C]// 27th International Conference on Software Engineering and Knowledge Engineering. 2015：674-679.

[4] GERGIC J. Towards a versioning model for component-basedsoftware assembly[C]// International Conference on Software Maintenance(ICSM 2003). IEEE，2003：138-147.

[5] MITCHELL，BRIAN S，SPIROS M. On the automatic modularization of software systems using the bunchtool[J]. IEEE Transactions on Software Engineering，2006，32(3)：193-208.

[6] TZERPOS V，HOLT R C. Accd：an algorithm forcomprehension-driven clustering[C]// Proceedings Seventh Working Conference on Reverse Engineering. IEEE，2000：258-267.

[7] MOKNI A，HUCHARD M，URTADO C，et al. An evolution-management model for multi-level component-based software architectures[C]// 27th International Conference on Software Engineering and Knowledge Engineering. 2015：674-679.

[8] MOKNI A，URTADO C，VAUTTIER S，et al. A formal approach for managing component-based architecture evolution [J]. Science of Computer Programming，2016，127：24-49.

[9] GÎRBA T，DUCASSE S. Modeling history to analyze software evolution[J]. Journal of Software：Evolution and Process，2006，18(3)：207-236.

[10] GÎRBA T. Modeling history to understand software evolution [D]. Bern：University of Bern，2005：13-19.

[11] ZHONG L H，XIE B，SHAO W Z. Supporting Component-based Software Development by Extending the CDL with Software Configuration Information [J]. Journal of Computer Research and Development，2002，39(10)：1361-1365.

[12] ZHONG L H，HUANG X M，XUE L B，et al. Research on automatic detection technology of multiple software reconstruction based on version [J]. Journal of Jiangxi Normal University (Natural Science Edition)，2018，42(5)：28-33，36.

[13] HATA H，MIZUNO O，KIKUNO T. Historage：fine-grained-version control system for java[C]// Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution. 2011：96-100.

[14] HAMMAD M. Design Observer：A Framework to Monitor Design Evolution[M]// Information Technology-New Generations. Springer，Cham，2018：635-640.

[15] HASSAN A，OUSSALAH M. Evolution Styles：Multi-View/Multi-Level Model for Software Architecture Evolution[J]. Journal of Software，2018，13(3)：146-155.

[16] WANG J S，AI W，PENG X，et al. Recovering Traceability Links among Multi-level Software Evolution Information [J]. Computer Science，2012，39(7)：135-139.

[17] JIANG Q，PENG X，WANG H，et al. Summarizing evolutionary trajectory by grouping and aggregating relevant code changes [C]// 2015 IEEE 22nd International Conference on Software Analysis，Evolution and Reengineering. IEEE，2015：361-370.

[18] PAN W，LI B，MA Y，et al. Multi-granularity evolution analysis of software using complex network theory[J]. Journal of Systems Science and Complexity，2011，24(6)：1068-1082.

[19] SERVANT F，JONES J A. Fuzzyfine-grained code-historyanalysis[C]// Proceedings of the 39th International Conferenceon Software Engineering. IEEE Press，2017：746-757.

[20] SERVANT F，JONES J A. History slicing：assistingcode-evolution tasks[C]// Proceedings of the ACMSIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM，2012：43.

[21] WEN W，CHEN J，YUAN J，et al. Evolution slicing-based

change impact analysis[C]// 2017 IEEE Third International Conference on Big Data Computing Service and Applications (Big Data Service). IEEE, 2017: 293-298.
[22] GUAN H, YANG H, WEN Z, et al. A dynamic model slicing approach for system comprehension during software evolution[J]. Multiagent and Grid Systems, 2018, 14(1): 1-29.
[23] AGHAJANI E, MOCCI A, BAVOTA G, et al. The code time machine[C]// Proceedings of the 25th International Conference on Program Comprehension. IEEE Press, 2017: 356-359.
[24] SCHNEIDER T, TYMCHUK Y, SALGADO R, et al. Cuboid Matrix: Exploring Dynamic Structural Connections in Software Components using Space-Time Cube[C]// 2016 IEEE Working Conference on Software Visualization. IEEE, 2016: 116-125.
[25] NAM D, LEE Y K, MEDVIDOVIC N. EVA: a tool for visualizing software architectural evolution[C]// Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. ACM, 2018: 53-56.
[26] LANZA M, DUCASSE S. Understanding software evolution using a combination of software visualization and software metrics[C]// Proceedings of LMO 2002. 2002.
[27] WANG W, LI T, HE Y, et al. A Hybird Approach for Ripple Effect Analysis of Software Evolution Activities [J]. Journal of Computer Research and Development, 2016, 53(3): 503-516.
[28] ZHONG L H, XIA J, PENG Y, et al. Research on a Method of Software Architecture Change Measure with Graph Edit Distance and Its Application[J]. Journal of Chinese Computer Systems, 2018, 39(3): 425-432.

ZHONG Lin-hui, born in 1974, Ph. D, professor, is a member of China Computer Federation. His main research interests include software architecture, software evolution and maintenance.

(上接第 534 页)

表 6 数据预测

Table 6 Data prediction

ARIMA(0,2,0)	2018	2019	2020
预测	31969	32919	33869
UCL	32834	34852	37104
LCL	31104	30986	30634

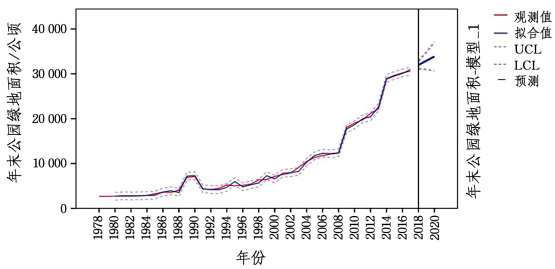


图 12 数据预测

Fig. 12 Data prediction

根据模型输出结果，2018－2020 年的公园绿地面积预测结果分别为 31969 公顷、32919 公顷、33869 公顷。

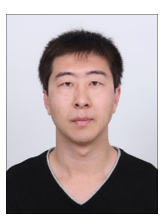
结束语 本文的时间序列分析探索不仅以 ARIMA 模型的理论作为基础，而且借鉴了机器学习中的调参思想，充分利用了 SPSS 软件的功能特性分别进行手动建模和专家建模器建模，检测离群值。从模型拟合角度看，本实验的最终预测为 ARIMA 模型理论和 SPSS 软件功能的综合结果。

本技术理论适用于对 ARIMA 模型要求不太严格的应用场景，可以将此分析过程应用于生活中的很多领域。由于 SPSS 软件对此模型的检验功能并不完善，一些检验指标无法直接通过系统输出，本实验也并没有自行构建复杂的计算，因此对模型的检验更多的还是以可视化结合模型拟合统计量的方式完成。

当然，本文分析探索技术仍然有改进空间，可以尝试使用参数估计、模型检验等功能支持性更好的软件，如 STATA，SAS 等；增加数据量，提高模型的拟合程度；留出一些末尾的时间序列数据不参与拟合，当作预测对比参考；在探索过程中使用单位根等其他检验方式等。

参 考 文 献

[1] HU C Q, SHU D. Predictive Analysis and Policy Orientation Research of Urban Landscaping[J]. Journal of Green Science and Technology, 2015(12): 108-109.
[2] ZHANG G H. Analysis and Prediction of Precipitation Trend in Weinan City Based on ARIMA Model[J]. Value Engineering, 2019(34): 197-199.
[3] ZHANG Y J. Based on ARIMA model Simple comparison analysis with index prediction results[J]. Marketing Research, 2019(11): 23-26.
[4] ZHENG M G, LI Q. Scenario prediction of China's oil resource demand in 2020－2030[J]. Advances in Earth Science, 2020, 35(3): 286-296.
[5] YOU Y L, LI J, GAO S Y J, et al. Application of the combination of ARIMA model and SVM model in the prediction of infectious diarrhea[J]. Journal of Medical Pest Control, 2020(5): 432-435.
[6] YANG X N, ZHANG K X, YANG H G, et al. Multiple Regression and ARIMA Model Prediction on the Yield of MSW in Xi'an[J]. Environmental Sanitation Engineering, 2020, 28(2): 37-41.
[7] ZHAO C Y. Research on the Stationarity of Smooth Transition Autoregressive Model[J]. The Journal of Quantitative & Technical Economics, 2012(1): 152-160.
[8] PEI W, CHEN F, CHENG L Q. Research on ARIMA prediction technology of traffic volume time series[J]. Shanxi Science and Technology, 2009(1): 75-79.
[9] LV S L. Application of ARIMA model in precipitation prediction [J]. Water Sciences and Engineering Technology, 2012(2): 6-8.



YAN Xiang-xiang, born in 1989, bachelor. He is a participant of the Advanced Course for In-service Staff. His main research interests include big data analysis and application.