一种基于模糊概念格和代码分析的软件演化分析方法*

许佳卿　彭鑫　赵文耘

(复旦大学计算机科学技术学院，上海，200433)

摘要 软件系统的演化分析是程序分析和程序理解的一个重要方面。通过演化分析可以了解系统需求和设计的演化趋势，从而更好地理解系统的需求和设计决策。本文在前期工作所提出的基于模糊概念格的程序分析方法基础上，进一步将其用于系统演化分析，提出了一种基于模糊概念格的软件演化分析方法。该方法利用基于概念相似度度量的松弛树匹配的方法建立不同版本概念格中概念和概念子格之间的映射关系，在此基础上通过结构差异分析来发现各种演化类型。实验表明，该方法能够有效地发现不同版本之间的高层演化信息，有助于开发人员理解系统的演化历史以及相关的设计决策。

关键字 程序理解；软件演化；演化分析；代码分析；概念格；树匹配；版本差异比较

中图法分类号 TP311.5

An Evolution Analysis Method based on Fuzzy Concept Lattice and Source Code Analysis

XU Jiaqing, PENG Xin, ZHAO Wenyun

(School of Computer Science, Fudan University, Shanghai 200433)

Abstract Evolution analysis of software systems is one of the important practice in program analysis and comprehension. By evolution analysis, one can find out the trends of the requirement and design evolutions, thus better understand the requirement and design decisions. In this paper, we propose to further introduce our previous work of program clustering based on fuzzy formal concept analysis into evolution analysis, and present an evolution analysis method based on fuzzy concept lattice and source code analysis. The method first constructs a fuzzy concept lattices for each software version, and then establishes concept mappings between different concept lattices by using a relaxation tree matching algorithm based on concept similarity. Based on the fuzzy concept lattices and concept mappings, the method can help identify various evolution types through structural difference analysis. Our experimental study has shown that the method can effectively discover high-level evolution information among different versions, thus help the developers better understand the evolution history and related design decisions.

Keywords Program Comprehension; Software Evolution; Evolution Analysis; Code Analysis; Concept Lattice; Tree Matching; Version Differencing

* 本课题得到国家“八六三”高技术研究发展计划项目基金（2007AA01Z125）、国家自然科学基金（60703092）、上海市重点学科建设项目（B114）资助。
许佳卿，男，1984 年生，硕士研究生，主要研究方向为软件再工程。Email: 062021114@fudan.edu.cn。彭鑫（通信作者），男，1979 年生，博士、讲师，主要研究方向为构件技术与软件体系结构、软件产品线、软件再工程。Email: pengxin@fudan.edu.cn。**赵文耘**，男，1964 年生，硕士、教授、博士生导师，主要研究方向为软件复用及构件技术。Email: wyzhao@fudan.edu.cn。

1 引言

在长期的软件演化和维护过程中，各种开发制品处于不断的变化之中，反映需求、设计的演化或者自身的改善和优化。分析软件系统的演化历史，可以辅助理解软件系统的需求和设计的演化过程及趋势，并指导未来的维护和开发制品复用。一般的演化历史分析需要借助于不同版本的需求、设计文档以及变更历史记录。然而，规范化文档的欠缺或不一致是一个常见的问题，特别是在长期的演化过程中保持相关文档的同步更新更是十分困难。针对这一问题，近些年来，相关学者提出了基于各种开发制品和开发过程信息的演化分析技术，包括基于 UML 设计模型的演化分析[1][2]、基于版本管理信息的分析方法[3][4][5]、基于代码分析的方法[6][7]等。

不同的演化分析方法所基于的分析对象以及分析目标都可能不同。例如，基于 UML 的分析方法以 UML 设计模型为基础，目标是发现面向对象设计的演化趋势；基于版本管理信息的分析方法借助于版本演化信息，覆盖各种演化类型；基于代码差异比较的方法以代码之间的具体差异为基础，目标是实现细粒度的代码演化分析。这些方法的基本步骤都是首先获取各个版本的分析模型，然后确定不同版本模型中元素之间的对应关系，发现相互之间的基本差异，然后以此为基础得到高层的演化模式或趋势分析结果。

本文主要关注于基于代码的系统需求及设计层次的高层演化分析。所谓的高层结构是指高于单个源文件代码或者计算单元所能表现出的与软件系统相关的信息，实际上在本文是指的是软件系统的业务需求，小到单元级的功能性需求，大到系统级的整体性需求。我们将概念定义为可以帮助软件开发人员和系统维护人员更好地理解需求、掌握业务的一种手段。软件体系结构为系统的不同参与者提供了交流的基础，也是系统理解和演化的基础[8]。随着系统规模的进一步扩大,系统维护与演化也变得更加复杂.如何将软件结构用于系统的自主维护与演化,日益成为研究者们关注的话题[9]。[10]提出了在构件描述语言 CDL 上扩充系统演化信息的方法, 使构件组装系统与配置管理系统形成有机的整体。而以代码为基础，是因为演化历史中不同代码版本通常都可以得到。传统的基于代码的演化分析[6][7]都是以代码自身的包和类等语法单元作为基本单位，缺点在于分析受到语法组织单元所限，难以发现高层需求和设计演化趋势。

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件；而概念格则是按照软件源代码文件的属性（与哪些关键字相关，即与哪些相应的业务需求相关）将其进行聚类归纳得到一个格型的结构，这样在进行需求变更、问题查找、业务理解的时候可以便于开发维护人员进行快速定位。

我们首先使用在前期工作[11]中所提出的基于模糊形式概念分析（Fuzzy Formal Concept Analysis）的程序聚类方法对各个版本的基本代码单元进行聚类。这些聚类在抽象层次上更接近于需求及设计单元，能够在很大程度上体现系统的高层结构。在此基础上，我们通过树匹配的方法在各个版本的聚类之间建立映射关系，并据此分析得到各种类型的差异。根据这些代码聚类的映射和差异信息，我们就可以发现各种设计和需求演化信息，例如引入对应于新的功能特征的代码模块、对各模块单元之间关系的结构调整、对已有模块的局部修改等。我们利用所提出的方法对一个商业软件系统的不同版本进行了演化分析，结果表明该方法可以有效地识别各种高层演化信息，对于理解软件系统演化历史及设计决策有很大的帮助。

2 背景知识

2.1 概念格与程序聚类

形式概念分析 FCA（Formal Concept Analysis）是一种基于格理论的聚类分析方法，通过对某一领域特定形式背景下对象和属性之间的关系进行聚类，从而得到相应的概念结构。对象和属性之间的关系可以通过概念格来表述[12]。概念格相对于传统的树型概念结构来说包含更丰富的信息，因为它支持多重继承[12]。FCA 方法非常适合用于概念聚类，目前已经广泛应用在数据挖掘、信息抽取（比如[12][13]）及程序聚类（比如[14][15]）等领域。

FCA 处理的是在一个对象集合 O 和一个属性集合 A 之间的二元关系 $I \subseteq O \times A$。元组 $C=(O,A,I)$被称为一个形式背景。对于一个对象集合 $O \subseteq O$，集合的公共属性 $\sigma(O)$被定义为：

$$\sigma(O)=\{a \in A \mid (o,a) \in I,\ \forall o \in O\}. \qquad (1)$$

同样的，对于一个属性集合 $A \subseteq A$，集合的公共对象 $\tau(A)$ 被定义为：

$$\tau(A)=\{o \in O \mid (o,a) \in I,\ \forall a \in A\}. \qquad (2)$$

当且仅当 $A=\sigma(O)$以及 $O=\tau(A)$时可以将一个元组 $c=(O,A)$称为是一个概念，也就是说元组 c 中的所有对象都共享元组 c 中的所有属性。对于一个概念

c=(O,A)，O 被称为是概念 c 的外延，用 extent(c)来表示，A 被称为是概念 c 的内涵，用 intent(c)来表示。

一个给定形式背景上的概念集合构成了一个偏序关系。给定形式背景上的概念集合以及这种偏序关系共同构成了一个完整的格，称为概念格。为了处理模糊信息，我们在前期工作[11]中将模糊形式概念分析方法引入到代码分析中，所得到的带有模糊属性的概念格称为模糊概念格。模糊概念格方法建立在以下这些概念基础上[12]。

定义 1. 模糊形式背景。一个模糊形式背景表示为 F=(O,A,I)，其中 O 为一个对象集合，A 为一个属性集合，映射 I 称为隶属度函数，这个函数满足：

I:$O\times A\rightarrow$[0,1]或者记作 I(o,a)=μ，其中 o$\in O$，a$\in A$，μ$\in$ [0,1]。

定义 2. 窗口。对于模糊形式背景中的属性，选取两个阈值 w_l 和 w_h，满足 $0\le w_l\le w_h\le 1$。w_l 和 w_h 构成窗口，w_l 和 w_h 分别称为窗口的下沿和上沿。

定义 3. 映射 f 和映射 g。在模糊形式背景 F=(O,A,I)中，O$\subseteq O$，A$\subseteq A$，在 O 和 A 之间可以定义属性映射函数 f 和对象映射函数 g：

$$f(O)=\{a \mid \forall o\in O,\ w_l\le I(o,a)\le w_h\}; \quad (3)$$

$$g(A)=\{o \mid \forall a\in A,\ w_l\le I(o,a)\le w_h\}. \quad (4)$$

定义 4. 模糊参数 σ 和 λ。 对于对象集合 O$\subseteq O$ 和属性集合 A$\subseteq A$，其中 A=f(O)，o$\in$O，a$\in$A，|O|和|A|分别是集合 O 和集合 A 的势，如果|O|和|A|都不为 0，则

$$\sigma_a=\frac{1}{|O|}\sum_{o\in O} I(o,a); \quad (5)$$

$\sigma=\sum_{a\in A}(\sigma_a/a)$，本式中的∑为模糊逻辑中的符号。 (6)

$$\lambda_a=\sqrt{\frac{\sum_{o\in O}(I(o,a)-\sigma_a)^2}{|O|}}; \quad (7)$$

$$\lambda=\frac{1}{|A|}\sum_{a\in A}\lambda_a. \quad (8)$$

定义 5. 模糊概念。如果对象集合 O$\subseteq O$ 和属性集合 A$\subseteq A$ 满足 O=g(A)且 A=f(O)，则 C=(O,A,σ,λ)被称为模糊形式背景 F 下的一个模糊概念。O 和 A 分别称为模糊概念 C 的外延和内涵，σ 和 λ 依据定义 4 计算。

定义 6. 模糊概念格。 F 的所有模糊概念的集合记为 CS(F)。CS(F)上的结构是一种偏序关系，可以简单地定义为：如果 $O_1\subseteq O_2$，则$(O_1,D_1)\le(O_2,D_2)$。通过这样的关系得到的有序集 CS(F)=(CS(F),≤)是一个格，称作模糊形式背景 F 的模糊概念格。

2.2 树匹配

树匹配的目的是发现多棵树之间所共享的模式以及对应的子树[16][17]。树匹配技术的应用十分广泛，包括模式识别、分子生物学、程序编译以及自然语言处理等[18]。在本文中，树匹配方法将用于发现不同版本概念格之间的概念映射关系。

2.2.1 树和树的映射

定义 7. 树（Tree）是由 n（n≥0）个数据元素组成的集合，可以表示为：T(V, E, Root(T))。其中，V 表示一个有限的结点集，Root(T)∈V 表示树的根结点，E 表示边集，它是 V 上的一个二元关系，它满足反自反、反对称和可传递性。如果(u, v)∈E 则称 u 结点为 v 结点的父结点，记为 u=parent(v)或 v=child(u)。如果树中兄弟结点左右顺序没有任何意义，则称此树为无序树。

定义 8. 设 T_1=(V, E, root(T_1))，T_2=(V',E', root(T_2))为两棵无序树，一个从 T_1 到 T_2 的映射 f 定义为 f$\subseteq$V×V'，并且对所有(v_1, v_1')，（v_2, v_2'）$\subseteq$E∪E'，满足以下的条件：

(1) $v_1=v_2\Leftrightarrow v_1'=v_2'$表示两棵树中参与映射的结点是一一对应的。

(2) v_1=ancestor(v_2)$\Leftrightarrow$ v_1'=ancestor(v_2')，表示映射保持结点对应之间的祖先后代关系。

映射 f 定义域 Domain(f)，定义为 $\text{Domain(f)}=\{v\in V \mid \exists v\in V' : (v,v')\in E'\}\subseteq V$

映射 f 值域 Domain(f)定义为 $\text{Range(f)}=\{v'\in V' \mid \exists v\in V : (v,v')\in E'\}\subseteq V'$

2.2.2 树匹配模型的概念[19][17][20]

在树映射定义的基础上，可以定义几种树匹配模型的经典定义。设 Q 和 T 是两棵树，Q_{sub} 和 T_{sub} 表示 Q 和 T 的子树集合。

定义 9. 子树匹配（Subtree-Matching）[19]。 如果存在从 Q_{sub} 到 T_{sub} 的一个映射 f 满足以下三个条件，则称 f 是从 Q 到 T 的一个子树匹配（如图 1 所示）。

(1) $v_1=v_2\Leftrightarrow f(v_1)=f(v_2)$, $v_1, v_2\in Q_{sub}$；

(2) v_1=parent(v_2) $\Leftrightarrow f(v_1)$=parent(f(v_2))；

(3) Q 中所有的叶子结点在 T 中存在映射结点，且在 T 中的映射结点也均为叶子结点。

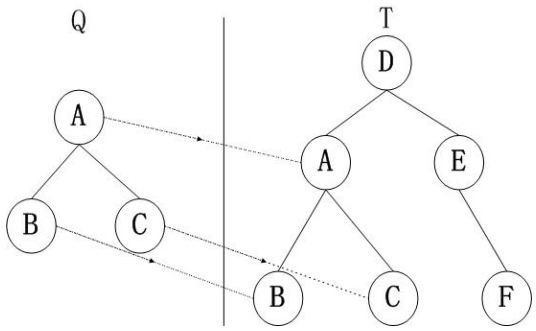


图 1 子树匹配

定义 10. 包含匹配（Inclusion-Matching）[17]。如果存在从 Q_{sub} 到 T_{sub} 的一个映射 f 满足以下三个条件，则称 f 是从 Q 到 T 的一个包含匹配（如图 2 所示）。

(1) $v_1= v_2 \Leftrightarrow f(v_1)=f(v_2), v_1, v_2 \in Q_{sub}$；

(2)$v_1=parent(v_2) \Leftrightarrow f(v_1)=ancestor(f(v_2))$；

(3) Q 中所有的叶子结点在 T 中存在映射结点，且在 T 中的映射结点也均为叶子结点。

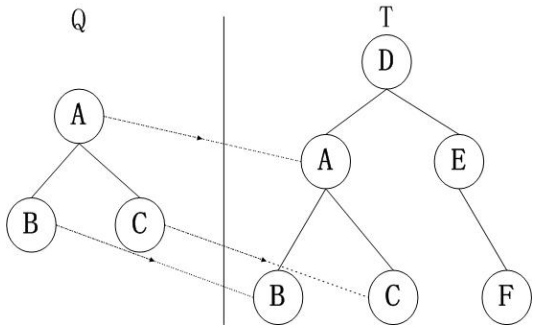


图 2 包含匹配

定义 11. 松弛匹配(Relaxation-Matching) [20]。如果存在从 Q_{sub} 到 T_{sub} 的一个映射 f 满足以下三个条件，则称 f 是从 Q 到 T 的一个松弛匹配（如图 3 所示）。

(1) $v_1= v_2 \Leftrightarrow f(v_1)=f(v_2), v_1, v_2 \in Q_{sub}$；

(2)$v_1=parent(v_2) \Leftrightarrow f(v_1)=ancestor(f(v_2))$；

(3) Q 中所有的叶子结点在 T 中存在映射结点，但允许在 T 中的映射结点不为叶子结点。

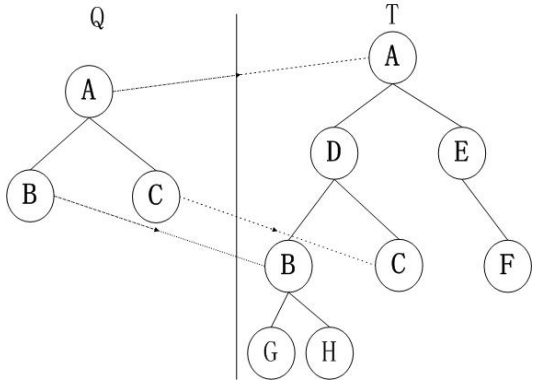


图 3 松弛匹配

2.3 基于树匹配的概念映射思路

为了分析不同版本之间的演化，我们首先利用在前期工作[11]中所提出的方法构造反映每个版本高层结构的模糊概念格，该方法是基于文本分析的，也就是将程序源代码作为对象、将源代码中的关键字（一般与软件系统的功能性或非功能性需求相关）作为属性。概念格中的每个概念都包含了在内涵（属性集合）上具有一定程度的体现的外延（对象集合）。概念的直接解释应该是可以直接对应功能需求的源代码文件集合，概念的外延对应于源代码文件而其内涵对应于带模糊性的业务术语集合。其中详细的模糊概念格构造和分析方法参见文献[11]。

为了分析软件系统不同版本概念格之间的差异，首先必须建立相应的映射关系，即找到不同版本所包含的概念之间的对应关系。以此为基础，我们才能进一步得到各种类型演化关系的判断。在本文中，我们利用树匹配方法实现不同版本概念之间的映射。概念格本身是格类型的数据结构（如图 4 所示），层次越深的概念拥有更多的内涵和更少的外延。由于支持多重继承，概念格并不完全符合树的结构定义。因此，我们将概念格中拥有多个父结点的概念认为是多个子树的共享结点，在分析过程中可以将它的复制拷贝加入到各个共享子树中，从而实现树的结构特性。

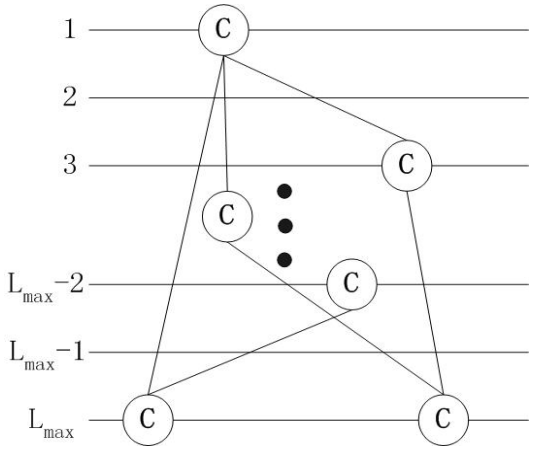


图 4 带有层次信息的概念格

由树匹配模型定义可以看出，定义 9-11 中的三种树匹配模型的约束条件是依次由强到弱的。包含匹配在子树匹配的基础上放宽了结点的信息之间的对应关系，只要求保持结点的祖先先后代关系，允许某些映射与映射的结点信息出现错层的情况，从而提高查全率。松弛匹配进一步放宽了对于叶结点的匹配要求，从而进一步提高查全率。由于高层需求和设计的演化常常会反映为概念间层次关系的较大变化，因此对于本文的概念映射而言松弛的树匹配模型是最合适的。

3 基于概念映射的演化分析方法

3.1 概念相似度计算

由于概念格的树形结构中的每个结点都是一个概念，那么如何定义概念与概念之间的相似度就成为了衡量两个概念是否相似的关键之处。对于两个概念 $C_1(O_1,A_1,\sigma_1,\lambda_1)$ 以及 $C_2(O_2,A_2,\sigma_2,\lambda_2)$以下给出概念相似度的定义。

定义 12. 外延相似度(Similarity on Extent)。两个概念的外延相似度是用其外延集合的交集的势的两倍除以两个概念的外延集合的势的和的结果。

$$Sim_{extent}(C_1, C_2)=\frac{|O_1 \cap O_2|*2}{|O_1|+|O_2|} \qquad (9)$$

例如对于 C_1 ($\{o_1,o_2,o_3\},\ldots$)以及 C_2 ($\{o_2,o_3,o_4\},\ldots$)两个概念，其外延相似度为：

$$Sim_{extent}(C_1, C_2)=\frac{|\{o_1,o_2,o_3\} \cap \{o_2,o_3,o_4\}|*2}{|\{o_1,o_2,o_3\}|+|\{o_2,o_3,o_4\}|}$$

$$=\frac{|\{o_2,o_3\}|*2}{|\{o_1,o_2,o_3\}|+|\{o_2,o_3,o_4\}|}=\frac{2*2}{3+3}=0.67。$$

定义 13. 内涵相似度(Similarity on Intent)。两个概念的内涵相似度的计算方法则相对复杂一些，是用其内涵集合的交集中的每个属性在各自概念中的 σ 的偏差值的绝对值减去 1 的绝对值的求和的两倍除以两个概念的内涵集合的势的和的结果。

$$Sim_{intent}(C_1, C_2)=\frac{(\sum_{a\in A_1\cap A_2}|1-|\sigma_a inC_1-\sigma_a inC_2||)*2}{|A_1|+|A_2|} \qquad (10)$$

例如对于C1(…,$\{a_1,a_2,a_3\}$,0.5/ a_1+0.6/ a_2+0.9/ a_3,…)以及 C2(…,$\{a_2,a_3,a_4\}$,0.7/ a_2+0.9/ a_3+0.2/ a_4,…)两个概念，其内涵相似度为：

$$Sim_{intent}(C_1, C_2)=\frac{((1-|\sigma_{a_2} inC_1-\sigma_{a_2} inC_2|)+(1-|\sigma_{a_3} inC_1-\sigma_{a_3} inC_2|))*2}{|\{a_1,a_2,a_3\}|+|\{a_2,a_3,a_4\}|}$$

$$=\frac{((1-|0.6-0.7|)+(1-|0.9-0.9|))*2}{3+3}=0.63。$$

由于在模糊概念格中不同的概念所包含的相同的属性还可能存在程度上的差异，因此这里在计算内涵相似度时用到了模糊参数 σ。

定义 14. 概念相似度(Similarity on Concept)。两个概念的总体概念相似度相应地可以根据其外延相似度以及内涵相似度来进行计算。

$$Sim(C_1, C_2)=Sim_{extent}(C_1, C_2)*\alpha+Sim_{intent}(C_1, C_2)*\beta \quad (\alpha+\beta=1) \qquad (11)$$

其中，α 和 β 分别表示外延和内涵相似度的权重，可以根据所分析的目标概念格的特点进行调解。一般情况下可以设置为 α=0.5，β=0.5，例如对于 C1($\{o_1,o_2,o_3\},\{a_1,a_2,a_3\}$,0.5/ a_1+0.6/ a_2+0.9/ a_3,…) 和 C2($\{o_2,o_3,o_4\},\{a_2,a_3,a_4\}$,0.7/ a_2+0.9/ a_3+0.2/ a_4,…)两个概念而言，其概念相似度为 $Sim(C_1, C_2)$=0.67*0.5+0.63*0.5=0.65。但一般而言内涵更能表达概念的意义，因此应该具有更高的权重。相关问题将在第四部分中进行讨论。

3.2 模糊概念格上的松弛匹配算法

在概念相似度基础上，我们进一步参考松弛匹配模型设计了模糊概念格松弛匹配算法。该算法实现了两个概念格之间的概念映射，可以用于确定概念之间的对应关系以及具有相同结构的概念偏序关系对。

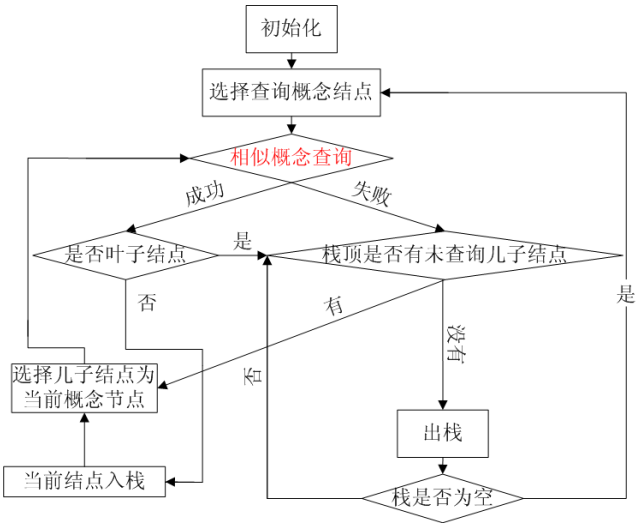


图 5 概念格松弛匹配算法图

侦测到的概念集合与偏序关系构成的同构的概念子格对可以帮助我们对软件版本差异进行一定程度的理解。概念格松弛匹配算法如图 5 所示。其形式化表述如图 6 所示。

显而易见该算法是一个典型的树的深度优先遍历。

在两个概念格中进行相似概念查询的算法如下：

1. 计算概念格 CL_2 第|A|层中所有的概念与概念 C 的概念相似度，取相似度最大的一个概念，并记录为 C’。

2. 将沿着概念 C’在概念格 CL_2 中的沿着父子关系的所有概念与概念 C 计算概念相似度，取相似度最大的一个概念，并记录为 C*。

3. 如果概念 C*与概念 C 的概念相似度 Sim (C,

C*)≥γ，γ 为概念的相似度阈值，则认为查询概念 C 在概念格 CL_2 中找到了对应相似的概念 C*，查询成功；否则认为查询概念 C 在概念格 CL_2 中没有对应的概念，查询失败。

图 7 为对于概念 C 运用相似概念查询算法的示意图。

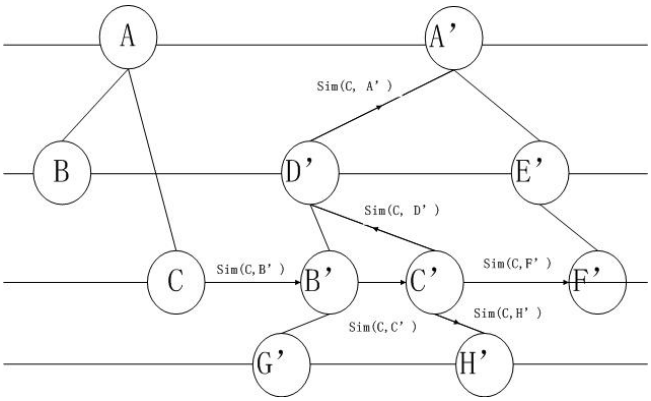


图 7 对概念 C 运用相似概念查询算法

Input: 概念格 CL_1 与 CL_2，概念栈 S，相似概念集合 Set1，相异概念集合 Set2，相似概念子格集合 Set3

Output: 相似概念集合 Set1，相异概念集合 Set2，相似概念子格对集 Set3

Initialize: L_{cur}=1, Set1=∅，Set2=∅, Set3=∅,C_{cur}=null, C*=null, CL_{cur}=null

Step 1: if L_{cur} 层中有概念未查询过 **then** C_{cur}=C(O, A, σ,λ), C 为未查询过的概念

else L_{cur}=L_{cur} +1

if L_{cur}>L_{max} **then** 算法结束

Step 2: C*= FindSimilarConcept(C_{cur}, CL_1, CL_2) //相似概念查询，C_{cur} 设为查询过

if C*=null **then** Set2=Set2∪C_{cur}, **goto step** 3 //查询失败

else Set1= Set1∪{(C_{cur}, C*)}, CL_{cur} = CL_{cur}∪{(C_{cur}, C*)}, C*=null

if child(C_{cur})=null **then goto step** 3 //当前结点为叶子结点

else push(C_{cur}, S), C_{cur}=child(C_{cur}), **repeat step** 2

Step 3: while(top(S)<>null)

if 有 child(top(S))未查询 **then** C_{cur}=child(top(S)), **goto step** 2

else pop(S)

if CL_{cur}<>null **then** Set3= Set3∪CL_{cur}, CL_{cur} = null

图 6 形式化的概念格松弛匹配算法

Input: 概念格 CL_1 中的查询概念 C(O, A, σ,λ)
Output: 概念格 CL_2 中的相似概念 C*
Step 1: 在概念格 CL_2 的第|A|层中选择与 C 相似度最大的结点，记为概念结点 C’ 。
Step 2: if *Sim* (C, C’) ≥γ，γ 为概念的相似度阈值 **then goto step 3**
else goto step 4
Step 3: a) if *Sim* (C, parent(C’)) ≤*Sim* (C, C’) **then** 放弃向上探测的路径
if *Sim* (C, child(C’)) ≤*Sim* (C, C’) **then** 放弃向下探测的路径
b) 在概念结点 C’，parent(C’)，child(C’)中选择新的结点 C_{new} 使：
Sim (C, C_{new})=Max(*Sim* (C, C’), *Sim* (C, parent(C’)), *Sim* (C, child(C’))
if C_{new}= C’ **then** C*=C’，查询成功，算法结束
else if C_{new}=parent(C’) **then** C’=parent(C’)，保留向上探测的路径
else if C_{new}=child(C’) **then** C’=child(C’)，保留向下探测的路径
repeat step 3
Step 4: a) if (*Sim* (C, parent(C’)) ≤*Sim* (C, C’) **and** *Sim* (C, child(C’)) ≤*Sim* (C, C’))
then C*=null，查询失败，算法结束
b) 在概念结点 parent(C’)，child(C’)中选择新的结点 C_{new} 使：
Sim (C, C_{new})=Max(*Sim* (C, parent(C’)), *Sim* (C, child(C’))
if C_{new}=parent(C’) **then** C’=parent(C’)，保留向上探测的路径
else if C_{new}=child(C’) **then** C’=child(C’)，保留向下探测的路径
c) if *Sim* (C, C_{new}) ≥γ **then goto step 3**
else repeat step 4
Note：if 概念 C’放弃向上探测的路径或没有父亲结点 **then** *Sim* (C, parent(C’))=0
if 概念 C’放弃向下探测的路径或没有儿子结点 **then** *Sim* (C, child (C’))=0

图 8 改进的相似概念查询算法

为了提高计算效率，我们对相似概念查询算法作了改进，形式化的表述如图 8 所示。

图 9 为对于概念 C 运用相似概念查询的改进算法的示意图。

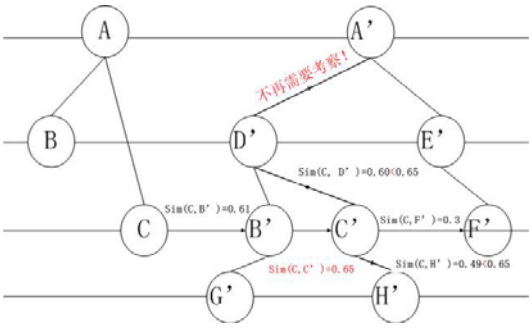


图 9 对概念 C 运用相似概念查询的改进算法

上述改进后的查询算法是利用了形式概念分析本身的特点，一般认为能够体现一个概念的特征的是其内涵部分，所以一旦两个概念之间的内涵偏差较大（在这里可以认为是内涵集合的秩的绝对差值较大），则认为两个概念不太可能具有较大的相似性。这样的改进，提高了计算效率，但也承担了一部分错失相似概念的风险。我们的方法主要是为了帮助开发人员对版本差异进行探测和理解，所以目标设为尽可能地发现准确的相似结构，从中来体会不同版本之间的设计意图差异或者实现方法区别。

利用上述递归算法，我们就可以在概念格上进行松弛匹配，并得到概念格之间所有的相似和相异的概念，以及同构的概念子格对。

3.3 演化类型

我们将模糊概念格中的每个概念作为系统的一个功能单元，而将识别出来的每个子格作为系统的功能模块。在树匹配所得到的概念和概念子格映射关系基础上，我们就可以根据不同版本概念格之间的结构差异识别出一些常见的演化类型，其中既包括实现上的细粒度演化又包括局部设计上的演化。

新增的功能单元（新增的概念）：这类演化往往体现为后一版本概念格中无法在前一版本概念格中找到对应映射的概念，如图 10 所示。

删除的功能单元（丢失的概念）：这类演化往往体现为前一版本概念格中无法在后一版本概念格中找到

对应映射的概念，如图 10 所示。

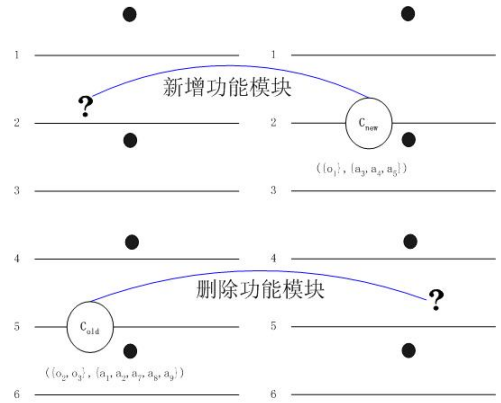


图 10 新增与删除功能单元或模块的图示

修改的功能单元（有变化的概念）。同一概念在两个概念格中都存在，但在内涵和外延上存在部分差异，如图 11 所示。

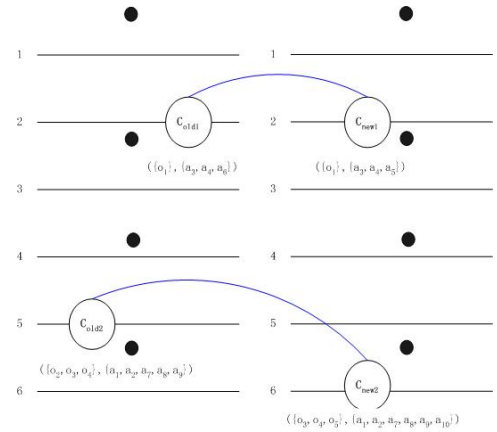


图 11 修改功能单元或模块的图示

稳定的功能模块（没有变化的概念子格）。两个版本中对应的概念子格所包含的概念以及概念之间的关系都能够完全匹配，体现出很强的稳定性，如图 12 所示。

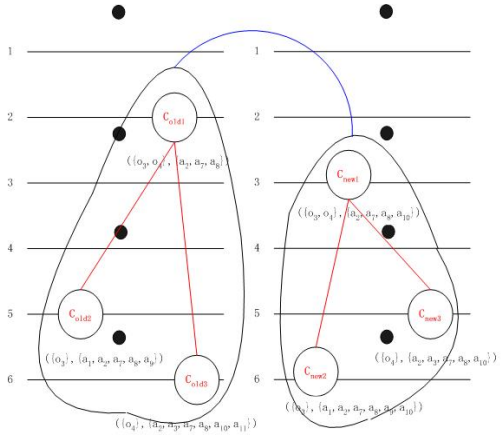


图 12 稳定的功能模块的图示

不稳定的功能模块（有变化的概念子格）。两个版本中对应的概念子格所包含的一部分概念或者概念之间的关系存在差异，表现出局部设计上的演化，如图 13 所示。

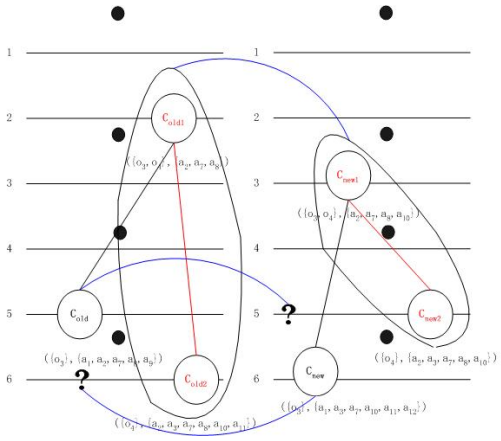


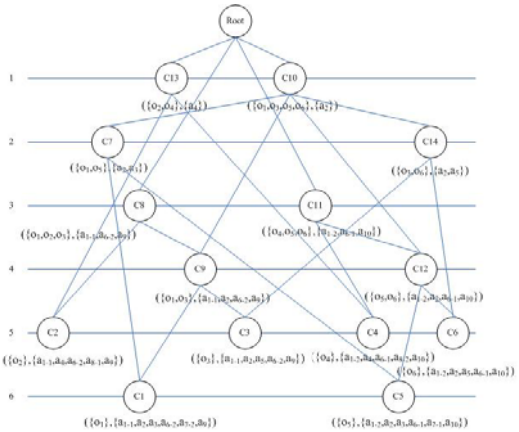
图 13 不稳定的功能模块的图示

4 实验

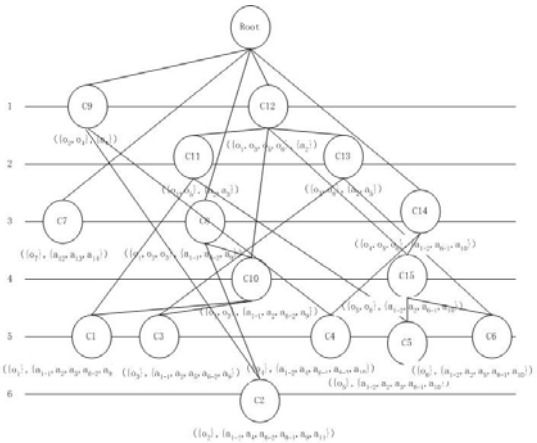
我们将算法应用到一个商用书店管理系统销售模块两个相近版本的源代码文件上。首先利用[11]中所提出的的聚类方法得到两个版本各自的概念格，然后对这两个概念格实施我们在第三部分提出的树匹配算法。

4.1 实验

使用我们所提出的方法，首先得到两个不同版本的模糊概念格，如图 14(a)和图 14(b)所示。两个版本的源代码与对象之间的映射，关键字与属性之间的映射，模糊参数分别列于表 1 至表 6 中。



图(a) 版本 1 的模糊概念格



图(b) 版本 2 的模糊概念格

图 14 两个版本的模糊概念格

表 1 版本 1 的源代码与对象之间的映射

文件名	对象
DealMailOrder.pas	o_1
DealRetailSaleList.pas	o_2
DealWholeSaleList.pas	o_3
ManageInstoreSaleList.pas	o_4
ManageMailOrder.pas	o_5
ManageWholeSaleList.pas	o_6

表 2 版本 1 的关键字与属性之间的映射

关键字	属性
book	a_1
client	a_2
mail	a_3
retail	a_4
wholesale	a_5
salelist	a_6
order	a_7
store	a_8
deal	a_9
manage	a_{10}

表 3 版本 1 的模糊参数列表

概念	模糊参数 σ	模糊参数 λ
C1	$0.8/a_{1\text{-}1}+0.2/a_2+0.8/a_3+0.2/a_{6\text{-}2}+0,2/a_{7\text{-}2}+0.8/a_9$	0
C2	$0.9/a_{1\text{-}1}+0.9/a_4+0.3/a_{6\text{-}2}+0.7/a_{8\text{-}1}+0.8/a_9$	0
C3	$0.9/a_{1\text{-}1}+0.3/a_2+1/a_5+0.3/a_{6\text{-}2}+0.9/a_9$	0
C4	$0.3/a_{1\text{-}2}+0.8/a_4+0.9/a_{6\text{-}1}+0.2/a_{8\text{-}2}+0.8/a_{10}$	
C5	$0.2/a_{1\text{-}2}+0.3/a_2+1/a_3+0.8/a_{6\text{-}1}+0.9/a_{7\text{-}1}+0.9/a_{10}$	0
C6	$0.3/a_{1\text{-}2}+0.3/a_2+0.8/a_5+1/a_{6\text{-}1}+1/a_{10}$	0
C7	$0.3/a_2+0.9/a_3$	0.07
C8	$0.9/a_{1\text{-}1}+0.3/a_{6\text{-}2}+0.9/a_9$	0.05
C9	$0.9/a_{1\text{-}1}+0.3/a_2+0.3/a_{6\text{-}2}+0.9/a_9$	0.07
C10	$0.3/a_2$	0.01
C11	$0.3/a_{1\text{-}2}+0.9/a_{6\text{-}1}+0.9/a_{10}$	0.08
C12	$0.3/a_{1\text{-}2}+0.3/a_2+0.9/a_{6\text{-}1}+1/a_{10}$	0.04
C13	$0.9/a_4$	0.05
C14	$0.3/a_2+0.9/a_5$	0.05

表 4 版本 2 的源代码与对象之间的映射

文件名	对象
DealMailOrder.pas	o_1
DealRetailSaleList.pas	o_2
DealWholeSaleList.pas	o_3
ManageInstoreSaleList.pas	o_4
ManageMailOrder.pas	o_5
ManageWholeSaleList.pas	o_6
SecurityCheck.pas	o_7

表 5 版本 2 的关键字与属性之间的映射

关键字	属性
book	a_1
client	a_2
mail	a_3
retail	a_4
wholesale	a_5
salelist	a_6
order	a_7
store	a_8
deal	a_9
manage	a_{10}
unionpay	a_{11}
clerk	a_{12}
password	a_{13}
log	a_{14}

表 6 版本 2 的模糊参数列表

概念	模糊参数 σ	模糊参数

		λ
C1	$0.9/a_{1-1}+0.2/a_2+0.8/a_3+0.3/a_{6-2}+0.8/a_9$	0
C2	$0.8/a_{1-1}+0.9/a_4+0.3/a_{6-2}+0.7/a_{8-1}+0.7/a_9+1/a_{11}$	0
C3	$0.9/a_{1-1}+0.3/a_2+1/a_5+0.3/a_{6-2}+0.9/a_9$	0
C4	$0.3/a_{1-2}+0.8/a_4+0.9/a_{6-1}+0.2/a_{8-2}+0.8/a_{10}$	0
C5	$0.3/a_{1-2}+0.3/a_2+1/a_3+0.9/a_{6-1}+0.9/a_{10}$	0
C6	$0.3/a_{1-2}+0.3/a_2+0.8/a_5+1/a_{6-1}+1/a_{10}$	0
C7	$1/a_{12}+1/a_{13}+1/a_{14}$	0
C8	$0.9/a_{1-1}+0.4/a_{6-2}+0.9/a_9$	0.04
C9	$0.9/a_4$	0.04
C10	$0.9/a_{1-1}+0.3/a_2+0.4/a_{6-2}+0.9/a_9$	0.06
C11	$0.3/a_2+0.9/a_3$	0.07
C12	$0.3/a_2$	0.01
C13	$0.3/a_2+0.9/a_5$	0.05
C14	$0.3/a_{1-2}+1/a_{6-1}+0.9/a_{10}$	0.08
C15	$0.3/a_{1-2}+0.3/a_2+1/a_{6-1}+1/a_{10}$	0.03

我们对于两个版本的模糊概念格使用松弛树匹配算法，外延考察因子 α 取 0.5，内涵考察因子 β 取 0.5，概念的相似度阈值 γ 取 0.6。得到了相似概念 11 对，相异概念 4 个，以及相似概念子格 6 对。图 15 至图 17 中是算法结果中的一些相似概念对，相异概念以及相似概念子格对。

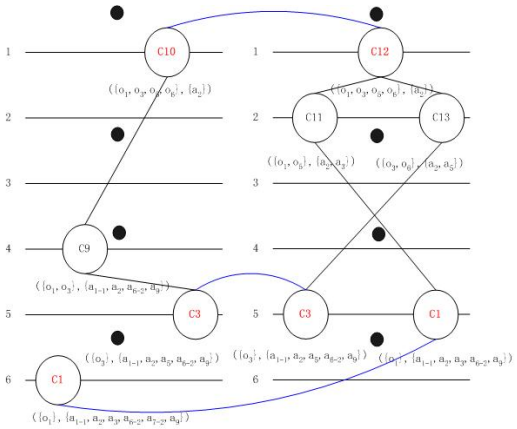


图 15 两个概念格之间的相似概念示例

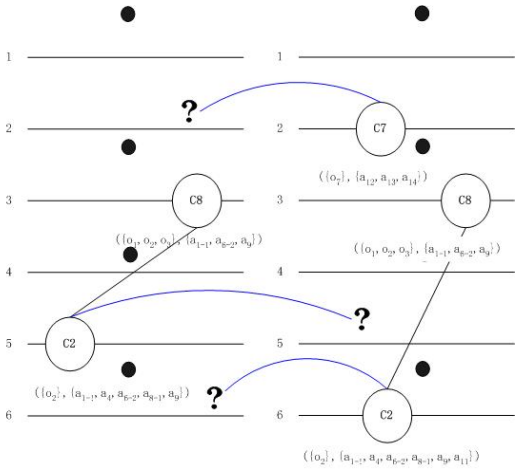


图 16 两个概念格之间的相异概念示例

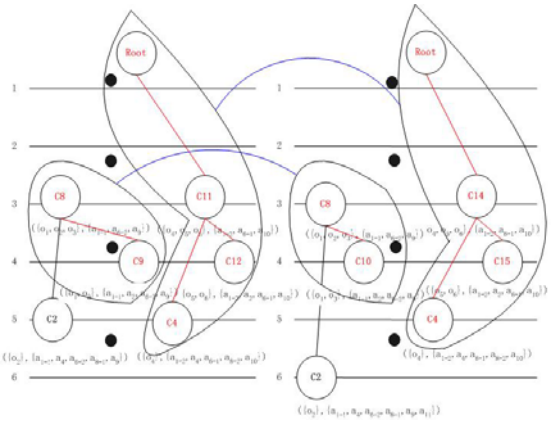


图 17 两个概念格之间的相似概念子格对示例

4.2 分析

我们将实验结果按照 3.3 节的演化类型进行分类统计结果列于表 3 中。我们从表 3 中选择一些演化类型的实例来进行分析。值得说明的是，我们的方法还无法支持对于聚类所得的概念进行智能命名的功能，目前的选择只能是使用概念的内涵以及外延集合以及模糊参数来表达其本身的意义。

表 3 书店系统的演化实例统计

演化类型	数量
新增的功能单元	2
删除的功能单元	1
修改的功能单元	5
稳定的功能模块	4
不稳定的功能模块	2

图 16 中 CL_2 中的概念 C2 ($\{o_2\},\{a_{1-1},a_4,a_{6-2},a_{8-1},a_9,a_{11}\}$) 、C7($\{o_7\},\{a_{12},a_{13},a_{14}\}$)就是新增的功能单元而 CL_1 中的概念 C2 ($\{o_2\},\{a_{1-1},a_4,a_{6-2},a_{8-1},a_9\}$)就是删除的功能单元。两个版本之间的演化差异在于但不仅限于：在后一版本中引入了关于银联支付（a_{11} 为“unionpay”）的支付方式，

而这在前一版本中是没有涉及的，所以该书店的支付手段得到了丰富。如果今后开发人员需要对银联支付方式相关的代码进行修改，就应该到源代码文件 o_2 中去进行处理。

同理 CL_2 中的概念 C7 与 CL_1 中的所有概念的相似度均非常低，通过观察其外延集合就可以发现 o_7 是在后一版本中新引入的关于安全认证的源代码，而其内涵集合中的关键字 a_{12}(“ClerkId”)、a_{13}(“ClerkPassword”)以及 a_{14}(“TransactionLog”)则既是该源代码文件中的变量名也是数据库表所使用的字段名，即安全认证需要职员的身份和密码并对交易进行日志记录。

图 15 为相似概念示例。其中 CL_1 中的概念 C10($\{o_1,o_3,o_5,o_6\},\{a_2\}$) 与 CL_2 中的概念 C12($\{o_1,o_3,o_5,o_6\},\{a_2\}$)的映射代表修改的功能单元，而 CL_1 中的概念 C3($\{o_3\},\{a_{1\text{-}1},a_2,a_5,a_{6\text{-}2},a_9\}$)与 CL_2 中的概念 C3($\{o_3\},\{a_{1\text{-}1},a_2,a_5,a_{6\text{-}2},a_9\}$)的映射代表未修改的功能单元。上述修改的功能单元蕴含了书店的业务信息：即邮购与批发业务与客户信息相关，必须有客户的注册信息才能进行，相应的用户可以在不注册成为客户的情况下就能在门店里通过零售的方式购买图书。而该概念之所以受到修改可能是由于实现细节上的差异。

图 17 中则分别显示了一个稳定的功能模块与一个不稳定的功能模块。CL_1 中的概念子格(Root-C11-C12-C4) 与 CL_2 中的概念子格(Root-C14-C15-C4)互为相似概念子格对，这是一个稳定的功能模块。我们从中获得了两个软件版本之间的需求和设计的不变性：在订单生成(关键字 a_9)模块中总是涉及较多对于图书(关键字 $a_{1\text{-}1}$)而较少对于订单（关键字 $a_{6\text{-}2}$）的处理，而在订单管理(关键字 a_{10})模块中则正好相反，总是涉及更多的订单相关的处理而更少的图书相关的操作；而批发批发模块与门店销售以及邮购销售模块最大的区别则始终保持在前者需要与门店的库存进行关联而后者不受制与该限制。这里对于相关度的模糊表述也正是模糊概念格相对于传统概念格的优势所在。

CL_1 中的概念子格(C8-C9)与 CL_2 中的概念子格(C8-C10)也互为相似概念子格对，这是一个不稳定的功能模块。相比之前一对概念格对，其区别在于 CL_1 中的概念 C8 的一个儿子结点 C2 与 CL_2 中的概念 C8 的一个儿子结点 C2 并非相似概念，从中也揭示了两个软件版本之间的需求和设计变化：除了订单处理模块这个整体设计的不变之外，在门店销售单独加入了有关银联支付的方式，而同为订单处理模块中的另外两种销售模式——批发销售与邮购销售则不提供有关银联支付的支持。

通过在模糊概念格上进行树匹配所得到的结果确实可以帮助我们初步探测和理解软件的版本差异以进行演化分析。

4.3 讨论

首先来看外延考察因子 α 和内涵考察因子 β，我们在本文的方法中将其简单的各设置为了 0.5 是为了淡化概念中究竟外延更多地表现了概念的属性还是内涵更多地表现了概念的属性。事实上，从形式概念分析的角度上来看，往往一个概念的内涵更具有变现其本身特点的能力，也就是说一个概念具有哪些属性，其与其它概念的区别更多地能从内涵上得到直观的体现。所以在实验中，我们可以有意识地调高内涵考察因子 β，调低外延考察因子 α，相应地，相似度阈值 γ 也可以被调高到诸如 0.7 甚至 0.8 的水平来判断两个概念究竟是否相似，阈值越高，结果也越准确，而查全率也越差。我们将在下一步的工作中对此进行研究，观察是否可以探测到更加准确又全面的版本差异。

再来讨论相似概念查询的算法，我们在其中使用了的优化手段是，首先在同层中寻找概念相似度最大的匹配概念，然后再沿着该概念的父子关系顺着概念格进行相似概念的查询，并且在查询的过程中考虑概念相似度的变化趋势，适时地中止算法，提高查询效率。而事实上，在同层中寻找最大的相似度匹配很有可能会是真正的最相似匹配无法被查询到，因为真正的最相似匹配可能是在沿着同层中次大匹配的概念结点的父子关系的子树上的。这里涉及到代价与回报的选择，我们认为在本算法中由于概念格的结构存在相对复杂的可能性，所以提高计算效率是本方法的首要因素，故可以承受某些概念无法找到最佳匹配的损失。

由于代码与设计单元之间通常总是存在某些对应关系的缺失，所以一般的聚类算法所获得的结果缺乏与真实概念或业务需求物理含义之间的对应能力。我们认识到这样的缺陷在聚类算法中本身是不可避免的，所以我们无法保证本文所提出的方法可以在最大程度上得到与软件系统的原业务需求的映射关系，但是应该肯定的是，在所能得到的映射关系中，绝大部分的聚类结果是正确且有意义的，而方法也能够成功地发掘软件生命周期中的一些演化信息，这在上述实验中均得到了验证。

文中所使用的演化分析方法是在前期工作的概念格构造算法的基础上通过树匹配来进行差异分析。由

于在概念格的构造算法中并不区分编程语言的语言特性，只是将所有的源代码文件当作文本文件来进行处理，唯一的编程语言相关性操作只有通过保留字列表来过滤编程语言中的保留字。由此可见，我们的方法是与语言无关的，无论是用过程式的编程语言或者是用面向对象的编程语言开发的软件系统，都可以使用该方法来进行演化分析。

5 相关工作

文本的研究是基于模糊概念格的，也就是从源代码文件以及其中的关键字出发以分析软件演化的，这是一种基于代码的分析方法。也有学者[5]使用更细粒度的程序切片技术来研究代码级别的演化分析，将不同版本之间的源代码按照是否发生变化进行切片分解。这样当演化发生时，需要关注的代码片段均被分解到会产生变化的切片当中。[6]则是从编译角度出发，提出了一种在抽象语法树上进行树形差异比较的算法来抽取不同版本源代码之间的变化，这与我们所使用的树匹配技术比较类似，而区别在于语法抽象树属于低层次的实现制品，而模糊概念格属于高层次的设计制品。

通过高层次的设计制品来进行演化分析自然也包括基于 UML 设计模型的演化分析。[1]中提出了一种启发式的算法来比较面向对象软件的逻辑模型，以发现版本之间的软件设计元素是否有增加、删除、匹配、移动或重命名，以及它们的属性是否产生了变化。还进一步分析了独立的类、聚类的类以及整个系统在演化中的长期趋势，并总结归纳了一些设计变化上的模式，我们也在文本中归纳了一些演化模式。与[1]类似，[2]虽然是以变更影响分析为目的，但其过程中仍然有对软件的两个版本的 UML 模型进行差异探测以发现设计上的改变。

除此之外，还有一类从版本管理和配置信息角度出发的演化分析方法。[3]提出了一种统一化的版本模型以及其对软件配置管理的支持架构，这种模型可以很好地支持版本演化中设计变化的表示。[4]提出与配置管理和过程管理系统相结合的软件变化管理系统结构,并介绍一个实际软件变化管理系统的实现。即使是诸如代码行数、文件个数等十分简单的版本信息，[5]也证明了在研究演化模式中也可以把它们当作必要的度量信息，类似的[11]将关键字的出现频率也当作对象与属性之间模糊关系的度量标准。

本文的研究还使用了松弛的树匹配模型并提出了适用的算法。事实上，所树匹配在很多相关领域中也有不同的应用。[21]中使用基于有序树匹配的方法来对模式查询进行评价，该算法可以方便地使用于映射环境中。[18]中提出了一个 VLSI 架构来进行近似的树匹配，该架构是一种动态编程算法的并行实现，它使用了其中的简单基本块的信息并且需要与最近的邻居进行常规的交流。在更为实际的应用方面，[22]中使用树匹配的方法进行 XML 文件的信息检索，通过其中的算法可以返回与用户的查询树最为相近的一些查询结果。[23]中，作者使用匹配抽象语法树来进行代码克隆侦测，并由此来生成更为结构化的代码，在逆向工程中帮助发现领域概念以及对应的实现。

6 总结与展望

软件系统的演化历史分析能够在很大程度上辅助我们对于系统需求和设计决策的理解。然而在长期的演化过程中，准确、及时的演化文档往往无法获得，因此基于不同版本程序代码分析的演化分析成为了一种重要的辅助手段。本文在前期工作所提出的基于模糊概念格的程序分析方法基础上，提出了一种基于模糊概念格的软件演化分析方法，并将其应用于一个商业软件系统的演化分析。实验表明，模糊形式概念分析能够准确捕捉系统的高层结构，在此基础上实现的演化分析能够较为准确地反映系统的需求及设计演化。在未来的研究工作中，我们将在保证算法效率的前提下，进一步提高概念映射中的查准率与查全率。其次，我们还将继续对软件系统演化的内涵和外延进行深入的研究，从而提出更加精确的分析方法。另一方面，我们还希望在演化分析基础上进一步实现高层结构的共性和可变性分析。

参考文献

[1] Zhenchang Xing. Supporting object-oriented evolutionary development by design evolution analysis [D]. Department of Computing Science, Edmonton, Alberta, 2008

[2] Briand, L.C. Labiche, Y. O'Sullivan, L. Impact analysis and change management of UML models// Proceedings of IEEE International Conference on Software Maintainance. Amsterdam, The Netherlands, 2003: 256-265

[3] Westfechtel, B. Munch, B.P. Conradi, R. A layered architecture for uniform version management.

IEEE Transactions on Software Engineering, 2001, 27(12): 1111-1133

[4] Zhaoqi Chen, Linhui Zhong, Lu Zhang, Bing Xie. Study of software change management system. Journal of Chinese Computer Systems, 2002.1, 23(1): 29-31 (in Chinese)

（陈兆琪，钟林辉，张路，谢冰 软件变化管理系统研究，小型微型计算机系统，2002.1, 23(1): 29-31.

[5] Herraiz, I. Robles, G. Gonzalez-Barahona, J.M. Capiluppi, A. Ramil, J.F. Comparison between slocs and number of files as size metrics for software evolution analysis//Proceedings of the 10th European Conference on Software Maintenance and Reengineering. Bari, Italy, 2006: 213

[6] K. B. Gallagher, J. R. Lyle. Using program slicing in software maintenance. IEEE Transaction on Software Engineering, 1991, 17 (8): 751-761

[7] Fluri, B. Wursch, M. Pinzger, M. Gall, H.C. Change distilling tree differencing for fine-grained source code change extraction. IEEE Transactions on Software Engineering. 2007, 33(11): 725-743

[8] Hong Mei, Feng Chen, Yaodong Feng, Jie Yang. ABC: an architecture based, component oriented approach to software development. Journal of Software. 2003, 14(04): 721-732 (in Chinese)

（梅宏,陈锋,冯耀东,杨杰. ABC:基于体系结构、面向构件的软件开发方法. 软件学报, 2003, 14(04): 721-732）

[9] Hong Mei, JunRong Shen. Progress of research on software architecture. Journal of Software. 2006, 17(6): 1257-1275 (in Chinese)

（梅宏,申峻嵘. 软件体系结构研究进展. 软件学报, 2006, 17(6): 1257-1275）

[10] Linhui Zhong, Bing Xie, Weizhong Shao. Supporting component-based software development by extending the cdl with software configuration information. Journal of Computer Research and Development. 2002, 39(10): 1361-1365 (in Chinese)

（钟林辉、谢冰、邵维忠扩充 CDL 支持基于构件的系统组装与演化. 计算机研究与发展，2002, 39(10): 1361-1365）

[11] Jiaqing Xu, Xin Peng, Wenyun Zhao, Program clustering for comprehension based on fuzzy formal concept analysis. Journal of Computer Research and Development. Accepted (in Chinese)

（许佳卿, 彭鑫, 赵文耘. 一种基于模糊形式概念分析的程序聚类方法. 计算机研究与发展, 已录用）

[12] T.T.Quan, S.C.Hui and T.H.Cao. A fuzzy fca-based approach to conceptual clustering for automatic generation of concept hierarchy on uncertainty data. CLA, 2004: 1-12

[13] A.Marcus and J.I.Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing//Proceedings of 25th International Conference on Software Engineering. Portland, Oregon USA, 2003: 125-135

[14] A.van Deursen and T.Kuipers.Identifying objects using cluster and concept analysis//Proceedings of 21st IEEE International Conference on Software Engineering. Los Angeles, CA, USA, 1999: 246-255

[15] C.Lindig and G.Snelting. Assessing modular structure of legacy code based on mathematical concept analysis//Proceedings of 19th IEEE International Conference on Software Engineering. Boston, Massachusetts, USA, 1997: 349-369

[16] Tyng-Luh Liu Geiger, D. Approximate tree matching and shape similarity//Proceedings of the Seventh IEEE International Conference on Computer Vision. Corfu, Greece, 1999, 1: 456 - 462

[17] Shasha, D. Wang, J.T.-L. Kaizhong Zhang Shih, F.Y. Exact and approximate algorithms for unordered tree matching. IEEE Transactions on Systems, Man and Cybernetics. 1994, 24: 668-678

[18] Sastry, R. Ranganathan, N. A vlsi architecture for approximate tree matching. IEEE Transactions on Computers. 1998, 47: 346 – 352

[19] Laszlo Kovacs, Tibor Rpai, Erika Baksa-Varga. Approximate subtree search for labeled trees with small depth value. Department of information Technology University of Miskolc

[20] Zhong-jie Wang, De-Chen Zhan. A component retrieval method based on feature matching. Research Centre of Intelligent Computing for Enterprises (ICE), School of computer Science and Technology Harbin Institute of Technology. O. Box315, No.92, 2006

[21] Yangjun Chen Cooke, D. Evaluation of twig pattern queries based on ordered tree matching//IEEE International Conference on Signal Image Technology

and Internet Based Systems. Bali, Indonesia, 2008: 24-31
[22] Ben Aouicha, M. Boughanem, M. Tmar, M. Abid, M. XML information retrieval based on tree matching//15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems. Belfast, Northern Ireland, 2008: 499-500
[23] Baxter, I.D. Yahin, A. Moura, L. Sant'Anna, M. Bier, L. Clone detection using abstract syntax trees//Proceedings of International Conference on Software Maintenance. Bethesda, MD, USA, 1998: 368-377

作者简介：

Xu Jiaqing, born in 1984. Master. His main research interest is program comprehension and reverse engineering.
许佳卿，男，1984 年生，复旦大学计算机科学技术学院硕士生，主要研究方向为程序理解及逆向工程。

Xin Peng, born in 1979. Ph.D. and assistant professor. Member of China Computer Federation. His current research interests include Software Product Line, Software Component and Architecture, Software Maintenance and Software Reengineering.
彭鑫，男，1979 年生，博士，CCF 会员，复旦大学计算机科学技术学院讲师。目前主要研究方向包括软件产品线、软件构件与体系结构、软件维护与再工程。

Zhao Wenyun, born in 1964. Professor and Ph.D. supervisor. Senior member of China Computer Federation. His main research interests include Software Reuse, Software Component and Architecture.
赵文耘，男，1964 年生，CCF 高级会员，复旦大学计算机科学技术学院教授、博导。目前主要研究方向包括软件复用、软件构件与体系结构。

Background

This research is sponsored by the National Natural Science Foundation of China under Grant No. 60703092, the National High Technology Research and Development Program (863 Program) under Grant No. 2007AA01Z125, and the Shanghai Leading Academic Discipline Project under Grant No. B114. The purpose of our research is to explore incremental construction of software product line (SPL) platforms through reverse engineering, refactoring and evolution management. The research scopes cover reverse engineering and reengineering, software product line, software maintenance and evolution.

In the research of incremental SPL development and evolution, the key problems include how to analyze the evolution history and evolution trend of software systems, especially for variability evolutions, and how to analyze the commonality/variability among several similar software products through reverse engineering. A straightforward approach for reverse evolution analysis and commonality/variability analysis is to abstract high-level requirement and design models first and then analyze the evolution trend and variations by identifying mappings among the high-level models of different products or product versions.

To this end, in this paper, we propose to further introduce our previous work of program clustering based on fuzzy formal concept analysis into evolution analysis, and present an evolution analysis method based on fuzzy concept lattice and source code analysis. Program clustering in our method is used to abstract a concept lattice as the high-level requirement model for each product version. Based on the recovered concept lattices, the method explores concept mappings between different concept lattices by using a relaxation tree matching algorithm, and then identifies various evolution types through structural difference analysis. Currently, our method is used for evolution analysis. In our future work, we will further extend the method for reverse variability analysis for multiple similar products in the same domain.