**Source code similarity measure using Plagiarism Detection Technique as in JPlag: Summary of Considerations**

**About the technique**

The Plagiarism detection approach for measuring code similarity applies the process of locating instances of plagiarism within a source code. Various algorithms are available for this approach and are classified based on the following:

Ability to match string segments; Ability to convert the program into tokens; Ability to build and compare parse trees; Ability to capture certain program metrics like “number of loops” and “number of variables”; Ability to combine similar features to achieve a single task in a better way (Hybrid capability)

**Algorithm for the technique and its application in JPlag:**

For this review, we consider the Greedy-String-Tiling (GST) algorithm. This algorithm was used in the development of JPlag, a Plagiarism Detection System that is discussed below.

The GST algorithm identifies the longest plagiarized sequence of substrings from the text of the source document and returns the sequence as tiles (i.e., the sequence of substrings) from the source document and the suspicious document.

In JPlag, all programs to be compared are parsed (or scanned, depending on the input language) and converted into token strings. These token strings are compared in pairs for determining the similarity of each pair. During each such comparison, JPlag attempts to cover one token stream with substrings (“tiles”) taken from the other as well as possible. The percentage of the token streams that can be covered is the similarity value. The front-end conversion of compared programs to token strings is the only language-dependent process in JPlag. As a rule, tokens should be chosen such that they characterize the essence of a program (which is difficult to change by a plagiarist) rather than surface aspects. For instance, whitespace and comments should never produce a token, since they are the most obvious points of attack. Some sorts of tokens may or may not be considered useful. For instance, the default token set for Java completely ignores expressions except for assignments and method calls.

When comparing two strings *A* and *B* with GST, the aim is always to find a set of substrings that are the same and satisfy the following rules:

* Any token of *A* may only be matched with exactly one token from *B*. This rule implies that it is impossible to completely match parts of the source text that have been duplicated in a plagiarized program.
* Substrings are to be found independent of their position in the string. This rule implies that reordering parts of the source code is no effective attack.
* Long substring matches are preferred over short ones, because they are more reliable. Short matches are more likely to be spurious.

**Technique’s Benefits and similarity detection Quality**

* It is easy to implement
* Effective at detecting similarity in source codes (though may be time-inefficient if not optimized)