论文阅读报告-glh

作者：郭礼华

文章：Improving Bug Localization using Structured Information Retrieval

Abstract

key insight: Structured information retrieval based on code constructs, such as class and method names, enables more accurate bug localization.

模型：BLUiR (Bug Localication Using information Retrieval)

输入：source code and bug reports

还使用了bug报告之间的相似度

在 Indri 这个IR工具的基础上构建，实验的数据集有提供

预处理

Bug report

1. Summary
2. Description

其中Summary代表错误报告中的一些关键信息，而描述则代表一下口语化的东西

Source code

1. Class
2. Method
3. Variable
4. Comments

对于每个分割出来的单词，有两种处理方式

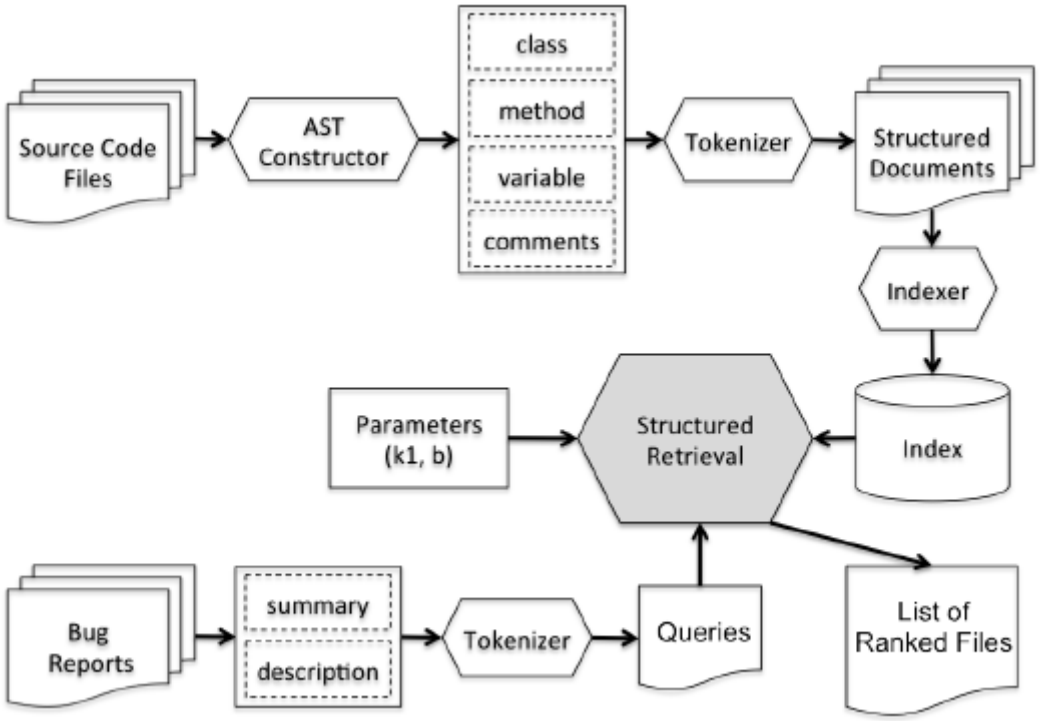
1. 按照原本格式不做处理
2. 按照驼峰命名将每个小单词抽取出来

Term Type	Summary		Description	
	Exact match	Token match	Exact mach	Token match
Class	27 (9.44%)	101 (35.31%)	148 (51.74%)	244 (85.31%)
Method	43 (15.03%)	205 (71.67%)	187 (65.38%)	277 (96.85%)
Variable	107 (37.41%)	125 (43.70%)	230 (80.42%)	252 (88.11%)
Comments	N/A	235 (82.16%)	N/A	278 (97.20%)

实验表明按照驼峰命名将一个长单词分割后（例如ConsoleView拆成Console和View），匹配的数目变多了

Approach

A. BLUiR Architecture



主要流程如下：

1. 将所有源代码解析出AST（利用Eclipse Java Developent Tools）
2. 遍历AST拿到类名、方法名、变量名、注释
3. 将所有identifiers和comments进行预处理（常规的预处理加上按照驼峰进行分割）
4. 将以上的信息存成xml保存起来

B. Source Code Parsing & Term Indexing

两点贡献

1. 通过AST拿到真正的identifier names
2. 将每个单词按照驼峰命名进行分割，分成token

C. Retrieval Model

用了一个改进的tf-idf算法

设 $\vec{v} = (t_1, t_2, ..., t_n)$ 为字母表

idf 改进

$$idf(t_i) = \frac{N+1}{n_t + 0.5}$$

---

tf 改进

设 $\vec{d} = (x_1, x_2, ..., x_n)$ 为源代码的向量，其中 x_i 表示 t_i 在源代码中出现的频数（count），其 tf 改进为如下形式

$$tf_d(x_i) = \frac{k_1 x_i}{x_i + k_1(1 - b + b\frac{l_d}{l_C})}$$

参数及含义

k_1	调整单词出现频数的影响因子
b	取值在(0,1)之间，调整文档长度的影响因子
l_d	Length of document
l_C	Average document length

k_1和b 的取值通过实验得到效果最好的，实验方式为 k_1 从0:2每隔0.1取值， b 从0:1每隔0.1取值，论文中最好效果的为 $k_1 = 1.0, b = 0.3$

设 $\vec{q} = (y_1, y_2, ..., y_n)$ 为错误报告的向量，其中 y_i 表示 t_i 在错误报告中出现的频数（count），其 tf 改进为如下形式

$$tf_q(y_i) = \frac{k_3 y_i}{y_i + k_3}$$

这个其实相当于在上一个公式中，将 b 取为 0

然后相似度计算公式变为

$$s(\vec{d},\vec{q}) = \sum_{i=1}^{n} tf_d(x_i) tf_q(y_i) idf(t_i)^2$$

D. Incorporating Structural Information

将 document 中的单词分成四类，构成四个集合

$$D = \{class, method, variable, comments\}$$

将 query 中的单词分成两类，构成两个集合

$$Q = \{summary, description\}$$

计算相似度的时候变成

$$s'(\vec{d},\vec{q}) = \sum_{r \in Q} \sum_{f \in D} s(d_f, q_r)$$

思想：同一个单词如果属于多个类别中，那就会被重复计算，那这样就相当于获得了一个较高的权重。