Journal of Computer Applications
计算机应用,2017,37(10):2958 -2963
ISSN 1001-9081
CODEN JYIIDU
2017-10-10
http://www.joca.cn

文章编号:1001-9081(2017)10-2958-06
DOI:10.11772/j.issn.1001-9081.2017.10.2958

基于语法和语义结合的源代码精确搜索方法

顾逸圣[1,2*],曾国荪[1]
(1.同济大学 计算机科学及技术系,上海 200092; 2.嵌入式系统与服务计算教育部重点实验室,上海 200092)
(*通信作者电子邮箱 qswy929@163.com)

摘 要:针对在编写软件、复用源代码的过程中仅依靠关键词无法精准搜索到适用源代码的问题,提出一种将语法和语义结合的源代码精准搜索方法。首先依据源代码语法语义的客观和唯一性,增加语法结构和"输入/输出"语义作为用户录入请求的一部分,并规范了具体的请求格式;然后在此基础上分别设计源代码语法匹配算法、"输入/输出"语义匹配算法、关键词兼容匹配,以及源代码搜索结果可信度计算算法;最后综合上述算法实现对源代码的精准搜索。测试结果表明:与单纯的关键词搜索相比,提出的方法对搜索的平均排序倒数(MRR)有超过62%的提升,有助于实现源代码的精准搜索。

关键词:软件编写;源代码复用;语法语义;匹配搜索
中图分类号:TP311.1 **文献标志码**:A

Accurate search method for source code by combining syntactic and semantic queries

GU Yisheng[1,2*], ZENG Guosun[1]
(1. *Department of Computer Science and Technology, Tongji University, Shanghai* 200092, *China*;
2. *Embedded System and Service Computing Key Laboratory of Ministry of Education, Shanghai* 200092, *China*)

Abstract: In the process of programming and source code reuse, since simple keyword-based code search often leads to inaccurate results, an accurate search method for source code was proposed. Firstly, according to the objectivity and uniqueness of syntax and semantics, the syntactic structure and semantics of I/O of a function in source code were considered as part of a query. Such query should be submitted following a regularized format. Secondly, the syntactic structure, semantics of I/O, keyword-compatible match algorithms along with the reliability calculation algorithm were designed. Finally, the accurate search method by combining syntactic and semantic queries was realized by using the above algorithms. The test result shows that the proposed method can improve Mean Reciprocal Rank (MRR) by more than 62% compared with the common keyword-based search method, and it is effective in improving the accuracy of source code search.

Key words: software programming; source code reuse; syntax and semantics; matching search

0 引言

在信息化时代,社会运作的方方面面都离不开软件。在软件编写的过程中,大量的源代码无疑是一笔笔宝贵的"财富"。为了尽可能利用这些"财富",有人试图将源代码看作普通的文本,借助通用搜索引擎的技术,建立基于关键词的源代码搜索引擎。这种方法技术成熟,实施成本低,但是基于关键词的搜索方法最大的问题是结果不精确,这个问题在目前的源代码搜索引擎中普遍存在:返回的结果中往往混杂着声明、类定义等,造成用户选择困难。为此,学者们利用例如语法、语义等其他多种信息,开展开源代码搜索,以提高搜索精度。

在基于源代码语法的搜索方面:Bajracharya 等[1]将源代码搜索从基于关键词的全文搜索变为基于源代码结构的搜索,支持"细节搜索",并将其分为"Components""Component Use"等五类。Farah 等[2]对源代码搜索引擎 Open Hub 深入研究,总结出其本质就是将搜索需求按函数、结构体、类、接口等进行分类。以上方法对搜索内容进行类别细分,但只是简单根据代码的组成并没有实质涉及语法的研究。Paul 等[3]则进一步通过定义一系列的通配符,对 C 语言的语法集进行抽象形成代码模式自动机(Code Pattern Automaton, CPA),通过解释器进行匹配,充分展示了利用语法进行抽象搜索的可行性,然而此方法接收的通配符描述过于复杂,难以推广。在基于源代码语义的搜索方面:Hill 等[4]提出的"自然语言源码定位"可应用于海量数据,将源代码中的变量名称进行语义猜测,构建一个包含海量源码变量使用关系与词素拓展集的数据库。胡翔等[5]将 Hill 的方法运用于源代码搜索,并结合传统的程序分析技术,提出一种海量源码分析的新方法。孟骁[6]通过以语义网络为文本关键字建立近义词索引的方法,改善了代码搜索问题中近义词无法被关键字匹配的问题。上述方法增加了源代码搜索输入的容错度,但是只是对其中某些元素的释义,并没有提取出源代码蕴涵的功能语义。刘石等[7]利用"超链接引导的主题搜索"(Hyperlink-Induced Topic Search, HITS)算法分析技术对 Java 语言的代码模型进行二

收稿日期:2017-04-21;**修回日期**:2017-06-09。
基金项目:上海市优秀学科带头人计划项目(10XD1404400);同济大学实验教改项目(0800104214)。
作者简介:顾逸圣(1992—),男,上海人,硕士研究生,主要研究方向:软件工程、代码搜索; 曾国荪(1964—),男,江西吉安人,教授,博士,博士生导师,主要研究方向:并行计算、可信软件、信息安全。

次周游,从而实现了对搜索应用程序编程接口(Application Programming Interface, API)实现代码和调用代码的区分、并从搜索结果摘要等方面进行了优化。Keivanloo 等[8]利用本体描述语言(Ontology Web Language, OWL)对面向对象的编程语言的各个属性进行语义分析并建模,解决了搜索到的源代码存在外部依赖关系从而不完整的问题。Stolee 等[9-10]则提出了一种全新的基于"输入/输出"的源代码搜索方式,编写轻量级的需求描述作为输入和期望的输出并对其分析,将程序的语义与描述其行为的约束进行映射;因为其分析过程的复杂性,目前该方法还只是停留在理论阶段。

可见,借助于源代码的语法或语义信息,搜索源代码的方式能够更加丰富,有助提高搜索的准确率。然而,纵观上述研究,目前的研究多数只针对语法、语义的其中一个方面展开。现阶段将语法与语义相结合,共同运用于源代码的搜索的工作现阶段开展较少。本文拟将函数作为源代码搜索的粒度,提取并利用函数源代码的语法结构和"输入/输出"语义信息进行搜索,通过语法和语义相结合的手段探索丰富源代码搜索录入形式、提高搜索精准度的途径。

1　源代码语法和语义的客观性

源代码是一段按照某种程序设计语言规范编写的形式化文本,因此反映了该语言的特征。记源代码 L 为三元组:$L=(T, G, S)$。其中:T 是整个源代码的文本;G 是文本的语法规则;S 是文本的语义规则。本章以 C 语言为例,说明源代码的语法和语义的客观性和唯一性。

1.1　语法和语义的客观性

自然语言中,为了确保某个区域内个体间信息交换的顺畅进行,对于用来交流、接收和传递信息的字符串,必须遵守相同的一组规则[11],如汉语规则、英语规则等。类似地,为了使计算机能够识别输入的源代码字符串,必须同样对源代码字符串的组成制定规则,这都属于语法的范畴。

源代码的语法是指有一组规则,用其能够形成和产生一个计算机程序。这组规则中包括了单词符号的形成规则,以及如何从单词符号形成表达式、语句、函数等语言规则。设 s 是程序设计语言源代码中的某一句,则语法的客观性体现在:$\forall s((s \in T) \Rightarrow \exists !f(x) \wedge (f(s) \in G))$,其中 $f(x)$ 是一个从单词符号串 x 到语言规则的映射。即每一条语句都能找到它对应的语言规则。映射 f 实际中可以采用巴科斯范式(Backus-Naur Form, BNF)等形式表现。很难想象,仅仅几十条 BNF 规则就可以描述所有 C 语言源代码的语法。如 C 语言的赋值语句就可以表示为以下的 BNF,通过它就能客观确定所有赋值语句的操作。

〈assignment_exp〉 ::=〈conditional_exp〉 | 〈unary_exp〉〈assignment_operator〉〈assignment_exp〉;

〈assignment_operator〉 ::= '=' | '*=' | '/=' | '%=' | '+=' | '-=' | '<<=' | '>>=' | '&=' | '^=' | '|='

源代码的语义是指能够用其定义源代码的功能和意义的一组规则[12]。无论是源代码执行前还是执行后的语义,都可以归类在操作语义、指称语义、公理语义之中。语义的客观性可以从源代码的功能、正确性等方面体现。例如对整型变量 Y 执行"$Y=5$"这条赋值语句就可用操作语义"$\langle Y:=5,\sigma\rangle \rightarrow \sigma'$"表示,其中:$\sigma$ 表示的是状态函数,σ' 表示执行完赋值命令后的终态。执行 1 到 100 整数求和的操作的语义正确性可以借助以下公理:"$P:=0;\ I:=1\ \{P=0 \wedge I=1\}$ (while¬ ($I=101$) do $P:=P+I;\ I:=I+1$)"验证。

1.2　语法和语义的唯一性

每个人在世上都是唯一的个体,在思想、性格、行为等方面的不同造就了其个性。由于个性差异对于同一件事不同人有全然不同的处理方式:到达同一个目的地会选择不同的出行方式;面对同样的题目能够写出不同的文章等。为实现相同的功能,不同的程序员会有全然不同的编写源代码的方式。因为每条语句都可能有区别,源代码这种近乎的唯一性也会传递到其中蕴涵的语法和语义的信息中,成为其相互区分的特征。如果认为不同的人很容易写出两段格式、变量、算法都一样的源代码,那就无异于忽略了人的个性因素。

1.3　语法和语义在源代码搜索中的用途

源代码语法和语义的客观性和唯一性对于源代码的搜索有很大的帮助。假设 T_1、T_2 是两段源代码文本,$G\circ S(X)$ 是对任意源代码文本串 X 进行语法和语义分析后的结果,那么有:$G\circ S(T_1)=G\circ S(T_2)\rightarrow T_1=T_2$。换而言之,对任意的源代码都可以分析其蕴涵的语法和语义信息;通过这两方面的信息,也可以唯一确定所需的源代码,因而通过语法和语义信息来搜索源代码理论上是可行的。本文的目的就是设计一种切实可行的方法以表达和分析这些信息,为搜索作铺垫。特别说明的是,本文中的源代码指的是"函数片段",即一段完整的函数源代码。

2　用户搜索录入需求的描述

2.1　用户搜索请求表达的困难性

虽然源代码的语法和语义具有客观性与唯一性,但正如人往往很难仅依靠语言或者文字表达自己的需求,同样,对于某时某刻需要什么样的源代码,搜索者也是很难完全清楚地描述。在描述时,由于交互方式的局限,关键词是一种为数不多的对文本信息的检索手段,因此现今用户采取录入关键词的方式提交搜索请求。单纯的关键词无法对问题精准描述,却又没有找到比之更简单有效的方法,这正是目前用户表达搜索请求的困难之处。

2.2　关键词搜索情况的特性分析

传统的利用关键词对所需源代码进行搜索,依靠的是关键词中对语法和语义信息的部分表达,是模糊不完整的。用户既不能准确地归结出与所需源代码密切相关的关键词,依靠关键词也多半不能获取相关的源代码。例如 $K=\{k_1, k_2, \cdots, k_n\}$ 是在搜索某一源代码时的一组关键词,R_s 为对应的结果,R_e 为用户期望的结果,C 为阈值常量,则 $\exists K((Input(K)\Rightarrow |R_s|) \wedge (|R_s| \cap |R_e| < C))$,且只是在不区分输入顺序的情况下返回 n 个关键词的并集,这便暴露了传统关键词搜索的不精确性。进而,如果能在搜索时考虑到关键词出现的位置及顺序,以区分关键词的重要程度,结合客观的语法和语义,那么这种改进后的搜索将能大幅提升搜索的精准度。

2.3　语法和语义搜索请求的表达法

仅凭关键词搜索源代码依然难以表达用户需求,搜索往往不精确,应当增加一些请求表达的方式,以便用户提供尽可能多的信息。本文提出一种将语法、语义、关键词三者结合用于源代码的搜索的方法。其中,关键词搜索的形式用户经常接触,不需要另行设计,而用户对语法和语义的搜索表达形式则需特别交代。

源代码的语法信息包括多种,本文选择结构信息用以描述源代码语法。其中,诸如抽象语法树信息虽是在源代码语法分析阶段一一对应生成,表示的结构信息完备且不存在二义性,但是图的结构使其注定不满足方便用户录入的要求。因而本文采取一种折中的方法,仅将源代码中分支和循环结构的描述作为用户录入的语法请求表达。用户的请求可以分为两种情形。第一种情形下用户清楚所需函数源代码的语法结构及顺序,此时定义符号“c:( )”表示一个分支结构,“l:( )”表示一个循环结构。用户录入这两个符号的先后及嵌套顺序就代表了源代码中分支和循环的组成:如请求“l:(c:( ))”就表示先出现循环并且其中嵌套一个分支,“l:( )c:( )”表示先循环,后跟随一个分支。第二种情形下用户只能提供每种结构出现的数目,则可以分别录入分支、循环等结构的数目,来搜索所需的源代码。这种以分支和循环结构作为源代码语法特征的方式,虽然不如语法树那样完备,比如无法表示代码中的递归结构,但通过确定这两种结构的数目或者出现的顺序及嵌套关系,都能辅助过滤过大量无关代码,是一种利用语法信息提高搜索精准度的可行之策。

同样,源代码的语义信息有多种,本文将代码的“输入/输出”功能语义作为让用户录入的语义信息。对函数参数表中的参数以及返回的参数的读/写通常反映了函数源代码的功能,所以本文中将这些参数视为源代码的“输入/输出”。在录入请求的阶段,用户使用文本串对“输入/输出”的参数类型和名称进行描述。定义语义请求为“$itype_1$ $iname_1$,$itype_2$ $iname_2$,…;otype oname”的形式,其中分号以左为函数接收的参数表,以右为函数的返回参数。type 是“输入/输出”的参数类型,name 是“输入/输出”参数名称。特别地,当参数为空时,规定 type 和 name 均为 void。这样定义语义请求,形式上直观全面地表达了“输入/输出”语义,用户操作时录入的难度也较低。不过参数的顺序和数目是“输入/输出”语义重要特征,因而用户在搜索录入时要注意尽量精确描述请求中的每个参数及它们的次序,这些都可作为语义匹配的依据。

2.4 用户搜索录入需求操作界面

根据 2.3 节的讨论,可以设计出图 1 中的界面,等待用户输入源代码搜索请求。按提示录入了对所需源代码的描述后,三个输入框中的文本分别记作 *kquery*、*squery* 和 *ioquery*,它们就构成了用户的请求表达语句,记作 *query*。下一步的工作就将针对用户在此提交的各项请求表达,分析源代码中的对应的信息,并按特定方法进行匹配。

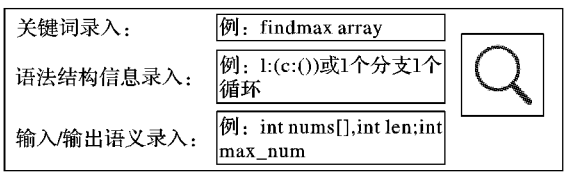

图 1 搜索请求录入界面

Fig. 1 Interface for inputting search requests

3 源代码语法和语义的分析

3.1 语法结构信息分析算法

在 2.3 节中明确了用户通过“c:( )”“l:( )”两种符号的组合表示源代码的语法结构请求,那么在分析目标源代码结构时,一种自然而然的思想,就是分析其中表示结构信息的保留字标志,将一段函数源代码中分支和循环结构信息的出现关系同样利用上述两种符号表达,形成语法结构串。这样在匹配时通过分别统计每一种符号出现的总数,以及符号的先后顺序、嵌套关系,比较所有的特征是否相符,就能够判定用户请求是否与当前源代码相匹配。以分析对象是一段 C 语言的函数源代码文件(用变量 *one_src_file*)为例,算法 1 描述了对其进行语法结构分析的整个过程。

算法 1 语法结构信息分析匹配算法 syntaxStructMatch。

输入 语法信息搜索录入 *squery*,一段函数的源代码文件 *one_src_file*。

输出 源代码文件是否语法结构匹配 True/False。

```
syntaxStructMatch(squery, one_src_file)
{if (isEmpty(squery))
   return True;                          //请求为空时直接返回“匹配”
  condition←{"if","else","case",…};      //分支结构保留字
  loop←{"for","do","while",…};           //循环结构保留字
  nest←{"{","}",…};                      //结构嵌套保留字
  text←readSourceCode(one_src_file);
  struct←geneStructString(text, condition, loop, nest);
                          //构造用3.3节两种符号表达的语法结构串
  if (squery = struct)      //源代码文件与用户语法搜索需求相同
   return True;                                    //语法结构匹配
  if (sameCount(squery, struct))
                          //录入的分支、循环的数目与目标相同
   return True;                          //近似认为语法结构匹配
  else
   return False;                                   //语法结构不匹配
}
```

执行算法 1 后,假设用户录入为“l:(l:( ))”,则有且仅有双重循环结构的源代码的分析结果为“l:(l:( ))”,即与搜索匹配。若录入的只是查找含有“2 个循环”的源代码,除了双重循环的源代码,单独循环两次的源代码也会被判定为匹配,这在实际应用过程中可能是合理的。

3.2 “输入/输出”语义信息分析算法

在 2.3 节中同样明确了用户“输入/输出”语义请求的表达方式,故在分析目标源代码时,也可先将其“输入/输出”按照相同的规则构成语义串。不过考虑到搜索请求中参数名、数量、类型等信息不一定完全精确,不能用请求与语义串是否完全相同来判定是否匹配。有时,源代码接收的参数名或是参数顺序即使存在出入,它所表示的“输入/输出”语义也是相同的。因而需要定义一个请求与源代码的“输入/输出”的匹配度 *degree*,作为语义信息之一,并将其与函数名中的词语、所有“输入/输出”的变量名因素综合起来,通过进一步的匹配才是更为科学的方法。获取源代码的函数名和参数表通常较为简单,至于匹配度 *degree* 的定义,设用户语义请求中包含的参数数目为 $c_{ioquery}$,源代码中包含的参数数目为 c_{inout},请求与源代码语义串中名称或类型相同的参数数目为 u,名称的出现位次一致的参数数目为 n,类型的出现位次一致的参数数目为 t,则 $degree = \frac{u*(n+t)}{2*c_{ioquery}*c_{inout}}$。例如对于一个 C 语言源代码文件 *one_src_file*,执行算法 2 便可获取其上述三方面的语义信息。

算法 2 “输入/输出”的语义信息分析匹配算法 inoutMatch。

输入 语义信息搜索录入 *ioquery*,一段函数的源代码文件 *one_src_file*。

输出 函数名 *Fname*,参数名 *Pname*,匹配度 *degree*。

inoutMatch(*ioquery*, *one_src_file*)。

```
{ line ← readDefineLine(one_src_file);        // 读取函数定义行
  Fname ← wordSegment(line);                  // 分词后的函数名
  text ← readSourceCode(one_src_file);
  ⟨Itype, Iname, otype, oname⟩ ← getParaInfo(line, text);
                    // 获取函数"输入/输出"参数的类型和名称
  if (isEmpty(ioquery))
    degree ← 0;                               // 输入请求为空,degree 为 0
  else
  { inout ← geneInoutString(Itype, Iname, otype, oname);
                    // 构造同 3.3 节用户语义请求形式的语义串
    degree ← u(ioquery, inout) * (n(ioquery, inout) + t(ioquery, inout)) / (2 * c_ioquery * c_inout);
                                              // 计算匹配度
  }
  Pname ← Iname + {oname};
  return Fname, Pname, degree;
}
```

例如,用户语义请求录入为"int a, float b; int c",而某段源代码的语义串经过分析为"byte a, float b; int c",那么 $c_{ioquery} = c_{inout} = 3$,$u = 2$(float b 和 int c),$n = 3$($a$、$b$ 和 c),$t = 2$(float 和 int),$degree$ 为 5/9。

3.3 关键词匹配及可信度计算

算法 2 返回的源代码三方面的语义信息,将会在关键词匹配和代码可信度计算中再次用到。本节引入 3 个匹配得分变量:*fscore* 是录入的关键词和源代码函数名的匹配得分,通过找出分词后的用户关键词和函数名中的公共单词,将这些单词在两者中的出现位次作为向量每一维的值,构造成两个向量,计算相似度而得。类似地,*pscore* 是录入的关键词和源代码参数名的匹配得分,利用分词后的用户关键词、"输入/输出"参数名和两者的公共单词,构造两个向量,计算相似度而得。*score* 则是总的可信度得分,由 *fscore*、*pscore* 传统的"词频-逆文档频率"(Term Frequency-Inverse Document Frequency, TF-IDF)关键词全文匹配度[13]和之前求得的语义匹配度 *degree*,四者的加权和构成。这是一种将多方面的有效信息充分合理利用的手段。以一个源代码文件 *one_src_file* 为对象,将关键词匹配及可信度计算的过程在算法 3 中展现。

算法 3　可信度计算算法 trustedScore。

输入　录入的关键词请求 *kquery*、*one_src_file*、*Fname*、*Pname*、*degree*。

输出　*one_src_file* 的可信度 *score*。

```
trustedScore(kquery, one_src_file, Fname, Pname, degree)
{ if (isEmpty(kquery))                        // 关键词搜索请求为空
  { fscore ← 0, pscore ← 0; }
  else
  { Words ← wordSegment(kquery);              // 析取出多个关键词
    Fwords ← {f | f ∈ Words ∩ Fname};
                    // 求关键词和函数名中公共及匹配的单词集合
    Pwords ← {p | p ∈ Words ∩ Pname};
                    // 求关键词和参数名中公共及匹配的单词集合
    if (| Fwords | = 1)
      fscore ← 1 / | Words |;
    else
      fscore ← findSimilarity(Words, Fname, Fwords);
                    // 根据录入关键词和源代码函数名计算匹配得分
    if (| Pwords | = 1)
      pscore ← 1 / | Words |;
    else
      pscore ← findSimilarity(Words, Pname, Pwords);
                    // 根据录入关键词和函数参数名计算匹配得分
  }
  score ← a1 * fscore + a2 * pscore +
          a3 * tf_idf(Words, one_scr_file) + a4 * degree;
                    // tf_idf():计算传统的关键词全文匹配度
  return score;
}
```

4　语法和语义结合的精确搜索算法

4.1 搜索匹配原理及过程

源代码搜索的最终目的是满足用户需求,既然用户在搜索时已经提供了语法、语义和关键词三方面的请求,那就应该充分利用这些信息,匹配用户的请求。整个源代码精准搜索的过程如下:1)提取和过滤用户的搜索请求,进行预处理,分别形成对应的规范格式。2)进行语法精确匹配:源代码语法结构信息可以精确获取,因此可以精确匹配用户的语法要求,只有当源代码的语法结构完全符合用户搜索时提供的语法要求,这样的源代码才是有效的,特别用户搜索时提供的语法要求为"空"时,则认为任何源代码都是有效的。如果语法匹配失败,则此次源代码搜索失败、结束。3)进行语义近似匹配:正如 3.2 节所述,源代码的语义信息很难精确获取,所以只能用语义匹配度来近似。本文通过分析源代码隐含的输入、输出变量特征,比较匹配用户搜索时提供的输入、输出请求,显然匹配度高对应的源代码更符合用户要求。4)进行关键字匹配,获得本次搜索源代码的可信度分值,为大量搜索结果排序提高了依据。关键字匹配过程一方面利用第 2)~3)步的语法、语义匹配的结果,提高了搜索的精度,另一方面也可兼容传统搜索引擎的方法。5)重复第 2)~4)步,直至搜索了规定个数的源代码文件,可信度高的源代码文件排在前面。

4.2 搜索算法

用户通过图 1 界面录入了请求表达式后,根据 4.1 节的匹配原理,利用算法 4"搜索精化算法",即可更精确地返回满足用户需求的函数源代码。特别地,该算法能够根据给定的值设置限制搜索的代码数量,以防搜索时间过长。

算法 4　搜索精化算法 searchRefine。

输入　用户录入搜索请求 *query*,待搜索的源代码库 *src_code_database*,限制搜索的代码个数 *n_max*。

输出　符合条件的函数源代码文件集合 *Ok_files*。

```
searchRefine(query, src_code_database, n_max)
{ Rfiles ← ∅, Rscores ← ∅;
                    // 记录待排序文件及得分的有序集
  searchCount ← 0;                            // 已搜索的次数
  kquery ← getKquery(query);                  // 获取预处理后的关键词
  squery ← getSquery(query);                  // 获取录入的语法请求表达式
  ioquery ← getIOquery(query);                // 获取录入的语义请求表达式
  if (isEmpty(query))                         // 用户请求全空
  { Ok_files ← random(src_code_database, n_max);
                                              // 随机返回若干文件
    return Ok_files;
  }
  while (one_src_file ← nextFile(src_code_database) and
         searchCount ≤ n_max)
  { if (syntaxStructMatch(squery, one_src_file))
                                              // 算法 1 语法结构分析匹配
    { ⟨Fname, Pname, degree⟩
```

```
        ← inoutMatch(ioquery, one_src_file);
                                    // 算法 2 语义信息分析匹配
        Rscores ← Rscores + {trustedScore(kquery, one_src_file,
          Fname, Pname, degree)};
                                    // 算法 3 可信度计算
        Rfiles ← Rfiles + {one_src_file};      // 记录对应的文件
      }
      searchCount ← searchCount + 1;
    }
    Ok_files ← sortByScore(Rfiles, Rscores);
                    // 根据可信度的降序,返回结果文件集合
    return Ok_files;
  }
```

5 搜索测试及分析

5.1 测试环境

关于测试的硬件环境,使用的计算机的 CPU 为 2.7 GHz Intel Core i5,内存为 8 GB。在软件方面,进行测试前编写了 Java 程序,接收用户的语法、语义和关键词请求,将用户的关键词请求转发到 SearchCode 源代码搜索引擎,把前 100 项结果对应的源文件保存到本地,按函数为粒度拆分后进一步调用算法 4 的 Java 实现版本进行搜索排序,并在最后以函数为粒度返回结果。

5.2 测试用例

本文将搜索表 1 中常见的 5 种算法的 C 语言代码作为测试的用例。测试分为两组:第一组在 SearchCode 引擎中录入表 1 的关键词部分进行搜索;第二组使用 5.1 节编写的程序,除了利用相同的关键词,同时加入每个算法用例可能包含的语法结构和"输入/输出"语义的描述信息。程序中进行可信度计算时的权值 $(a_1, a_2, a_3, a_4) = (0.3, 0.15, 0.25, 0.3)$,根据每一项的重要程度按经验设置;限制搜索的代码个数 *n_max* 设置为 100。对于两组的返回结果,通过统计首条与搜索请求相关结果的排名,并对 5 次搜索的平均排序倒数(Mean Reciprocal Rank, MRR)[14]进行计算,来衡量搜索的精准程度。

表 1 搜索测试用例

Tab. 1 Test cases for search

用例序号	用例描述	录入信息		
		关键词	语法	语义
1	对整型数组的冒泡排序	bubbleSort array	1;(1;(c;()))	int * data,int size;void void
2	判断一个数是否为素数	isPrime	1;(c())	int num;bool result
3	删除单链表指定的节点	delete linklist node	1;(c(,))	linklist list,int loc;void void
4	迪杰斯特拉算法	Dijkstra distance	不录入	graph g,int nodeNum;void void
5	KMP 模式匹配算法	kmp match pattern	不录入	string srcstr,string findstr;int pos

5.3 结果分析

图 2 为各个用例的搜索测试结果。可以看出相对于第一组,第二组中有 4 个用例的首条相关结果的排名更靠前,一个用例排名持平。分别记两组的 MRR 为 MRR_1 和 MRR_2,通过图 2 的排名计算可得 $MRR_1 = 0.3785$,$MRR_2 = 0.6167$。测试中本文算法对该指标有超过 62% 的提升。

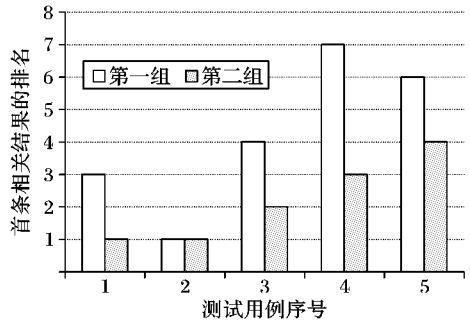

图 2 搜索测试的排名结果

Fig. 2 Rank result of the search test

进一步以用例 1 为例,列出第二组测试中返回的前 10 项函数结果、对应的可信度,以及在 SearchCode 中的原始排名(通过在其搜索页面录入关键词,并筛选 C 语言源代码后获得),见表 2。

经过统计,排名前 10 的结果中只有 4 条结果同样在 SearchCode 中排在前十,其余都是原先排名靠后的结果。但通过分析这些结果的源代码发现,它们又的确都是和用户搜索需求高度匹配的,反而是在原始排名的前十项中包含了与请求无关的信息。究其原因,不外乎是在 SearchCode 引擎中单单录入关键词不能利用到源代码蕴含的语法和语义信息,导致搜索的不精准,而本文结合了源代码语法和语义的方法则改善了这一问题。测试结果说明:利用了源代码语法、语义信息搜索得到的结果更准确可信。

表 2 第二组测试中用例 1 的详细结果

Tab. 2 Detailed result of case 1 in the 2nd test

测试排名	返回的源代码函数	可信度 *score*	原始排名
1	void bubble_sort(int * array, int size)	0.7792	18
2	void bubbleSort(int s[], int size)	0.5292	25
3	void bubblesort(int array[])	0.4057	11
4	void bubbleSort(int numbers[], int array_size)	0.3847	5
5	int bubbleSort(int a[], int size)	0.3670	24
6	void bubblesort(int numbers[])	0.3224	4
7	void bubblesort(int a[], int n)	0.3223	32
8	void bubble_srt(int a[],int n)	0.2890	3
9	void bubble(int v[], int n)	0.2070	43
10	void bubsort(int v[], int n)	0.2056	8

6 结语

当前,每天都会产生无数的源代码,人们对搜索源代码的需求愈发渴望,然而目前基于单纯关键词的搜索却无法达到精准搜索。本文提出了一种基于语法和语义结合的源代码精确搜索方法。该方法通过利用源代码本身的语法和语义信

息,接收经过设计的用户语法结构信息、“输入/输出”语义信息及关键词搜索请求表达式,分别提出了相应的算法将用户的请求与源代码进行匹配、计算可信度,并最终按可信度排序。对于相同的测试对象,将本文方法与单纯的关键词搜索比较,MRR 指标提升显著,且排名靠前的那些源代码确为所需,以此验证了方法的可行性。当然,源代码的语法和语义信息类型有很多,本文对于源代码其他方面的语法语义尚未充分发掘。下一步的研究工作将是对源代码中相关更深层次的信息的挖掘和利用。

参考文献(References)

[1] BAJRACHARYA S, NGO T, LINSTEAD E, et al. Sourcerer: a search engine for open source code supporting structure-based search [C]// OOPSLA 2006: Companion To the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications. New York: ACM, 2006: 681 - 682.

[2] FARAH G, TEJADA J S, CORREAL D. OpenHub: a scalable architecture for the analysis of software quality attributes [C]// MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories. New York: ACM, 2014: 420 - 423.

[3] PAUL S, PRAKASH A. A framework for source code search using program patterns[J]. IEEE Transactions on Software Engineering, 1994, 20(6): 463 - 475.

[4] HILL E, POLLOCK L, VIJAY-SHANKER K. Improving source code search with natural language phrasal representations of method signatures [C]// ASE 2011: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. Washington, DC: IEEE Computer Society, 2011: 524 - 527.

[5] 胡翔, 舒礼莲. 基于语义网络的海量源代码搜索引擎[J]. 计算机与现代化, 2014, 30(7): 19 - 23. (HU X, SHU L L. Search engine of massive source code based on semantic network[J]. Computer and Modernization, 2014, 30(7): 19 - 23.)

[6] 孟骁. 基于语义网络的智能搜索引擎研究[D]. 长春: 东北师范大学, 2011. (MENG X. Research of intelligent search engine based on semantic Web[D]. Changchun: Northeast Normal University, 2011.)

[7] 刘石, 李合, 王啸吟, 等. 基于语法与语义分析的代码搜索结果优化[J]. 计算机科学, 2009, 32(8): 165 - 168. (LIU S, LI H, WANG X Y, et al. Enhancement of code search results using syntax and semantic analysis[J]. Computer Science, 2009, 32(8): 165 - 168.)

[8] KEIVANLOO I, ROOSTAPOUR L, SCHUGERL P, et al. SECodeSearch: a scalable semantic Web-based source code search infrastructure [C]// ICSM 2010: Proceedings of the 2010 26th IEEE International Conference on Software Maintenance. Piscataway, NJ: IEEE, 2010: 1 - 5.

[9] STOLEE K T, ELBAUM S, DOBOS D. Solving the search for source code[J]. ACM Transactions on Software Engineering and Methodology, 2014, 23(3): 26 - 26.

[10] STOLEE K T, ELBAUM S, DWYER M B, et al. Code search with input/output queries: generalizing, ranking, and assessment [J]. Journal of Systems and Software, 2016, 116(6): 35 - 48.

[11] 满海霞, 梁雅梦. 乔姆斯基层级与自然语言语法——从短语结构语法到非转换语法[J]. 外国语文, 2015, 31(3): 84 - 89. (MAN H X, LIANG Y M. Chomsky hierarchy and natural language grammars: from phrase structure grammar to non-transformational grammar[J]. Foreign Language and Literature, 2015, 31(3): 84 - 89.)

[12] 陈火旺. 程序设计语言编译原理[M]. 北京: 国防工业出版社, 2000. (CHEN H W. Programming Language and Compile Principles[M]. Beijing: National Defense Industry Press, 2000.)

[13] RAJESHWARKAR A, NAGORI M. Optimizing search results using Wikipedia based ESS and enhanced TF-IDF approach[J]. International Journal of Computer Applications, 2016, 144(12): 23 - 28.

[14] FREITAS A, OLIVEIRA J G, ORIAIN S, et al. Querying linked data using semantic relatedness: a vocabulary independent approach [C]// NLDB 2011: Proceedings of the 16th International Conference on Application of Natural Language to Information Systems. Berlin: Springer, 2011: 40 - 51.

This work is partially supported by the Program of Shanghai Subject Chief Scientist (10XD1404400), the Experimental Teaching Reform Project of Tongji University (0800104214).

GU Yisheng, born in 1992, M. S. candidate. His research interests include software engineering, source code search.

ZENG Guosun, born in 1964, Ph. D., professor. His research interests include parallel computing, trusted computing, information security.

(上接第 2925 页)

[17] YU Y H, ZHAO H D, LIU H, et al. Translational motion estimation of moving object based on windowed phase correlation algorithm with kernel regression[C]// CCPR 2010: Proceedings of the 2010 Chinese Conference on Pattern Recognition. Washington, DC: IEEE Computer Society, 2010: 163 - 167.

[18] 余应淮, 谢仕义, 梅其祥. 基于核回归修正的上采样相位相关精确运动估计算法[J]. 计算机应用, 2016, 36(8): 2316 - 2321, 2326. (YU Y H, XIE S Y, MEI Q X. Accurate motion estimation algorithm based on upsampled phase correlation with kernel regression refining[J]. Journal of Computer Applications, 2016, 36(8): 2316 - 2321, 2326.)

This work is partially supported by the Project of Enhancing School with Innovation of Guangdong Ocean University (GDOU2016050222), the Science and Technology Program of Zhanjiang (2015B01009).

YU Yinghuai, born in 1981, M. S., experimentalist. His research interests include image processing, pattern recognition, computer vision.

XIE Shiyi, born in 1963, M. S., professor. His research interests include digital city, ocean remote sensing, image processing.