软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(8):2428−2452 [doi: 10.13328/j.cnki.jos.005511]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向模式软件体系结构合成中的冲突消解方法*

徐永睿[1,2], 梁 鹏[1,2]

[1](武汉大学 计算机学院,湖北 武汉 430072)

[2](软件工程国家重点实验室(武汉大学),湖北 武汉 430072)

通讯作者: 梁鹏, E-mail: liangp@whu.edu.cn

摘 要: 面向模式的软件体系结构合成主要包括两个核心活动:(1) 将软件职责分配到对象类的职责合成活动;(2) 减少体系结构模式约束违背的模式合成活动.但如何从以上两个核心活动生成的候选方案中无冲突地组合出最终的软件体系结构设计方案,是面向模式的软件体系结构合成所面临的挑战.以基于搜索的软件工程技术为框架,提出了基于学习的协作式协同演化方法(CoEA-L),以解决自动化软件体系结构合成中面临的上述问题.CoEA-L 使用学习运算子扩展了传统遗传算法中的运算子.在学习运算子中,使用数据挖掘的关联算法自动发现软件职责间的关系,并用于解决面向模式的软件体系结构合成中的冲突问题.实验结果表明,该方法能够有效地消解面向模式的软件体系结构合成中产生的冲突.

关键词: 面向模式的软件体系结构合成;冲突消解;协作式协同演化;基于搜索的软件工程;数据挖掘

中图法分类号: TP311

中文引用格式: 徐永睿,梁鹏.面向模式软件体系结构合成中的冲突消解方法.软件学报,2019,30(8):2428−2452. http://www.jos.org.cn/1000-9825/5511.htm

英文引用格式: Xu YR, Liang P. Conflict resolution approach in pattern-oriented software architectural synthesis. Ruan Jian Xue Bao/Journal of Software, 2019,30(8):2428−2452 (in Chinese). http://www.jos.org.cn/1000-9825/5511.htm

Conflict Resolution Approach in Pattern-oriented Software Architectural Synthesis

XU Yong-Rui[1,2], LIANG Peng[1,2]

[1](School of Computer Science, Wuhan University, Wuhan 430072, China)

[2](State Key Laboratory of Software Engineering (Wuhan University), Wuhan 430072, China)

Abstract: There are two core activities in pattern-oriented software architectural synthesis (AS): responsibility synthesis which attempts to assign responsibilities to classes, and pattern synthesis which tries to prevent violations of pattern constraints. One of the major challenges of providing automated support for architectural synthesis is how to compose a final architectural solution from generated solutions of the two activities without inconsistencies. In this study, a learning based cooperative co-evolution approach (CoEA-L) is proposed for automated AS by leveraging search-based software engineering (SBSE) techniques. CoEA-L extends the traditional genetic operator of the genetic algorithm with a learning operator, and employs an association algorithm from data mining in the learning operator to discover the relations between responsibilities. The relations are further used to address the inconsistency issues during pattern-oriented AS. The experiment results show the effectiveness of learning for addressing the inconsistency issues during automated pattern-oriented architectural synthesis.

Key words: pattern-oriented architectural synthesis; conflict resolution; cooperative co-evolution; search-based software engineering; data mining

* 基金项目: 国家自然科学基金(61472286)
Foundation item: National Natural Science Foundation of China (61472286)
收稿时间: 2016-02-26; 修改时间: 2017-04-21; 采用时间: 2017-10-31; jos 在线出版时间: 2019-03-27
CNKI 网络优先出版: 2019-03-27 16:39:55, http://kns.cnki.net/kcms/detail/11.2560.TP.20190327.1639.001.html

软件体系结构作为控制软件复杂性、提高软件系统质量的重要手段,日益受到软件研究者和实践者的关注,并发展成为软件工程的一个重要的研究领域[1].在软件体系结构设计中,软件体系结构合成活动通过将软件体系结构设计相关需求转换成软件体系结构设计方案,连接了问题空间和对应的解空间,是软件体系结构设计中的核心活动[2].但是在传统的软件体系结构合成过程中,如何合成软件体系结构解决方案,极大地依赖于架构师的设计经验.为了在软件体系结构合成中减少对架构师经验的依赖,同时复用设计知识,软件体系结构模式被广泛地使用(如 MVC、管道过滤器、分层模式等)[3].因此,面向模式的软件体系结构合成应运而生[4].尽管如此,软件体系结构模式的使用存在着一些困难和挑战:1) 如何将候选模式中的抽象设计元素映射到软件体系结构设计的具体实现;2) 架构师设计经验的缺乏会导致在采用特定模式进行软件体系结构设计时,违背该模式的设计约束.另一方面,如何将软件职责(即面向对象分析得到的行为和属性)分配到类,也是软件体系结构合成中的一个难点问题.目前已有大量研究关注于解决软件职责的分配问题[5,6],但这些研究都是独立地考虑软件职责的分配,而没有考虑职责分配对软件体系结构设计质量的影响(相同的职责分配在采用不同的模式时,系统质量需求的满足可能完全不同).因此,面向模式的软件体系结构合成必须同时考虑软件职责的分配以及系统质量需求的满足.

在我们之前的工作中[7],提出了基于协作式协同演化的方法(CoEA)来自动化合成面向模式的软件体系结构候选设计方案.在我们提出的方法中,主要包含两个自动化活动:职责合成(responsibility synthesis,简称 RS)和模式合成(pattern synthesis,简称 PS).职责合成负责处理软件职责到类的分配,而模式合成关注于软件体系结构层面的模式实现.我们提出的自动化合成方法能够生成接近专家设计方案的体系结构设计方案[7],但是文献[7]中提出的方法存在一定局限:在模式合成生成的方案 *ps* 和职责合成生成的方案 *rs* 进行组合,生成候选软件体系结构方案时,*ps* 和 *rs* 可能存在冲突.这里,我们通过一个简单的例子来说明该冲突问题及其影响.

假设对包含 5 个软件职责(R_1 到 R_5)的设计问题进行软件体系结构合成.当采用面向模式的软件体系结构合成时,假设架构师选择分层模式来满足该设计问题的质量需求,那么一个可能的职责合成方案 *rs* 如图 1(a)所示.在该职责合成方案中,5 个软件职责被分配到 2 个类中.而一个可能的模式合成方案 *ps* 如图 1(b)所示.在该模式合成方案中,5 个职责被分配到分层模式的 2 个层中.

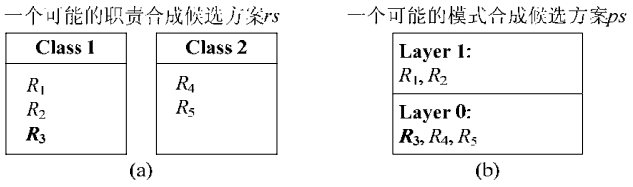


Fig.1 Possible design solutions of RS and PS

图 1 可能的职责合成和模式合成设计方案

当架构师需要组合职责合成的设计方案 *rs*(如图 1(a)所示)和模式合成的设计方案 *ps*(如图 1(b)所示),从而生成软件体系结构设计方案时,必须确保 *rs* 和 *ps* 方案之间不存在冲突.在职责合成设计方案(如图 1(a)所示)中,R_3 与 R_1,R_2 被分配到同一个类中,但在模式合成设计方案(如图 1(b)所示)中,R_3 与 R_1,R_2 却被分配到分层模式的不同层中.因为同一个类中的软件职责应该被分配到分层模式的同一层中,所以架构师无法直接利用图 1(a)和图 1(b)的设计方案,组合生成最终面向模式的软件体系结构设计方案.如果架构师需要利用 *rs* 和 *ps* 方案来生成体系结构设计方案,则要对两者之间的冲突进行消解.例如在该例子中,架构师可以调整职责合成的设计方案 *rs*,将 R_3 分配到 Class 2 中;或者调整模式合成的设计方案 *ps*,将 R_3 分配到 Layer 1 中.

通过上述包含 5 个软件职责的简单例子,我们直观地介绍了面向模式的体系结构合成中的冲突问题.在实际设计问题中,职责合成需要将大量软件职责分配到许多类中,而模式合成则需要将大量软件职责分配到某一模式的不同角色中(例如分层模式的不同层).这导致了 *rs* 和 *ps* 方案在进行组合生成最终的面向模式的软件体系结构设计方案时可能包含大量冲突.因此,对于面向模式的软件体系结构合成,自动化方法不仅需要自动生成

rs 和 *ps* 方案,同时必须具备自动消解 *rs* 和 *ps* 方案间冲突的能力.在我们之前的工作中[7],采用协作式协同演化的方法较好地解决了 *rs* 和 *ps* 方案的生成问题.但当组合 *rs* 和 *ps* 方案生成最终的体系结构设计方案时,文献[7]中提出的方法不能对冲突进行自动化消解,而只能组合那些没有冲突的 *rs* 和 *ps* 方案来生成软件体系结构设计方案.该策略虽然能避免设计冲突的产生,但缺陷是限制了软件体系结构设计方案的多样性,而这也是采用协作式协同演化方法所面临的一个常见困难与挑战[8,9].因此,本文的工作尝试解决面向模式的软件体系结构合成中的冲突自动化消解问题.

为了解决面向模式软件体系结构合成中的冲突自动化消解问题,本文提出了基于学习的协作式协同演化软件体系结构合成方法(CoEA-L),用于消解职责合成和模式合成产生的设计方案组合时的潜在冲突.在本文中,我们基于鲍德温进化理论[10],提出了学习运算子的概念,扩展了传统遗传算法(genetic algorithm,简称 GA)中的选择、突变、交叉运算子.通过在学习运算子中采用一种关联算法来发现职责合成种群和模式合成种群中个体间频繁同时出现的软件职责,进而利用提取到的知识消解 *ps* 和 *rs* 方案组合过程中产生的冲突.

本文的主要贡献如下:在面向模式的软件体系结构合成问题中,通过在搜索过程中使用关联算法挖掘软件职责间的关系,引入了职责合成与模式合成间的学习效应作为解决方案空间的启发式搜索方法,进而消解职责合成和模式合成产生的设计方案间的冲突.同时,不同于现有基于搜索的软件工程(SBSE)研究试图通过改进对问题的表达,定义更好的适应度函数,或在搜索过程中采用更好的启发式规则来提升候选解决方案的质量[11],本文提出的方法利用个体间的关系来改进候选方案的设计质量.个体通过学习其他种群中个体的特征,从而提升个体设计解决方案的质量.

本文第 1 节介绍本文工作的相关研究背景.第 2 节提出面向模式的软件体系结构合成中的冲突消解方法.第 3 节说明实验的设计.第 4 节给出实验结果和分析.第 5 节介绍相关工作.第 6 节总结全文及下一步工作.

1 研究背景

针对面向模式的软件体系结构候选设计方案的自动化合成,本节介绍了我们之前的工作[7]中提出的 CoEA 方法.

在我们之前的工作[7]中,CoEA 被用于面向模式软件体系结构候选方案的合成.如图 2 所示,CoEA 主要包含 7 个步骤.

1) 初始化种群(如图 2 的步骤 1 所示).

在 CoEA 方法中,我们使用两个独立的种群(职责合成种群和模式合成种群)分别对应面向模式软件体系结构合成活动中的 RS 和 PS.因此,图 2 中 *m* 取值为 2.职责合成种群中的每个个体即为一个候选的 *rs* 方案;同理,模式合成种群中的每个个体即为一个候选的 *ps* 方案.对于任意的 *rs*(RS 种群表现型),我们使用整数向量(RS 种群基因型)对其进行编码.整数向量的长度取决于软件系统中软件职责的数目,而向量中的每一位表示某个具体的单一职责,该位的取值即为该职责所被分配到的类.如果向量中不同的位具有相同的取值,则说明这些位所对应的软件职责被分配到了同一个类中.类似地,对于任意的 *ps*(PS 种群表现型),我们也使用整数向量(PS 种群基因型)对其进行编码.该整数向量的长度仍然取决于软件系统中软件职责的数目,但不同的是,向量中每一位的取值表示的是在所选取的模式中,该职责所对应的模式角色[4].模式角色独立于具体的软件体系结构候选方案合成问题,而只与选取的模式相关.在我们之前的工作[12]中,研究了包括分层模式在内的常见软件体系结构模式的模式角色及模式角色之间的关系.在图 3(a)中,我们给出了图 1(a)中的职责合成候选方案 *rs* 的表示实例.在该例子中,由于系统包含 5 个软件职责,所以候选方案所对应的向量长度为 5,而每一位的取值,则表示了对应的软件职责所被分配到的具体类的编号.同理,由于分层模式中基本的模式角色为层[12],图 1(b)中的模式合成候选方案 *ps* 的表示如图 3(b)所示.

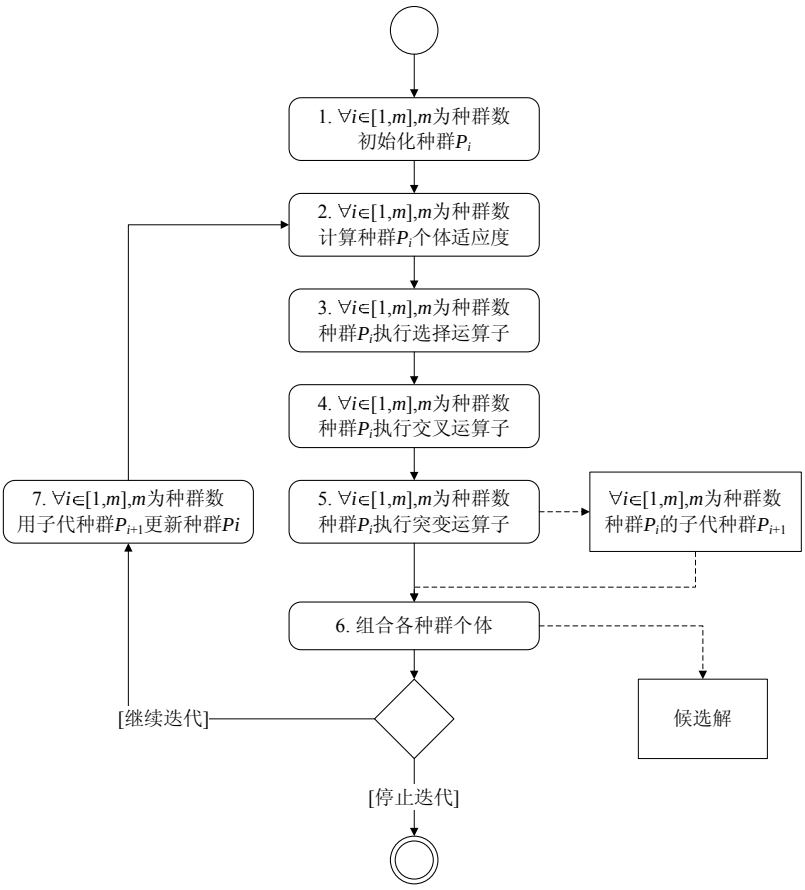


Fig.2 CoEA for automated pattern-oriented architectural synthesis

图 2 面向模式软件体系结构 CoEA 自动化合成方法

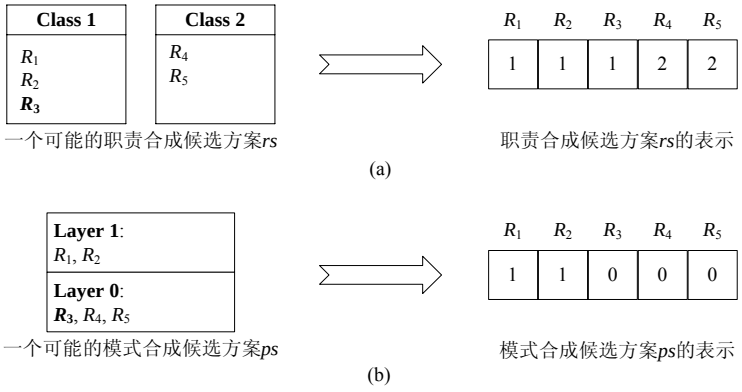


Fig.3 Representations of individuals in RS and PS populations

图 3 职责和模式合成种群个体的表示实例

当采用整数向量对两个种群候选解决方案进行编码表示时,需要处理以下问题.

(1) 整数向量每一位的取值范围.在职责合成中,向量每一位的取值范围为[1,*R*],其中,*R* 表示软件职责的个数.在模式合成中,向量每一位的取值范围为[1,*P*],其中,*P* 表示所选取的模式中包含的不同模式角

色的数目.在分层模式中,系统可能存在的最大层数等同于软件职责的个数(在该极端情况下,每个软件职责被分配到单独一层),因此 $P=R$.

(2) 相同候选方案可能具有不同的整数向量表示,即相同种群表现型对应不同的种群基因型.这里以图3(a)中的职责合成候选方案 *rs* 为例.*rs* 作为一个候选的 RS 种群表现型,其可能的一种基因型如图 3(a)所示为[11122],而不同的基因型[11133][22211][33355][55533]也可以表示 *rs*.针对该问题,我们将每一个基因型进行额外的转换.在转换中,通过逐个扫描原始基因型的每一位,令第 1 个扫描到的整数值为 *i*,将所有取值为 *i* 的向量位的值替换为 1;同理,令第 2 个扫描到的整数值为 *j*,将所有取值为 *j* 的向量位的值替换为 2.以此类推.通过该方法的转换,基因型[11133][22211][33355][55533]都将被转换为[11122],从而保证两个种群中表现型与基因型之间的一一对应关系.

2) 计算种群适应度(如图 2 的步骤 2 所示).

在 CoEA 方法中,职责合成种群和模式合成种群分别采用不同的适应度函数计算种群内个体的适应度.

针对职责合成,目前已有的职责合成相关研究文献[5,6]均采用耦合[13]、内聚[14]、复杂度[6]等指标对候选方案进行度量.根据文献[15]中提出的系统独立性原则,职责合成的适应度函数如公式(1)所示.

$$\text{适应度}_{RS}=\text{内聚}-\text{耦合}-\text{复杂度} \tag{1}$$

一个职责合成候选方案如果拥有较高的职责合成适应度值,则意味着该方案拥有较高的设计质量.每一指标具体使用的度量见表 1.

Table 1 Definitions of the metrics for RS

表 1 职责合成相关度量定义

度量	描述	定义	属性
CBO(c)[16]	针对每一个类 *c*,统计与它耦合的类的数目	$\frac{\lvert\{(m,a)\in U \mid \forall c\in C((m\notin c)\vee(a\notin c))\}\rvert}{\lvert U\rvert}$	耦合
TCC(c)[17]	针对每一个类 *c*,统计使用相同属性的方法对的数目	$\frac{\lvert\{(m_1,m_2) \mid m_1,m_2\in M(c)\wedge m_1\neq m_2\wedge cau(m_1,m_2)\}\rvert}{k(k-1)/2}$	内聚
LCOM5(c)[14]	针对每一个类 *c*,统计类的内聚缺失程度.因此,较低的值意味着类 *c* 具有较高的内聚性	$\frac{k-\frac{1}{l}\sum_{a\in A(c)}\lvert\{m \mid m\in M(c)\wedge a\in I_m\}\rvert}{k-1}$	内聚
CSC[6]	任意两个类的规模差异与系统总职责数目的比值.该度量用于控制类的规模	$\frac{\sum_{c_i,c_j\in C}\left\lvert\frac{\lvert c_i\rvert-\lvert c_j\rvert}{\lvert R\rvert}\right\rvert}{\lvert C\rvert\lvert C-1\rvert/2}$	复杂度

在一个包含|*R*|个职责的职责合成解决方案中,*C* 是类的集合,其中,*c* 是 *C* 中的一个类,即 $c\in C$.令 *m* 和 *a* 表示任意的方法和属性,则 *M*(*c*)表示类 *c* 中包含的所有 *k* 个方法,而 *A*(*c*)是类 *c* 中包含的所有 *l* 个属性.所有职责间的依赖的集合由 *U* 表示.I_i 表示方法 *i* 所引用的所有属性.如果方法 m_1 和 m_2 同时引用了任意类 *c* 中的某个属性,则 $cau(m_1,m_2)$值为 1,否则为 0

另一方面,目前仅有较少文献关注于软件模式合成的自动化度量[18].文献[18]也仅研究了分层模式的自动化度量.针对模式合成的自动化度量,在我们之前的工作[12]中,通过研究包括分层模式在内的常见软件体系结构模式的模式角色、模式约束以及模式角色之间的关系,提出了一个通用的模式度量的定义过程.利用该模式度量定义过程,针对某个特定的模式,我们可以定义与该模式相关的度量,并利用模式的约束设置每个度量的权值 *w*.因此,模式合成的适应度函数如公式(2)所示.

$$\text{适应度}_{PS}=\sum_{k=1}^{n} w_k\text{度量}_k \tag{2}$$

该适应度函数利用模式相关的度量计算出一个模式合成候选方案的模式约束违背代价.一个模式合成候选方案如果拥有较低的模式合成适应度值,则意味着该方案拥有较高的模式实现质量.

由于软件体系结构可以被视为一系列设计规则空间的集合[19],而每个空间中的设计元素(例如模块、包、类等)以层次化的关系进行交互[19],因此,分层模式作为使用频率最高的软件体系结构模式,适用于绝大多数系统的软件体系结构设计.表 2 列出了利用文献[12]中模式度量定义过程定义的分层模式相关的度量.

Table 2 Definitions of the metrics of the layer pattern for PS

表 2 分层模式合成相关度量的定义

度量	描述	定义	属性
IndexOfUse(*i*,*j*)	针对每一层 *i*,统计层 *i* 对其相邻低层 *j*(即 *j*=*i*−1)的依赖数目	$\|\{(m,a)\in U\|m\in L(i)\wedge a\in L(j)\}\vee\{(m_1,m_2)\in U\|m_1\neq m_2\wedge m_1\in L(i)\wedge m_2\in L(j)\}\|, j=i-1$	相邻层依赖
SkipUse(*i*,*j*)	针对每一层 *i*,统计层 *i* 对其非相邻低层 *j*(即 *j*<*i*−1)的依赖数目	$\|\{(m,a)\in U\|m\in L(i)\wedge a\in L(j)\}\vee\{(m_1,m_2)\in U\|m_1\neq m_2\wedge m_1\in L(i)\wedge m_2\in L(j)\}\|, j<i-1$	跨层依赖
BackUse(*i*,*j*)	针对每一层 *i*,统计层 *i* 对其高层 *j*(即 *i*<*j*)的依赖数目	$\|\{(m,a)\in U\|m\in L(i)\wedge a\in L(j)\}\vee\{(m_1,m_2)\in U\|m_1\neq m_2\wedge m_1\in L(i)\wedge m_2\in L(j)\}\|, j>i$	反向依赖
IntraUse(*i*)	针对每一层 *i*,统计层 *i* 对其同层 *j*(即 *i*=*j*)的依赖数目	$\|\{(m,a)\in U\|m\in L(i)\wedge a\in L(j)\}\vee\{(m_1,m_2)\in U\|m_1\neq m_2\wedge m_1\in L(i)\wedge m_2\in L(j)\}\|, j=i$	同层依赖
在一个包含|*R*|个职责的模式合成解决方案中,*U* 表示职责间依赖的集合,*L*(*i*)表示包含于层 *i* 中的所有职责.*m* 和 *a* 表示任意的方法和属性			

在使用分层模式进行模式合成时,根据分层模式的模式约束,需要尽可能地减少候选方案中存在的跨层依赖和反向依赖.因此,在定义关于分层模式的适应度函数时,需要对跨层依赖和反向依赖度量设置较高的权值(即跨层依赖和反向依赖会对模式的实现方案的质量产生较大的影响),同时对同层依赖和相邻层依赖度量设置较低的权值.我们通过使用在决策分析领域常用于权重设置的层次分析法[20],设置分层模式中的上述度量的权值分别为 0.145,0.277,0.508,0.07.因此,分层模式的适应度函数定义如公式(3)所示.

$$\text{适应度}_{ps}=\left.\begin{aligned}&0.145\sum_{i=1}^{n}IndexOfUse(i,i-1)+0.277\sum_{i=1}^{n}\sum_{j=1}^{i-2}SkipUse(i,j)+\\&0.508\sum_{j=1}^{n}\sum_{i=1}^{j-1}BackUse(i,j)+0.07\sum_{i=1}^{n}IntraUse(i)\end{aligned}\right\}\tag{3}$$

其中,*i*,*j* 为分层模式中的任一层(*i*≠*j*),*n* 为候选方案中包含的总层数.同理,我们可以根据公式(2)及文献[12]中的模式度量定义过程,定义出其他软件体系结构模式的适应度函数.

3) 执行选择运算子(如图 2 的步骤 3 所示).

在 CoEA 方法中,职责合成种群和模式合成种群选择运算子均使用锦标赛选择方法来选择个体.相对于其他选择方法,锦标赛选择更加高效且易于并行实现[21,22].在锦标赛选择方法中,随机选择种群中的任意两个个体,并保留适应度较优的个体作为父代中的个体.重复该操作,直到父代中个体的数目等于种群数目.

4) 执行交叉运算子(如图 2 的步骤 4 所示).

在 CoEA 方法中,职责合成种群和模式合成种群交叉运算子均使用单点交叉方法[22]从父代个体生成子代个体.在单点交叉方法中,父代个体的整数向量编码随机产生一个交叉位置,并根据该交叉位置交换配对的父代个体的整数向量表示,来生成子代个体.

5) 执行突变运算子(如图 2 的步骤 5 所示).

在 CoEA 方法中,职责合成种群和模式合成种群突变运算子均使用交换突变方法让子代个体产生变异.在交换突变方法中,个体的整数向量编码随机产生一个突变位置,并基于概率确定该位置是否需要突变.当突变发生时,随机修改个体整数向量该位置的值.对于职责合成种群,交换突变意味着将候选方案 *rs* 的某一职责交换到不同的类中;对于模式合成种群,交换突变意味着将候选方案 *ps* 的某一职责交换到不同的模式角色中.当突变运算子执行完成后,职责合成和模式合成种群产生下一代个体.

6) 组合各种群个体(如图 2 的步骤 6 所示).

在 CoEA 方法中,通过组合职责合成和模式合成种群个体,产生面向模式软件体系结构的候选方案.需要强调的是,在 CoEA 方法中,由于不存在有效的冲突消解机制,因此只能组合不包含冲突的 *rs* 和 *ps* 来产生体系结构设计方案.

7) 更新种群(如图 2 的步骤 7 所示).

当需要继续迭代时,针对职责合成和模式合成种群,CoEA 用子代种群更新当前种群,从而跳转到图 2 的步骤 2 开始新一轮迭代.

2 协作式协同演化软件体系结构合成中的冲突消解

本节首先介绍了基于学习的协作式协同演化软件体系结构合成方法(见第 2.1 节),然后说明了如何使用该方法自动化消解面向模式的软件体系结构合成中产生的冲突(见第 2.2 节).

2.1 基于学习的协作式协同演化体系结构合成方法

在进化生物学中,生命体的特征会随着生命体与环境的交互而发生改变.鲍德温在文献[10]中研究了生命体具备的学习能力,提出了一种特别的生命体进化机制.在该机制中,后代具备从环境中获得新特征的学习能力,而不仅仅是直接依赖于继承遗传编码所获得的对应特征.

相对于单种群进化,鲍德温效应对于协同演化计算模型的意义更加突出:在协同演化计算中,其他种群扮演了某一特定种群的外部环境,每个种群内部的个体能够从外部环境中学习从而改变自身特征.在我们之前提出的方法中[7],职责和模式种群采用的遗传算法[22]基于达尔文进化理论,个体特征依赖于从父代继承的遗传编码,而无法从另一种群中学习.种群间交互的缺失,导致了协同演化软件体系结构合成中的冲突问题.因此,基于鲍德温效应,针对面向模式的软件体系结构自动化合成问题,我们提出了 CoEA-L 方法,从而扩展了 CoEA 方法.CoEA-L 方法的主要步骤如图 4 所示.

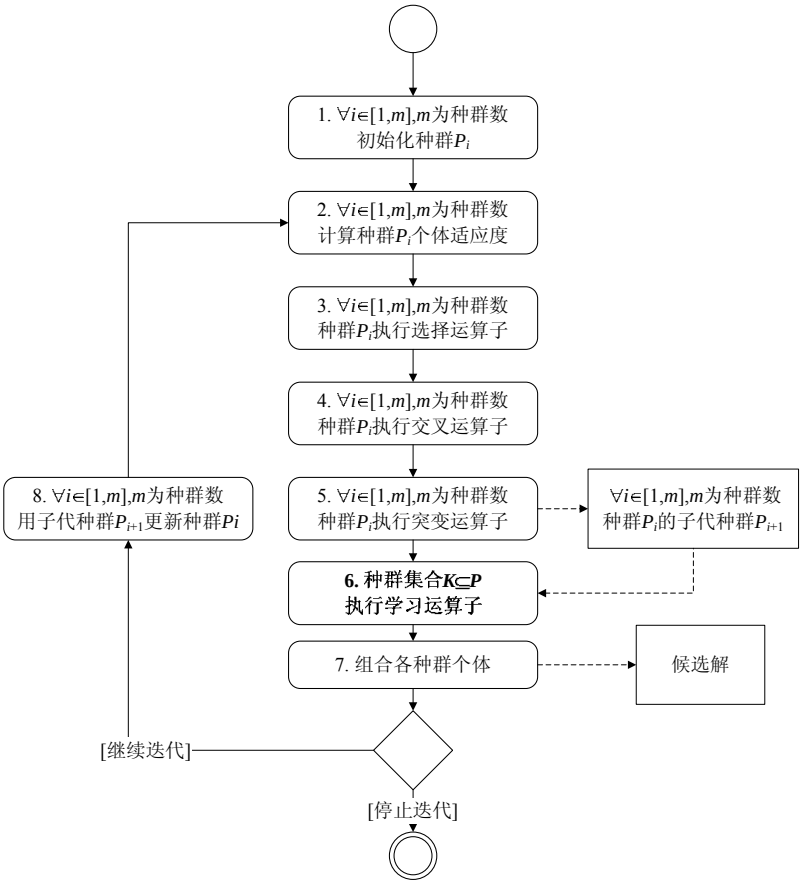


Fig.4 Proposed approach for automated pattern-oriented architectural synthesis

图 4 本文提出的面向模式软件体系结构自动化合成方法

相对于 CoEA,在 CoEA-L 中,我们根据鲍德温效应提出了新的学习运算子(如图 4 的步骤 6 所示),从而扩展了遗传算法的选择、交叉和突变运算子.作为 CoEA 和 CoEA-L 方法的唯一区别,学习运算子的提出,为职责合

成和模式合成种群个体引入了学习机制.当突变操作完成后,生成的子代可以从外部环境(其他种群)中学习,自适应地对种群个体的特征进行调整.当职责合成种群和模式合成种群中的个体进行组合,生成面向模式的候选软件体系结构设计方案时,如果检测到冲突,个体可以利用从其他种群中的个体学习到的知识对自身进行自适应调整,从而完成冲突消解.需要注意的是,当进行冲突消解时,只需要任意1个或多个种群(即种群集合P的子集K)在学习运算子中进行消解即可.在本文中,由于 CoEA-L 方法只需要使用两个种群,我们假设由模式种群来进行冲突消解.

2.2 基于学习的协作式协同演化体系结构合成方法中学习运算子的实现

如图 5 所示,为了消解面向模式的软件体系结构合成中,职责种群个体和模式种群个体组合生成候选体系结构方案时的冲突.学习运算子主要由 4 个步骤组成.

1) 提取种群信息数据集;
2) 生成种群频繁项集;
3) 生成学习规则集;
4) 对组合的种群个体应用学习规则消解冲突.

这 4 个步骤以顺序工作流的形式实现了图 4 中的学习运算子(即图 4 中的步骤 6),因此在后文中,我们以步骤 6.1~步骤 6.4 来标示学习运算子中的每个具体步骤.本小节的剩余部分将详细介绍如何利用这 4 个步骤来消解冲突(即第 2.2.1 节~第 2.2.4 节),并给出一个冲突消解实例(见第 2.2.5 节).

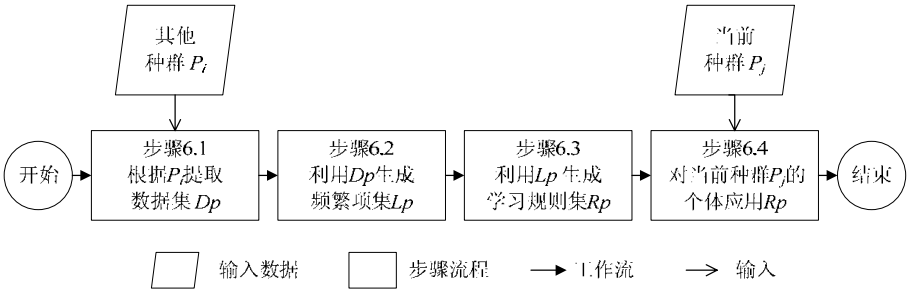


Fig.5 Implementation of the learning operator

图 5 学习运算子的实现

2.2.1 提取种群信息数据集(步骤 6.1)

对于职责种群个体,首先需要从模式种群个体中提取出模式种群的数据信息.通过后续步骤将提取出的信息进一步处理,提供了职责种群个体在与软件体系结构模式种群个体进行组合时消解冲突的必要信息.同理,对于模式种群个体,我们也需要类似地从职责种群个体中提取出职责种群的数据信息.由于我们假设由模式种群进行冲突消解,步骤 6.1 需要从职责种群个体中提取出职责种群的数据信息,再将提取到的信息传递给模式种群用于后续的冲突消解.职责种群提取出的数据信息包含职责种群所有个体中职责到类的分配信息.当职责种群的种群信息数据集被提取后,可以在步骤 6.2 中进一步生成职责种群的频繁项集.种群信息数据集提取操作的时间复杂度为 $O(n)$.

2.2.2 生成种群频繁项集(步骤 6.2)

在关联数据挖掘中,频繁项集表示那些在数据集中出现次数大于等于用户定义的阈值的项集[23].频繁项集意味着项集里包含的元素有较高的概率在不同的记录中同时出现.在一个包含 n 个软件职责的面向模式软件体系结构合成中,项集表示软件职责的任意组合,因此所有可能的不同项集数为 2^n–1,而频繁项集除了需要满足关联数据挖掘中的定义外,还需要保证该项集中包含的元素数目大于 1(即由单一软件职责构成的项集不能算作频繁项集).在我们提出的方法中,职责种群信息数据集中的频繁项集意味着该项集所包含的软件职责在职责种群的不同个体中,拥有较高的概率被分配到相同的类.类似地,模式种群信息数据集中的频繁项集意味着该项

集所包含的软件职责在模式种群的不同个体中,有较高的概率被分配到相同的模式角色.

在步骤 6.2 中,我们使用了数据挖掘中的 Apriori 关联规则挖掘算法[23]从步骤 6.1 提取出的职责种群信息数据集中生成职责种群的频繁项集.提取出的职责种群频繁项集将在步骤 6.3 中生成学习规则集,便于模式种群个体在组合生成软件体系结构设计方案时学习以消解冲突.需要说明的是,Apriori 算法主要的连接和剪枝操作具有较高的性能开销,文献[23]中对这些操作进行了详细的性能分析评估.

2.2.3 生成学习规则集(步骤 6.3)

在步骤 6.3 中,需要将提取出的职责种群信息数据集中的频繁项集转化成可以用于冲突消解的学习规则集.在频繁项集中,我们只知道哪些软件职责具有较高的概率被分配到同一个类,因此,频繁项集中软件职责之间的关系是相关关系.而在学习规则集中,必须把软件职责的相关关系转换成软件职责间的因果关系,才能被用于后续的冲突消解.图 6 给出了本文提出的学习规则集生成算法.

输入:L_p:步骤 6.2 生成的职责种群频繁项集.
输出:R_p:职责种群学习规则集.

```
01  for frequent itemset fi∈Lp
02      S←{s|s⊆fi∧s≠∅}
03      for set s∈S
04          Rule r←"s⇒(fi−s)"
05          if (cos←cosine(s,fi−s))>0.65
06              conf←confidence(s⇒(fi−s))
07              Rp←Rp∪{r,conf,cos}
08  for Rule ri∈Rp
09      R←{r|r∈Rp∧r⊆ri∧r.cos⩾ri.cos∧r.conf⩾ri.conf}
10      if R≠∅
11          Rp←Rp−{ri,ri.conf,ri.cos}
12  return Rp
```

Fig.6 Proposed algorithm for generating the rule set

图 6 本文提出的学习规则集生成算法

在该算法中,针对职责种群频繁项集中的每一个频繁项 *fi*,生成该频繁项的所有非空子集构成集合 *S*,如算法第 2 行所示.在第 4 行中,对 *S* 集合中的每一元素 *s*,生成形如"*s*⇒(*fi*−*s*)"的学习规则.*fi*−*s* 构成一个软件职责集合,包括那些在频繁集 *fi* 中但不在 *s* 中的软件职责.在算法第 4 行中,大量的学习规则会被生成,因此,在算法第 5 行中,通过使用 *cosine* 度量来评估生成的学习规则和其对应的频繁项 *fi* 的相关性,从而在算法中仅保留那些具有学习意义的强关联学习规则.

针对任意生成的学习规则"*s*⇒(*fi*−*s*)",该学习规则 *cosine* 度量的定义如公式(4)所示.

$$cosine(s, fi-s)=\frac{P(s\cup(fi-s))}{\sqrt{P(s)\times P(fi-s)}} \tag{4}$$

其中,*P*(*i*)表示集合 *i* 中的所有软件职责在步骤 6.1 生成的种群信息数据集中同时出现的概率.根据 Merceron 和 Yacef 的经验准则[24],算法仅保留 *cosine* 度量大于临界值 0.65 的学习规则.在后续的研究中,我们将进一步研究 *cosine* 度量的临界值,从而找到冲突消解效率最高的 *cosine* 临界值取值.此外,由于不同的学习规则具有不同的优先级,在第 6 行中,算法利用置信度[25]定义学习规则的优先级.置信度计算公式如公式(5)所示.

$$confidence(s⇒(fi-s))=P((fi-s)|s) \tag{5}$$

当有多条规则可以被学习时,优先级最高的规则将会被学习.最后,算法第 8 行到第 11 行对重复的规则进行过滤.假设规则集中有两条学习规则 r_1 和 r_2,如果 r_1 是 r_2 的子集并且 r_1 的 *cosine* 和置信度度量大于等于 r_2 的 *cosine* 和置信度,那么算法将会从学习规则集中去除重复规则 r_2.步骤 6.3 学习规则生成算法的最坏时间复杂度为 $O(n\times 2^n)$.

2.2.4 应用学习规则消解冲突(步骤 6.4)

当学习规则生成后,学习运算子可以应用生成的学习规则消解职责种群和模式种群个体组合生成软件体系结构候选方案时产生的冲突.图 7 给出了应用学习规则的冲突消解算法.在该算法中,假定当职责种群个体和

模式种群个体的组合发生冲突时,模式种群个体通过学习职责种群个体所产生的规则,从而自适应地调整.换言之,当检测到冲突时,由模式种群个体尝试进行冲突消解(反之,职责种群个体也可以通过学习模式种群个体所产生的规则来消解冲突).

```
输入:R_res:职责种群学习规则集,ind_p:模式种群个体,ind_r:职责种群个体.
输出:newInd_p:利用学习规则进行学习后的模式种群个体 ind_p 的新状态.
01  newInd_p←ind_p
02  if isConflicted(ind_p,ind_r)=FALSE
03      return newInd_p
04  R_s←sort(R_res)
05  rulesBanningList←∅
06  ruleSearch(newInd_p)
07  return newInd_p
08
09  procedure ruleSearch(ind_p)
10      for Rule r∈R_s
11          if r∉rulesBanningList∧ind_p satifies the precondition of r
12              tmpInd_p←applyRule(r,ind_p)
13              if tmpInd_p≠ind_p
14                  if isConflicted(tmpInd_p,ind_r)=FALSE
15                      newInd_p←tmpInd_p
16                  rulesBanningList.update(r)
17                  ruleSearch(tmpInd_p)
18  end procedure
```

Fig.7 Algorithm for conflict resolution

图 7 冲突消解算法

首先,在算法的第 1 行~第 3 行,算法检测待组合的职责种群个体和模式种群个体是否存在冲突:如果不存在冲突,则可以直接进行组合生成软件体系结构候选方案;否则,需要在职责种群个体和模式种群个体组合前进行冲突消解.在算法的第 4 行,按照规则的优先级对职责种群的学习规则集进行排序,确保高优先级的规则被优先学习.算法的第 5 行使用了列表来记录最近被学习的学习规则.为了防止单一的规则在短时期内多次被学习从而影响到生成个体的多样性,该列表确保最近被学习的学习规则在短时间之内不可再次被个体学习.如第 6 行所示,算法的主体部分使用深度优先搜索对职责种群学习规则集进行了遍历,从而尽最大可能根据职责种群学习规则集自适应调整模式种群个体.在算法的第 10 行和第 11 行,通过找到优先级相对最高且最近没有被学习过的学习规则对模式种群个体进行学习.需要说明的是,每条学习规则有其前置条件,如果模式种群个体尝试学习该规则,则个体必须满足该学习规则的前置条件.假设学习规则为$\{R_i,R_j\}\Rightarrow\{R_k\}$,那么规则的左部$\{R_i,R_j\}$即为该规则的前置条件.如果一个模式种群个体尝试学习该条规则,意味着R_i和R_j必须属于相同的模式角色.通常情况下,模式种群个体需要通过多次学习规则集中的规则才能消解与职责种群个体间的冲突.在最坏的情况下,Ind_p始终无法学习到合适的规则来消解冲突,我们将在第 4.2 节对此进行讨论.如图 7 所示的冲突消解算法的最坏时间复杂度为$O(n^n)$.

2.2.5 方法示例

在本节中,我们采用引言部分提出的例子(包含 5 个软件职责R_1~R_5的体系结构合成问题),通过示例的方式进一步解释 CoEA-L 中的步骤 6.1~步骤 6.4.需要说明的是,实际软件项目中的软件职责数目非常巨大,这里的示例只是为了简洁地解释 CoEA-L 中步骤 6.1~步骤 6.4,因此,示例中假设只包含 5 个软件职责.

由于本文提出的方法假设由模式种群进行冲突消解,在步骤 6.1 中,需要从职责种群中提取出职责种群所有个体中职责到类的分配信息,生成职责种群信息数据集.这里,我们假设职责合成种群的个体总数为 3(在演化计算中,种群个体数一般较大,这里为了以例子形式直观进行描述,假设职责种群的个体总数只有 3 个).职责合成种群及其包含的 3 个可能个体如图 8 所示.

在步骤 6.1 中,针对职责种群的每一个个体的每一个类,使用一个记录来记录该类所包含的软件职责.因此,个体的记录数目等于该个体所对应的职责候选方案中的类的数目.例如,图 8(a)中职责种群个体 1 包含 2 条记

录,分别是{R_1,R_2,R_3},{R_4,R_5};而图 8(b)中职责种群个体 2 因为有 3 个类,所以包含 3 条记录,分别是{R_1,R_4},{R_2,R_3},{R_5}.同理,图 8(c)中职责种群个体 3 的记录为{R_1,R_2,R_3},{R_4},{R_5}.只需要将职责种群的所有个体的所有记录组合,就生成了职责种群信息数据集.在图 8 中,提取到的职责种群信息数据集为{{R_1,R_2,R_3},{R_4,R_5},{R_1,R_4},{R_2,R_3},{R_5},{R_1,R_2,R_3},{R_4},{R_5}}.

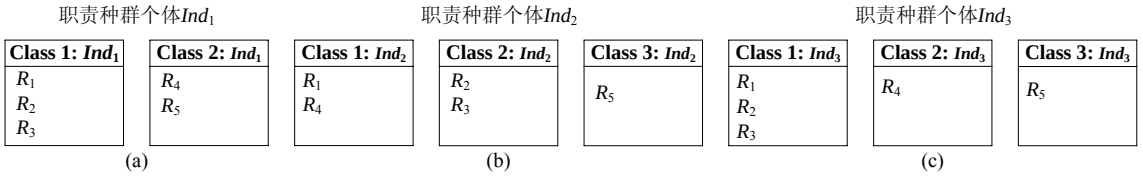


Fig.8 Three possible individuals in responsibility population

图 8 职责种群的 3 个可能个体

在步骤 6.2 的职责种群频繁项集的生成中,假设架构师定义的频繁项集的阈值为 2,那么图 8 生成的职责种群信息数据集{{R_1,R_2,R_3},{R_4,R_5},{R_1,R_4},{R_2,R_3},{R_5},{R_1,R_2,R_3},{R_4},{R_5}}中,频繁项集为{{R_1,R_2},{R_1,R_3},{R_2,R_3},{R_1,R_2,R_3}}.因为在所有二元项集中,{R_1,R_2},{R_1,R_3}在职责种群信息数据集中出现次数等于频繁项集的阈值(2),分别 2 次出现在项集{R_1,R_2,R_3}中,而{R_2,R_3}在职责种群信息数据集中出现 3 次,2 次出现在项集{R_1,R_2,R_3}中,1 次出现在{R_2,R_3}中.同时,在该例子中不存在除{R_1,R_2,R_3}以外的其他三元及三元以上的项集出现次数大于等于频繁项集的阈值.由于频繁项集的所有非空子集必然也是频繁的,因此,步骤 6.2 最终生成的职责种群频繁项集只需要包含{R_1,R_2,R_3}.

由于步骤 6.2 生成的频繁项集中只包含频繁项{R_1,R_2,R_3}(图 6 算法第 1 行中的 *fi*),该频繁项 *fi* 对应 7 个非空子集,则步骤 6.3 中图 6 算法第 2 行中的 *S* 集合包含 7 个元素 *s*,分别为{R_1},{R_2},{R_3},{R_1,R_2},{R_1,R_3},{R_2,R_3},{R_1,R_2,R_3}.这里以其中的一个元素 *s*({R_1,R_2})为例,因为在图 6 算法第 4 行中,只有职责 R_3 在 *fi* 中,同时又不在 *s* 中,则该元素 *s* 对应生成的学习规则为{R_1,R_2}⇒{R_3}.同理,假设元素 *s* 为{R_1,R_3},则对应的学习规则为{R_1,R_3}⇒{R_2}.通过进一步应用图 6 算法第 5 行中对学习规则 *cosine* 度量的计算、图 6 算法第 6 行中对学习规则 *confidence* 度量的计算以及图 6 算法第 8 行到第 11 行中对冗余学习规则的消除,步骤 6.3 可以生成用于后续冲突消解的学习规则集,并将这些学习规则按照 *confidence* 度量进行优先级排序.

在步骤 6.4 中,如图 9 所示,假设一个职责种群个体 Ind_r 和一个模式种群个体 Ind_p 尝试进行组合生成软件体系结构设计候选方案.同时,假设步骤 6.3 生成的职责种群学习规则集包含 3 条不同优先级的学习规则.当 Ind_r 和 Ind_p 进行组合时,这两个个体之间存在着两个冲突:1) 因为在职责种群个体 Ind_r 中,软件职责 R_2 和 R_3 被分配到同一个类,所以与模式种群个体 Ind_p 中 R_2 与 R_3 被分配到分层模式的不同层产生冲突;2) 同理,R_4 与 R_5 也产生冲突.因此,冲突消解算法需要利用职责种群学习规则集来调整模式种群个体 Ind_p,从而消解 Ind_p 和 Ind_r 之间的所有冲突.

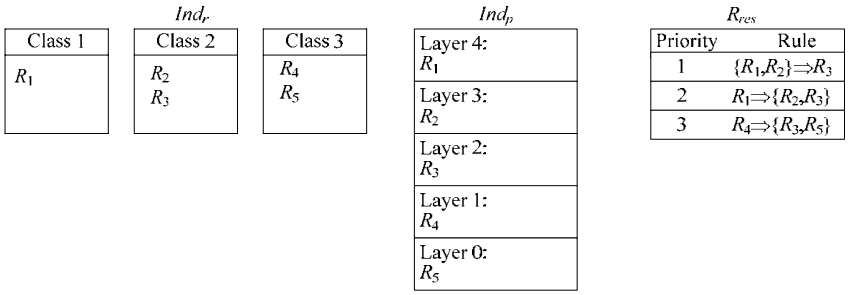


Fig.9 An initial state when ind_p and ind_r try to resolve inconsistencies

图 9 职责种群个体 ind_p 和模式种群个体 ind_r 尝试冲突消解时的初始状态

图 9 中的模式种群个体冲突消解实例如图 10 所示.在图 10 中,Ind_p 尝试学习优先级最高的规则 $\{R_1,R_2\}\Rightarrow\{R_3\}$,由于该规则的前置条件是软件职责 R_1 和 R_2 必须属于同种模式角色(分层模式的同一层),因此 Ind_p 只能继续尝试学习第二优先级的规则 $\{R_1\}\Rightarrow\{R_2,R_3\}$.于是,$Ind_p$ 通过学习,将自身调整为图 10(a)所示状态.在该状态下,由于 R_2,R_3 已被分配到分层模式的同一层里,上文提到的第 1 个冲突被成功消解.但 R_4 与 R_5 的冲突依旧存在,Ind_p 继续利用职责种群学习规则集进行第 2 轮学习.在第 2 轮学习中,优先级最高的规则 $\{R_1,R_2\}\Rightarrow\{R_3\}$ 由于无法改变 Ind_p 产生新的状态(图 7 第 13 行),所以依然无法被 Ind_p 学习,而第二优先级的规则 $\{R_1\}\Rightarrow\{R_2,R_3\}$ 因为在上一轮已被学习,因此在这一轮的学习中,Ind_p 只能利用优先级最低的 $\{R_4\}\Rightarrow\{R_3,R_5\}$ 进行学习,产生新的状态如图 10(b)所示.在该状态下,由于 R_4 与 R_5 已被分配到分层模式的同一层里,上文提到的第 2 个冲突被成功消解.但 R_2 与 R_3 的冲突又重新产生,Ind_p 需要继续进行第 3 轮学习.在第 3 轮学习中,Ind_p 利用优先级最高的规则 $\{R_1,R_2\}\Rightarrow\{R_3\}$ 产生新的状态,如图 10(c)所示.在该状态下,Ind_r 和 Ind_p 之间的两个冲突全部被消解.因此对于该例子中的 Ind_r 和 Ind_p,本文提出的方法可以无冲突地使用它们进行软件体系结构候选方案的合成.但是在 Ind_p 的某一(中间)状态下,可能所有的规则都无法被学习.当学习无法继续时,Ind_p 的当前状态需要回溯到上一状态,重新利用不同的规则进行学习(即优先级相对较低的候选规则).

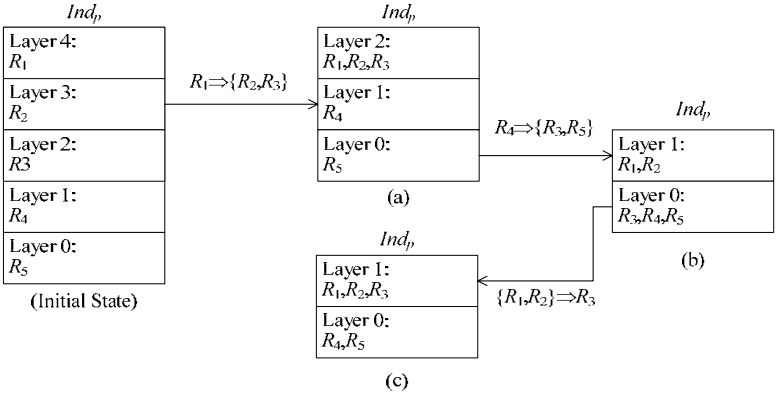


Fig.10 Procedure of conflict resolution for Ind_p

图 10 模式种群个体 Ind_p 的冲突消解过程

3 实验设计

本节主要描述如何通过实验设计对本文所提出的方法进行有效性验证,包括研究问题的提出(第 3.1 节)、设计问题的选择(第 3.2 节)、竞争方法介绍(第 3.3 节)、方法运行参数的设置(第 3.4 节)以及对方法有效性评估的度量指标(3.5 节).最后介绍了对实验结果进行分析的统计方法和检验(第 3.6 节)以及效度威胁(第 3.7 节).

3.1 研究问题

我们提出 3 个研究问题(research question,简称 RQ),用于验证本文所提出方法的有效性.

- RQ1:职责种群个体和模式种群个体组合生成软件体系结构候选方案时,冲突发生的频率?RQ1 验证本文提出的 CoEA-L 方法是否具备研究意义:如果职责种群个体和模式种群个体组合时很少或基本不发生冲突,则尝试进行冲突消解的研究意义不大.RQ1 是验证我们所提出的方法意义的基本研究问题.
- RQ2:基于学习的协作式协同演化软件体系结构合成方法是否可以在不同的问题实例中有效的消解冲突?RQ2 主要研究 CoEA-L 方法的有效性,即在多大程度上消解职责种群个体和模式种群个体组合时所产生的冲突.同时,通过在不同规模的问题实例上应用本文提出的方法,RQ2 也研究了本文提出的方法是否具备一定的适用性.
- 余下的研究问题 RQ3 主要关注于本文所提出的 CoEA-L 方法和其他竞争方法(见第 3.3 节)在解决面

向模式的软件体系结构合成问题中的差异.RQ3 包含两个子研究问题.

- RQ3.1:基于学习的协作式协同演化软件体系结构合成方法最终生成的面向模式的软件体系结构方案相比于其他竞争方法生成的设计方案,是否具有更好的设计质量?如果 CoEA-L 方法最终生成的软件体系结构的设计质量低于其他方法生成方案的设计质量,那么即使本文提出的方法能够较好地进行冲突消解,方法的实用性依然存在问题.
- RQ3.2:基于学习的协作式协同演化软件体系结构合成方法相比于其他竞争方法,是否具有显著的额外系统开销?RQ3.2 主要研究不同方法用于解决面向模式的软件体系结构合成问题时的系统性能开销,在该研究问题中,我们主要关注本文提出方法所引入的学习运算子是否会显著增加系统的性能开销.如果额外增加的性能开销过大,将影响本文提出方法的实用性.

3.2 实验设计问题

在实验验证中,我们选取了 3 个设计问题(实例)进行验证,分别是电影院预订系统(cinema booking system,简称 CBS)[26]、毕业生管理系统(graduate development system,简称 GDS)[27]和邮轮选择系统(select cruises system,简称 SCS)[28].选择这 3 个设计问题进行实验验证的理由如下.

1) 这 3 个设计问题非本文作者设计,这样可以减少实验结果可能存在的偏向性.
2) 这 3 个系统的人工设计方案公开[29],便于作为基准设计与各种自动化设计方法的结果进行统一比较.
3) 软件体系结构合成领域已有研究工作[30−33]使用这些设计问题来评估所提出的方法.

选取的这 3 个设计问题具有不同的问题规模,表 3 给出了每个设计问题的方法、属性、软件职责以及职责间的依赖数目.在文献[31]中,Simons 等人分析了职责合成的问题复杂度以及不同问题实例所对应的解空间的大小[31].Simons 等人通过一个包含 16 个软件职责的具体例子,指出职责合成问题的解空间大小将随着软件职责数目的增加而呈指数级增长.例如,将这 16 个职责分配到 5 个类中,该实例所对应的解空间大小也超过了 132 300 000[31].而在实际职责合成中,由于类的数目并不固定,因此即便是对 16 个软件职责进行职责合成,问题也具有非常高的复杂度.类似地,软件职责的模式合成问题也具有非常高的问题复杂度.综上,本文所选取的这 3 个不同规模的设计问题都足以验证本文提出方法的有效性,从而保证实验结果具有实用意义.

Table 3 Problem instances with their sizes

表 3 设计问题(实例)的规模

问题实例	方法数目	属性数目	职责总数	依赖数目
CBS	15	16	31	39
GDS	12	43	55	107
SCS	30	62	92	141

3.3 竞争方法介绍

本文选择的竞争方法包括 RS,CoEA 和 BO.Arcuri 等人指出,在进行实验验证时,任何基于搜索的算法都需要和 RS 算法进行比较,从而确保改进算法获得的理想实验结果并不是由于选取的目标实验研究问题过于简单导致的[34,35].而 CoEA 和本文提出的 CoEA-L 的唯一差异就是是否引入学习运算子(图 4 中的步骤 6).通过比较 CoEA 和 CoEA-L 的性能差异,我们可以验证学习运算子的有效性.BO 作为一种广泛使用的分层优化方法[36,37],适合解决面向模式软件体系结构设计方案的合成问题.因此在实验验证中,我们选择了 RS,CoEA 和 BO 作为 CoEA-L 的竞争方法.

3.4 实验参数设置

实验运行在配置酷睿 2.3G 双核处理器,10GB 物理内存的计算机上.在实验中,包含 3 个不同的对照组,分别采用了第 3.3 节介绍的 3 种竞争方法,而实验组采用了本文提出的 CoEA-L.实验参数设置见表 4(其中,斜体字表示仅适用于本文所提出的方法(即 CoEA-L)的参数及其设置).

Table 4 Experimental parameter settings of studied approaches

表 4 方法运行的实验参数设置

方法	实验参数
基于学习的协作式协同演化方法(CoEA-L)(本文提出的方法)	职责种群大小:100 模式种群大小:100 最大种群代数:200 交叉运算子:单点交叉 交叉概率:0.9 突变运算子:交换突变 突变概率:1/染色体长度 选择运算子:锦标赛选择 频繁项集阈值:5(仅适用于本文提出的方法)
协同演化算法(CoEA)	
双层优化法(BO)	
随机搜索(RS)	适应度函数评估次数:40 000

3.5 实验度量指标

为回答第 3.1 节提出的研究问题(RQ),我们需要对实验组与对照组进行了如下 4 个方面的比较.

1) 职责种群个体和模式种群个体组合生成候选软件体系结构方案时冲突发生的频率.

2) 当冲突发生时,不同方法能够有效地进行冲突消解的能力.

3) 不同方法生成的候选软件体系结构方案的设计质量.

4) 不同方法运行导致的系统性能开销.

由于性能开销主要比较方法运行时间,因此这里主要介绍与前 3 个方面相关的实验度量指标.

1) FC(frequency of conflicts)度量

在 FC 度量中,我们使用两个计数器 $C_{conflict}$ 和 $C_{combine}$:当一个职责种群个体 r 和一个模式种群个体 p 尝试组合生成候选软件体系结构方案时,$C_{combine}$ 计数器始终递增;如果 r 和 p 发生冲突,则同时递增 $C_{conflict}$ 计数器.因此,FC 度量的定义如公式(6)所示.

$$FC=\frac{C_{conflict}}{C_{combine}}\times 100\% \tag{6}$$

FC 度量的取值范围为 0~1,FC 比值越高,则职责种群个体和模式种群个体组合时发生冲突的概率越大.

2) ERI(effectiveness of resolving inconsistencies)度量

在 ERI 度量中,我们使用两个计数器 $C_{conflict}$ 和 $C_{resolved}$:当一个职责种群个体 r 和一个模式种群个体 p 尝试组合生成候选软件体系结构方案发生冲突时,$C_{conflict}$ 递增;如果 r 和 p 的冲突可以成功消解,则 $C_{resolved}$ 递增.因此,ERI 度量定义如公式(7)所示.

$$ERI=\frac{C_{resolved}}{C_{conflict}}\times 100\% \tag{7}$$

ERI 度量的取值范围为 0~1,ERI 比值越高,则职责种群个体和模式种群个体组合发生冲突时,冲突被消解概率越大.

3) DQ(design quality of solution)度量

软件体系结构设计方案的设计质量难以自动化度量.通常,设计质量是根据架构师自身的设计经验进行主观评价.文献[38]证实:对于软件设计质量评估,目前依旧缺乏有效的自动化度量指标.因此,我们参照软件体系结构合成领域相关文献[6,32],采用将自动化设计的软件体系结构候选方案与专家设计方案进行相似度比较的方法来度量自动化方案的设计质量.我们认为,专家设计方案能够尽可能地满足软件设计原则,同时,针对特定设计问题,专家方案的软件设计质量能够满足设计需求.因此,自动化生成的设计方案与专家设计方案的相似度越高,则可以认为设计质量越高.

本文所采用的实验设计问题(见第 3.2 节)全部具有作为基准设计的公开的专家职责合成方案[29].根据专家职责合成方案类及其依赖关系,形成以类为节点、类间依赖为边的有向图 G,进而对 G 执行拓扑排序,从而生成专家模式合成方案.在拓扑排序中,最早找到的没有前驱(入度为 0)的节点构成节点集合 V_i,V_i 中节点对应的类构

成采用分层模式的专家模式合成方案中的第 i 层(i>0).在 G 中删除节点集合 V_i 及所有以 V_i 中节点为起点的有向边后,找到的入度为 0 的节点构成节点集合 V_{i-1},V_{i-1} 中节点对应的类构成采用分层模式的专家模式合成方案中的第 i–1 层.重复该过程,直到 G=∅.当 G 中存在环导致无法找到入度为 0 的节点时,选择入度最小出度最大的节点 v 连同 v 的所有前驱节点构成节点集合 V_i.

对于职责种群个体,我们使用 F-Score 来比较该设计方案与专家职责合成方案的相似度.F-Score 的定义如公式(8)所示.

$$F\text{-}Score(Cls,C)_{RS}=\sum_{cls_i\in Cls}\frac{|cls_i|\times\max_{c_j\in C}\{F(cls_i,c_j)\}}{|R|} \tag{8}$$

其中, $F(cls_i,c_j)=2\times\frac{Recall(cls_i,c_j)\times Precision(cls_i,c_j)}{Recall(cls_i,c_j)+Precision(cls_i,c_j)}$, $Precision(Cls_i,c_j)=\frac{n_{ij}}{|c_j|}$, $Recall(cls_i,c_j)=\frac{n_{ij}}{|cls_i|}$.

在公式(8)中,R 表示软件职责;Cls 表示专家设计方案;cls_i 表示专家设计方案中的任意一类(class),而该类所拥有的软件职责数目为$|cls_i|$.类似地,C 表示职责合成自动化生成的一个候选方案,c_j 表示其中的一个类,n_{ij} 记录了专家设计方案 Cls 中的类 cls_i 和自动化生成的一个候选方案 C 中的类 c_j 共同拥有的软件职责数目.F-Score 的计算依赖于 F 值,而 F 值的计算依赖于信息检索领域的精确率和召回率的计算[39].在我们的公式中,通过重新定义精确率和召回率的计算方法来计算 F 值.

类似地,我们仍然使用 F-Score 来计算模式种群个体和专家模式合成方案间的相似度.F-Score 的定义如公式(9)所示.

$$F\text{-}Score(Rol,Ro)_{PS}=\sum_{rol_i\in Rol}\frac{|rol_i|\times\{F(rol_i,ro_i)\}}{|R|} \tag{9}$$

其中, $F(rol_i,ro_i)=2\times\frac{Recall(rol_i,ro_i)\times Precision(rol_i,ro_i)}{Recall(rol_i,ro_i)+Precision(rol_i,ro_i)}$, $Precision(rol_i,ro_i)=\frac{n_{ii}}{|ro_i|}$, $Recall(rol_i,ro_i)=\frac{n_{ii}}{|rol_i|}$.

在公式(9)中,Rol 表示专家设计方案中存在的所有模式角色,Ro 表示模式合成自动化生成的一个候选方案中的所有模式角色,而 rol_i 表示专家设计方案所用模式的某一特定模式角色,ro_i 表示模式合成自动化生成的一个候选方案所用模式的某一特定模式角色.

F-Score 公式可以对自动化生成的任意软件体系结构设计方案计算其与专家方案的相似度,从而评估其设计质量.通过评估种群中所有软件体系结构设计方案与专家方案的相似度,就可以评价不同方法生成的候选软件体系结构方案的设计质量.

3.6 统计分析方法

对于具有随机性特征的搜索算法,为了可靠地回答第 3.1 节提出的研究问题(RQ),针对每一个设计问题(实例)的每次实验验证,我们采用多次运行并使用统计分析的方法给出对应的实验结果[40].根据 Arcuri 等人提出的随机性算法实验评估建议[35],所有实验验证次数设置为 30 次.比较两组样本时,最常使用的统计检验是 t-test 和 Mann-Whitney U-test.其中,t-test 是有参检验,而 U-test 是无参检验.U-test 在统计推断过程中不涉及有关总体分布的参数,因而对样本数据的分布假设较少.根据文献[40]中的建议,我们使用 p 值为 0.05 的双尾 t-test 和 U-Test 进行两组样本间差异的显著性检验.根据文献[40,41]的建议,我们同时报告了效应量 $\hat{A}_{12}$ [42]. $\hat{A}_{12}$ 度量可以定量分析两组样本之间差异的显著性,其取值为 0~1.假设得到的 $\hat{A}_{12}$ 度量值为 0.8,则意味着在不同的实验验证中,80%的运行第 1 组样本能取得更优的结果.因此,取值越大,表明样本间差异越显著.根据文献[42]中的建议,当 $\hat{A}_{12}$ 取值小于 0.56,0.64,0.71 时,我们分别认为第 1 组样本和第 2 组样本之间差异较小、差异一般,或具有较大差异.

3.7 效度威胁

根据文献[41]的指南,本节讨论了本文实验设计的局限以及对实验结果效度可能产生的威胁和应对措施.

- 构造效度是指实验与理论间的一致性.在基于搜索的软件工程研究中,构造效度威胁主要来自于使用了不可靠的有效性度量[43].在本文的实验中,主要包含两个该类威胁.

1) 我们使用的评估候选体系结构设计方案质量的*F*-Score度量是否能够准确地度量设计方案的设计质量.为了缓解该效度威胁,在所有的设计问题(实例)中,当评价不同的自动化方法时,都使用了相同的适应度函数.此外,*F*-Score 度量的定义是基于信息检索领域广泛使用的准确率和召回率度量[39],从而缓解了 *F*-Score 度量的构造效度威胁.

2) 针对同一设计问题,不同的架构师会给出不同的专家设计方案,因此,当专家方案由不同的架构师来设计时,自动化方法生成的候选体系结构方案的 *F*-Score 会发生改变.但由于我们选取的设计问题(实例)以及相关的专家设计方案作为基准设计被广泛采用[30−33],所以该效度威胁在一定程度上得以缓解.

- 内部效度指实验的自变量和因变量之间存在明确因果关系或相关关系的程度,它表明因变量的变化在多大程度上来自自变量的影响.在本文的实验中,我们并未研究冲突消解效率和设计质量提升是由什么因素造成,因此不需要考虑内部效度威胁.另一方面,通过比较CoEA-L和CoEA算法的运行时间差异,我们研究了性能开销与学习运算子实现之间的因果关系.我们通过固定实验相关的其他自变量,并采用 30 次(多次)运行的方式,尽量消除其他可能变量产生的干扰.
- 外部效度是指实验结果类推到其他环境的有效性,强调实验结果是否具有普遍适应性.在基于搜索的软件工程研究中,外部效度威胁主要来自于缺乏对目标设计问题进行清晰的定义、实验设计问题的随意性选择、实验选取的设计问题在规模等方面没有差异[43].在本文的实验设计中,通过使用 3 个具有不同规模的设计问题来缓解外部效度威胁.选取的 3 个设计问题在已有的相关研究中也被广泛采用作为实验案例[30−33].
- 结论效度是关于研究结果的数据分析与方法的有效性.在基于搜索的软件工程研究中,结论效度威胁主要来自于使用了较少的设计问题进行实验验证、设计问题在规模或复杂度方面缺乏代表性、实验中没有考虑到生成数据的随机性、缺乏有效的统计显著性检验等[43].因此,采用更多不同规模、不同复杂度的设计问题(实例)进行实验验证,对于缓解结论效度威胁非常重要.在本文的实验设计中,主要采取 3 项措施来缓解结论效度威胁:1) 对每次实验验证采用多次重复实验,从而尽可能消除实验生成数据的随机性;2) 通过文献[41]中定义的标准实验流程和统计显著性检验方法,减少实验过程中不规范的数据分析;3) 使用具有不同规模的设计问题进行实验.同时,选取的设计问题的专家设计方案都是公开的,以便于重复验证[26−29].

4 实验结果和分析

本节基于上一节描述的实验设计对 3 个设计问题(实例)进行了实验.实验的结果及分析如下.

4.1 冲突频率

利用第 3.5 节提出的 FC 度量指标,我们研究了职责种群个体和模式种群个体组合生成软件体系结构候选方案时的冲突情况,如图 11 所示.

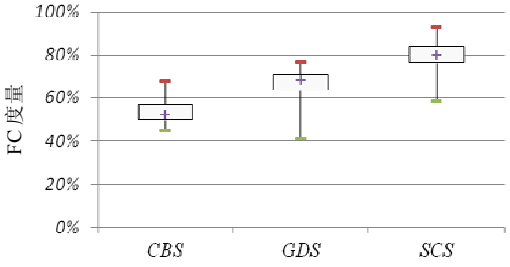


Fig.11 FC measurement for problem instances

图 11 设计问题的 FC 度量

从图 11 中可以看到,当职责种群个体和模式种群个体组合生成软件体系结构候选方案时,对于 3 个设计问

题(实例)都存在大量的冲突.在所有的 3 个设计问题中,*FC* 度量的最小值都超过了 40%,这意味着在职责种群个体和模式种群个体进行组合时,至少 40%的组合会发生冲突.此外,从图 11 中我们还可以发现,相对于软件规模较小的设计问题(CBS),较大的设计问题(GDS 和 SCS)在合成过程中存在更多的冲突.这表明采用协作式协同演化方法进行面向模式的软件体系结构候选方案合成时,有效的冲突消解至关重要,特别是对于规模较大的软件系统.这个结果佐证了本文研究工作的研究价值和实用意义.

4.2 冲突消解效率

在图 12 中,针对 3 个设计问题,我们给出了 CoEA-L 在职责种群个体和模式种群个体产生冲突时的冲突消解效率.在所有的 3 个设计问题中,ERI 度量的最小值均超过了 70%,意味着 70%以上的冲突都可以被 CoEA-L 方法有效地消解.而 ERI 度量在 3 个设计问题中的平均值接近 90%,表明了在所有的 30 次不同实例运行中产生的绝大多数冲突都可以被有效地消解.此外,在所有的 3 个设计问题中,ERI 度量的最大值均能达到 100%,表明针对这 3 个设计问题的某些运行中,所有冲突都被成功消解.

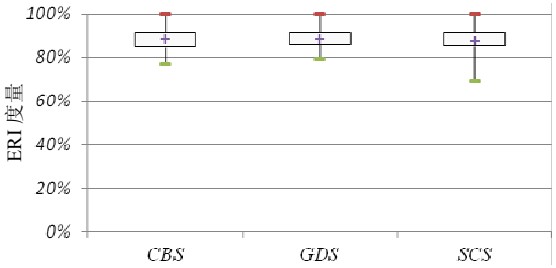


Fig.12 ERI measurement for problem instances with the proposed approach

图 12 本文方法用于不同设计问题得到的 ERI 度量

我们希望进一步研究本文提出的冲突消解方法的冲突消解效率是否会受到设计问题规模的影响,我们提出了如下原假设.

- H_{01}:CBS 设计问题的 ERI 度量和 GDS 设计问题的 ERI 度量差异没有统计意义上的显著性.
- H_{02}: CBS 设计问题的 ERI 度量和 SCS 设计问题的 ERI 度量差异没有统计意义上的显著性.
- H_{03}: GDS 设计问题的 ERI 度量和 SCS 设计问题的 ERI 度量差异没有统计意义上的显著性.

针对每个设计问题(实例),30 次重复实验得到的 ERI 度量结果作为每个设计问题对应的 ERI 度量值样本,然后计算 *p* 值进行比较.每个原假设的 *p* 值见表 5.从表 5 可以看到,所有的 *p* 值都大于 0.05,所以无法拒绝原假设,没有证据表明,在不同规模的设计问题之间,冲突消解的效率存在差异.

Table 5 *p*-values for the hypotheses of insignificant difference on *ERI* metric between problem instances

表 5 不同设计问题间 *ERI* 度量没有统计差异的原假设 *p* 值

统计检验方法	研究假设		
	H_{01}	H_{02}	H_{03}
t-test	0.357 5	0.599 0	0.200 7
Mann-Whitney *U*-test	0.456 1	0.824 7	0.304 3

基于上述实验和统计分析结果,本文提出的 CoEA-L 方法针对不同规模的设计问题都有较好的冲突消解效率;同时,设计问题本身的规模对该方法的冲突消解效率的影响较小.因此,本文提出的方法可以有效消解面向模式的软件体系结构合成中产生的冲突,并且适用于解决实际应用中不同规模的设计问题,具有一定的适用性.

需要说明的是,由于 3 种竞争方法在包含冲突的职责种群个体和模式种群个体进行组合时,没有提供额外的冲突消解策略.因此 3 种竞争方法的 ERI 度量值始终为 0.

4.3 软件体系结构候选方案的设计质量

针对 3 个设计问题,我们比较了 CoEA-L 和 3 种竞争方法所生成的候选软件体系结构方案的 $F\text{-Score}_{ps}$ 和

F-Score$_{rs}$ 的度量结果.实验验证过程针对每个设计问题,将每次实例运行生成的候选软件体系结构方案与专家设计方案的相似度的平均值作为该次运行的相似度,从而得到该次实例运行的 F-Score 值(相似度).

在表 6 中,我们给出了在不同的设计问题中,CoEA-L 和 CoEA 30 次实例运行得到的相似度的平均值、30 次实例运行得到的相似度的中位数、不同方法统计差异显著性的 p 值以及效应量 $\hat{A}_{12}$.表 6 的结果表明,针对软件规模较小的设计问题(如 CBS 实例),CoEA-L 和 CoEA 都能生成接近专家方案的候选体系结构设计方案(F-Score$_{rs}$ 和 F-Score$_{ps}$ 的平均值和中位数都大于 0.8),并且设计质量不存在明显的差异(t-test 和 U-test 得到的 p 值大于 0.05).可能的原因是:(1) 规模较小的设计问题其解空间也较小,使得自动化生成的候选解决方案的设计质量差异较小;(2) 在较小规模的设计问题中,模式种群个体和职责种群个体进行组合时发生冲突的概率相对较少,因此减弱了引入学习效应对软件体系结构候选方案设计质量的影响.

Table 6 CoEA-L vs. CoEA—Comparison in terms of the design quality metrics: F-Score for responsibility and pattern synthesis

表 6 职责合成和模式合成的设计质量度量 F-Score 在 CoEA-L 和 CoEA 方法上的比较

		CoEA-L		CoEA		CoEA-L vs. CoEA		CoEA-L vs. CoEA
		均值	中位数	均值	中位数	p-value		$\hat{A}_{12}$
						t-test	U-test	
CBS	F-Score$_{RS}$	0.807	0.801	0.801	0.801	0.595 3	0.636 1	0.52
	F-Score$_{PS}$	0.888	0.887	0.878	0.879	0.139 0	0.094 8	0.54
GDS	F-Score$_{RS}$	0.764	0.758	0.730	0.734	<0.0001	<0.0001	0.78
	F-Score$_{PS}$	0.848	0.848	0.820	0.822	<0.0001	<0.0001	0.75
SCS	F-Score$_{RS}$	0.722	0.723	0.638	0.647	<0.0001	<0.0001	0.94
	F-Score$_{PS}$	0.719	0.703	0.608	0.663	<0.0001	<0.0001	0.83

针对规模相对较大的设计问题(如 GDS 和 SCS 实例),CoEA-L 和 CoEA 也都能生成接近专家方案的候选体系结构设计方案.但是 CoEA-L 能够生成更接近专家方案的候选体系结构设计方案(F-Score$_{rs}$ 和 F-Score$_{ps}$ 的平均值和中位数都更大,并且 p 值小于 0.05).可能的原因是:

1) 针对某一特定设计问题,适合该设计问题的较优的候选体系结构方案往往都包含适用于该设计问题的软件体系结构知识.个体从模式种群和职责种群中学习,除了学习如何消解冲突外,也学习到种群中个体间共有的软件体系结构知识,从而改善了设计质量.
2) 相对于较小规模的设计问题,较大规模的设计问题拥有更大的解空间,学习效应的引入,能够更加有效地对解空间进行探索.
3) 相对于较小规模的设计问题,在较大规模的设计问题中,当职责种群个体和模式种群个体进行组合时,会发生更多的冲突.由于 CoEA 只组合那些没有冲突的个体,限制了候选体系结构方案的多样性,从而影响了生成方案的设计质量.

综上,当问题空间更加复杂时,CoEA-L 方法相对于 CoEA 方法的优势更明显.对于规模较大的软件系统,CoEA-L 能够生成比 CoEA 设计质量更高、更接近专家方案的候选体系结构设计方案.

在表 7 中,我们给出了在不同的设计问题中,CoEA-L 和 RS 30 次实例运行得到的相似度的平均值、30 次实例运行得到的相似度的中位数、不同方法统计差异显著性的 p 值以及效应量 $\hat{A}_{12}$.表 7 的结果表明,针对 3 个规模不同的设计问题,相对于 RS,CoEA-L 都能生成更接近专家方案的候选体系结构设计方案(CoEA-L 方法生成的候选体系结构方案的均值和中位数都大于 RS 方法生成的候选体系结构方案的均值和中位数;同时,t-test 和 U-test 得到的 p 值远小于 0.000 1).值得注意的是,由 CoEA-L 生成的候选体系结构方案的 F-Score 值所组成的 F-Score 样本与由 RS 生成的候选体系结构方案的 F-Score 值所组成的 F-Score 样本之间的效应量 $\hat{A}_{12}$ 的值始终为 1.这意味着针对 3 个规模不同的设计问题的任意一次实例运行中,CoEA-L 都优于 RS.这说明:(1) 面向模式的软件体系结构合成问题具有一定的复杂性,简单的应用随机搜索方法并不能获得理想的结果;(2) 针对第 3.1 节提出的研究问题(RQ),实验所选取的 3 个设计实例都具备足够的复杂性.

Table 7 CoEA-L vs. RS—Comparison in terms of the design quality metrics: *F*-Score for responsibility and pattern synthesis

表 7 职责合成和模式合成的设计质量度量 *F*-Score 在 CoEA-L 和 RS 方法上的比较

		CoEA-L		RS		CoEA-L vs. RS		CoEA-L vs. RS
		均值	中位数	均值	中位数	*p*-value		$\hat{A}_{12}$
						t-test	*U*-test	
CBS	$F\text{-Score}_{RS}$	0.807	0.801	0.612	0.633	<0.0001	<0.0001	1
	$F\text{-Score}_{PS}$	0.888	0.887	0.591	0.604	<0.0001	<0.0001	1
GDS	$F\text{-Score}_{RS}$	0.764	0.758	0.506	0.537	<0.0001	<0.0001	1
	$F\text{-Score}_{PS}$	0.848	0.848	0.493	0.480	<0.0001	<0.0001	1
SCS	$F\text{-Score}_{RS}$	0.722	0.723	0.446	0.421	<0.0001	<0.0001	1
	$F\text{-Score}_{PS}$	0.719	0.703	0.417	0.429	<0.0001	<0.0001	1

在表 8 中,我们给出了在不同的设计问题中,CoEA-L 和 BO 30 次实例运行得到的相似度的平均值、30 次实例运行得到的相似度的中位数、不同方法统计差异显著性的 p 值以及效应量 $\hat{A}_{12}$.

Table 8 CoEA-L vs. BO—Comparison in terms of the design quality metrics: *F*-Score for responsibility and pattern synthesis

表 8 职责合成和模式合成的设计质量度量 *F*-Score 在 CoEA-L 和 BO 方法上的比较

		CoEA-L		BO		CoEA-L vs. BO		CoEA-L vs. BO
		均值	中位数	均值	中位数	*p*-value		$\hat{A}_{12}$
						t-test	*U*-test	
CBS	$F\text{-Score}_{RS}$	0.807	0.801	0.775	0.784	<0.0001	<0.0001	0.76
	$F\text{-Score}_{PS}$	0.888	0.887	0.893	0.885	0.114 2	0.162 7	0.47
GDS	$F\text{-Score}_{RS}$	0.764	0.758	0.558	0.614	<0.0001	<0.0001	1.00
	$F\text{-Score}_{PS}$	0.848	0.848	0.806	0.794	<0.0001	<0.0001	0.81
SCS	$F\text{-Score}_{RS}$	0.722	0.723	0.513	0.507	<0.0001	<0.0001	1.00
	$F\text{-Score}_{PS}$	0.719	0.703	0.632	0.641	<0.0001	<0.0001	0.95

表 8 的结果表明,针对软件规模较大的设计问题(如 GDS 和 SCS 实例),CoEA-L 明显优于 BO(CoEA-L 方法生成的候选体系结构方案的均值和中位数都大于 BO 方法生成的候选体系结构方案的均值和中位数;同时,得到的 p 值均小于 0.000 1).其中,针对职责合成,由 CoEA-L 生成的候选体系结构方案的 *F*-Score 值所组成的 *F*-Score 样本优势更加明显(效应量 $\hat{A}_{12}$ 的值为 1).我们认为,由于我们的 BO 方法实现将职责合成作为了外层优化问题,职责合成所生成的解决方案必须保证满足模式合成问题生成的最优解的约束(因为在面向模式的软件体系结构合成中,为了组合生成最终的解决方案,职责合成候选方案在与模式合成方案进行组合时不能有冲突,因此,这里的约束意味着外层职责合成生成的最优解不可与内层模式合成生成的最优解产生冲突),而 BO 方法又不具备内外层优化问题冲突消解的机制,因此这限制了职责合成候选解决方案的多样性.另一方面,在 CoEA-L 中,虽然职责合成和模式合成的候选解之间也存在大量冲突(见第 4.1 节),但由于具有较好的冲突消解效率(见第 4.2 节),职责合成和模式合成候选解决方案的多样性并不会受到限制,所以 CoEA-L 具有更高的概率生成接近专家方案的候选体系结构设计方案.

4.4 运行开销

在本小节中,我们使用 3 个设计问题中复杂程度最高的 SCS 来评估基于学习的协作式协同演化软件体系结构合成方法和其他 3 个竞争方法的系统运行开销.我们采用两种方式来评估各种方法的运行开销.

- 首先,在第 4.4.1 节中,我们根据第 3.4 节描述的各方法的参数设置,通过固定各种方法适应度函数的总评估次数,比较各种方法的系统运行时间.在该评估方法中,由于不同方法间适应度函数的总评估次数相同,因此方法间运行时间的差异主要由方法内部的实现差异决定.例如,当适应度函数的总评估次数相同时,CoEA-L 与 CoEA 将运行相同的种群代数;同时,由于两种方法在种群的每一代中使用相同的选择、交叉、突变运算子,则两种方法运行时间的差异由 CoEA-L 新增的学习运算子实现所决定.
- 此外,对于软件架构师而言,自动化方法的一个重要优势是能够在尽可能短的时间向架构师提供可行

的候选体系结构解决方案.因此在第 4.4.2 节中,我们比较了不同方法生成可行候选体系结构方案的运行时间.

4.4.1 固定适应度函数评估次数的运行时间比较

在图 13 中,我们给出了不同方法在 SCS 设计问题中进行相同次数的适应度函数评估所消耗的运行时间.针对每种方法,我们同样运行 30 次实例,图 13 中的箱线图展示了每种方法 30 次运行所消耗的运行时间的最大值、最小值和四分位.

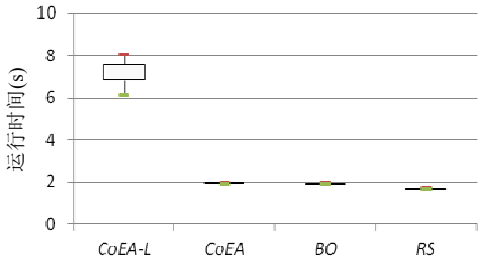


Fig.13 Running time for CoEA-L and the three rival approaches with the same fitness evaluations

图 13 CoEA-L 与 3 种竞争方法进行相同次数适应度函数评估所消耗的时间

图 13 的结果表明,(1) 3 种竞争方法进行相同次数的适应度函数评估所消耗的运行时间差异不大;(2) 每种竞争方法内部 30 次运行所消耗的运行时间几乎没有差异(最大值、最小值及四分位基本相等);(3) 本文提出的 CoEA-L 与 3 种竞争方法相比,完成相同次数的适应度函数评估需要消耗更多的运行时间;(4) CoEA-L 内部 30 次运行所消耗的运行时间存在一定的差异.我们认为,以上结果的原因是:由于适应度函数的评估次数固定,意味着对于 CoEA,BO 和 RS 而言,它们采用的算法每次运行的搜索次数是恒定的.同时,这 3 种算法基本的搜索运算操作独立于生成解,导致在不同的运行中,虽然可以生成不同的候选方案,但运行时间几乎没有差异.这里以 CoEA 方法中的算法为例,在每一子代的生成中,该算法包含的选择、交叉、突变运算子在任意的生成解上运行都不会有显著的运行时间差异,而生成子代的数目确定,使得该算法不同运行所消耗的运行时间差异很小.另一方面,本文提出的 CoEA-L 不同运行间存在较大的运行时间差异.我们认为,主要原因在于,CoEA-L 中,学习运算子进行学习消解冲突时,职责合成和模式合成生成的候选方案对冲突的消解效率具有较大的影响.每一次的实例运行由于生成了不同的职责合成和模式合成候选方案,导致了不同运行所消耗的运行时间有较大差异.

在图 13 中,通过比较 CoEA-L 和 CoEA 方法的算法运行时间差异,我们可以评估学习运算子的实际执行代价.通过比较两种算法的运行时间可以发现,相比于选择、交叉和突变运算子,学习运算子的执行代价较高.学习运算子的引入导致 CoEA-L 相比于 CoEA 方法的平均运行时间增长了 2 倍以上.通过进一步分析,我们认为,CoEA-L 和 CoEA 方法所采用算法的公共部分,即选择、交叉和突变运算子的实现非常简单,任一操作的平均时间复杂度不会超过 $O(n)$,而 CoEA-L 方法中的学习运算子则由图 5 所示的 4 个步骤组成,这些步骤的时间复杂度讨论见第 2.2.1 节~第 2.2.4 节.

虽然本文提出的 CoEA-L 方法在运行相同的适应度函数评估时性能开销大于已有的方法,但在实际应用中,对于具有相当规模的设计问题(比如 SCS),额外的性能开销并不影响方法的实用性(额外 6 秒左右的运行开销不会影响方法的实用性),尤其是本文提出的 CoEA-L 方法能够有效地消解面向模式的软件体系结构合成问题中产生的冲突(见第 4.2 节)并能生成更优的候选体系结构解决方案(见第 4.3 节).另一方面,在实际应用中,能够尽快生成可行体系结构候选方案对架构师而言更具有实用性.在下一小节中,我们比较了不同方法生成可行候选体系结构方案的运行时间.

4.4.2 生成可行候选体系结构方案的运行时间比较

在图 14 中,我们给出了不同方法在 SCS 设计问题中生成可行候选体系结构方案所消耗的运行时间.针对每种方法,我们同样运行 30 次实例,图 14 中的箱线图展示了每种方法 30 次运行所消耗的运行时间的最大值、最小值和四分位.图 14 的结果表明,本文提出的 CoEA-L 相比 3 种竞争方法,可以在更短的时间内生成可行的体系

结构候选方案.我们认为,产生该结果的原因在于,CoEA-L 具有有效的冲突消解机制,当职责合成和模式合成候选方案存在冲突时,CoEA-L 可以立即消解冲突并迅速合成候选体系结构方案.

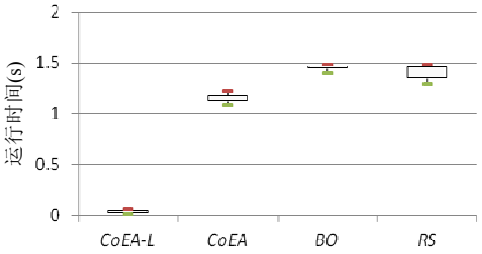


Fig.14 Running time for CoEA-L and the three rival approaches for generating feasible solutions

图 14 CoEA-L 与 3 种竞争方法生成可行体系结构方案所消耗的时间

综上,我们认为,虽然本文提出的 CoEA-L 相对于已有的 3 种竞争方法会产生额外的性能开销(见第 4.4.1 节),但并不影响该方法的实用性.另一方面,由于本文提出的方法具有有效的冲突消解机制(见第 4.2 节),可以更快地生成可行的候选体系结构方案(见第 4.4.2 节,即本小节),同时,生成的候选体系结构方案具有更优的设计质量(见第 4.3 节).因此,本文提出的 CoEA-L 方法与 3 种竞争方法相比,具有较为明显的优势.

5 相关工作

目前有大量的相关研究关注于软件体系结构设计中的冲突消解问题.在文献[44]中,作者研究了软件体系结构模型的不同版本之间合并时产生的冲突问题,并提供了冲突消解方法.文献[44]中的方法可以自动发现软件体系结构模型中需要更改的元素,从而消解冲突.该方法基于修复生成技术,通过检测模型的静态、动态结构,在准备合并的模型中建立一系列的修复操作,并在模型合并时消解冲突.与文献[44]中的工作类似,本文提出的方法也是用来消解软件体系结构的不同模型之间合并时产生的冲突(职责合成和模式合成模型).但不同于文献[44],本文提出的冲突消解方法是用于不同类型模型之间的合并,而不是相同模型的不同版本之间的合并.

在发生冲突时,每个冲突都有其本质根源.而冲突的根源通常揭示了软件体系结构模型中错误的来源.因此,在冲突消解之前了解冲突的根源非常重要.在文献[45]中,作者分析了软件体系结构中不一致设计规则的基本结构以及由此产生的相关行为.作者进而提出了一种算法来识别冲突产生的根源.与文献[45]的工作重点不同,本文的工作并不是去尝试发现软件体系结构设计模型中的缺陷与错误,而是消解自动化体系结构合成中的冲突,因此无须分析职责合成种群个体与模式合成种群个体组合时的冲突根源.

由于冲突消解会造成软件系统运行时额外的性能开销,因此软件体系结构设计和体系结构建模领域的许多研究重点关注于冲突检测、冲突追踪,而不是冲突消解.文献[46]是其中比较有代表性的工作,作者提出了一种可以实时检测和追踪冲突的自动化方法.在该方法中,用户只需要利用任意的形式化语言定义一致性规则,而自动化方法则可以侦测模型的变化是否违反这些自动化规则.与文献[46]的工作类似,本文的工作也可以在职责种群个体和模式种群个体组合时检测到存在的冲突,但是本文的工作不仅仅关注于冲突检测,更关注于在职责种群个体和模式种群个体进行组合时,对检测到的冲突进行消解.

在软件体系结构领域,构件模型作为一种常见的视图用于描述软件体系结构.同时,已有的软件体系结构研究通过对软件体系结构设计决策进行建模,从而捕捉设计原理和软件体系结构演化过程中的体系结构知识[47].当需要同时用构件模型和设计决策模型描述软件体系结构设计时,可能造成构件模型、软件体系结构设计决策模型、软件体系结构设计之间的冲突.在文献[48]中,作者提出了一种基于约束的方法来检测设计决策和相关构件之间是否存在冲突.当决策模型发生改变时,文献[48]提出的方法可以重新生成构件模型的约束,从而使得构件模型相应更新消解冲突.而文献[49]则提出了一种软件体系结构知识的转换语言消解软件体系结构设计决策模型和对应的软件体系结构设计之间的冲突.不同于文献[48,49]的工作,本文提出的方法通过学习的方式调整职责种群和模式种群个体,而不需要在职责种群和模式种群间建立显式的复杂约束关系.

除了软件构件模型和设计决策模型以外,在软件体系结构领域,还有多种不同的视图对软件体系结构进行描述[3].消解不同类型视图之间的冲突,对于软件体系结构的理解具有重要的意义.在文献[50]中,作者通过利用软件体系结构中的层次结构信息,将不同的软件体系结构视图转换成树形结构.然后,作者利用提出的算法修正了树形结构之间的不一致,实现了软件体系结构视图之间的冲突消解和视图间差异化的显示.与本文工作不同的是,本文主要关注软件体系结构合成中候选解决方案的冲突消解.

针对软件体系结构设计中产生的冲突,文献[51]采用体系结构自动化重构的方法对冲突进行消解.作者通过结合使用形式化的冲突检测技术以及搜索算法,自动发现适合消解冲突的重构操作集合.但是文献[51]中候选的重构操作仅仅包含了类的移动重构操作,这限制了冲突消解的灵活性.类似文献[51],本文提出的基于学习的协作式协同演化体系结构合成方法中也采用了自动化的搜索算法,但不同的是,本文通过在搜索算法中引入学习机制来消解冲突,种群个体间的学习并不限定于通过形式化方法定义的学习规则集合.

针对运行时的软件体系结构,文献[52]利用本体[53]、UML 建模语言[54]、着色 Petri 网[55]建模技术,提出了一个描述系统结构及行为的一致性框架.该框架通过使用不同的建模技术,采用映射的方式在体系结构的不同构件和体系结构的不同视图间建立关联,从而确保软件体系结构不同构件及视图之间不存在冲突.类似于文献[48],文献[52]通过利用多种建模技术,采用形式化语言定义模型元素间的约束,来确保体系结构构件和视图间的一致性.对于规模较为复杂的软件体系结构模型,人工定义模型元素之间的约束极易出错.本文所提出的 CoEA-L 方法并不需要通过建模语言来定义不同模型间的约束关系进行冲突消解.

当前,许多大型软件系统采用了面向服务的软件体系结构技术(service-oriented architecture,简称 SOA)[56]进行构建.为了促进 SOA 的标准化及可复用性,针对不同的业务领域,软件工程研究社区提出了 SOA 的参考软件体系结构[57].如何消解采用 SOA 技术构建的软件体系结构和相关参考软件体系结构之间的冲突,是 SOA 领域的重要研究问题.文献[58]是其中比较有代表性的工作,文献[58]通过利用自动化的映射技术,将软件体系结构设计元素映射到 SOA 参考软件体系结构的不同角色上,通过参考软件体系结构元素间的规则和约束,自动化修改软件体系结构设计元素,从而消解软件体系结构设计与相关参考软件体系结构之间的冲突.与文献[58]不同的是,本文提出的冲突消解方法不需要依赖业务领域的参考软件体系结构.本文工作与相关工作的比较见表 9.

Table 9 Comparison of our work with existing work.

表 9 本文工作与相关工作的比较

	研究目标	研究对象	研究方法
本文工作	冲突消解	不同类型的多个架构设计模型	数据挖掘获取知识进而消解冲突
文献[44]	冲突消解	相同类型不同版本的架构设计模型	分析模型一致性约束生成冲突修复规则
文献[45]	冲突根源识别	架构设计模型元素	通过自定义算法分析设计模型元素是否与形式化方法定义的设计规则冲突
文献[46]	冲突追踪	架构设计模型元素	通过自定义算法跟踪设计模型元素和与之对应的设计规则
文献[48]	冲突检测与消解	架构构件模型与架构决策	构建架构决策与构件元素之间的约束
文献[49]	冲突消解	架构构建模型与架构决策	定义架构知识转换语言,检测架构决策与构件元素一致性
文献[50]	冲突消解	架构视图	利用架构层次结构信息将不同的视图转换成树形结构.通过自定义算法实现树的转换,修正树形结构之间的不一致
文献[51]	冲突消解	架构设计模型元素	利用搜索方法找到一系列架构元素的 move 重构操作序列消解冲突
文献[52]	冲突消解	架构设计模型元素	本体,UML,着色 Petri 网建立架构一致性框架
文献[58]	冲突消解	SOA 架构设计模型元素	映射模型元素至 SOA 参考模型,利用参考模型约束消解冲突

6 结束语

软件体系结构合成活动连接了问题空间和解决方案空间,是软件体系结构设计中的关键性活动.为了在软件体系结构合成中复用已有的体系结构知识,架构师通常使用软件体系结构模式进行面向模式的软件体系结

构合成.职责合成和模式合成是面向模式的软件体系结构合成中的两大主要活动.由于软件系统规模不断增大,软件职责不断增加,使得自动化进行职责合成和模式合成并组合生成候选软件体系结构方案变得非常重要.但难点问题是:当组合职责合成和模式合成的设计方案时,如何消解设计方案之间的冲突.针对该问题,本文提出了基于学习的协作式协同演化软件体系结构合成方法.该方法利用鲍德温进化作为理论基础,提出了学习运算子概念,扩展了协作式协同演化软件体系结构合成方法[7].在本文中,我们通过在学习运算子中采用数据挖掘的关联算法,自动化发现哪些软件职责具有较高的概率在职责合成中被分配到相同的类或者哪些软件职责具有较高的概率在模式合成中被分配到相同的模式角色.提取到的这些信息进而用于冲突消解.

为了验证本文提出方法的有效性,我们使用了 3 个不同规模的设计问题作为研究案例.实验结果表明,与 3 种竞争方法相比,本文提出的方法能够有效地消解冲突,并且能够生成更加接近专家方案的候选软件体系结构设计方案.

在下一步工作中,我们将针对 3 个方面展开研究.1) 提升冲突消解的性能.一方面,在本文中,我们使用了数据关联挖掘中的 Apriori 算法,在下一步,我们尝试对 Apriori 算法及其参数进行进一步分析,并考虑使用其他数据关联挖掘算法来提升冲突消解的性能;另一方面,基于搜索的软件工程的最终目标之一是实现知识表示与处理的自动化[59],在下一步,我们将考虑如何在搜索过程中高效地表示和处理搜索过程中自动化产生的冲突消解相关知识,从而提升冲突消解性能.2) 使用更多不同规模、领域和复杂度的设计问题(实例)来评价本文提出的方法,考虑采用工业项目或开源项目.3) 基于本文提出的方法,实现面向模式的软件体系结构合成的支撑工具,并将该工具集成到已有的设计工具或集成开发环境中.

作者注 本文是我们于 2016 年 2 月 26 日投到《软件学报》的论文.该文是武汉大学梁鹏老师(本文第二作者)指导的 2018 届(2018 年 6 月)毕业的研究生徐永睿(本文第一作者)的博士论文《基于模式的软件体系结构自动化合成》工作成果的一部分,特此说明.

References:

[1] Mei H, Shen JR. Progress of research on software architecture. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[2] Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, America P. A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software, 2007,80(1):106−126.

[3] Bass L, Clements P, Kazman R. Software Architecture in Practice. 3rd ed., Addison-Wesley Professional, 2012.

[4] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-oriented Software Architecture: A System of Patterns. Wiley, 1996.

[5] Bowman M, Briand LC, Labiche Y. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE Trans. on Software Engineering, 2010,36(6):817−837.

[6] Masoud H, Jalili S. A clustering-based model for class responsibility assignment problem in object-oriented analysis. Journal of Systems and Software, 2014,93(7):110−131.

[7] Xu YR, Liang P. A cooperative coevolution approach to automate pattern-based software architectural synthesis, Int'l Journal of Software Engineering and Knowledge Engineering, 2014,24(10):1387−1411.

[8] Mitchell M, Thomure MD, Williams NL. The role of space in the success of coevolutionary learning. In: Proc. of the 10th Int'l Conf. on the Simulation and Synthesis of Living Systems (ALIFE). 2008. 118−124.

[9] De Jong K. Co-evolutionary algorithms: A useful computational abstraction? In: Proc. of the 7th Int'l Symp. on Search Based Software Engineering (SSBSE). 2015. 3−11.

[10] Baldwin JM. A new factor in evolution. The American Naturalist, 1896,30(354):441−451.

[11] Harman M, Mansouri SA, Zhang Y. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012,45(1):1−61.

[12] Xu YR, Liang P. Automated software architectural synthesis using patterns: A cooperative coevolution approach. In: Proc. of the 26th Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE). 2014. 174−180.

[13] Briand LC, Daly JW, Wüst JK. A unified framework for coupling measurement in object-oriented systems. IEEE Trans. on Software Engineering, 1999,25(1):91−121.

[14] Briand LC, Daly JW, Wüst JK. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering, 1998,3(1):65−117.

[15] Choi M, Kim IJ, Hong J, Kim J. Component-based metrics applying the strength of dependency between classes. In: Proc. of the 2009 ACM Symp. on Applied Computing. 2009. 530−536.

[16] Chidamber SR, Kemerer CF. Towards a metrics suite for object oriented design. In: Proc. of the '91 ACM SIGPLAN Conf. on Object-oriented Programming Systems, Languages, and Applications. 1991. 197−211.

[17] Bieman JM. Kang BK. Cohesion and reuse in an object-oriented system. ACM SIGSOFT Software Engineering Notes, 1995,20(SI): 259−262.

[18] Belle A, Boussaidi GEl, Desrosiers C, Mili H. The layered architecture revisited: is it an optimization problem? In: Proc. of the 25th Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE). 2013. 344−349.

[19] Xiao L, Cai Y, Kazman R. Design rule spaces: A new form of architecture insight. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE). 2014. 967−977.

[20] Saaty T. How to make a decision: The analytic hierarchy process. European Journal of Operational Research, 1990,48(1):9−26.

[21] Črepinšek M, Liu S, Mernik M. Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys, 2013, 45(3):1−33.

[22] Mitchell M. An Introduction to Genetic Algorithms. MIT Press, 1998.

[23] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: Proc. of the 20th Int'l Conf. on Very Large Data Bases (VLDB). 1994. 487−499.

[24] Merceron A, Yacef K. Interestingness measures for association rules in educational data. In: Proc. of the Educational Data Mining. 2008. 57−66.

[25] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. ACM SIGMOD Record, 1993,22(2):207−216.

[26] Simons C. Cinema booking system. 2015. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/CinemaBookingSystem.htm

[27] Simons C. Graduate development system. 2015. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/GraduateDevelopmentProgram.htm

[28] Apperly H. Service- and Component-based Development Using Select Perspective and UML. Addison-Wesley Professional, 2003.

[29] Manual designs. 2015. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/ManualDesigns.pdf

[30] Simons CL, Parmee IC. Elegant object-oriented software design via interactive, evolutionary computation. IEEE Trans. on Systems, Man, and Cybernetics, 2012,42(6):1797−1805.

[31] Simons CL, Parmee IC, Gwynllyw R. Interactive, evolutionary search in upstream object-oriented class design, IEEE Trans. on Software Engineering, 2010,36(6):798−816.

[32] Tawosi V, Jalili S, Hasheminejad SMH. Automated software design using ant colony optimization with semantic network support. Journal of Systems and Software, 2015,109(11):1−17.

[33] Smith JE, Simons CL. The influence of search components and problem characteristics in early life cycle class modelling. Journal of Systems and Software, 2015,103(5):440−451.

[34] Solis FJ, Wets RJB. Minimization by random search techniques. Mathematics of Operations Research, 1981,6(1):19−30.

[35] Arcuri A, Briand L. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability, 2014,24(3):219−250.

[36] Bard JF. Practical Bilevel Optimization: Algorithms and Applications. Springer Science & Business Media, 2013.

[37] Colson B, Marcotte P, Savard G. An overview of bilevel optimization. Annals of Operational Research, 2007,153(1):235−256.

[38] Lanza M, Marinescu R. Object-oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems. Springer-Verlag, 2006.

[39] Kowalski G. Information Retrieval Systems: Theory and Implementation. Kluwer Academic Publishers, 1997.

[40] Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proc. of the 33rd Int'l Conf. on Software Engineering (ICSE). 2011. 1−10.

[41] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in Software Engineering. Springer Science & Business Media, 2012.

[42] Vargha A, Delaney HD. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics, 2010,25(2):101−132.

[43] Barros MO, Dias-Neto AC. Threats to validity in search-based software engineering empirical studies. Technical Report, No. 0006/2011, Rio de Janeiro, 2011. http://www.seer.unirio.br/index.php/monografiasppgi/article/viewFile/1479/1307

[44] Dam HK, Reder A, Egyed A. Inconsistency resolution in merging versions of architectural models. In: Proc. of the 11th Working IEEE/IFIP Conf. on Software Architecture (WICSA). 2014. 153−162.

[45] Reder A, Egyed A. Determining the cause of a design model inconsistency. IEEE Trans. on Software Engineering, 2013,39(11): 1531−1548.

[46] Egyed A. Automatically detecting and tracking inconsistencies in software design models. IEEE Trans. on Software Engineering, 2011,37(2):188−204.

[47] Cui XF, Sun YC, Mei H. Decision-centric software architecture design method. Ruan Jian Xue Bao/Journal of Software, 2010, 21(6):1196−1207 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3588.htm [doi: 10.3724/SP.J.1001.2010.03588]

[48] Lytra I, Tran H, Zdun U. Constraint-based consistency checking between design decisions and component models for supporting software architecture evolution. In: Proc. of the 16th European Conf. on Software Maintenance and Reengineering (CSMR). 2012. 287−296.

[49] Lytra I, Tran H, Zdun U. Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations. In: Proc. of the 7th European Conf. on Software Architecture (ECSA). 2013. 224−239.

[50] Abi-Antoun M, Aldrich J, Nahas N, Schmerl B, Garlan D. Differencing and merging of architectural views. Automated Software Engineering, 2007,15(1):35−74.

[51] Herold S, Mair M. Recommending refactorings to re-establish architectural consistency. In: Proc. of the 8th European Conf. on Software Architecture (ECSA). 2014. 390−397.

[52] Khan I, Haider S. On building a consistent framework for executable systems architecture. Journal of Systems and Software, 2014, 98(12):155−171.

[53] Spyns P, Meersman R, Jarrar M. Data modelling versus ontology engineering. ACM SIGMod Record, 2002,31(4):12−17.

[54] Rumbaugh J, Jacobson I, Booch G. Unified Modeling Language Reference Manual. Pearson Higher Education, 2004.

[55] Jensen K, Kristensen LM. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Science & Business Media, 2009.

[56] Erl T. Service-oriented Architecture: Concepts, Technology, and Design. Pearson Education India, 2005.

[57] MacKenzie CM, Laskey K, McCabe F. Reference Model for Service Oriented Architecture 1.0. OASIS Standard, 2006.

[58] Weinreich R, Buchgeher G. Automatic reference architecture conformance checking for SOA-based software systems. In: Proc. of the 11th Working IEEE/IFIP Conf. on Software Architecture (WICSA). 2014. 95−104.

[59] Li Z, Gong DW, Nie CH, Jiang H. The research progress and development trend of search-based software engineering. In: Computer Science and Technology Discipline Development Report (2014−2015). Beijing: China Science and Technology Press, 2016 (in Chinese).

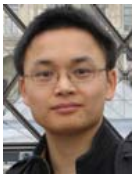
附中文参考文献:

[1] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[47] 崔晓峰,孙艳春,梅宏.以决策为中心的软件体系结构设计方法.软件学报,2010,21(6):1196−1207. http://www.jos.org.cn/1000-9825/3588.htm [doi: 10.3724/SP.J.1001.2010.03588]

[59] 李征,巩敦卫,聂长海,江贺.基于搜索的软件工程研究进展与趋势.见:计算机科学技术学科发展报告(2014−2015).北京:中国科学技术出版社,2016.

徐永睿(1987－),男,四川成都人,博士,主要研究领域为软件体系结构设计优化,基于搜索的软件工程.



梁鹏(1978－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件体系结构,知识驱动的软件工程.