I. Related Work

A. Types of Detection Tool

Based on the four 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.

Among all the techniques and tools, token-based method performs well in both clone type detection and time cost.

B. Token-based Detection Tool

Many token-based methods have been developed, like CCFinder[**?**], CMCD[**?**], Boreas[**?**], RTF[**?**].

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.

C. Similarity Algorithm

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 v_a and v_b:

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

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 does not fit for the token frequency case. For example, if two vectors have components $< a, 1 >, < b, 2 >$ and $< a, 2 >, < b, 4 >$ respectively, then this algorithm will give a similarity of 1, which is not true.

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

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

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

The similarity is given by:

$$Sim(v_a, v_b) = \frac{1}{1 + ED(v_a, v_b)} \tag{1}$$

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.

Take the example in our implementation:

$$\begin{aligned} \text{List}_x &= < a, 3 >, < b, 3 >, < c, 3 >, < d, 2 > \\ \text{List}_y &= < a, 3 >, < b, 3 >, < c, 3 >, < d, 5 > \\ \text{Sim}(\text{List}_x, \text{List}_y) &= \frac{1}{1+3} = 0.25 \end{aligned} \tag{2}$$

$$\begin{aligned} \text{List}_x &= < a, 3 >, < b, 1 >, < c, 3 >, < d, 1 > \\ \text{List}_y &= < a, 0 >, < b, 5 >, < c, 2 >, < d, 5 > \\ \text{Sim}(\text{List}_x, \text{List}_y) &= \frac{1}{1+6.48} = 0.13 \end{aligned} \tag{3}$$

Intuitatively we'll take the first case as clone, but the actual similarity is only 0.25.

Thus a similarity related to Levenshtein Distance is designed in our work.