计算机科学
COMPUTER SCIENCE

# 面向风格的软件体系结构演化路径生成方法

钟林辉，杨超逸，夏子豪，黄淇轩，屈乔乔，李方云，孙文彬

引用本文

钟林辉，杨超逸，夏子豪，黄淇轩，屈乔乔，李方云，孙文彬. 面向风格的软件体系结构演化路径生成方法[J]. 计算机科学，2024，51(11A)：240100130-9.

ZHONG Linhui, YANG Chaoyi, XIA Zihao, HUANG Qixuan, QU Qiaoqiao, LI Fangyun, SUN Wenbin. Style-oriented Software Architecture Evolution Path Generation Method[J]. Computer Science, 2024, 51(11A): 240100130-9.

---

**相似文章推荐（请使用火狐或 IE 浏览器查看文章）**

**Similar articles recommended (Please use Firefox or IE to view the article)**

基于小生境算法的空气质量模糊认知图预测

Air Quality Fuzzy Cognitive Map Forecasting Based on Niche Genetic Algorithm

计算机科学, 2024, 51(11A): 240300120-6. https://doi.org/10.11896/jsjkx.240300120

基于EBRCG的API结构模式信息增强方法研究

Study on Information Enhancement Method of API Structural Pattern Based on EBRCG

计算机科学, 2024, 51(11A): 230900121-10. https://doi.org/10.11896/jsjkx.230900121

基于遗传算法的低碳导向的物流中心配送优化

Optimization of Low-carbon Oriented Logistics Center Distribution Based on Genetic Algorithm

计算机科学, 2024, 51(11A): 231200035-6. https://doi.org/10.11896/jsjkx.231200035

基于改进遗传算法的家庭用电调度优化方法

Scheduling Optimization Method for Household Electricity Consumption Based on Improved Genetic Algorithm

计算机科学, 2024, 51(6A): 230600096-6. https://doi.org/10.11896/jsjkx.230600096

基于深度强化学习的自学习排课遗传算法研究

Study on Genetic Algorithm of Course Scheduling Based on Deep Reinforcement Learning

计算机科学, 2024, 51(6A): 230600062-8. https://doi.org/10.11896/jsjkx.230600062

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240100130

# 面向风格的软件体系结构演化路径生成方法

钟林辉　杨超逸　夏子豪　黄淇轩　屈乔乔　李方云　孙文彬
江西师范大学计算机信息工程学院　南昌 330022

**摘　要**　软件体系结构风格是对软件通用结构的泛化,软件的结构风格通常与结构特征密切相关,通过向某种风格演化能够使其软件的结构特征更加明显。传统的面向软件体系结构风格的演化方法在构建演化路径时,需要人工构造目标软件体系结构,因此缺少自动化的支持。目前,亦未提出针对软件体系结构风格的度量方法。因此,文中以正交化软件体系结构风格为例,提出了一种遗传算法与规划领域定义语言(Planning Domain Definition Language-PDDL)相结合的软件体系结构风格演化路径生成方法。该方法提出了一种基于语义相似度的遗传变异算子和正交软件体系结构风格的度量方法,提出了软件体系结构与PDDL的映射规则。实验证明,提出的遗传变异算子相比通用变异算子能更好地提升算法前期的收敛效率,正交软件体系结构风格演化完成后,软件的计算变动代价、正交风格距离以及McCabe度量等指标得以改善。

**关键词**:软件体系结构;软件体系结构风格;遗传算法;软件演化;基于搜索的软件工程
**中图分类号**　TP311

## Style-oriented Software Architecture Evolution Path Generation Method

ZHONG Linhui,YANG Chaoyi,XIA Zihao,HUANG Qixuan,QU Qiaoqiao,LI Fangyun and SUN Wenbin
College of Computer and Information Engineering,Jiangxi Normal University,Nanchang 330022,China

**Abstract**　Software architecture style is a generalization of the common structure of software,and the structure style of software is usually closely related to the structural characteristics. By evolving to a certain style,the structural characteristics of the software can be more obvious. Traditional software architecture style evolution methods not only require manual construction of the target software architecture when building the evolution path,which lack the automation support,but also no measurement method for software architecture style has been proposed. Therefore,this paper takes orthogonal software architecture style as an example and proposes a software architecture style evolution path generation method that combines genetic algorithm and planning domain definition language(PDDL). This method proposes a genetic mutation operator based on semantic similarity and a measurement method for orthogonal software architecture style,and proposes the mapping rules between software architecture and PDDL. Experiments show that the proposed genetic mutation operator can better improve the convergence efficiency of the algorithm in the early stage,and after the orthogonal software architecture style evolution is completed,the software is improved in terms of change cost,orthogonal style distance and McCabe measurement.

**Keywords**　Software architecture,Software architecture style,Genetic algorithm,Software evolution,Search-based software engineering

## 1　引言

软件体系结构风格(SA)是对某一类应用领域系统组成方式的常见模式,作为“可复用的组织模式和习语”[1]。成功的应用程序会随着时间的推移发生软件体系结构演化,其结构会发生偏移。为了应对这种偏移,需要对软件体系结构进行重构,并检查其是否仍然有效[2]。随着SA的研究已经扩展到软件生命周期的每个阶段,SA重构问题开始受到关注。软件体系结构的重构会产生一系列的版本,形成软件演化路径。

与此同时,软件体系结构中的演化路径也成为了一个重要的研究方向,它描述了软件系统在其生命周期中经历的变化和发展的轨迹。具体而言,演化路径描述了软件系统从起始 $SA_0$ 到目标 $SA_k$ 的连续变化和演化过程。本文认为 $SA$ 重构问题中的演化路径是由多个里程碑 $SA_i$(其中 $0\leqslant i\leqslant k$,其中 $SA_0$ 为起始 $SA$,$SA_k$ 为目标 $SA$)共同构成演化路径上的关键结点,关键结点之间存在着动作序列描述具体的演化过程。

Mancoridis 等是第一个在重构软件体系结构设计方面使用基于搜索的软件工程方法(Search Based Software Engi-

基金项目:国家自然科学基金(62062039,61966017);江西省自然科学基金(20212BAB202017,20224BAB202013,20212BAB202018);校教改课题(JXSDJG2044)
This work was supported by the National Natural Science Foundation of China(62062039,61966017),Natural Science Foundation of Jiangxi Province,China(20212BAB202017,20224BAB202013,20212BAB202018) and School Education Reform Project(JXSDJG2044).
通信作者:钟林辉(shiningto@jxnu.edu.cn)

neering,简称 SBSE ),将软件体系结构问题转换为优化问题[3]。使用软件体系结构的指标作为待优化的适应度函数(fitness),用于寻找到指标优秀的软件体系结构。他们的研究针对算法本身以及 SA 搜索问题与算法的适配。本文认为通过向优化算法的待优化值中引入 SA 风格距离指标可以得到一系列用于寻找目标风格 SA 的算法。Barnes 等较早提出在软件体系结构层通过 PDDL 规划 SA 演化过程来减少 SA 演化中的代价[4]。Chondamrongkul 等随后提出利用 PDDL 向具有特定风格的 SA 演化并对演化过程中的 SA 进行形式化验证,从而保证了演化的每一个中间 SA 的正确性[5]。在他们的研究中主要考虑 $SA_0$ 和 $SA_k$ 已知的情况,通过定义域文件与问题文件,再由 PDDL 解释器生成 SA 演化序列。他们的方法需要给定 $SA_k$,且本文发现该方法在处理较大规模 SA 演化问题时可能会出现解释器无法求解的问题。

为此,本文提供了一种遗传算法与规划领域定义语言(Planning Domain Definition Language-PDDL)相结合的 SA 风格转换方法(GA-PDDL 方法)。在该方法中,本文专注于 SA 风格转换,由优化算法找到合适的里程碑 $SA_i$ 再通过 PDDL 找到合适的演化路径。本文的贡献概括如下:

1)提出了 GA-PDDL 方法,该方法以 $SA_0$ 为起点寻找更接近软件体系结构风格的里程碑 $SA_i$,通过多轮迭代达到满足软件体系结构风格的 $SA_k$。

2)提出了一种用于针对软件体系结构演化的基于语义相似度的变异算子,该算子可加快算法前期的收敛速度。

3)构造用于表示 $SA_i$ 与 $SA_{i+1}$ 差异的反射模型,通过裁剪无关结构缩减 PDDL 问题的规模。

本文第 2 章介绍了相关工作;第 3 章详细描述了 GA-PDDL 方法;第 4 章介绍了实验的设计与结果;最后总结全文。

## 2 相关工作

### 2.1 基于搜索的软件工程方法

采用基于搜索的软件工程方法处理软件体系结构重构问题首次由 Mancoridis 等[3]提出,该方法主要是通过将软件体系结构重构问题转换为基于搜索的优化问题并且通过元启发算法来解决。此后,Mahdavi 等[6]将软件系统转换为加权和非加权的构件连接图,并使用多峰优化器来寻找基于模块化指标(MQ)作为适应度函数的良好划分解决方案。Abdeen 等[7]于 2008 年实现了一种基于启发式搜索的方法来自动优化包间连接,通过最小化直接包循环依赖来优化软件的包结构。Prajapati 等[8]于 2020 年提出了使用和声搜索算法寻找多种指标较优的结构。此类问题的解决重心在于寻找能良好反映 SA 的指标以及改进相关的算法。

### 2.2 软件体系结构规划

Barnes 等[4]首先提出基于 PDDL 规划软件演化的方法,自动生成 SA 的演化路径。它通过预先定义软件演化过程中的动作、动作的约束以及动作的代价,在给定起始 SA 与目标 $SA_k$ 的 PDDL 描述基础上利用 PDDL 解释器生成代价较小的演化序列,以此处理 SA 演化路径规划问题。Lin 等[9]于 2016 年开发了一套工具 Refactoring Navigator,它以给定的实现为起点,以所需的高级设计为目标,建立用于表示 SA 差异的反射模型并通过爬山法迭代推荐一系列重构步骤,重构过程中使用用户的反馈交互式的推荐重构步骤。2021 年 Chondamrongkul 等[5]提出了一种重构 SA 设计和规划演化过程的自动化方法,通过对 PDDL 生成的演化路径上的中间状态进行形式化方法验证来保证结构迁移的有效性,即主要采用两种方式,一种是通过 PDDL 规划,另一种是通过反射模型利用爬山法推荐重构步骤。

## 3 GA-PDDL 方法

本文提出的 GA-PDDL 方法包括两个阶段:遗传算法搜索最优种群阶段和 PDDL 演化任务规划阶段。前者寻找里程碑 SA,后者规划 $SA_i$ 与 $SA_{i+1}$ 的动作序列。在 GA-PDDL 方法中两个阶段轮换进行,其过程可视为:寻找里程碑 $SA_1$、规划 $SA_0$ 到 $SA_1$ 之间的演化路径、寻找下一轮里程碑 $SA_2$、规划 $SA_1$ 到 $SA_2$ 的演化路径…直至开发人员认为里程碑 $SA_k$ 的 $SA$ 风格明显时可认为当前的 $SA_k$ 为目标 SA。最终得到的演化路径形式如图 1 所示,演化路径由 $SA$ 的关键节点以及节点之间的动作序列共同组成,并且在路径中可能存在多种里程碑方案。

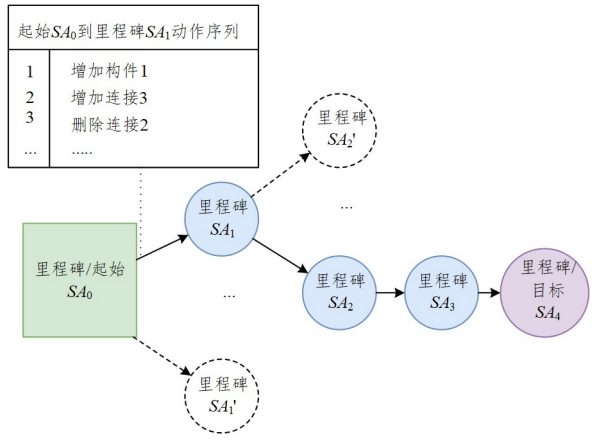



图 1 演化路径示例

Fig. 1 Example of evolution path

遗传算法搜索最优种群阶段的任务是对问题进行建模并通过遗传算法搜索出最优种群,这一阶段需要重点研究染色体编码方式和适应度函数的设计、起始种群的初始化、遗传算法相关算子的设计及执行终止条件的定义等,最终生成最优解种群作为本阶段的 $SA_{i+1}$;PDDL 演化任务规划阶段的任务是通过反射模型[10]生成软件演化的 PDDL 问题描述,最终利用 PDDL 解释器得到从上一轮 $SA_i$ 到里程碑 $SA_{i+1}$ 的动作序列。

### 3.1 基本思想与整体流程

传统的 SA 风格转换方法主要是使用 UML 通过关注结构中的分离点来设计相应结构,并使之具有相应的结构风格。GA-PDDL 方法的思想则是首先假设构件是由多个不可再分的程序元素(Program Element)组成,其中一个程序元素调用另一个程序元素从而形成构件之间的连接,并将构件的调整看作是程序元素在构件之间的转移,由此软件体系结构风格转换问题转化为可以被遗传算法处理的组合优化问题。针对这个问题初始化种群、迭代,最终得到最优种群,最后将最优

种群的 SA 作为里程碑 SA,将其与起始 $SA_0$ 共同构造为用于表示 SA 差异的反射模型,利用反射模型提取差异部分转换为 PDDL 问题描述,随后由 PDDL 解释器得到从 $SA_i$ 到风格距离更小的 $SA_{i+1}$ 的演化路径。

但在求解之前需要解决如下 3 个问题:

1)如何评价一个候选解在开销最小的同时最接近 SA 风格,即 fitness 函数的设计问题。

2)如何将 $SA_i$ 与 $SA_{i+1}$ 的差异映射到 PDDL 的问题描述。

3)如何定义 SA 演化路径中的动作序列类型,即 PDDL 的动作设计问题。

对于问题 1,本文基于图编辑距离[11]定义了变动代价,记作 EP(Evolutionary Penalty)作为变动的开销,并在此基础上针对每一处变动所影响的代码计算行数作为变动的权值;对于 SA 风格相似度的评估,本文定义了风格距离,记作 SD(Style Distance)。对于问题 2,本文通过构造 $SA_i$ 与 $SA_{i+1}$ 的反射模型后,删除未发生变更部分,再生成 PDDL 问题中的起始状态与目标状态。对于问题 3,本文的操作对象为 SA,其中的成分有构件、接口、连接,对于这 3 种成分的操作可定义为插入构件、删除构件、插入 provide 接口、删除 provide 接口、插入 require 接口、删除 require 接口、插入连接和删除连接这 8 种。

GA-PDDL 方法的核心在于定义 SA 风格的指标和定义合理的 PDDL 动作,图 2 给出了整个方法的流程图,其共分为两个阶段,第 1 个阶段首先使用 $SA_i$ 初始化种群;然后进行迭代执行交叉、变异、选择算子,生成新个体并加入种群中,如果达到迭代上限,则将得到 $SA_{i+1}$。然后进入第 2 阶段,生成能表现 $SA_i$ 与 $SA_{i+1}$ 差异的反射模型,最后生成 PDDL 问题文件并配合 PDDL 域文件使用解释器进行求解。

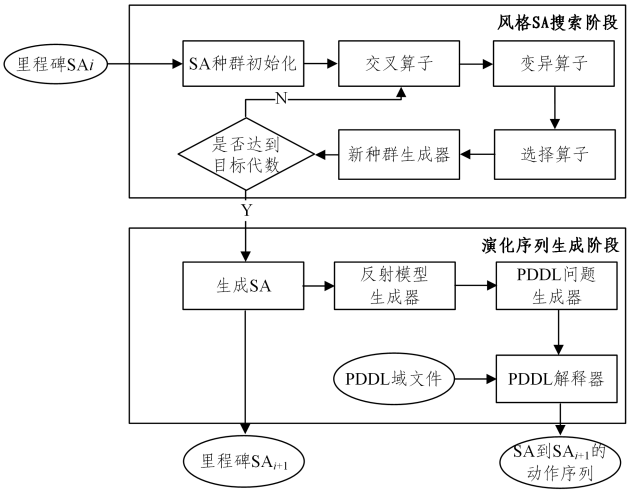



图 2 GA-PDDL 方法的流程图

Fig. 2 Flowchart of GA-PDDL method

## 3.2 遗传算法搜索最优种群阶段

本节主要介绍 GA-PDDL 方法的第 1 阶段计算过程,首先介绍对 SA 的编码方式,然后给出具体的适应度值函数,最后描述搜索的整体流程并介绍基于语义趋向的变异算子。

GA-PDDL 方法首先构造一个维数为程序元素(Program Element)总数的自然数向量,该向量表示一个 SA 中的构件组成方式。其中向量下标代表程序中的一个组成元素,对应下标的值代表这个程序元素所属的构件;设 SA 由 $m$ 个构件组成,这 $m$ 个构件又由 $n$ 个程序元素组成,则所有可能的解有 $n^m$ 个。目标就是去寻找开销最小和 SA 风格最接近的解,以得到合理的重构目标。这便是将传统的 SA 风格转换问题转化为一类搜索问题,可用常见的遗传方法来求解。

### 3.2.1 编码

GA-PDDL 方法将 SA 中的程序元素表示为一个自然数向量:

$$\boldsymbol{S}=\{p_1,p_2,\cdots,p_n\},p_i\in[0,m+t) \tag{1}$$

其中,$n$ 为当前 SA 中程序元素数量,$p_j$ 表示 SA 中第 $j$ 个程序元素所属的构件号,$m$ 代表起始 $SA_0$ 中的构件数,$t$ 表示可能出现的新构件的数量。例如 $S=\{3,2,3\}$,分别表示 0 号程序元素存在于构件 3 中,1 号程序元素存在于构件 2 中,2 号程序元素存在于构件 3 中。即一条染色体可完整地表示出一个 SA。

### 3.2.2 适应度函数

本文采用变动代价(EP)和风格距离(SD)两个指标作为 fitness。变动代价代表传入算法的 $SA_i$ 到 $SA_{i+1}$ 的差异大小。风格距离代表 SA 风格的接近程度。两个指标都是越小越好,对这两个指标进行加权求和,可以表达为让遗传算法在变动代价尽可能小的情况下最接近 SA 风格。

为使用变动代价(EP),需要定义节点与边关于插入、删除操作的编辑代价。在遗传算法中,设迭代 j 代,每一代群体数为 $m$,个体 SA 的构件连接图中边数为 $n$,若直接采用图编辑距离的算法则会导致算法时间代价接近 $O(jmn^3)$。因此本文中节点、边以及编辑代价的定义与图编辑距离指标有部分差异。

**定义 1**(软件体系结构图) $G=(V,E)$ 由顶点的非空集 $V$ 和有向边集 $E$ 构成。每条边有两个顶点与它相连,这样的顶点称为端点。在软件体系结构的语境下,节点集合和边集合的定义如下。

**定义 2**(节点集合 $V$) 表示软件体系结构中的构件、每个构件对外提供的接口以及对外所需的接口。

**定义 3**(边的集合 $E$) 表示节点之间的关系,根据构件、接口之间的关系,可以区分为以下几种关系:接口之间的连接关系、接口之间的映射关系、构件和接口之间的对外所需的功能和对外提供的功能。

**定义 4**(节点之间的编辑代价) 区分为插入代价、删除代价、替换代价 3 种,即:

$$c(\varepsilon\rightarrow V_i)=c(V_i\rightarrow\varepsilon)=codeOfLine(V_i) \tag{2}$$

$$\begin{aligned}c(V_i\rightarrow V_j)&=c(V_i\rightarrow\varepsilon)+c(\varepsilon\rightarrow V_j)\\&=codeOfLine(V_i)+codeOfLine(V_j)\end{aligned} \tag{3}$$

**定义 5**(边之间的编辑代价) 区分为插入代价、删除代价、替换代价 3 种,即:

$$c(\varepsilon\rightarrow E_i)=c(E_i\rightarrow\varepsilon)=\sum_{k\in\{k|V_k\in E_i\}}codeOfLine(V_k) \tag{4}$$

$$c(E_i\rightarrow E_j)=c(E_i\rightarrow\varepsilon)+c(\varepsilon\rightarrow E_j)=\sum_{k\in\{k|V_k\in\{E_i,E_j\}\}}codeOfLine(V_k) \tag{5}$$

**定义 6**(软件体系结构变化性度量 $EP(S_i,S_{i+1})$) 表示软件系统 $S$ 的软件体系结构从第 $i$ 版本到第 $j$ 版本的变化度

量值。EP 的计算式如下：

$$EP(S_i,S_{i+1})=\sum_{k\in\{k|V_k\in\Delta S\}}codeOfLine(V_k)+\sum_{k\in\{k|V_k\in E_j,E_j\in\Delta S\}}codeOfLine(V_k) \quad (6)$$

为使用风格距离(SD)，本文以一种正交体系结构风格为例，给出了正交化体系结构的风格距离。

如图 3 所示，正交体系结构风格的图形表示具有直观性，易于转化为数学描述。这种数学描述有助于深入理解系统的组成部分、交互关系和功能分配。并且正交体系结构风格具有模块化和分层、独立性、标准化接口、可扩展性以及清晰的责任分配等特点。

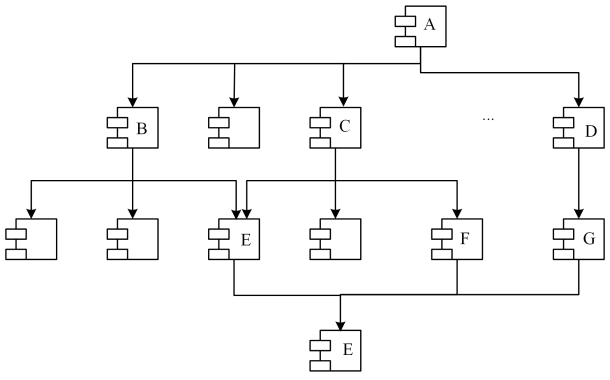



图 3 正交体系结构

Fig. 3 Orthogonal architecture

正交体系结构由组织层与线索的组件构成[12]。同一层的组件之间的连接较少或完全没有，例如 $B,C,D$ 同为第 2 层。系统中从顶层至底层的一条连接路径可视为线索，例如 $A,B,E,H$ 为一条线索。在正交体系结构中线索被视为系统的一部分功能，如果线索与线索之间没有任何相互的调用，我们可以称这个结构为完全正交的体系结构。我们可以找到正交体系结构风格有 3 个特征：1)不存在没有线索的构件；2)同一层之间的连接越少越好；3)相交线索越少越好。由此，正交体系结构的风格距离如下：

$$SD(S_j)=\sum_{i=1}^{3}w_iv_i \quad (7)$$

其中，$v_1$ 为孤立构件数，$v_2$ 为同一层构件间的连接数，$v_3$ 为线索相交数；$w$ 为权值。即，如果当前 SA 中的孤立构件数、同一层构件间连接数、线索相交数减少，SD 会相应减少，SA 也越接近正交体系结构风格。值得注意的是，正交体系结构风格层的划分可视为构件连接图中进行生成树算法时的层划分，本文采用文献[13]中构建 BFS 的相关技术得到 SA 的层次结构。

最终的 fitness 由式(6)和式(7)可得：

$$fitness(S_i,S_{i+1})=W_1\times EP(S_i,S_{i+1})+W_2\times SD(S_{i+1}) \quad (8)$$

其中，$W_1,W_2$ 为正权值。$W_1$ 与 $W_2$ 比值越大，实现同样的 SA 风格转换时里程碑更少，但得到的演化路径每一轮推进的动作序列会更多。反之，$W_1$ 与 $W_2$ 比值越小实现同样的 SA 风格转换时里程碑更多，但得到的演化路径每一轮推进的动作序列会更少。因此，当开发人员希望以较少的里程碑实现 SA 风格转换时可设置较小的 $W_1$ 和较大的 $W_2$。当开发人员希望每一个里程碑变动较小，通过更多的里程碑实现风格转换时可设置较大的 $W_1$ 与较小的 $W_2$。

由式(8)可知，如当前的 SA 与 $SA_i$ 差异越大，fitness 会增大，当前的 SA 越接近正交体系结构风格，fitness 会减少。即 fitness 越小，代表可以用较小的编辑代价得到一个更接近正交体系结构的 SA。

### 3.2.3 遗传操作算子及最优种群搜索过程

如 3.2.2 节所述，GP-PDDL 方法中的 fitness 的计算式可计算出每一个 SA 从起始到正交体系结构的变动代价以及正交体系结构的风格距离。本节所述方法就是对构件组成方式进行搜索，找出其中变动代价较小且 SA 风格距离较小的种群，不断重复得到中间里程碑 SA，直到最终得到满足正交风格的 $SA_k$。

本文采用的遗传算法由 $SA_i$ 作为初始种群开始。初始种群产生后，按照适者生存和优胜劣汰的原理，逐代地演化以产生新的个体，并保留更好的个体进入下一代。在每一代中，根据个体的适应度值来选择个体进行均匀交叉和基于语义相似度的突异，从而产生新的个体。这一过程迭代进行，直到达到终止条件，终止条件预设为迭代次数等。最终代种群中的最优个体经过解码，作为问题的近似最优解。

**算法 1** GA-PDDL 目标 $SA_k$ 搜索算法

输入：$S_0\leftarrow SA_i$

输出：$SA_{i+1}\leftarrow S_g$

```
1. pop←Initialze(S0)
2. fitnessCaculate(pop)
3. eliteSolution←pop.getEliteSolution()
4. size←pop.size()
5. i←0
6. while(i<Ngen)
7.     offspring←Ø
8.     while(offspring.size()<size)
9.   p1,p2←select(2,pop)
10.      p1,p2←crossover(p1,p2,Pcrossover)
11.      p1,p2←mutate(p1,Pmutate),mutate(p2,Pmutate)
12.      offspring.add(c1,c2)
13.   pop←eliteSolution∪offspring
14.   fitnessCaculate(pop)
15.   eliteSolution←pop.getEliteSolution()
16.   i←i+1
17. Sg←resolve(pop)
```

### 3.2.4 基于语义相似度的变异算子

为使算法进一步增强变异的有效性，利用当前待变异的程序元素与其他构件的语义相似度与所有相似度之和的比值作为变异至某一构件的概率。最后采用赌轮盘的方式选择要变异到的构件号。

**算法 2** 基于语义相似度的变异算子

输入：当前后代 p，变异率 $p_{mutated}$

输出：变异后的后代

```
1. pmutated←p.copy()
2. i←0
3. while(i<p.length)
4.     if(random(0,1)<Pmutate)
5.         j←0,similarities←Ø
```

```
6.      while(j<component.size())
7.        sim←SIM(class.get(i),component.get(j))
8.        similarities.put(j,(sim+2)³)
9.        j←j+1
10.     r←rand(0,similarities.values().sum())
11.     t←0,m←0
12.     while(t≤r)
13.       t←t+similarities.get(m)
14.       m←m+1
15.     p_mutated[i]←m−1
```

其中 SIM 主要用于计算 $i$ 号程序元素与 $j$ 号构件的语义相似度。本文认为构件的功能基本可以由构件的接口名判断,因此构件可以近似看作由构件内各个接口名字组合的一段语句,这段语句中包含了构件的功能。程序元素也可以近似看作由程序元素下所有函数名组成的一句话。由此可通过构件的接口计算出构件的段向量,以及程序元素的句向量。

SIM 的计算方式主要通过面向软件工程领域的 word2vec 模型[14]将每个词转换为词向量,其次通过平滑倒词频(SIF)[15]计算构件的段向量以及程序元素的句向量,最后计算二者的余弦相似度。SIM 算法如算法 3 所示。

**算法 3** 构件与程序元素的语义相似度算法

```
输入:程序元素 c,构件 j
输出:相似度 similarity
1. cNames←c.getMethodsName()
2. cWords←camelWordSpliter(cNames)
3. i←0,cWordsVec←∅
4. while(i<cWords.length)
5.     cWordsVec.add(word2vec.get(cWords[i]))
6.     i←i+1
7. jName←j.getInterfacesName()
8. jWords←camelWordSpliter(jName)
9. i←0,jWordsVec←∅
10. while(i<jWords.length)
11.     jWordsVec.add(word2vec.get(jWords[i]))
12.     i←i+1
13. similarity←cos(SIF(cWordsVec),SIF(jWordsVec))
```

### 3.3 PDDL 演化任务规划阶段

本节主要介绍 GA-PDDL 方法的第 2 阶段计算过程。PDDL(Planning Domain Definition Language)是一种用于描述计划问题领域的形式语言。它在人工智能领域中被广泛应用于描述问题领域的状态、动作和目标,以便让规划算法能够解决各种问题。PDDL 的输入主要分为域(domain)文件和问题(question)文件,首先介绍用于表示 SA 操作的域文件,其次介绍用于描述问题状态的问题文件。

GA-PDDL 方法中,第 1 阶段的输出是一个满足软件体系结构风格的里程碑 SA。第 1 阶段在搜索的过程中 SA 会逐渐向一个较好的 fitness 演化,但并非所有的演化都是向好的 fitness 演化,算法迭代过程中可能会出现部分个体 fitness 变坏,它们有较小概率会存活至下一代用于克服算法中的局部较优解,因此 GA 在处理 SA 的演化路径是曲折的。并且由于算法中存在均匀交叉过程,而这一过程体现在 SA 上则是随机地提取两个 SA 的部分特征用于构造新的 SA,这导致要从 GA 中提取最终较优个体的演化路径更加困难。因此,本文结合 GA 找到满足目标风格的 SA,并构造 $SA_i$ 与 $SA_{i+1}$ 的反射模型使用 PDDL 进行规划得到演化路径上的演化动作序列。

从 $SA_i$ 与 $SA_{i+1}$ 的演化分为多步,其中存在多种操作,如增加构件、删除构件、增加 Provide 接口等。这些操作存在着一定的约束,如在删除构件时构件中的接口应事先删除,接口能否立即删除又与当前接口上有无连接相关。因此,本文需要对 SA 的各项操作进行定义。然后构造一个用于表示 SA 差异的反射模型,并利用反射模型删减模型中未发生变动的构件与连接并生成简化的 PDDL 问题,最后由 PDDL 解释器得到演化动作序列。

#### 3.3.1 域文件定义

如上所述,PDDL 中的域文件用于描述当前问题中的对象、谓词、函数以及动作。本节主要介绍针对 SA 演化问题的域文件定义。

首先是对象定义,如下所示:

```
(:types
    Element SA-objects
    Component Interface-Element
    ProvideInterface RequireInterface-Interface
)
```

在本文中针对 SA 实体定义为对 SA 类型,其他 SA 的所有组成元素均为 Element 类型,组成元素主要有构件 Component 和构件接口 Interface,而 Interface 又可以分为两种分别为对外部构件提供的接口 ProvideInterface 和构件产生依赖的接口 RequireInterface。这里本文将接口分为 Provide 和 Require 类型,主要用于区分接口是调用类型还是提供类型。

其次是谓词定义,如下所示:

```
(:predicates
  (interface-connect ?interfaceA-RequireInterface ?interfaceB-ProvideInterface)
  (has-provide-interface ?component-Component ?provideInterface-ProvideInterface)
  (has-require-interface ?component-Component ?requireInterface-RequireInterface)
  (is-in ?component-Component ?sa-SA)
)
```

本文使用 4 种谓词用于描述 SA 中元素所处的状态,首先 is-in 代表构件存在于 SA 中,其次 has-provide-interface 以及 has-require-interface 代表软件中存在接口,最后 interface-connect 代表两个接口之间存在连接。值得注意的一个细节是,PDDL 不允许动作创建新对象,许多演化都涉及创建新的架构元素或废除现有元素[4]。在这种情况下本文通过定义谓词来定义构件、接口和连接的存在。

函数的定义如下所示:

```
(:functions
  (modify-cost)
  (provide-interface-connect-num ?interface-ProvideInterface)
  (require-interface-connect-num ?interface-RequireInterface)
  …
```

```
  (add-component-cost)
  (del-component-cost)
  ...
)
```

本文定义了 11 个函数。这里要注意，区别于常规编程语言，这里的函数是用于存储值。首先是 modify-cost 代表执行 PDDL 的动所带来的变更代价，其次是 interface-connect-num 代表接口所存在的连接数，再次是 interface-num 代表构件上存在的接口数，最后是各项操作的代价。其中存放端口数以及连接数的函数用于约束构件以及端口的删除操作。

最后是动作定义，如下所示：

```
;为构件增加 require 接口
(:action add-require-interface-to-component
  :parameters(?component-Component ?interface -RequireInterface)
  :precondition(and
  ;当前接口未连接
  (forall(?provideInterface-ProvideInterface)
  (not(interface-connect ?interface ?provideInterface)))
  ;当前接口未被分配
    (forall(?every-component-Component)
    (not(has-require-interface ?every-component ?interface)))
  )
  :effect(and
  ;增加接口
  (has-require-interface ?component ?interface)
  ;增加构件的接口数
    (increase(require-interface-num ?component) 1)
  ;初始化接口连接数
  (assign(require-interface-connect-num ?interface)0)
  ;增加代价
  (increase(modify-cost)(add-interface-cost))
  )
)
(:action add-provide-interface-to-component..)
(:action del-require-interface-to-component..)
(:action del-provide-interface-to-component..)
(:action add-connection..)
(:action del-connection..)
(:action add-component..)
(:action del-component..)
```

本文定义了共 8 种动作，分别为增加 require 接口，增加 provide 接口，删除 require 接口，删除 provide 接口，增加连接，删除连接，增加构件，删除构件。这里以增加接口为例：在增加接口时需要指定接口名以及要分配到的构件名，动作执行的前提是当前接口未被分配且不存在连接，最终改变当前的 SA 状态，增加接口、构件上接口数、初始化接口连接数、增加动作代价。

### 3.3.2 问题文件定义

本节主要介绍问题文件的生成，其中问题文件要定义的分为 4 处。首先要定义问题的实体对象；即当前演化问题中出现的构件和接口对象；其次需要初始化当前问题中的变动代价以及各个动作执行的代价；然后要给出当前 SA 的状态，即 $SA_i$ 的谓词描述；最后给出 $SA_{i+1}$ 的状态。

通过 SA-PDDL 第 1 阶段可得 $SA_{i+1}$ 为提取二者的差异，用于构造 PDDL 问题描述，本文通过构造反射模型将其与 $SA_i$ 进行比对可得到如图 4 所示的模型。原始的反射模型能很好地体现前后 SA 的差异，但其中包含大量的无关信息，如图 4 中的 comp2 和 comp6 在 SA 前后差异中显示无变化。为突出 SA 的差异，可以删减反射模型中未发生变更的部分，可得到如图 5 所示的反射模型。

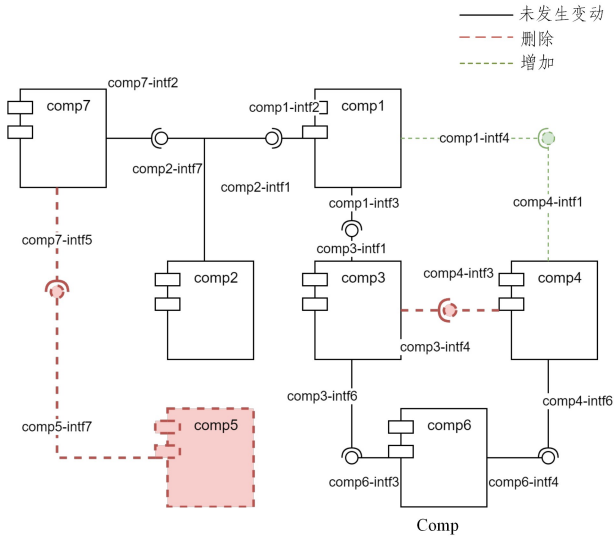



图 4 反射模型

Fig. 4 Reflective model

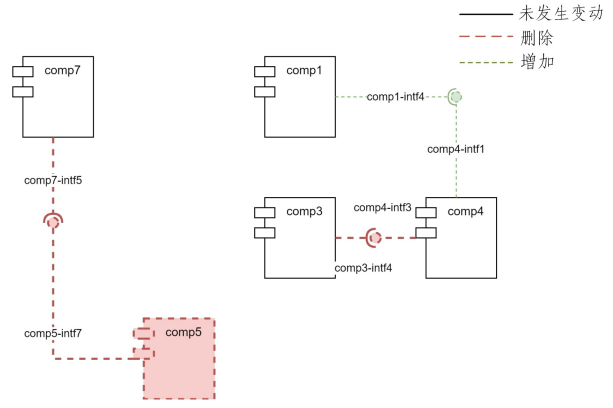



图 5 删减后的反射模型

Fig. 5 Reduced reflective model

首先是对象声明，依照反射模型可生成 PDDL 问题文件，以图 5 为例首先应该提取出模型中的所有对象，即反射模型中的所有构件及接口，要注意的是 PDDL 中无法通过动作进行创建或删除，本文通过谓词来定义对象的存在状态或删除状态，故在对象 objects 部分要声明所有出现过的对象：

```
(:objects
  comp1 comp3 comp4 comp5 comp7-Component
  comp1-intf3 comp1-intf4 comp3-intf4 comp7-intf5 -RequireInterface
  comp3-intf1 comp4-intf1 comp4-intf3 comp5-intf7 -ProvideInterface
)
```

其次提取出模型中的所有起始状态，即反射模型中除了增加的构件、接口和连接，其他的所有状态，以及初始化增删构件、接口和连接的代价，并提取出模型中所有增加的成份和

所有删除的成份作为目标。最后设置寻找能够最小化修改代价的动作序列。

```
(:init
;初始化总的修改代价
  (=(modify-cost) 0)
;初始化每种动作的代价
  ( =(add-component-cost) 50)
…
;存在 SA_i内的构件
(is-in comp1sa)
…
;起始的构件中存在的接口
  (has-require-interface comp1 comp1-interface3)
  (has-provide-interface comp3 comp3-interface1)
  (interface-connect comp1-interface3 comp3-interface1)
    …
)
(:goal(and
  (not(is-in comp5sa))
;SA_{i+1}中构件 1 和 4 存在用于连接的端口及连接
  (has-require-interface comp1 comp1-interface4)
  (has-provide-interface comp4 comp4-interface1)
  (interface-connect comp1-interface4 comp4-interface1)
…
)
  (:metric minimize(modify-cost))
```

### 3.3.3 动作序列生成

由于本文定义的域文件需要解释器支持 PDDL2.1 语法,本文最终选择 LPG-TD[16] 作为解释器。最后将以上定义的两个文件传入 PDDL 解释器可以生成如图 6 所示的结果。

```
  Time: (ACTION) [action Duration; action Cost]
0.0000: (DEL-CONNECTION COMP7-INTERFACE5 COMP5-INTERFACE7) [D:1.00; C:1.00]
0.0000: (DEL-CONNECTION COMP3-INTERFACE4 COMP4-INTERFACE3) [D:1.00; C:1.00]
0.0000: (ADD-PROVIDE-INTERFACE-TO-COMPONENT COMP4 COMP4-INTERFACE1) [D:1.00; C:10.00]
0.0000: (ADD-REQUIRE-INTERFACE-TO-COMPONENT COMP1 COMP1-INTERFACE4) [D:1.00; C:10.00]
1.0000: (DEL-PROVIDE-INTERFACE-TO-COMPONENT COMP4 COMP4-INTERFACE3) [D:1.00; C:10.00]
1.0000: (DEL-REQUIRE-INTERFACE-TO-COMPONENT COMP7 COMP7-INTERFACE5) [D:1.00; C:10.00]
1.0000: (DEL-REQUIRE-INTERFACE-TO-COMPONENT COMP3 COMP3-INTERFACE4) [D:1.00; C:10.00]
1.0000: (DEL-PROVIDE-INTERFACE-TO-COMPONENT COMP5 COMP5-INTERFACE7) [D:1.00; C:10.00]
1.0000: (ADD-CONNECTION COMP1-INTERFACE4 COMP4-INTERFACE1) [D:1.00; C:1.00]
2.0000: (DEL-COMPONENT COMP5) [D:1.00; C:50.00]
```

图 6 PDDL 运行结果

Fig. 6 PDDL execution results

通过分析不难发现,PDDL 解释器的结果能满足动作执行的逻辑。即这里的 Time 代表任务的先后时刻,Time 相同代表动作可同时执行,不会有冲突。第 0 时刻可以删除 comp7 到 comp5 以及 comp3 到 comp4 的连接,并且增加接口 comp4-interface1 和 comp1-interface4。第 1 时刻由于之前的连接已经删除,故删除无效的端口,并且由于 0 时刻 comp4 和 comp1 上用于连接的端口已经建立,因此 1 时刻可以建立连接。最后第 2 时刻删除 comp5,最终到达 $SA_{i+1}$。

## 4 实验设计和结果分析

为了对所提出的 GA-PDDL 方法的有效性进行适当的研究,建立了一个适当的实验。实验是在一台装有 ubuntu 20.4 64 位操作系统和英特尔酷睿 i5-6500 3.20 GHz CPU 的电脑上进行的。由于所提出的方法和现有的方法都是随机优化器,它们可能不会在同一问题实例的不同运行中产生相同的输出。因此,本文将针对 4 种不同的开源数据库项目进行处理,并应用 McCabe 度量[17]指标来评估软件架构重构目标的质量。最后本文分析 PDDL 的运行结果。

选择开源数据库项目作为实验样本的决定是基于其领域专注性、数据可靠性、研究目的和可比性等优势,以全面评估软件体系结构风格转换的效果和影响。未考虑其他类型项目的原因在于,开源数据库项目具有较高的复杂性和代表性,能更好地反映实际软件开发挑战,提高研究结果的实用性和推广性。

### 4.1 测试对象

本文为有效地评估 GA-PDDL 方法的有效性,选择了 4 种与数据库相关的基于 JAVA 的面向对象开源项目作为演化素材。为了避免实验结果的偏差,实验对象包括了具有不同规模和复杂程度的软件项目。测试样本信息如表 4 所列。

表 4 测试样本信息

Table 4 Test sample information

| 测试对象 | 版本号 | 类数 | 依赖数 |
|---|---|---|---|
| Cassandra | 0.7.0 | 812 | 2138 |
| H2 | 1.3.176 | 568 | 3611 |
| neo4j_kernel | 1.4 | 425 | 814 |
| orientdb_core | 1.0 | 561 | 2089 |

本文选用这些测试对象的原因在于这 4 个测试对象功能清晰,并且通过演化风格匹配方法[18]发现这 4 个样本适合向正交体系结构风格转换。

由于用于验证所提方法的软件系统的大小(即实体的数量)不相同,对总体和其他参数使用相同的大小是不合适的。因此,算法的种群数是根据特定软件系统中存在的程序元素的数量来定义的[8]。

### 4.2 实验结果分析

为了客观地验证本文提出的风格距离,本文首先对这 4 个样本的正交风格距离和 McCabe 度量进行计算,如表 5 所列。通过对这 4 个样本进行独立重复 50 轮实验,设置表 6 中的参数进行最优种群搜索,再对结果进行变动代价、正交风格距离以及 McCabe 度量计算并取平均值,可以发现最优种群的正交风格距离减少了,且当软件规模较小时距离减小更明显。并且本文发现 McCabe 度量均出现一定程度的改善。

表 5 测试样本重构前指标

Table 5 Metrics for test samples before refactoring

| 测试对象 | 正交风格距离 | McCabe 度量 |
|---|---|---|
| Cassandra | 8274 | 174 |
| H2 | 13674 | 169 |
| neo4j_kernel | 3034 | 74 |
| orientdb_core | 10412 | 236 |

表 6 遗传算法参数设置

Table 6 Genetic algorithm parameter settings

| 变异率 | 交叉率 | 种群数与类数比 | 迭代次数 |
|---|---|---|---|
| 0.8 | 0.001 | 1 | 100 |

对比经典的遗传算法,本文提出的基于语义相似度的变异算子改进的遗传算法在参数均相同的情况下得到的结果更优。表 8 中,标注[+]代表数值更大,标注[-]代表数值更小,3 个指标都是越小越好。

表 7 通过遗传算法重构后指标

Table 7 Metrics after refactoring using genetic algorithm

| 测试对象 | 变动代价 | 正交风格距离 | McCabe 度量 |
|---|---|---|---|
| Cassandra | 8194.18 | 5773.68 | 137.48 |
| H2 | 24721.8 | 7880.96 | 128.7 |
| neo4j_kernel | 2944.96 | 1703.00 | 70.32 |
| orientdb_core | 8727.94 | 7527.72 | 198.28 |

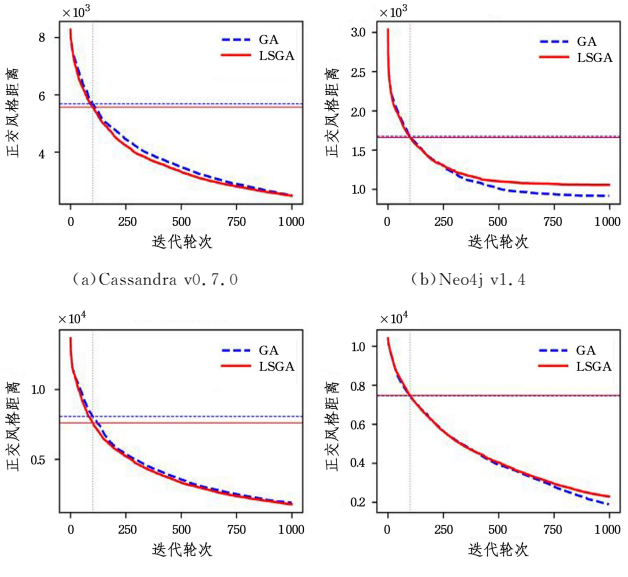
表 8 通过改进的遗传算法重构后指标

Table 8 Metrics after refactoring using improved genetic algorithm

| 测试对象 | 变动代价 | 正交风格距离 | McCabe 度量 |
|---|---|---|---|
| Cassandra | 7901.72[−] | 5684.92[−] | 133.06[−] |
| H2 | 25909.76[+] | 7756.92[−] | 126.02[−] |
| neo4j_kernel | 2888.4[−] | 1682.64[−] | 67.10[−] |
| orientdb_core | 8847.74[+] | 7475.00[−] | 191.16[−] |

由表 8 可知，算法在 100 轮的迭代可加速算法收敛，并且多项指标均更优。此外，本文在针对 cassandra 的实验中发现，当针对类 org. apache. cassandra. service. CassandraDaemon 所在的构件为起始构件时算法出现无法收敛的情况。而当本文使用类 org. apache. cassandra. cli. CliMain 所在的构件为起始构件时，cassandra 的 fitness 能收敛 25%左右。本文认为出现这种现象的原因在于 CliMain 是 cassandra 的命令入口，即所有的功能可以以 CliMain 为起点向其他构件发出。而正交体系结构风格的特点在于，以最顶层结点为起点向下层发起调用。因此，起点的选择对向正交体系结构风格转换的影响较大。

为分析算法的收敛效果，本文设置迭代 1000 轮重复 10 次并取均值。结果证明，本文基于语义相似度变异算子改进的遗传算法(Lexical Static Genetic AlgorithmLSGA)可在算法前期加快收敛。本文认为基于语义相似度变异算子在算法迭代 100 轮后与 GA 相比不具优势的主要原因在于，算子中的构件语义向量计算位于算法起始阶段，即计算的是 $SA_i$ 的构件语义向量。随着迭代的进行，SA 差异过大，语义信息不再有效。因此，本文的结果取，第 100 轮结果，以评估基于语义相似度变异算子的有效性。

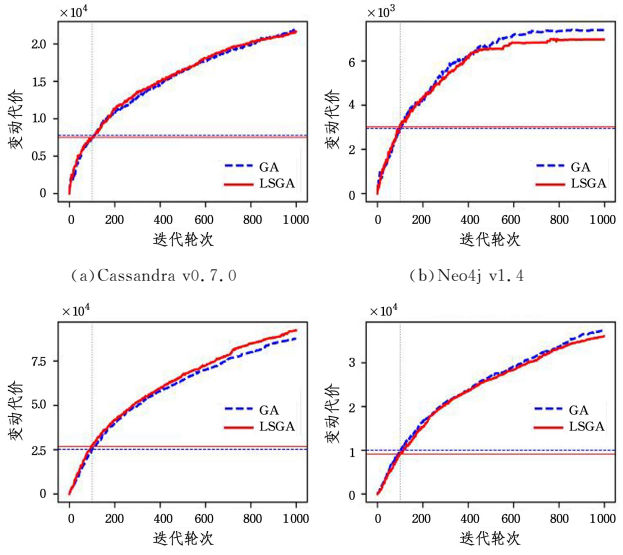



(a)Cassandra v0.7.0 (b)Neo4j v1.4

(c)h2 v1.3.176 (d)orientdb v1.0

图 7 算法迭代过程中风格距离的比较

Fig. 7 Comparison of style distance during algorithm iterations




(a)Cassandra v0.7.0 (b)Neo4j v1.4

(c)h2 v1.3.176 (d)orientdb v1.0

图 8 算法迭代过程中变动代价的比较

Fig. 8 Comparison of change cost during algorithm iterations

最后，本文以上述实验结果中 Cassandra v0.7.0 得到的 $SA_1$ 为例，通过 PDDL 得到演化路径中 $SA_0$ 到 $SA_1$ 的动作序列。由于动作序列结果较多，此处从路径中选出删除构件 org. apache. cassandra. db. ss 与 org. apache. cassandra. cli. cliclient. ss 之间的连接为例子，给出如下所示结果。

…

(DEL-CONNECTION ORG_APACHE_CASSANDRA_DB_SS_ORG_APACHE_CASSANDRA_CLI_CLICLIENT_SS ORG_APACHE_CASSANDRA_CLI_CLICLIENT_SS_ORG_APACHE_CASSANDRA_DB_SS)

…

(DEL-PROVIDE-INTERFACE-TO-COMPONENT ORG_APACHE_CASSANDRA_CLI_CLICLIENT_SS ORG_APACHE_CASSANDRA_CLI_CLICLIENT_SS_ORG_APACHE_CASSANDRA_DB_SS)

…

(DEL-REQUIRE-INTERFACE-TO-COMPONENT ORG_APACHE_CASSANDRA_DB_SS ORG_APACHE_CASSANDRA_DB_SS_ORG_APACHE_CASSANDRA_CLI_CLICLIENT_SS)

…

结果显示连接件删除后由于端口处于未占用状态，PDDL 解释器会先后删除 org. apache. cassandra. db. ss 上的 provide 接口以及 org. apache. cassandra. cli. cliclient. ss 上的 require 接口。动作满足约束即先删除连接，再删除构件上的接口，由此可判断动作有效。

**结束语** 本文提出一种基于遗传算法与 PDDL 相结合的向软件体系结构风格转换的重构方法，本文称为 GA-PDDL 方法。该方法利用遗传算法生成演化路径上的里程碑体系结构，利用 PDDL 生成从起始到目标的演化动作序列，结合二者找到满足风格的软件体系结构以及向其演化的路径。本文还设计了一种基于语义相似度的变异算子，提升了算法的前期收敛速度。最后所提方法中，我们提出软件体系结构风格距离指标，通过遗传算法结合风格距离指标和变动代价指标作为 fitness。实验结果表明，以正交风格为例，第一阶段搜索得到的目标结构具有较好的正交风格特征，并且由于正交风格概念中强调结构中不应出现同一层的构件间存在连

接,因此正交风格的 McCabe 度量较起始结构有部分提升。本文提出的基于语义相似度的变异算子改进的遗传算法能有效提升正交风格的搜索结果的各项指标。第二阶段的演化任务规划得到的演化序列有明显的动作依赖关系,能在一定程度上协助开发者规划重构过程。本文方法适用于一定规模的风格转换,但对于大规模的重构问题,第二阶段可能出现问题规模超出 PDDL 解释器的处理上限。因此,对于所提方法有待进一步改进。未来的工作包括考虑使用多目标优化算法、改进 fitness 的指标、优化 PDDL 的动作设置、设计其他结构明显的软件体系风格指标。

## 参 考 文 献


[1] MEI H,SHEN J R. Progress of Research on Software Architecture[J]. 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[J]. 软件学报,2006,17(6):1257-1275.

[2] DUCASSE S,POLLETD. Software architecture reconstruction: A process-oriented taxonomy[J]. IEEE Transactions on Software Engineering,2009,35(4):573-591.

[3] MANCORIDIS S,MITCHELL B S,RORRES C,et al. Using automatic clustering to produce high-level system organizations of source code[C]// Proceedings 6th International Workshop on Program Comprehension (IWPC'98) (Cat. No. 98TB100242). IEEE,1998:45-52.

[4] BARNES J M,PANDEY A,GARLAN D. Automated planning for software architecture evolution[C]// 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE,2013:213-223.

[5] CHONDAMRONGKUL N,SUN J,WARRENI. Software architectural migration: An automated planning approach[J]. ACM Transactions on Software Engineering and Methodology (TOSEM),2021,30(4):1-35.

[6] MAHDAVI K,HARMAN M,HIERONS R M. A multiple hill climbing approach to software module clustering[C]// International Conference on Software Maintenance,2003. IEEE,2003:315-324.

[7] ABDEEN H,DUCASSE S,SAHRAOUI H,et al. Automatic package coupling and cycle minimization[C]// 2009 16th Working Conference on Reverse Engineering. IEEE,2009:103-112.

[8] PRAJAPATI A,GEEM Z W. Harmony search-based approach for multi-objective software architecture reconstruction [J]. Mathematics,2020,8(11):1906.

[9] LIN Y,PENG X,CAI Y,et al. Interactive and guided architectural refactoring with search-based recommendation[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016:535-546.

[10] MURPHY G,NOTKIN D,SULLIVAN K. Software reflexion models: bridging the gap between design and implementation [J]. IEEE Transaction on Software Engineering,2001,27(4):364-380.

[11] 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. (in Chinese)
钟林辉,夏鲸,彭云,等. 一种图编辑距离的软件体系结构变化性度量方法及应用研究[J]. 小型微型计算机系统,2018,39(3):425-432.

[12] ZHANG Y S. Models of Orthogonal Software Architecture[J]. Computer Applications,2004,24(6):96-98. (in Chinese)
张友生. 正交软件体系结构模型[J]. 计算机应用,2004,24(6):96-98110

[13] XUAN R,CHEN L,SHI H H. The reusable design and implementation of graph algorithms family [J]. Journal of Jiangxi Normal Uniyersity (Natural Science),2023,47(1):52-60. (in Chinese)
轩瑞,陈磊,石海鹤. 图类算法可重用设计及其实现[J]. 江西师范大学学报(自然科学版),2023,47(1):52-60.

[14] EFSTATHIOU V,CHATZILENAS C,SPINELLIS D. Word embeddings for the software engineering domain[C]// Proceedings of the 15th International Conference on Mining Software Repositories. 2018:38-41.

[15] ARORA S,LIANG Y,MA T. A simple but tough-to-beat baseline for sentence embeddings[C]// International Conference on Learning Representations. 2017.

[16] GEREVINI A,SERINA I. LPG: A Planner Based on Local Search for Planning Graphs with Action Costs[C]// AIPS. 2002,2:281-290.

[17] MCCABE T J. A complexity measure[J]. IEEE Transactions on Software Engineering,1976(4):308-320.

[18] ZHONG L H,QI J,YE H T,et al. The Study on the Method for Matching the Software Evolutionary Style Based on Multi-Dimensional Evolutionary Tree[J]. Journal of Jiangxi Normal Uniyersity(Natural Science),2021,45(1):55-59. (in Chinese)
钟林辉,齐杰,叶海涛,等. 基于多维属性演化树的软件演化风格匹配方法研究[J]. 江西师范大学学报(自然科学),2021,45(1):55-59.


**ZHONG Linhui**, born in 1974, Ph. D, professor, is a member of CCF (No. 09772M). His main research interests include software architecture, software evolution and maintenance.