文章编号：2095-2163(2020)03-0132-06 中图分类号：TP391 文献标志码：A

一种语句分裂变更模式的分类框架

段卫华，杨春花
（齐鲁工业大学（山东省科学院）计算机科学与技术学院，济南 250353）

摘 要：语句分裂变更模式是一种常见的将一条代码语句分裂成多条语句的代码变更模式，该模式有多种呈现形式，而不同的呈现形式又可能对应不同的变更目的。提出一种分类框架，从分裂语句的类型、语句变更的行为和新增语句类型三个维度对该模式进行分类，并设计了基于该框架的分类算法。最后，将该算法应用于 4 个开源项目，对其中所包含的语句分裂变更模式进行了分类和分析，实验结果呈现出较高的分类准确率。

关键词：软件演化；语句分裂变更模式；分类框架；代码变更块

A classification framework of statement split change pattern

DUAN Weihua, YANG Chunhua
(School of Computer Science and Technology, Qilu University of Technology(Shandong Academy of Sciences), Jinan 250353, China)

【Abstract】 Statement splitting change pattern is a common code change pattern that splits a code statement into multiple statements. This pattern has many forms of presentation, and different forms of presentation may correspond to different purposes of change. This paper proposes a classification framework, which classifies the pattern from three dimensions: the type of split statement, the behavior of statement change and the type of new statement, and designs a classification algorithm based on the framework. Finally, the algorithm is applied to four open source projects to classify and analyze the statement splitting change patterns. The experimental results show a high classification accuracy.

【Key words】 software evolution; statement split change pattern; classification framework; Hunk

0 引 言

现代大型软件的开发和维护一般由多人协作完成，软件工程师每天将变更的代码提交到版本管理系统，对代码变更进行理解是软件工程师日常工作的基础。然而，一次变更提交中往往聚集了多种修改模式，如缺陷修复、重构、增加特征等。多种变更模式掺杂在一起使得对软件变更的理解变得困难。因此，为了更好地理解变更的内容，有必要对一些常见的变更模式进行分类，如此一来即与其它代码修改分离开来，从而使得变更理解变得容易。

代码变更中较典型的模式包括重构[1](Refactoring)和缺陷修复(bug-fixing)等。对于重构模式的分类研究包括：文献[2]对软件重构研究进行的较为全面的总结，文献[3]对代码坏味对软件演化影响的研究和文献[4]对常见的重构操作的简单介绍和分类等。缺陷修复的研究包括：文献[5]对软件缺陷分类应用于缺陷预测方法的研究；文献[6]对面向软件自动修复的缺陷分类方法的研究；以及文献[7]对具体领域软件代码中缺陷分类的研究等。上述研究对常见的代码变更模式制定了相应的分类体系，这些体系为代码变更的研究提供了参考，并且通过将其应用于各种变更模式的自动识别当中，从而对软件演化的理解产生更多积极作用。

语句分裂是一种常见的代码变更模式，将一条语句分裂为两条或多条语句。该模式有许多呈现形式，而不同的呈现形式对应的变更目的不同：可以是一种重构，也可以是一种缺陷修复。然而，现有的变更模式分类工作没有涵盖对该模式的系统研究。

本文在对大量语句分裂变更模式进行观察分析的基础上，总结了常见的语句分裂变更模式的呈现形式以及一些附加的变化，提出一个语句分裂变更模式的三维分类框架以及基于该框架的分类算法，并用该算法对开源项目中包含的语句分裂变更模式进行了抽取、分类和分析。

下面首先介绍语句分裂变更模式，然后提出语

基金项目：山东省自然科学基金面上项目(ZR2017MF056)。

作者简介：段卫华(1993-)，男，硕士研究生，主要研究方向：代码变更分析、软件演化；杨春花(1974-)，女，博士，教授，主要研究方向：代码变更分析、软件演化、软件开发。

通讯作者：段卫华 Email：860043430@qq.com

收稿日期：2019-01-10

句分裂变更模式的分类框架和识别算法，最后是实验验证和分析。

1　语句分裂变更模式

一个源代码文件修改前后的 2 个版本相应位置的差异即为变更，而当前对代码变更的提取一般是借助于差异化分析工具来进行。目前存在的差异分析工具主要有 2 种：文本差异分析（Textual－Differencing）和树差异分析（Tree－Differencing），前者将 2 个版本的源代码视为 2 个字符串，通过计算 2 个字符串的差异得到代码变更，著名的工具有 GNU－Diff[8] 等；后者比较 2 个版本的源代码所对应的抽象语法树，计算二者的差异作为代码变更，代表性的工具有 Change－Distiller[9] 等。由于文本型差异分析工具目前广泛应用到版本管理系统中辅助用户查看 diff，因此本文的研究基于文本差异分析工具的输出结果，对其中的语句分裂变更模式进行研究。

文本型差异分析工具的变更输出单位为 hunk（代码变更块），一个 hunk 由删除行和添加行构成。图 1 是 GNU－Diff 的一个 hunk 输出示例。由"－"开头的行为被删除的语句行，"＋"开头的行为新增的语句行，hunk 中"－"左边的行号 88 表示该行在旧版本中的位置，"＋"右边的行号 88～89 表示新增的行在新版本的位置。

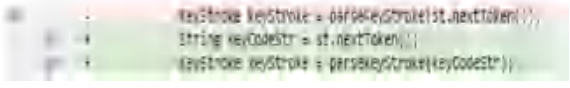

图 1　hunk 示例

Fig. 1　hunk example

经过对人工识别出的开源项目中包含语句分裂变更模式的 hunk 集进行分析，发现该模式包含 2 种基本形式，将其分别命名为拆分形式和替换形式。对此可做阐释分述如下。

（1）拆分形式。就是将一条赋值语句拆分成多条赋值语句。图 2 为拆分形式的示例，可以看到删除行 872 行的赋值语句被拆分为 2 条赋值语句（增加行 872～873 行）。

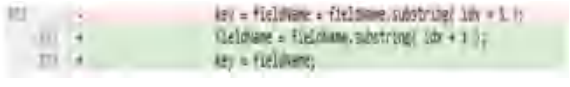

图 2　拆分形式

Fig. 2　Split form

（2）替换形式。使用一个新声明的变量替换一条语句中部分内容使之变为一条新的语句。如图 3 所示，老版本的第 398 行方法调用语句中的 getPluginExtensionComponents（plugin）被新版本行号 402 的变量声明语句所替换，因此原语句变为第 404 行的形式。

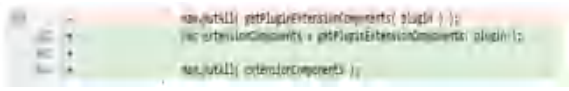

图 3　替换形式

Fig. 3　Replacement form

2　一个语句分裂变更模式的分类框架

为了从呈现形式和变更行为上全面理解语句分裂变更模式，提出一个三维分类框架，从分裂语句的类型、语句变更行为和新增语句类型三个维度对该模式进行分类，分类框架如图 4 所示。

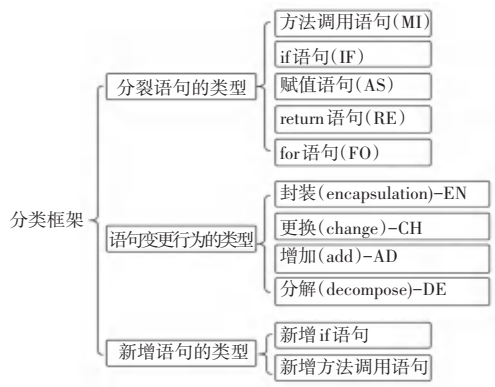

图 4　分类框架

Fig. 4　Classification framework

下面将对框架中的每个分类维度进行解释，并给出对应的实例进行说明。

2.1　分裂语句的类型

分裂语句是指 hunk 删除行中被分裂的语句（下文用"原句"表示）。通过观察，分类模式常发生在以下 5 种语句类型中：

（1）方法调用语句（MI）。

（2）if 语句（IF）。

（3）赋值语句（AS）。

（4）return 语句（RE）。

（5）for 语句（FO）。

值得注意的是，拆分形式的语句分裂一般出现在赋值语句中，而其他类型语句则常以替换形式进行分裂变更。

2.2　语句变更行为的类型

对于语句分裂变更模式，可根据 hunk 的增加行中，声明新变量的赋值内容和参与原句更改的方式，总结出替换形式的 4 种变更行为类型，详述如下。

（1）封装（encapsulation）－EN：新变量的赋值内容取自原句中，并且这部分内容没发生任何变化，如图 5 所示。

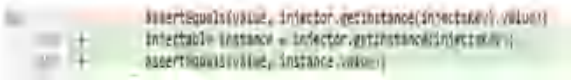

图 5 封装行为

Fig. 5 Encapsulation behavior

(2)更换(change)-CH:新变量的赋值内容与原句中任何部分都不完全相同或完全不同,如图 6 所示。

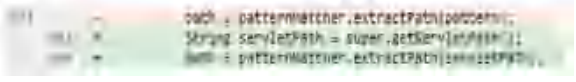

图 6 更换行为

Fig. 6 Change behavior

(3)增加(add)-AD:只是在原句的基础上增加了与新变量相关的部分,从而语句原有的内容没有发生变化,(此种类型中新变量的赋值内容可能与原句中的内容有关,也可能无关)。如图 7 所示。

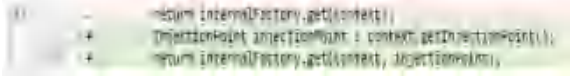

图 7 增加行为

Fig. 7 Add behavior

(4)分解(decompose)-DE:将原句直接拆分成两条或多条语句,如图 2 所示。这种行为一般只出现在拆分形式的语句分裂变更模式中。

2.3 新增语句的类型

这个维度只针对替换形式的语句分裂变更模式,因为此维度所研究的新增语句一般是与替换形式中新声明赋值的变量相关的语句,而拆分形式中不存在新变量赋值声明的语句,所以也就不会有与之相关的新的语句出现,所以新增语句指的是 hunk 增加行部分,除新变量声明语句和替换后的原句之外的其它新增加的语句,当然新增的语句是不一定存在的。

新增语句一般分为以下 2 种类型:

(1)新增与新变量相关的 if 语句,如图 8 所示。

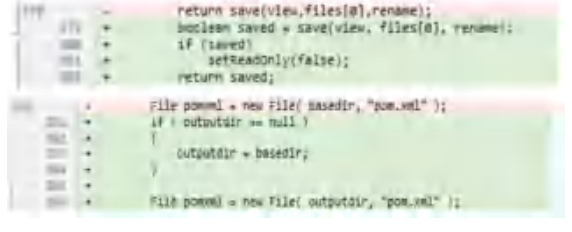

图 8 新增 if 语句

Fig. 8 Add if statement

(2)新增新变量相关的调用方法的语句,如图 9 所示。

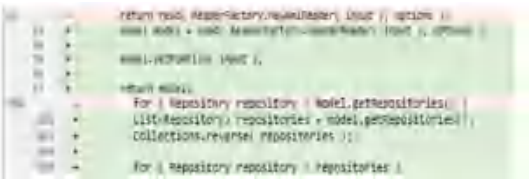

图 9 新增方法调用语句

Fig. 9 Add method invocation statement

2.4 其他类型

这部分主要探讨替换形式中存在的另外 2 种新增内容的类型。研究给出内容表述如下。

(1)第一种是对上述语句变更行为类型中的“更换”类型的补充,这种类型中的“不完全相同”一般是赋值内容中除了取自原句中的部分之外还有新增加的内容,下面就介绍常见的这部分新增内容的类型:

① 新变量的赋值内容中新增方法调用,如图 10 所示。

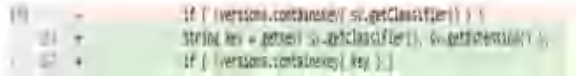

图 10 新增方法调用

Fig. 10 Add method invocation

② 新变量的赋值内容中新增逻辑或算术运算,如图 11 所示。

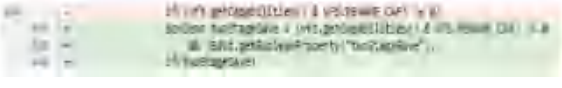

图 11 新增逻辑或算术运算

Fig. 11 Add logical or arithmetic operation

(2)第二种是针对原句变更时,除部分内容被新变量替换外还增加了新的内容。对此拟做重点论述如下。

① 被替换后的原句新增了与新变量相关的方法调用,如图 12 所示。

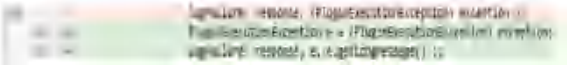

图 12 新增方法调用

Fig. 12 Add method invocation

② 被替换后的原句新增了与新变量相关的逻辑或算术运算,如图 13 所示。

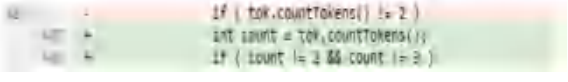

图 13 新增逻辑或算术运算

Fig. 13 Add logical or arithmetic operation

3 语句分裂变更模式的分类算法

根据上述分类框架,设计了一个分类算法,该算

法用四元组 $< p,t,b,a >$ 的结构表示每个包含语句分裂变更模式的 hunk 的分类结果，并将这些结果存入集合 Q 中，其中变量 p 是语句分裂变更模式识别算法（已有文章对此识别算法进行论述）输出的语句分裂变更模式替换形式的 hunk 集 P_R 和拆分形式的 hunk 集 P_S 中的元素，变量 t、b、a 都是字符串类型的变量，t 代表分裂语句的类型；b 代表语句变更行为的类型；a 代表新增语句的类型。

经观察拆分形式的语句分裂变更模式三维判别结果单一，即 $< p,AS,DE,NULL >$，所以分类算法只对替换形式的语句分裂变更模式的 hunk 集 P_R 中的元素进行 3 个维度的判别，算法的伪代码如下。

输入：同一文件的一次更改前后两个版本的源文件 *Fileold* 和 *Filenew*

输出：对 2 个源文件的变更中包含的语句分裂变更模式的 hunk 进行 3 个维度分类判别的结果集 Q

```
1   P_R ← identifySplitpatten(Fileold, Filenew)
2   for each p ∈ P_R   p = < h, o_, m, c >
3     t ← get Type(o_)
4     if true← contains(c, o_)
5           b = "AD"
6           z ← getVarValue( )
7           if true← contains(o_, z)
8                   b = "EN"
9           else b = "CH"
10                  if true ← hasIfstmt(m)
11                        a = "IF"
12                  else if true ← hasMistmt(m)
13                        a = "MI"
14                        else a = "NULL"
15                              Q ← < r, t, b, a >
16                        end if
17                  end if
18          end if
19      end if
20  end for
```

算法第 1 行利用 *identifySplitpatten*( ) 算法（即语句分裂变更模式的识别算法）获取 2 个源文件的变更中包含的替换类型的语句分裂变更模式的 hunk 集 P_R，其中每个元素 p 可表示成一个四元组 $< h,o_-,m,c >$ 的形式，变量 h 代表一个 hunk，这是一个四元组 $h = < L_-,L_+,R_-,R_+ >$ 的结构（其中变量 L_- 和 R_- 分别表示 hunk 删除行的文本内容和行号范围，变量 L_+ 和 R_+ 则分别表示 hunk 增加行的文本内容和行号范围），变量 o_- 代表发生替换分裂的原句的全部节点信息，变量 m 代表 hunk 增加行中新声明变量的变量名，变量 c 代表变更后的原句的全部节点信息。

算法第 2 行利用 for 循环对集合 R 中的每个元素 r 进行分类处理。

第 3 行利用 *getType*( ) 函数获得分裂语句 o_- 即原句的类型 t，即对第一维度的判别。

第 4~9 行是对第二个维度的判别，其中 4、5 行利用 *contains*( ) 函数判断原句分裂变更后是否还包含了原来的全部节点信息（*contains*( ) 方法，当且仅当此字符串包含指定的 char 值序列时，返回 true），如果返回 true 则新的变量没有替换掉原句中的任何内容，并且识别算法已确定变更后的原句包含了新声明的变量 m，则可以确定其语句变更行为类型为增加 "*AD*"；第 6 行利用 *getVarValue*( ) 函数获取 hunk 对应的抽象语法树片段中，给变量 m 赋值的语句节点的赋值内容 z；第 7、8 行利用 *contains*( ) 函数判断原句 o_- 是否包含了字符序列 z，如果返回 true 则确定其语句变更行为类型为封装 "*EN*"；第 9 行如果返回 false，则确定其语句变更行为类型为更换 "*CH*"。

第 10~14 行是对第三个维度的判别，其中 10、11 行利用 *hasIfstmt*( ) 函数通过遍历 hunk 增加行对应语法树中的节点信息，找出是否存在新增的包含新声明变量 m 子节点的 if 语句节点，如果返回 true 则确定其新增语句类型为 if 语句 "IF"；第 12、13 行利用 *hasMistmt*( ) 函数找出是否存在新增的包含新声明变量 m 子节点的方法调用语句节点，如果返回 true 则确定其新增语句类型为方法调用语句 "MI"；第 14 行如果非上述情况则判断不存在新增的语句 "NULL"。

第 15 行将上述的三个维度的判别结果以四元组 $< r,t,b,a >$ 的形式存入集合 Q 中。

4　算法的实现及验证

该算法当前采用 Java 语言来实现。在此实现的基础上，将提出的分类框架应用到 4 个开源项目，对其中的语句变更分裂模式进行了识别和分类，并对识别和分类出的结果进行了分析。

数据集包含了 4 个开源项目：J Edit（http://sourceforge.net/project/jedit/）、Apache maven（http://maven.apache.org/）、Google-guice（https://github.com/apress/google-guice）和 Eclipse（http://

www.eclipse.org/downloads/eclipse-packages/)。输出结果见表 1。

表 1 识别算法的输出结果

Tab. 1 Results of recognition algorithm

项目名称	语句分裂变更模式的个数	
	拆分形式	替换形式
J Edit	3	98
maven	4	52
guice	0	29
Eclipse	10	284

4.1 分裂语句类型的分类结果

算法对上述数据集进行分裂语句类型的判别结果如图 14 所示。

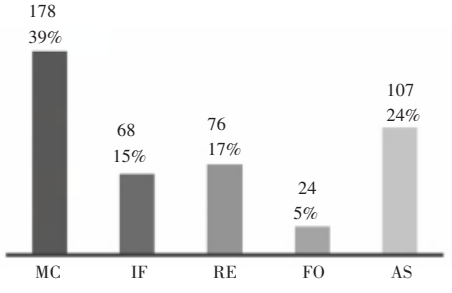

图 14 各类型分裂语句的分类结果

Fig. 14 Classification results of various types of split statements

经人工观察验证分裂语句类型的分类结果后发现没有出现错误分类的情况,表明算法在此维度的分类具有较高的准确性。

根据图 14 所示的分裂结果,语句分裂变更模式发生在方法调用语句(MC)和赋值语句(AS)中的次数较多,分别占到了总数的 39%和 24%,return 语句(RE)和 if 语句(IF)次之,分别占到了总数的 17%和 15%,for 语句(FO)最少,只占到了总数的 5%。

4.2 语句变更行为类型的分类结果

算法对语句变更行为类型的判别结果如图 15 所示。

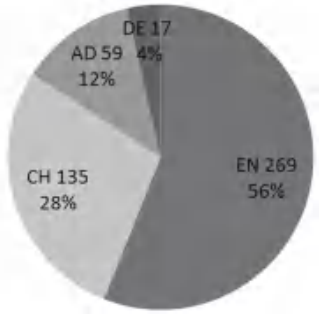

图 15 各类型语句变更行为分类结果

Fig. 15 Classification results of various types of statement change behavior

经人工观察验证语句变更行为类型的分类结果后发现没有出现错误分类的情况,表明算法在此维度的分类具有较高的准确性。

根据图 15 的结果,不难发现封装(EN)行为出现的次数最多,占到了总数的 56%;而更换(CH)行为和增加(AD)行为次之,分别占到了总数的 28%和 12%,这两种行为一般会改变原语句的作用;而出现次数最少的分解(DE)行为只占到了总数的 4%。

4.3 新增语句类型的分类结果

算法对新增语句类型的判别结果如图 16 所示。

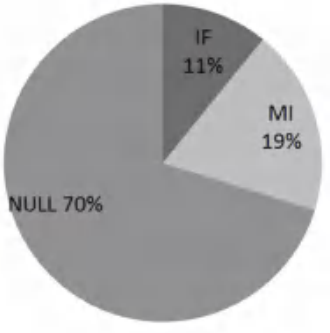

图 16 2 类新增语句的分类结果

Fig. 16 Classification results of two kinds of added statements

经人工观察验证新增语句类型的分类结果后发现,有极个别包含与新声明变量无关的新增语句的 hunk 没有被识别出来,这是因为设计的算法只识别包含新声明变量的新增语句,而这些极个别与新变量无关的新增语句对本课题的研究基本无意义,所以图 16 是将这些 hunk 排除后所得到的结果。

根据图 16 可知,没有新增语句的 hunk 占到了 70%,而出现新增语句的例子大约占到总数的 30%,其中与新变量相关的方法调用语句(MI)占到了 19%,if 语句(IF)占到了 11%。

4.4 结果分析及结论

根据表 1 可知,语句分裂变更模式的拆分模式出现次数极少,这种形式不会改变语句的行为并且不会伴随额外的变化。

通过观察图 14 中 3 种占比较少的语句类型的 hunk 可知,替换形式的语句分裂模式大多数都发生在这些语句中的复杂的方法调用和赋值部分,并且结合方法调用语句和赋值语句占比较多的情况可以说明:语句中发生替换的基本都是复杂的方法调用和赋值部分。

根据图 15 中封装类型(DE)占比较多的情况可知,多于半数的替换形式本质上是没有发生算法更

换的，只是通过替换简化了语句中较为复杂的内容，这种操作不会改变语句行为所以也可以看作是一种重构行为，而更换类型（CH）才更加类似于替换算法重构模式，这两种语句变更行为类型一共占到了总数的 84%，所以可以认为：替换形式的语句分裂变更多数情况是一种重构操作。

图 15 中的增加行为（AD）基本呈现 2 种形式：一种形式是在 if 语句中增加了与新变量相关条件表达式，则此变更是由重构模式与 IF-CC[10]（更改条件表达式）缺陷修复模式组合而成的变更；另外一种形式是在其它语句类型的方法调用部分增加了与新变量相关的新的参数，则此变更是由重构模式与 MC-DNP[10]（具有不同数量参数或不同类型参数的方法调用）缺陷修复模式组合而成的变更。

图 16 中 30%的包含新增语句的替换形式的语句分裂变更模式，其意图是为了便于引入语句中复杂内容的方法调用或条件判断，从而使用简单的变量将其替换，新增语句实际上是一种缺陷修复行为，例如新增 if 语句的语句分裂变更类似于 IF-APC[10]（增加前提条件检查）缺陷修复模式，所以有新增语句的语句分裂变更也是由重构模式与缺陷修复模式组合而成的变更。

上述的增加行为和新增语句的行为都是基于替换操作基础上的额外的变更，所以可知：包含额外变更的替换形式的语句分裂变更多数情况是一种由重构模式与缺陷修复模式组合的变更模式。

5　结束语

本文对语句分裂变更模式进行了分类研究，根据人工筛选出的数据集提出了语句分裂变更模式的三维分类框架，并且根据该框架设计了语句分裂变更模式的分类算法，最后用该分类算法对语句分裂变更模式的 hunk 集进行分类判别，在观察分析判别结果后得出一些结论。后续工作还需要结合代码变更块的上下文内容，对语句分裂变更模式的各种形式进行变更发生情境、产生因素等方面做进一步的经验分析，使得对语句分裂变更模式的研究更为完善。

参考文献

[1] FOWLER M. Refactoring: Improving the design of existing programs [M]. USA: Addison-Wesley, 1999.

[2] MENS T, TOURWÉ T. A survey of software refactoring[J]. IEEE Transactions on Software Engineering, 2004, 30(2): 126.

[3] 章晓芳，朱灿. 代码坏味对软件演化影响的实证研究[J]. 软件学报，2019，30(5)：1422.

[4] 阮航，陈恒，彭鑫，等. 面向设计的开源软件项目重构经验研究[J]. 计算机科学与探索，2017，11(9)：1418.

[5] 李伟清，郭鸿昌. 基于邻域三支决策粗糙集模型的软件缺陷预测方法[J]. 数据采集与处理，2017，32(1)：166.

[6] 易昕，毛晓光，纪涛. 面向程序自动修复的缺陷分类方法研究[J]. 计算机应用研究，2016，33(6)：1748.

[7] 贺仁亚，唐龙利. 故障注入的软件代码缺陷模式[J]. 指挥信息系统与技术，2015，6(6)：23.

[8] HUNT J W, SZYMANSKI T G. A fast algorithm for computing longest common subsequences[J]. Communications of the ACM, 1977, 20(5): 350.

[9] FLURI B, GALL H C. Classifying change types for qualifying change couplings [C]//14th IEEE International Conference on Program Comprehension (ICPC'06). Athens, Greece: IEEE, 2006, 35.

[10] PAN Kai, KIM S, JR E J W. Toward an understanding of bug fix patterns[J]. Empirical Software Engineering, 2009, 14(3): 286.

（上接第 131 页）

4　结束语

列车弓网关系的稳定直接影响列车的安全运行。而弓网间接触力的稳定则尤为关键。比较上述 3 种控制算法结合接触力评价指标可以得出结论：对于列车弓网系统，施加主动控制能够极大提高弓网系统的稳定性。对比分析 3 种控制器可发现，PID、模糊滑模控制及模糊滑模自适应 PID 的控制效果依次提高。为研究高效智能的弓网主动控制系统提供科学理论依据。

参考文献

[1] 郭京波. 高速机车受电弓稳定受流与控制研究[D]. 北京：北京交通大学，2006.

[2] 乔枫，郭慧佳，李界家，等. 二级倒立摆系统的模糊滑模变结构控制[J]. 沈阳建筑大学学报（自然科学版），2010，26(4)：792.

[3] 崔营波. 高速列车受电弓半主动控制方法研究及应用[D]. 北京：北京交通大学，2018.

[4] 刘浩，钱存元，施招东. 基于模糊自适应 PID 控制的 ATO 系统控制算法[J]. 城市轨道交通研究，2017(3)：40.

[5] 孙磊，孙冬梅，袁倩，等. 基于模糊滑模变结构的磁轴承振动控制研究[J]. 科技通报，2018，34(9)：186.

[6] 张兆东，徐小亮，杨杨，等. 基于模糊 PID 控制策略的液压缸试验台加载系统设计[J]. 南京理工大学学报，2019，43(1)：78.

[7] 张红，徐海军，刘淑荣，等. 基于改进模糊 PID 控制的循环水优化仿真[J]. 电气应用，2014，33(24)：134.

[8] GUAN Kaizhong, LUO Zhiwei. Stability results for impulsive pantograph equations[J]. Applied Mathematics Letters, 2013, 26(12): 1169.

[9] 刘浩，钱存元，施招东. 基于模糊自适应 PID 控制的 ATO 系统控制算法[J]. 城市轨道交通研究，2017(3)：40.

[10] 郭桂林. 高速受电弓控制系统研究[D]. 成都：西南交通大学，2014.

[11] 韩南南. 基于接触网动力学参数变化的弓网系统振动特性研究[D]. 上海：上海工程技术大学，2016.