A Token Frequency Based Code Clone Detection

He Feng*, Liuqing Li†,
*Department of Physics,
Virginia Tech, Blacksburg, VA, 24061
Email: fenghe@vt.edu
†Department of Computer Science,
Virginia Tech, Blacksburg, VA, 24061
Email: liuqing@vt.edu

***Abstract*—Code clone detection remains an active topic in software engineering. It is not only common in the programming process, but also leading to low maintainability. In this paper, we propose a simple and efficient tool in Java to detect cloned codes, basing on the token frequency in source code. Our tool extracts variables in each block, performs pretreatment on them, as well as counts keywords and symbols. After transforming these pieces of information into vectors, different weights are given to these components, then a similarity algorithm is introduced to find out similar codes. As a counting-based detection tool, it compares lists of names and numbers instead of source code directly, so we can expect less time cost compared to other detection tools. Also components have different weights, thus precision and recall can be maintained to some extent.**

I. Introduction

It is very common that people clone codes in their programs, i.e. reuse some code fragments by copying with or without modification in the coding process. Previous research shows a significant fraction of the code being cloned even in some well-known software systems. The fraction of duplicated code in X Windows System is about 19%[1], while in some core parts of Linux the number is between 15% and 25%[2]. Even if code clone can improve the efficiency of a program in some ways, for example, reducing execution time for no need to call functions, it is quite difficult for developers to maintain a software in its life cycle: if fixing a bug, adding or cutting a feature, it has to be applied to all the similar parts. To solve this problem, clone detection has become an important tool for programmers, since it is much easier if provided all the copied parts when making a change.

A number of techniques and tools have been implemented, using various algorithms to find out code clones in different level. Token-based method shows a good comprehensive performance among all of them. However, each tool has its own limitations, just like the time cost of CCFinder. Then we provoked our research, to develop and implement a scalable clone detection based on tokens to detect code clone in token variation and statement modification.

While many researchers have done similar job before, our tool is new: since most developers would not make dramatic changes in code clone process except for some token modifications including keywords, types, variables and operators, we try to design and improve a simple statistical method to calculate the similarity between code fragments. Specifically, to detect whether fragment A is a clone of fragment B or not, we tokenize each fragment and design a flexible token filter. Any useful tokens, such as types, variables, identifiers, operators, will be counted, so the information is transformed into a token frequency vector. Next is pretreatment of variable names, take v_A and v_B, compare and find out similar variables with small variations in prefix, suffix and substitution. After finding matching variable names, we use a similarity algorithm to measure the two vectors, by putting different weights on different levels of tokens, the accuracy is enhanced, and this method can be improved after actual testing. Finally, a threshold is introduced and optimized, which can also be decided through experiments.

Our tool has its own advantages: the first part of this process is just counting, which has very low time cost; the next part is to find out similar variable names, which is the most time-consuming part in our process, but is still not complicated; the last part is to compare token frequency vectors, which is simple math and pretty straightforward. For the whole process, we can expect low time cost.

For a counting based detection tool with low time cost, the accuracy can still be maintained to some extent, in our algorithm, the weighting part does contribute.

Our solution is new and we believe it's aggressive in clone detection field.

II. Background

To detect cloned codes, the first thing is to know clone types: there are two main kinds of similarity between code fragments. The first kind has textual similarity, and often comes from copying and pasting. Textual similarity can fall into three types[3]:

Type 1: simply includes the variations in whitespace, layout and comments.

Type 2: allows more variations in identifiers, literals and types, in addition to *Type 1*.

Type 3: contains *Type 2* and allows further modifications such as changed, added or removed statements.

An example of *Type 1, 2, 3* clones is shown in Fig 1.

The second main kind is functional similarity that is described by *Type 4* clone[4], or semantic clone, which is more complicated.

Type 4: code fragments perform the same computation or results but are implemented in different ways.

Few tools have been developed to detect *Type 4* clones so far.

```
public static int Num(int X, int Y) {
    int a = 100;
    int b = 120;
    if ( X > Y ) {
        return b + X;
    }
    else {
        a = 80;
        return a + Y;
    }
}
```

```
public static int Num(int X, int Y) {
    int a=100;
    // no blanks          _Type 1
    int c = 120;
    // b to c             _Type 2
    if ( X > Y ) {
        X = X - Y;
        // Add Stmt       _Type 3
        return c + X;
    }
    else {
        a = 80;
        return a + Y;
    }
}
```

Fig. 1. An example of textual clones

Based on the cloned types, researchers have developed different techniques and tools.

Text-based, which means comparing whole lines to each other textually.

Token-based, it is still line-based comparison, but each sequence is summarized by a functor that abstracts the identifiers and literals, this encoding process preserves the structure of the whole sentence.

Metric-based, by gathering different metrics for code fragments, these metric vectors are compared.

Tree-based, usually means the abstract syntax trees(AST), a program is transformed into an abstract syntax tree, then the leaves and subtrees are compared.

Graph-based, program dependency graphs(PDG), control and data flow dependencies of a function can be represented by a program dependency graph, and comparison of subgraphs will be performed.

There are also other techniques but will not be discussed here. Among all the techniques and tools, token-based method performs well in both clone type detection and time cost.

III. Related Work

Many token-based methods have been developed, like CCFinder[5], CMCD[6], Boreas[7], RTF[8].

CCFinder is a classical tool, it first makes a parameter-replacement, like replaces identifiers with a token, and compares each code portion to all other portions. This is very long and detailed comparison, which results in a high accuracy, however that could also costs a huge amount of time.

CMCD uses Count Matrix, it counts many aspects for each variable, and constructs a count vector for that variable, then the count vectors are compared to find out corresponding variables. This method works well in extracting variables those are copied but in different names, but is doesn't include other contributions, like keywords and symbols.

Boreas is based on CMCD, it uses CMCD to compare variables, as well as taking keywords and symbols into account, but still has room for improvement, both in executing time, and detection accuracy.

RTF uses flexible tokenization, but it doesn't use access modifiers like *private, protected, public* and type names *int, short, long, float, double*, so it loses information to some degree.

As for the similarity calculation, different methods such as Cosine similarity, Jaccard similarity, Euclidean distance and Manhattan distance can be applied in specific cases. The Cosine similarity function calculates cosine of the angle between two vectors:

$$CosSim = cos(\alpha) = \frac{v_a \cdot v_b}{||v_a|| \cdot ||v_b||} \tag{1}$$

If v_a and v_b are in the same direction, then $cos(0)$ gives a similarity of 1, otherwise the cosine is between 0 and 1. This function takes care of vector length, but may not fit for the token frequency case.

CMCD gives a very good equation to derive the difference between two count vectors v_a and v_b, which is expressed by the Euclidian Distance between them in space:

$$ED(v_a, v_b) = ||v_a - v_b||_2 = \sqrt{\sum_{i=1}^{n}(v_{1i} - v_{2i})^2} \tag{2}$$

where n is the dimension of the vectors, v_i is the ith component of v.

However the distance of two vectors may not reflect their actual difference because of their lengths, long vectors are more likely to have longer distance, while short vectors distance tend to be shorter, so it leads to some trouble while setting the threshold. A suitable similarity measurement will be designed in our work.

References

[1] Brenda S. Baker, *On Finding Duplication and Near-Duplication in Large Software Systems*, Working Conference on Reverse Engineering(WCRE), pp. 86-95, 1995.

[2] G. Antoniola, U. Villanob, E. Merloc, M. Di Penta, *Analyzing cloning evolution in the Linux kernel*, Special Issue on Source Code Analysis and Manipulation(SCAM), 44(13): 755-765, 2002.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, *Comparison and Evaluation of Clone Detection Tools*, Transactions on Software Engineering, 33(9): 577-591, 2007.

[4] M.Gabel, L.Jiang and Z.Su, *Scalable Detection of Semantic Clones*, Proceedings of the 30th International Conference on Software Engineering(ICSE), pp. 321-330, 2008.

[5] T. Kamiya, S. Kusumoto and K. Inoue, *CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code*, IEEE Transactions on Software Engineering, Vol. 28, No. 7, 2002.

[6] Y. Yuan and Y. Guo, *CMCD: Count Matrix based Code Clone Detection*, Software Engineering Conference(APSEC), pp. 250-257, 2011

[7] Y. Yuan and Y. Guo, *Boreas: An Accurate and Scalable Token-Based Approach to Code Clone Detection*, Automated Software Engineering(ASE), pp. 286-289, 2012

[8] H. A. Basit, S. J. Puglisi, W. F. Smyth, A. Turpin and S. Jarzabel, *Efficient Token Based Clone Detection with Flexible Tokenization*, Proceedings of the 6th European Software Engineering Conference and Foundations of Software Engineering(ESEC/FSE), pp. 513-515, 2007