Software Engineering and Applications 软件工程与应用, 2022, 11(3), 602-610
Published Online June 2022 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2022.113064

Hans汉斯

学生程序分析与修复研究

胡建鹏，魏　龙，林　渤

上海工程技术大学电子电气工程学院，上海

收稿日期：2022年4月20日；录用日期：2022年6月13日；发布日期：2022年6月20日

摘　要

针对高校C语言程序设计课程教师很难对课堂上每一个学生遇到的问题进行点评，同时课下的编程练习也缺乏对程序错误的指导等问题，设计了一种针对学生程序分析与程序修复的方法。在静态代码分析软件Cppcheck所能检测缺陷类型的基础上，添加了一些学生代码常见的缺陷类型。对于学生程序中出现的语法错误，本文在以往提交的学生代码数据库中，使用编辑距离算法获取修复提示，学生根据修复提示信息对代码进行改正。并基于此方法实现了相关模块应用到C语言在线实验系统，取得了不错的应用效果。

关键词

程序修复，静态代码分析，Cppcheck，编辑距离

Student Program Analysis and Repair Research

Jianpeng Hu, Long Wei, Bo Lin

School of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai

Received: Apr. 20th, 2022; accepted: Jun. 13th, 2022; published: Jun. 20th, 2022

Abstract

Currently, in C language programming courses at colleges and universities, it is difficult for teachers to comment on the problems encountered by each student in the class, and the programming exercises after class also lack guidance on programming errors. To solve these problems, we present a method designed for student program analysis and program repair. Based on the types of errors that can be detected by the static code analysis software Cppcheck, some common types of errors in student code have been added into our work. This paper employs the editing distance algorithm to provide a repair prompt for syntax problems in students' program, and students correct the code

文章引用: 胡建鹏, 魏龙, 林渤. 学生程序分析与修复研究[J]. 软件工程与应用, 2022, 11(3): 602-610.
DOI: 10.12677/sea.2022.113064

according to the repair prompt. Then, using this method, the relevant module is applied to the C language online experiment system, resulting in a positive application effect.

Keywords

Program Repair, Static Code Analysis, Cppcheck, Edit Distance

1. 引言

在科技飞速发展的背后，软件源代码中还存在安全漏洞，对信息的使用安全构成威胁[1]。相关学者通过用户评价结果，对代码质量进行安全识别与质量分析；还有学者提出一种基于深度学习的缺陷预测方法，通过分析安全缺陷报告中的数据，实现对代码质量管理[2] [3]。静态分析工具在过去十几年中迅速成熟，它已经从简单的词汇分析发展到使用更复杂的技术，随着对代码静态信息研究的不断发展，各类静态代码检测的技术和工具不断更新问世，部分静态代码检测的工具已经进行商业化应用，根据SAMATE 的统计信息，目前市面上常用的静态代码检测工具有超过六十多种，但这些软件检测的缺陷类型较固定，且检测的方式较单一。

现有的软件自动修复方法大多面向工业软件应用，而由于学生程序存在缺陷类型复杂、缺陷种类较多等特点，直接将现有的软件自动修复方法应用到学生程序中必然会出现各种问题[4]。部分学者对学生程序缺陷的修复进行了研究，并且将部分工业级应用的修复方法应用到学生程序的修复中。Singh 等人提出了一种 APEX [5]方法，能够基于符号执行、动态分析和最大满足求解自动为程序存在的问题提供反馈。该方法需要程序作业的参考实现以及错误模型，利用这些信息，系统会自动对学生的错误分析产生的原因和生成解决方案，并进行最小限度的修正。受符号执行的原因，该方法只能对简单的程序错误提供修改建议。Yi 等人使用当前比较流行的工业软件自动修复工具对学生程序的修复进行研究[6]，分析了GenProg [7]、Prophet [8]、Angelix [9]和 AE [10]这四个修复工具，它们均是采用测试用例驱动的方式进行修复，但由于学生程序存在缺陷类型复杂、缺陷种类较多等特点，直接将现有的软件自动修复方法应用到学生程序中必然会出现各种问题，因此，这些工具不适合修复学生程序。

由于新冠疫情的影响，各大高校开始使用线上教学的方式进行授课，但对于实践性较强的 C 语言程序设计课程，线上教学中仍存在一些问题。一方面，由于时间有限，教师很难对课堂上每一个学生遇到的问题进行点评，另一方面，学生课下的编程练习也缺乏对程序错误的指导。针对学生程序存在的问题，本文设计了一种新的程序分析和修复方法。并基于此方法实现了相关模块应用在 C 语言在线实验系统，该系统可以对学生实验编程题目中的错误给予修复提示，在疫情期间的线上教学中发挥了较大作用，取得了不错的应用效果。

2. 方法概要

如图 1 所示，本文设计并实现了一种新的代码检测和修复方法，首先，使用 Cppcheck 对学生的源程序进行静态分析，静态分析是将源程序与 Cppcheck 中所有的缺陷模式进行逐条匹配，当源程序与缺陷模式匹配成功，则输出缺陷提示信息，学生根据提示进行修改，并对修改后的程序再次进行静态分析，直到程序

匹配不到缺陷。然后，执行修改后的程序，当程序存在语法错误时，使用编辑距离算法从学生以往提交的代码库中获取五条修复提示信息，学生根据提示信息对错误修复后继续执行代码，直到程序不报语法错误。

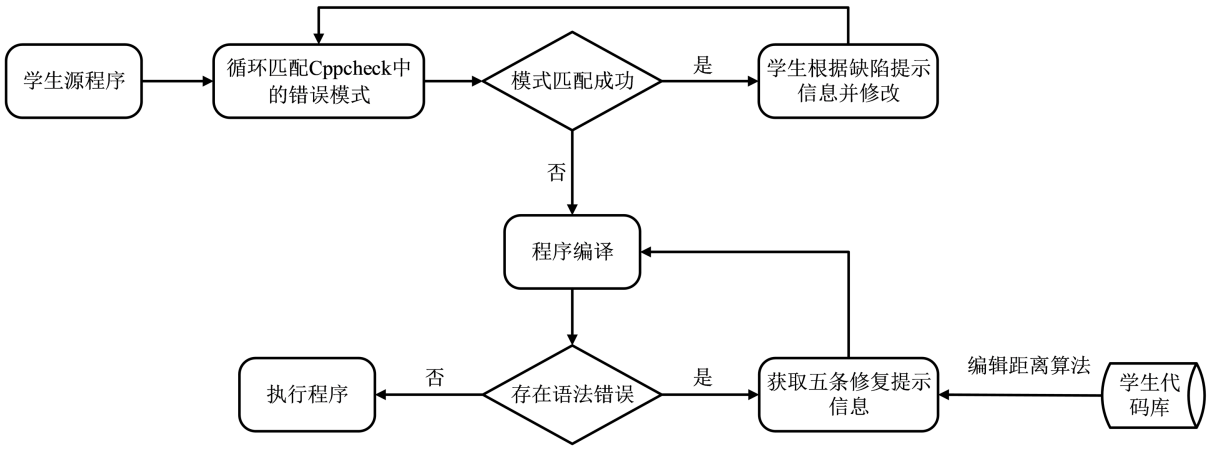

Figure 1. Overall execution process
图 1. 整体执行流程

本文的学生代码库是在面向初学者的 C 语言在线实验系统[11]中学生提交的程序实验数据，如图 2 所示，包括源代码的 id 号、错误行代码和错误行的修复代码等信息。我们对提交的实验数据进行分析，人工总结学生程序中常出现的缺陷模式和修复提示，然后，将总结的缺陷模式和修复提示添加到 Cppcheck 中，当源程序与 Cppcheck 中的缺陷模式匹配成功时，则输出指定的提示信息。使用编辑距离算法计算相似度是将源代码的错误行和代码库中的错误行代码进行相似度计算，然后选择相似度高的错误行代码所对应的修复代码做为修复提示信息。

d_mistak	code	row	olum	wcode	fcode	console
1442	# include "stdio.h"int main() { int r	7	5	printf("%d",i);	i=(n%10)*10+n/10;	expected ';' before 'printf'
5665	# include <stdio.h>int main(void) {	9		printf("sum=%d,sum");	printf("sum=%d",sum);	sum=37,sum
4010	# include <stdio.h>int main () { int	12	8	if(i%2==0)	if((i+1)%2==0)	
2496	# include <stdio.h>int main() {int x,y,	9		case '2':	case 2:	无报错，语法错误
6708	# include <stdio.h>int main(void) {	12		sum=sum+flag*(2i-1)/(3i-2	sum=sum+flag*(2*i-1)/	输入:10输出:Enter n:sum=0.000000
4469	# include<stdio.h>int main( ) {int n, i	21	1	printf("%d\n", &sum);	printf("%d\n", sum);	null
6610	# include<stdio.h>	15	30	if(year%4==0&&year%100!=(if(year%4==0&&year%10		expected expression before '
3118	#include <stdio. h>int main() { int	8	32	if(i/100==1||i/100==2||i/	if(i/100==1||i/100==2	expected expression before '
3561	#include "stdio.h"	1	6	"stdio.h"	<stdio.h>	[Error] #include expects "FILENAME" or
3405	#include "stdio.h"	1		#include "stdio.h"	#include <stdio.h>	
3486	#include "stdio.h"	1	7	#include "stdio.h"	#include<stdio.h>	In function 'main'
214	#include stdio.h	4	1	Int i,j, s = 1;	int i,j, s = 1;	unknown type name 'Int'; did you mean
5056	#include "stdio.h"int main(void) {int i	6	1	for(j = 2; j <= i / 2; j-	s=1;	for(j = 2; j <= i / 2; j++)
5058	#include "stdio.h"int main(void) {int i	9	3	if(i / j == 0) s = s + j;	if(i%j==0) s=s+j;	if(i%j==0) s=s+j;
5057	#include "stdio.h"int main(void) {int i	9	4	if(s==i)	if(i==s)	if(s==i)
5055	#include "stdio.h"int main(void) {int i	4	9	inti,j,s=1;	int i,j,s;	inti,j,s=1;
7005	#include "stdio.h"int main() {int n,x,s	10		}while(n&&((!x) =n%2));	}while(n&&((!x) == n	
3524	#include "stdio.h"int main(void) {	6	10	for(j=2;j<=i/2;j++)	s=1; for(j=2;	
3523	#include "stdio.h"int main(void) {	8	18	if(i/j==0) s=s+j;	if(i%j==0) s=s+j;	
5046	#include "stdio.h"int main(void) { i	13		if(i / j == 0)printf("+%d	if(i % j == 0)printf(	
5045	#include "stdio.h"int main(void) { i	9		if(i / j == 0) s = s + i;	if(i % j == 0) s = s	

Figure 2. Student codebase information
图 2. 学生代码库信息

3. 学生程序分析

3.1. Cppcheck 简介

Cppcheck 是一款开源的针对 C/C++语言的静态代码检测的工具，主要是检测一些编译器不能检测出

来的高级的逻辑错误或者安全漏洞等，不能够检测出代码的语法错误。Cppcheck 的执行流程如图 3 所示，首先使用 Preprocessor 类对源码预处理，然后对预处理后的源码中的每个字符串信息建立 Token 对象、接着对 Token 对象建立双链表，SybmbolDatabase 类对 Token 对象建立的双链表进行语义分析，最后使用 Cppcheck 注册的检测类和规则进行检测并且输出检测信息。根据检测缺陷的不同类型，Cppcheck 对软件缺陷的检查划分为 12 个内嵌检查类[12]，共可检测出 85 种缺陷，如表 1 所示。对这 12 个检查类所检测到缺陷严重性程度的不同分别赋予不同的级别，按照等级的由高到低依次分别为：error，warning，style，performance，portability，information。Cppcheck 有两种缺陷添加方式，一种是添加注册类，一种是基于规则，本文使用基于规则的缺陷添加方式，首先使用正则表达式来表示缺陷模式，然后将缺陷模式添加到规则文件中。

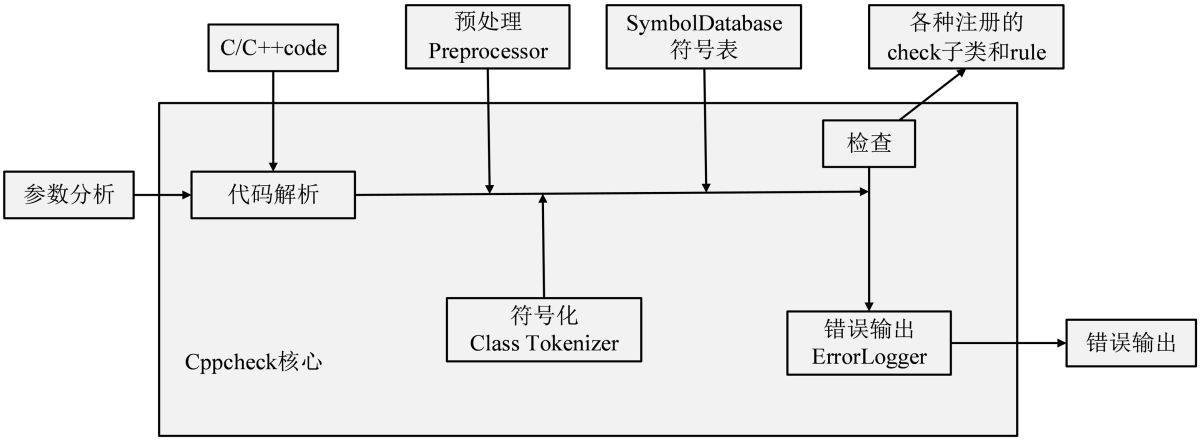

Figure 3. Cppcheck system execution flow chart
图 3. Cppcheck 系统执行流程图

Table 1. Cppcheck inline check class
表 1. Cppcheck 内嵌检查类

序号	检查类名称	检测数量
01	Checkunusedfunction	1
02	Checkautovariables	6
03	Checkuninitvar	3
04	Checkbufferoverrun	3
05	Checkstl	13
06	Checkclass	9
07	Checkpostfixoperator	11
08	Checkexceptionsafety	3
09	Checkother	34
10	Checkmemoryleak	9
11	Checkobsoletefunctions	1
12	Checknullpointer	3

3.2. 缺陷模式表示

本文使用 Cppcheck 基于规则的缺陷添加方式，首先，将总结的缺陷模式使用正则表达式来表示，本文总结了 6 种学生程序中常见的缺陷模式，缺陷模式的正则表达式如表 2 所示。然后，给每个缺陷模式创建 xml 格式的规则文件，并将正则表达式添加到规则文件中，规则文件中除正则表达式外还包括缺陷修复提示信息和错误等级等信息。最后，将规则文件添加到 Cppcheck 指定目录下。表中的 type 表示缺陷类型，pattern 是缺陷模式的正则表达式。

Table 2. Defect code information representation
表 2. 缺陷代码信息表示

type	pattern
输入语句缺少&符号	^scanf\("(\%[c|d|f])+"(\,[a-z])+\)
输入语句变量个数不一致	^scanf\("((\%[c|d|f])+)"(\,&[a-z])+\)
输出语句缺添加&符号	^ prinf ("%(c|d|f)+"(,& [a-z])+)
输出语句变量个数不匹配	^print\("(\%[c|d|f]){2}"(\,&[a-z]){2}\)
输出语句缺少输出变量	^print\("(\%(c|d|f))+")
返回变量地址	^return\s&[a-z]

4. 编辑距离算法及程序修复提示

编辑距离也称 Levenshtein Distance 是用来度量两个字符串之间相似度大小的指标，例如两个字符串表示为<a, b>，则编辑距离是由字符串 a 转换为 b 所需要的最小编辑操作的次数。字符串的编辑操作主要有三种：插入(Insertion)、删除(Deletion)和替换(Substitution)。两个字符串的编辑距离(Levenshtein Distance)表示为 $lev_{a,b}(i,j)$，其中的|a|和|b|分别表示字符串 a 和 b 的长度，则这两个字符串之间的编辑距离公式 $lev_{a,b}(|a|,|b|)$ 表示为：

$$lev_{a,b}(i,j)=\begin{cases}\max(i,j) & \text{if } \min(i,j)=0\\ \min\begin{cases}lev_{a,b}(i-1,j)+1\\ lev_{a,b}(i,j-1)+1\\ lev_{a,b}(i-1,j-1)+1_{(a_i\neq b_j)}\end{cases} & \text{otherwise}\end{cases} \tag{1}$$

公式中的 $lev_{a,b}(i,j)$ 表示 a 中前 i 个字符串和 b 中前 j 个字符串之间的的编辑距离，其中字符串中的第一个字符的索引从 1 开始。当公式中 $\min(i,j)=0$ 时则对应字符串 a 中的前 i 个符和 b 中前 j 个字符，那么在 i,j 中会有一个值为 0，即表示在 a 中和 b 中有一个是空字符串，则字符串 a 转换为字符串 b 只需要进行 $\max(i,j)$ 次编辑操作即可，那么两字符串之间的编辑距离大小即为 $\max(i,j)$。当 $\min(i,j)\neq 0$ 时，字符串 a 和 b 的编辑距离表示如下三种情况下的最小值：

1) $lev_{a,b}(i-1,j)+1$ 表示删除 a 的第 i 个字符。
2) $lev_{a,b}(i,j-1)+1$ 表示插入 b 中的第 j 个字符。
3) $lev_{a,b}(i-1,j-1)+1_{a\neq b}$ 表示替换 b 中的第 j 个字符。

其中，$1_{a\neq b}$ 表示指示函数，即当 $a_i=b_j$ 时取 0，当 $a\neq b$ 时，其值为 1。

我们使用编辑距离算法获取正确的修复提示代码前需要获取错误代码的行号、正确代码数据集信息以及需要对源代码去除注释等操作。在比较前需要对正确代码数据集进行分类，即只在提交的相同题目

中进行相似度比较，这样可以减少资源的消耗，提高搜索效率。去除代码注释可以减少编辑操作的次数，同时排除其他无关字符串的影响，C 语言的注释有多种，本文主要是单行代码的相似度的获取，所以只分析代码行之后的单行注释以及输出语句双引号中的内容。本文根据编辑距离算法来获取五个最优的修复推荐，即编辑距离最小的前五个推荐。基于编辑距离的代码推荐算法步骤如下所示：

1) 获取原始数据集。原始数据集中具有以往提交的题目信息，几乎包含了学生所有可能犯错的代码，以及对错误行代码的修复信息，同时每道题目和错误代码行都有唯一的 id 号。

2) 获取在线练习题。每道在线练习题目都有唯一的 id 号，此 id 号和系统以往提交的相同题目的 id 号一致，一致的 id 号保证在相同题目中进行相似度比较。学生在线提交练习题目，同时系统获取 GCC 输出的错误代码行，并对错误行代码进行保存。

3) 屏蔽注释。对获取的在线练习错误代码行进行去除注释，去除注释的算法在上一节已介绍。

4) 计算编辑距离。对步骤 1 和步骤 2 获取的代码行采用编辑距离算法进行相似度计算，将步骤 2 的错误行代码与步骤 1 中的错误行代码逐条进行编辑距离的计算，获取其中编辑距离最小的五个错误代码行的 id 号。

5) 正确提示信息。获取步骤 4 中的五个错误代码行的 id 号，并根据错误代码行的 id 号获取其对应的正确修复代码行。

5. 实验结果与分析

5.1. 缺陷检测结果

为了验证本文所设计的方法的有效性，以改进的 Cppcheck 系统为实验工具，我们以在线实验课堂上的 80 名学生为实验对象，获取其中错误比较典型的 3 道题目进行验证，共计 240 道题，所有学生提交的 3 道题目共计缺陷个数 67 个，并且与未改进的 Cppcheck 系统进行对比实验，以查准率(PR)、误报率(FNR)和漏报率(FPR)作为评价指标，各指标描述信息如下所示：

查准率(PR)表示的是正确检测到的缺陷个数占代码中总的缺陷个数的比例，公式表示为：

$$\mathrm{PR} = \frac{\mathrm{Defects}}{\mathrm{SumDefects}} \tag{2}$$

误报率(FNR)表示检测到的缺陷不是真实缺陷的数量与代码中实际缺陷数量的比例，公式表示为：

$$\mathrm{FNR} = \frac{\mathrm{WroDefects}}{\mathrm{SumDefects}} \tag{3}$$

漏报率(FPR)表示未能检测到的缺陷，即代码中实际缺陷的数量减去正确检测到的缺陷数量与代码中实际缺陷数量的比例，公式表示如下：

$$\mathrm{FPR} = \frac{\mathrm{SumDefects} - \mathrm{Defects}}{\mathrm{SumDefects}} \tag{4}$$

3 道题目的查准率、误报率和漏报率的结果如表 3 所示，与未改进的 Cppcheck 缺陷检测率对比结果如表 4 所示。

由于本文的缺陷数据集多为学生容易出错的类型，且未改进的 Cppcheck 系统无法检测，而改进后的 Cppcheck 系统检测到的缺陷数量较未改进后的 Cppcheck 系统多。由表 4 中的查准率 PR 所示的结果可知，改进后的 Cppcheck 系统缺陷检测的平均准确率为 63.7%，未改进的 Cppcheck 系统缺陷检测平均准确率为 26%，本文改进 Cppcheck 后缺陷检测的能力有了显著的提升，能够检测出更多的学生代码缺陷误。

Table 3. Cppcheck defect detection results
表 3. Cppcheck 缺陷检测结果

名称	改进的 Cppcheck			未改进的 Cppcheck		
	SumDefects	Defects	WroDefects	SumDefects	Defects	WroDefects
Q19	21	11	3	21	4	0
Q57	19	13	4	19	7	1
Q68	27	19	3	27	6	1

Table 4. Improved and unimproved Cppcheck defect detection rates
表 4. 改进与未改进的 Cppcheck 缺陷检测率

名称	改进的 Cppcheck(%)			未改进的 Cppcheck(%)		
	PR	FNR	FPR	PR	FNR	FPR
Q19	52.4	1.4	47.6	19	0	90
Q57	68.4	2.1	31.6	36.8	5.3	63.2
Q68	70.4	1.1	29.6	22.2	3.7	77.8

由表 4 误报率 FNR 数据可知，改进后的系统的平均误报率为 1.5%，相较于未改进的系统的误报率为 3%，误报率降低不是非常明显，主要是由于本文的缺陷类型是学生题目中常见的，改进后的系统可以检测出其中的多数，而未改进的系统只能检测出其中的少数，不会出现太多的检测到的缺陷不是真实缺陷的情况。

由表 4 漏报率 FPR 数据可以看出，改进后的系统的平均漏报率为 36.3%，而未改进后的系统的平均漏报率为 77%，本文改进后的平均漏报率有了明显的降低。

通过实验结果可以看出，本文改进后的 Cppcheck 系统在各个方面较未改进的系统有了很大的提升，能够检测到更多的代码缺陷，提高了查准率，降低了漏报率，可以对学生在线练习的题目进行检测。

5.2. 缺陷检测结果

本文使用基于编辑距离的代码推荐算法显示五条语法错误的代码修复提示信息，系统可以对一道题目的语法错误进行提示，同时也可以对多道题目的语法错误进行提示，错误提示使用的是 HTML 页面展示的样式，采用 HTML 页面中的表格可以清晰的展示代码推荐的详细信息，并根据编辑距离的大小倒叙排列推荐结果。

学生代码库中保存了多种学生程序出现的语法错误，学生可以根据编辑距离算法获取修复提示，但是对于一些代码库中未出现过的语法错误，学生需要借助 GCC 的错误提示，或者老师的指导对其进行修复，并且系统会把修复结果保存到代码库中，后续学生练习再次遇到此类语法错误时可以获取到修复提示。

图 4 展示了三个语法错误的提示，第一个是代码行 if(b>='A' && b<='Z)的错误，本行代码是由于 if 语句中的<=符号后面的变量 Z 缺少引号导致的语法错误，由于此种错误出现的较多，且修复较固定，使得获取的五条修复都是一样的。第二个是代码行 z=x+y 后缺少分号导致出现语法错误。第三个是输出语句 printf(“%d,t)的括号内缺少双引号导致出现语法错误。图中属性 codeId 表示题目的 id 号，line 表示错误代码行号，code 表示错误行代码，recommend 表示五条修复提示。

codeId	line	code	recommend
096	7	if(b>='A' && b<='Z)	if(b>='A' && b<='Z')
			if(b>='A' && b<='Z')
			if(b>='A' && b<='Z')
			if(b>='A' && b<='Z')
			if(b>='A' && b<='Z')
109	3	z=x+y	z=x+y;
			z=x+y;
			z=y+x;
			y=x+z;
			y=x+z;
043	9	printf("%d,t);	printf("%d",t);
			printf("%d",t);
			printf("%d",t);
			printf("%f",x);
			printf("%f",z);

Figure 4. Code recommendation result graph
图 4. 代码推荐结果图

6. 结语

本文基于编辑距离算法和 Cppcheck 静态代码检测工具设计了一种新的 C 语言程序分析与修复方法，该方法主要面向所有专业学生及各类程序设计的初学者，旨在快速高效的对学生程序中的错误给与修复提示信息，提高了学生的编程效率，并且减轻了教师的教学负担。同时，基于以上技术设计并实现了相关模块应用在 C 语言在线实验系统中，在疫情期间的线上教学中发挥了较大作用，取得了不错的应用效果。在未来的工作中，将继续优化代码辅助纠错功能，总结更多的程序错误模式，增加代码逻辑错误的自动修复功能。

基金项目

上海工程技术大学教学建设项目 y202202002 混合教学模式下数据驱动的教学手段数字化改革研究与探索——以程序设计基础课程为例。

参考文献

[1] 曾新励. 基于 SonarQube 平台提升企业软件项目质量的应用研究[J]. 电脑知识与技术, 2019, 15(11): 78-80. https://doi.org/10.14004/j.cnki.ckt.2019.1106

[2] 徐海燕, 姜瑛. 基于用户评论的代码质量识别与分析[J]. 计算机科学, 2020, 47(3): 41-47.

[3] 郑炜, 陈军正, 吴潇雪, 陈翔, 夏鑫. 基于深度学习的安全缺陷报告预测方法实证研究[J]. 软件学报, 2020, 31(5): 1294-1313. https://doi.org/10.13328/j.cnki.jos.005954

[4] 王甜甜, 许家欢, 王克朝, 苏小红. 示例演化驱动的学生程序自动修复[J]. 软件学报, 2019, 30(5): 1256-1268. https://doi.org/10.13328/j.cnki.jos.005716

[5] Kim, D., Kwon, Y., Liu, P., *et al*. (2016) Apex: Automatic Programming Assignment Error Explanation. *ACM SIGPLAN Notices*, **51**, 311-327. https://doi.org/10.1145/3022671.2984031

[6] Yi, J., Ahmed, U.Z., Karkare, A., Tan, S.H. and Roychoudhury, A. (2017) A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. *ESEC/FSE* 2017: *Proceedings of the* 2017 11*th Joint Meeting on Foundations of Software Engineering*, 740-751. https://doi.org/10.1145/3106237.3106262

[7] Weimer, W., Nguyen, T.V., Le Goues, C. and Forrest, S. (2009) Automatically Finding Patches Using Genetic Program-

ming. 2009 *IEEE* 31*st International Conference on Software Engineering*, Vancouver, 16-24 May 2009, 364-374. https://doi.org/10.1109/ICSE.2009.5070536

[8] Long, F. and Rinard, M. (2016) Automatic Patch Generation by Learning Correct Code. *ACM SIGPLAN Notices*, **51**, 298-312. https://doi.org/10.1145/2914770.2837617

[9] Mechtaev, S., Yi, J. and Roychoudhury, A. (2016) Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis. *IEEE*/*ACM* 38*th International Conference on Software Engineering* (*ICSE*), Austin, May 2016, 691-701. https://doi.org/10.1145/2884781.2884807

[10] Weimer, W., Fry, Z.P. and Forrest, S. (2013) Leveraging Program Equivalence for Adaptive Program Repair. 2013 28*th IEEE*/*ACM International Conference on Automated Software Engineering* (*ASE*), Silicon Valley, 11-15 November 2013, 356-366. https://doi.org/10.1109/ASE.2013.6693094

[11] 林渤, 薛斌, 林逸滔, 胡建鹏. 面向初学者的 C 语言在线实验系统设计与实现[J]. 智能计算机与应用, 2020, 10(10): 108-111.

[12] 张仕金, 尚赵伟. Cppcheck 的软件缺陷模式分析与定位[J]. 计算机工程与应用, 2015, 51(3): 69-73.