Frequency-based Highlight Extraction in Software Plagiarism Detection

Bachelor's Thesis of

Elisabeth Hermann

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr. Ralf Reussner
Second examiner:	Prof. Dr.-Ing. Anne Koziolek
First advisor:	Robin Maisch, M.Sc.
Second advisor:	Dr.-Ing. Timur Sağlam

21. April 2025 – 21. August 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

Determining whether programming submissions addressing the same task were created independently or copied from one another is challenging. This task can be made easier with the use of plagiarism detection programs. These programs compare the submissions and identify similarities in sections between two submissions. However, to date, they do not take into account whether an identical section appears in more than two submissions. We assume that if a similarity occurs in only a few submissions, the probability of plagiarism is increased, and vice versa. The frequency of matches is counted across all comparisons. We integrate this approach into the token-based plagiarism detector *JPlag* to see how different strategies for detecting and weighting the frequency distribution of matches can be used to better separate plagiarism from inconspicuous matches. The weighting is incorporated into the *Similarity Score*, which assesses the similarity between two submissions. The results show that this approach can improve plagiarism detection.

Zusammenfassung

Festzustellen, ob Programmierabgaben, die dieselbe Aufgabenstellung bearbeiten, eigenständig erstellt oder voneinander abgeschrieben wurden, ist eine Herausforderung. Diese Aufgabe kann durch den Einsatz von Plagiatsdetektorprogrammen erleichtert werden. Diese Programme vergleichen die Abgaben miteinander und erkennen die Ähnlichkeiten von Abschnitten zwischen zwei Abgaben. Bisher wird jedoch nicht berücksichtigt, ob ein gleicher oder übereinstimmender Abschnitt in mehr als zwei Abgaben vorhanden ist. Wir gehen davon aus, dass wenn eine Übereinstimmung in nur wenigen Abgaben vorkommt, die Wahrscheinlichkeit erhöht ist, dass es sich um ein Plagiat handelt und umgekehrt. Die Häufigkeiten von Übereinstimmungen wird über alle paarweise Vergleiche hinweg gezählt.

Wir integrieren diesen Ansatz in den tokenbasierten Plagiatsdetektor *JPlag*, um zu sehen, wie sich verschiedene Strategien zur Erkennung und Gewichtung der Häufigkeitsverteilung von Übereinstimmungen eignen, um Plagiate und unverdächtige Übereinstimmungen besser zu trennen. Das Ergebnis der Gewichtung fließt in den *Similarity Score* ein, der die Ähnlichkeit zwischen zwei Abgaben bewertet. Die Ergebnisse zeigen, dass dieser Ansatz die Plagiatserkennung verbessern kann.

.

Contents

List of Figures

List of Tables

1 Introduction

Today, there are many areas where programming skills are useful. As a result, students, especially those in technical degree programs, need to gain basic programming skills. These are assessed in many courses as coursework in the form of assignments. Learning new programming languages or working on programming tasks can be time-consuming, and can therefore pose a significant challenge for students [1].

In order to support each other, students often work together in study groups and exchange ideas [9]. However, most students are also aware that their assignments are sent through a plagiarism checker at the end. The boundaries between what is still considered collaborative and independent work and what is already considered plagiarism are not always clear to students. Especially when, for example, time pressure increases, students can be increasingly inclined to copy code from other students [12]. Sometimes, students hope to pass their assignments with little effort by using so-called obfuscation attacks, in which the code is changed with the target of not showing up in the plagiarism check [15]. Various tools and AIs can help change code relatively quickly without changing the structure and logic of the program. It is therefore important that plagiarism checkers are able to recognize copied code parts as reliable as possible, and thus offer correctors the best possible support in the correction and fair assessment of programming tasks [12].

Some programming tasks, especially if the tasks are very clearly formulated or not very long, often have only a limited number of obvious solutions. As a result, it can happen that several students choose the same or a very similar solution strategy for parts of the program [11]. Also, information like variable names or function names, may be lost during plagiarism detection. In token-based detection, for example, the syntactic structure of a program is represented as a token list. For this reason, it may happen that students who have not plagiarized from one another will still have sections with the same or very similar token sequences.

In our approach, we consider different approaches to determine the frequency of token sequences across all submissions of a dataset. Since plagiarism detection tools typically only provide values for the similarity between two submissions of a dataset, it is possible that submissions that have been completed independently of each other may also achieve a relatively high value [2]. We consider the frequency of the sequences that match in a pairwise comparison between two submissions (so-called matches). The objective is to differentiate between rare occurrences, which may possess particularly informative structures, and frequent ones, which are more likely to occur by chance. Since matches can be contained within other token sequences, we consider different strategies to take

different partial token sequences into account. The matches are then weighted according to the frequency of their occurrences.

We integrate this approach into the token-based plagiarism detector JPlag to see how different frequency distribution detection strategies and weighting strategies affect plagiarism detection in the submissions of datasets used. This allows us to determine whether the additional information improves plagiarism detection.

2 Foundation

The foundations represent the basic concepts needed to understand the approaches pursued in this bachelor's thesis.

2.1 Software Plagiarism

Because code can often be easily shared or forwarded, either with the intention of copying parts, or helping each other with examples of ideas or solutions when working on tasks together, students, for example, have several ways of obtaining other people's solutions. In addition, different universities, courses, and other contexts may have individual definitions of how they assess the reuse of code and when it is considered plagiarism [7].

In a general formulation of the characteristics that constitute software plagiarism, it is irrelevant with what intention the code was initially made available. This also includes working on a task together if the individual's own performance required to pass the assignment is not guaranteed. In case of software plagiarism, code or parts of code have been reused without stating this in sufficient form. The translation of code into another programming language, reusing code, or obfuscation of code does not constitute a sufficient personal contribution [6, 7].

Obfuscation attacks can occur at either lexical or structural level and can be divided into four categories. These cover obfuscation attacks that target formatting, small additions and meaningful replacements, paraphrasing, inserting dead code or converting it to a logically equivalent structure [7, 10].

The copied code being not an individual solution, independent of the amount of effort that may have been spent to ensure that plagiarism is less likely to be recognized during a check. Sometimes students do not even know that they plagiarized, due to unclear boundaries when working together, or accidentally forget to or do not adequately acknowledge the author [6].

2.2 Token-Based Plagiarism Detection with JPlag

As can be seen in the previous section, there are many different causes for the emergence of software plagiarism. Since plagiarism can be considered an academic offense and lead

to unfair performance assessments on assignments or exams, it is important to detect it reliably.

Given that the number of submissions in courses is generally high, and the effort required for pairwise comparison increases with $\frac{n(n-1)}{2}$ and thus scales to $O(n^2)$, the time required for comparing grows very quickly to a level were comparing the submissions manually is too time consuming [1, 15].

The high effort makes it crucial to detect plagiarism reliable with plagiarism detection tools. However, plagiarism is difficult to detect when sufficient time and effort have been invested in concealing it. The goal is that it takes similar time to modify the copied code sufficiently so that it cannot be easily identified as plagiarism as writing an original contribution and therefore would not have an advantage regarding the work to be spent. Some widespread software plagiarism detection systems rely on determining a similarity score. This score is intended to indicate that plagiarized works are more similar to each other and thus more likely to receive a higher similarity score than submissions where no code was copied. However, similarity alone is not enough to confidently identify which submissions contain plagiarism or where accidental similarities occur, so it is almost impossible to avoid a personal assessment of a smaller portion of submissions based on the results of a plagiarism detection tool. Various approaches exist to calculate such similarity scores. For example, token-based methods and Abstract Syntax Tree (AST)-based methods are widely used [7, 4, 1].

JPlag is a token-based plagiarism detection tool that is often referred to in the literature. Token-based plagiarism detection tools convert the source code into a sequence of tokens using a tokenizer specific to the programming language. Using tokens makes plagiarism checks more robust as details such as variable names, formatting, and comments are ignored. As a result, attacks involving these methods can be detected.

JPlag compares the token sequences of submitted assignments pairwise. The Greedy String Tiling (GST) algorithm iteratively looks for the longest possible common sequences, called matches between two submissions that are not already part of previously found matches. A minimum match length is enforced, since very short token matches, such as a single token, are more likely to occur by chance and are not specific enough to indicate plagiarism.

To improve runtime efficiency, a hash function is used to detect matching sequences more efficiently. JPlag combines the Karp-Rabin matching algorithm with the Greedy String Tiling approach to provide a similarity score for each pairwise comparison. The similarity score is a quantitative measure of similarity, enabling the user to assess the degree of overlap or resemblance between the submissions. The similarity score is represented on a scale from 0% to 100%. This percentage value reflects the extent to which two submission share common sequences. Therefore the length of the matched regions is considered relative to the length of the submissions. A score of 100% indicates perfect similarity, while 0 indicates complete dissimilarity. In plagiarism detection, the similarity score is assigned for each pairwise comparison of two submissions. Is the similarity score is above a certain threshold, it indicates that there might be plagiarism and the similarities between the submissions may need further review [14]. The resulting similarity score is shown in the report viewer,

allowing reviewers to judge whether two submissions are similar enough to raise suspicion of plagiarism or to manually inspect specific submission pairs more closely. JPlag also provides a front-end view that displays the matched segments between two submissions for manual comparison. [8, 14].

3 Related Work

This bachelor's thesis focuses on plagiarism detection and uses a hash function to store and count matches. A similar approach is also used in MOSS [16], a widely used plagiarism detection tool. Their idea is to look at all k-long substrings of a text after removing irrelevant information, such as spaces. These k-grams are hashed, and so-called fingerprints are created on the basis of these hash values, for example, by selecting a subsequence of these hash values. The fingerprints can then be used to identify matches between documents. For creating a fingerprint, MOSS uses a sliding-window approach in which the smallest hash value within a window is selected. This makes the method robust against minor changes to the text, while still maintaining a recognizable structure that can identify significant matches above a certain length without having to store all hash values [16]. In this bachelor's thesis, we also work with hash functions, but with a different focus: instead of comparing pairwise similarities between two submissions, the frequency of identical matches across all submissions in a dataset is examined.

As in the MOSS approach, it is assumed that there is a reasonable minimum length for matches; in this thesis, we use the threshold from the configuration of JPlag. Unlike MOSS, however, fingerprinting is not only used to evaluate similarity; additionally, we consider counting the absolute frequency of individual (partial) matches. While MOSS stores the positions of individual hash values [16], in the thesis approach, this is limited to one ID for each submission, as this is already unique and sufficient for the evaluation.

With TF-IDF-inspired weighting [8], the rarity of individual tokens in submissions is used as an indication of possible plagiarism. There, rarity is not considered across programming languages. The matches have a different influence on plagiarism detection due to the weighting of their individual tokens. The tokens are weighted in relation to the number of submissions and the number of submissions that contain the corresponding token. Matches are indirectly weighted by the tokens they contain [8]. In this bachelor's thesis, weighting by rarity is also used, but only the rarity of the discovered matches is taken into account and weighted, not that of the individual tokens contained. As the weighting is applied to the matches found and not to the tokens, the approach described in this thesis is based on token types and thus language independent.

In the Sherlock N-overlap approach [2], the similarity calculation is also based on tokens. They combine several tokens into a so-called signature, which is a numerical representation of the token, with a fixed size. Those signatures are compared for the plagiarism detection. For the N-overlap they consider in the similarity score based on their frequency [2]. In our approaches for the frequency determination, we also consider a strategy (window of matches strategy) which considers subsequence of a fixed size, and let the found frequencies

influence the similarity score of the submissions. In contrast to their approach we choose our sequences directly on the token sequences and only consider the matches, that where already calculated before. Therefore we do not try to find similarities between the submissions, but try to get more information from the matching areas.

4 Concept

This chapter outlines the approach taken in the bachelor's thesis. The goal is to provide an overview of the main idea. The Chapters 5 and 6 discuss the details of the concept and Chapter 7 then explores the implementation.

Matches are the areas of two submissions that the GST algorithm labels as similar at the token level in at least one submission pair. The same match can occur multiple times across all submissions. More frequent matches can occur, for example, when a sequence of commands is used very often, such as variable definitions before loops. In these cases, they are usually standard solutions for building logical structures. In addition, the task definition for certain sections may be very precisely defined, making it likely that several students choose the same or a very similar implementation for a subsection. Also, if the basecode, the code provided with the task definition, which the GST algorithm is not supposed to take into account, is not ignored by the plagiarism detection tool, it would appear as matches in all comparisons. If these passages are long enough, they appear very frequently as matches between two submissions. Rare matches, on the other hand, may indicate that the algorithm is finding a more unique solution or that the student is using less common structures that still appear in at least one other submission. In this way, a very noticeable match, which might have a unique solution approach and token structure, has little influence on the similarity score if the two submissions otherwise have few matches. Conversely, submissions that use many standard or obvious approaches for each subtask are more likely to be labeled as possible plagiarism, even if there is no plagiarism.

Previous token-based plagiarism detectors mainly compare pairs of submissions to find matches. However, these detectors do not take into account the likelihood that many submissions of a dataset might have the same matches. They also do not consider whether particularly rare and therefore suspicious structures appear in the matches. To address this issue, the idea behind this bachelor's thesis is to determine how frequently matches occur over all comparisons and incorporate these frequencies into the final result. We compare different approaches to calculate the frequencies for the matches. The most intuitive way might be to count how frequent the same match occurs. However, the strategy of the GST algorithm is to find the longest possible matches between two submissions. This strategy leads to cases where the token sequences of matches can have several different subrelationships. For example, shorter matches can be part of a longer match. Since those subrelationships of the token sequences may lead to different behavior, depending on whether we take them into account or not, we will discuss various approaches in Chapter 5.

Since the user of the plagiarism detection tool shall benefit from the detected frequency, we incorporate the result into the calculation of the similarity score of the comparisons. The approach we consider in this bachelor's thesis is to regard rare matches as more essential and frequent matches as less significant. We assume that matches with low frequencies are more likely to indicate plagiarism, while frequent matches are more likely to have occurred by chance. However, since we cannot decide on a case-by-case basis, we cannot rule out the possibility that a frequent match may not result from plagiarism. If a student is plagiarizing a submission, the student would likely also copy the passages where the original author chose an obvious solution. Since we do not know how significant the influence of the frequency of matches is on the probability of plagiarism without looking at actual data, we consider various possibilities to take the frequency into account in the similarity. To this end, we define multiple approaches and desired properties of a weighting function in Chapter 6. With this weighting function for individual matches, we can then use the frequency of a match to influence the similarity score between two submissions. In this way, we want to ensure that submissions in which plagiarism has occurred but was not previously detected, for example, because a student copied only a few distinctive passages, are now detected because we can recognize these passages as rare matches.

For better expandability, we keep the approaches to determine the frequency of a match and the influence of the frequency on the similarity score separate. It shall be possible to combine the various methods for determining the frequency with all methods for influencing the weighting of matches based on their frequency. In this bachelor's thesis, we implement our various approaches into an existing token-based plagiarism detection tool. This implementation allows us to discuss essential implementation details of the methods. It also enables us to assess the practicality of the approaches and evaluate their use on different datasets.

5 Frequency Determination

In this section, we look at different methods for determining the frequency distribution of all matches across all pairwise comparisons.

5.1 Motivation

The data structures with which we operate are the matches found by the GST algorithm. Matches are relevant for us, because they have been identified as possible instances of plagiarism and we want to improve plagiarism detection by highlighting sections that are particularly likely to have been plagiarized. The inconspicuous areas of the code are not relevant for our approach.

Plagiarism detection is based on a similarity score that is displayed to the user. If we want to influence this score based on frequency, we first have to define frequencies for the matches. We look at matches and not entire comparisons, as comparisons can contain multiple matches which can have different frequencies. A situation where this occurs could be caused, for example, by students working together in groups and different sections being similar because they were discussed or plagiarized. This can also result in sections with unusual structures that are used in only a few submissions and are later recognized as rare matches. According to our theory, random matches are more likely to be frequent matches than copied sections. In the following, we will consider matches as token sequences. The set of all matches across all pairwise comparisons represents the set of sequences considered, over which the frequency is determined. Since the GST algorithm can generally only determine the longest possible match sequences, shorter match sequences may be subsequences of longer match sequences. This happens for example, in cases of plagiarism, where one student plagiarizes a large part of the solution and another only a smaller part of the same section. However, since the students plagiarized the same passage in both cases, we want to take this into account in our frequency counts. If we only count the matches of entire sequences, this would not be taken into account. Since there are several possible types of partial relationships between sequences, we explore four different approaches of frequency determination in this bachelor's thesis. We consider different types of subsequences as comparison units in order to compare them against each other.

The frequency distribution over the set of sequences, depends on two factors: the frequency determination approach used and the dataset used. In general, it is possible to apply all frequency determination strategies described in this section to any dataset.

5.2 Methods for Determining the Frequency of Matches

In this bachelor's thesis, we examine various approaches to determine the frequency of matches. The determination of the frequencies across all matches consists of three logical steps. First, we assign a frequency to all different possible comparison units. Second, we set the frequency to the number of occurrences of the comparison unit. Third, we determine a frequency value for each match, by calculating the mean of all comparison units that equal the match sequence or subsequence, depending on the frequency determination strategy. In this approach, we consider the count of matches across all submissions.

The GST algorithm only considers the longest possible matches. We defined the set of all match sequences as set of sequences S. In addition, we also consider subsequences of the sequences in S to improve the quality of detection. We consider four types of subsequences. We discuss each of those subsequence possibilities as an independent strategy to count frequencies in the sequence set S. We consider the following subrelationships:

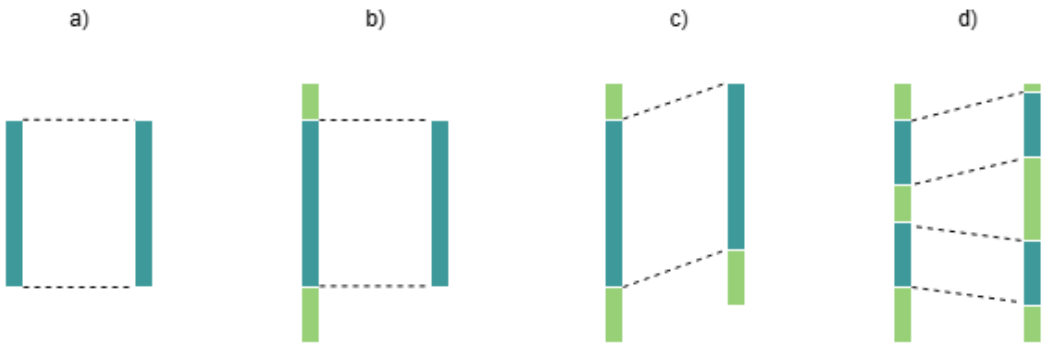


Figure 5.1: Substring Relationships for Determining the Frequency of Matches

a) Two sequences are identical.

b) One sequence is a subsequence of another.

c) A subsequence is identical to part of another sequence.

d) Fixed-size windows of the matches are compared to each other.

Two sequences are identical: In this sequence relationship, we consider the sequence set S as the comparison unit U_c, which contains all sequences that get assigned a frequency during comparison. We do not consider subrelationships. The comparison unit can be defined as:

$$U_c := S\,.$$

In this case, the considered sequences F_c that are compared against the sequences in U_c and count towards increasing the frequency of a match are defined identically:

$$F_c := S\,.$$

One sequence is a subsequence of another: In this strategy, we consider the relationship that a sequence is a subsequence of a larger sequence. Our considered subsequences are all sequences of the sequence set and all subsequences of a sequence that are identical to a whole sequence of the sequence set. Since the subsequence must be in the sequence set S, the comparison units are all elements of the sequence set that are longer than the minimum length considered for the subsequences. We also need to know for each subsequence of a sequence whether it equals a complete sequence in S. Checking this would result in high computational costs.

Therefore, we consider all subsequences, and mark them different in the algorithm depending on whether they equal a complete sequence in S, or not. The evaluation unit considers therefore all subsequences and can be described as:

$$U_p := \{\, x \mid \exists\, y \in S\colon\ x \subseteq y \ \wedge\ |y| \geq |x| \geq m \,\}.$$

where: $m :=$ minimum length and x is a continuous subsequence.

The minimum length is the minimum length for the subsequences that are considered. Therefore, the minimum length should not be smaller than the minimum length of the matches the GST algorithm considers. Since x needs to be an element of the sequence set S, we define the sequence set S as the set of match sequences that meet this condition. We make the minimum length changeable. Sequences shorter than the minimum length are not taken into account when calculating the frequency. This can be done without the risk of overlooking plagiarism that would otherwise be found, since if the frequency detection would not find them, those sequences still have the same influence as without frequency detection. The considered sequences for the frequency of a match are as follows:

$$F_p := \{\, x \in S \mid \exists\, y \in S\colon x \subseteq y \ \wedge\ |y| \geq |x| \geq m \,\}.$$

where: $m =$ minimum length and x is a continuous subsequence.

A subsequence is identical to part of another sequence: In this strategy, we consider subsequences. In contrast to the previous strategy, subsequences are also part of the comparison unit U_s, since they do not have to be in the sequence Set.

$$U_s := \{\, x \mid \exists\, y \in S\colon x \subseteq y \ \wedge\ |y| \geq |x| \geq m \,\}.$$

where: $m =$ minimum length and x is a continuous subsequence.

We can make the same assumptions as in the previous strategy. The only exception is that we cannot require that the minimum length is larger than the minimum length in the GST Algorithm. So the minimum length has to be considered, due to the fact that choosing a smaller size would consider smaller subsequences that have not sufficient significance. Therefore, the considered sequences for the frequency of a match are defined identically to U_s:

$$F_s := \{\, x \mid \exists\, y \in S\colon x \subseteq y \ \wedge\ |y| \geq |x| \geq m \,\}.$$

where: $m =$ minimum length and x is a continuous subsequence.

Fixed-size windows of the matches are compared with each other: In this sequence relationship, we consider all subsequences of the sequence set that have a fixed size. These form the comparison unit U_w. We decide to make the size flexible. But, the minimum size cannot be smaller than the minimum size for a match in the GST algorithm. As with the other strategies, ignoring matches is not a problem if the minimum size has been chosen to be larger than the sequence length. In contrast to the other strategies, the comparison unit does not contain all whole sequences from the sequence set. For continuous subsequences x the comparison unit U_w can be defined as:

$$U_w := \{ x \mid \exists y \in S \colon x \subseteq y \ \wedge \ |x| = m \}.$$

where: m = minimum length and x is a continuous subsequence.

So the considered sequences for the frequency of a match are as follows:

$$F_w := \{ x \mid \exists y \in S \colon x \subseteq y \ \wedge \ |x| = m \}.$$

where: m = minimum length and x is a continuous subsequence.

where x is a continuous subsequence.

Frequency of a Match If the minimum size is chosen small enough, the relationships between the matches found by the strategies is shown in Figure 5.2. Complete matches are contained within the contained matches, which are themselves part of the submatches. The windows of matches overlap only partially with the complete matches and are outside otherwise. The frequencies of the sequences that are taken into account in the frequency analysis.

Since we have defined different comparison criteria in the various strategies, the question arises as to how we determine the frequencies of the matches. To do this, we must map our comparison units back to the matches, i.e. our sequence set. In the following, we will discuss in more detail what the special features of the strategies are and how they can be implemented algorithmically. Since we map our evaluation units U to the frequencies of the sequences in F, a hash map is a suitable data structure here. We can determine the frequencies in two logical steps. First, create the hash map with the evaluation unit U as keys. Second, assign the frequencies to these by counting the set of sequences and subsequences considered for frequency in F.

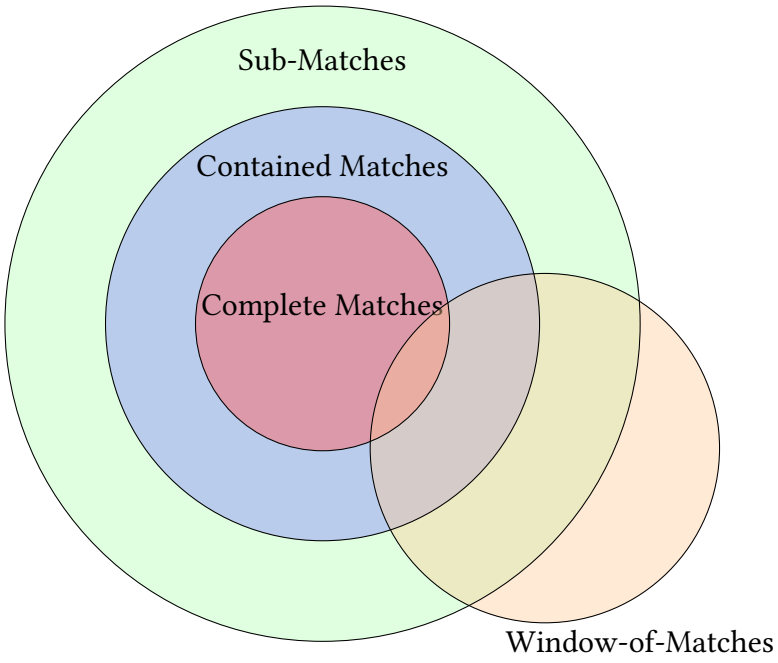


Figure 5.2: Relationship of Matches

5.3 Strategy A - Complete Matches

In the *complete matches strategy*, only complete matches are considered. The matches found in the pairwise comparison are not divided into smaller subsequences. The evaluation unit U are also the sequences whose frequency we count. This strategy is the most intuitive for determining how frequent matches occur. It checks whether the same sections appear in multiple submissions at the token level. However, in order to appear in the frequency distribution, the sequences must be identical, i.e., they must also be the same length. In cases of plagiarism, this can occur when several people have shared the same code section. In cases of non-plagiarism, it could occur when structures are used very commonly.

Because we take the sequence set as a unit and do not consider subsequences, this strategy is particularly efficient. The strategy is less critical in terms of performance and unlike in the other strategies, we do not need a minimum length, as the minimum length is determined by the GST algorithm. Because the set of keys U is identical to the set of sequences we consider F, populating the hash map with keys and incrementing their counts can be done in a single step. As each can be added, its entry in the map is created (if necessary) and its occurrence count is immediately increased by one. We then count exactly how many times each whole match appears across all submissions, any difference of even a single token means the match is considered different and will not be merged. Since the key set U and the match set F coincide, the weighting procedure reduces to retrieving the corresponding map entry, and the frequency can then be directly assigned to the match.

Algorithm 1 Complete Matches Strategy

```
1: comparisons ← pairwise comparisons of submissions
2: for all comparison in comparisons do
3:     for all match in comparison do
4:         map.put(matchTokenSequence, map.getOrDefault(matchTokenSequence, 0) + 1)
5:     end for
6: end for
```

5.4 Strategy B - Contained Matches

The *contained matches strategy*, unlike the complete matches strategy, also accounts for the fact that shorter matches may be fully contained within longer ones. Since the GST algorithm always seeks only the longest possible match, these shorter matches are ignored by default. By treating entire matches as sequences, we now also check whether any complete sequence is a contiguous subsequence of another. Since we need to know which of the subsequences equal a sequence in the sequence set, we initialize these values with an impossible frequency value, to mark them as not occurred. We regard as subsequences all coherent token subsequences whose length exceeds the minimum length. This ensures that we count only segments relevant to the similarity measure between two submissions. Since we have potentially more overhead than the previous strategy, the minimum length can be increased, to reduce the effort and only partly apply this strategy, for example in large datasets. The threshold for the minimum length may, for instance, be set to the same minimum match length used by the GST algorithm, since this defines when a similarity is considered as significant rather than merely coincidental. Then, all full matches are included and every sequence considered by the complete matches strategy will also be taken into account. If the value is set to a higher value, shorter matches will not be considered and the strategy will only consider a subset of the matches. Algorithm 2 first generates all contiguous subsequences of the token sequence, including the original sequence itself, only if its length meets or exceeds the minimum length. This collection of sequences constitutes the key set for the strategy. Next, all keys are added to the hash map. They will be initialized with zero, since all sequences that will be part of the sequence set will be found at least once because we count all whole sequences. However, since we only wish to count the occurrences of entire matches, we increment the hash count only for the token sequence of the original sequence. As a result, some keys may appear in the map with a frequency of zero, reflecting that they do not correspond to any full-match occurrences. We added them additionally, to avoid iterating twice over all sequences. This is no problem since we can later tell them apart as subsequences that are not contained in the sequence set.

Algorithm 2 Contained Matches Strategy

```
1:  comparisons ← pairwise comparisons of submissions
2:  for all comparison in comparisons do
3:      for all match in comparison do
4:          subsequences ← getAllContinuousSubsequencesLongerThanMinLength()
5:          for all subsequence in subsequences do
6:              map.putIfAbsent(subsequence, 0)
7:          end for
8:          if length(matchTokenSequence) ≥ minimumLength then
9:              map.put(matchTokenSequence, map.get(matchTokenSequence) + 1)
10:         end if
11:     end for
12: end for
```

The frequency value for each match is calculated as the average of the frequencies of the full sequence of the match and its strategy-specific subsequences, considering only those with a frequency greater than zero. In other words, we include every match and every subsequence that appears as at least one full match. Subsequences with zero occurrences are excluded, since counting them would wrongly penalize longer sequences, whose own frequency may be high, by classifying them as rare merely because they contain no direct matches.

5.5 Strategy C - Sub-Matches

The sub-matches strategy treats each coherent subsequence of a match as a sequence. Both the sequences from the sequence set and their sub-sequences are taken into account. If the minimum length is set greater than the minimum length of the GST algorithm, sequences and sub-sequences that are shorter are not taken into account. The set of evaluation units corresponds to U. Since we want to consider the frequencies of all subsequences, the set U is equal to the set F.

This strategy is improving detection because it considers all possible matches and not just the longest ones, as the GST algorithm does. Thus, all possible matches are compared to see how often they actually occur in all comparisons.

Algorithm 3 Sub-Matches Strategy

```
1: comparisons ← pairwise comparisons of submissions
2: for all comparison in comparisons do
3:     for all match in comparison do
4:         subsequences ← getAllContinuousSubsequencesLongerThanMinLength()
5:         for all subsequence in subsequences do
6:             if length(subsequence) ≥ minimumLength then
7:                 map.put(subsequence, map.getOrDefault(subsequence, 1) + 1)
8:             end if
9:         end for
10:    end for
11: end for
```

This strategy has the same number of evaluation units as the contained matches strategy. In contrast to the contained match strategy, however, all submatches are taken into account in the frequency detection, so we do not have to initialize the entries beforehand.

When computing a match's frequency under this strategy, we take the average of the frequencies of the full match and all of its subsequences. Since each of these sequences is guaranteed to occur at least once, each subset contributes meaningfully to the final frequency value of the match.

5.6 Strategy D - Window-of-Matches

The *window-of-matches strategy* is based on finding all subsequences of a fixed window size inside all match sequences. All subsequences of the specified length are taken into account. In this case, full matches are no longer considered unless they exactly match the defined window size.

This approach has the advantage that subsequences occurring within other sequences are still detected as long as they meet the window size. In this strategy, cases can occur in which the frequencies of the window sequences of one match are very different. For example, when all window sequences have a low frequency and only one has a higher frequency, the influence of the frequent window sequence may not be very high, especially for long matches. But this effect is significantly lower in contrast to the same effect in the sub-matches strategy. Since we consider fewer subsequences of a match, the impact of a single window sequence is higher in the similarity score of a match.

The match frequency is again calculated using the average of the individual window sequences frequency. If all matches are of equal length and this length matches the window size, the strategy effectively behaves like the complete matches strategy.

Algorithm 4 Window-of-Matches Strategy

```
1: comparisons ← pairwise comparisons of submissions
2: for all comparison in comparisons do
3:     for all match in comparison do
4:         windowSequences ← createAllContinuousSubsequencesOfWindowSize()
5:         for all windowSequence in windowSequences do
6:             map.put(windowSequence, map.getOrDefault(windowSequence, 1) + 1)
7:         end for
8:     end for
9: end for
```

As we can see the algorithm 4 is very similar to algorithm 3 of the sub-matches strategy. The main difference are the considered sequences and subsequences. Since all windows have the same length in the window of matches strategy, we do not need to check the length in the algorithm. Since the sequences and subsequences from the sequence set that we want to consider match the set of sequences used for the frequency calculation, U is equal to F. This is something that all strategies, except the contained strategy, have in common.

6 Frequency-Based Similarity

After assigning frequency values to the matches in the previous chapter, we now use these values to influence the similarity score of a comparison.

6.1 Motivation

Our approach assumes that rare matches are more likely to indicate plagiarism than frequent ones. Therefore, these should have a higher influence on the similarity score between two submissions. Frequent matches, resulting, for example, from the use of obvious code solutions should have less influence. Students may often choose such structures because the assignment descriptions may provide relatively clear guidelines or instructions for these parts of the submissions. In contrast, rare matches indicate submissions that contain a more unique solution, which nevertheless also appear in a few or at least one other submission. We assume that such less common solutions, when they are similar enough to be detected as matches, are very likely to indicate possible plagiarism in that part of the submission. However, since individual parts of a submission may contain plagiarism while others do not, we decide for each match whether it should be weighted more strongly as a potential indicator of plagiarism or not. We also consider an approach where frequent matches are taken less into account. This idea is based on the assumption that, although these structures appear very similar and could be detected as potential plagiarism, they have little evidential value because they are commonly used. Therefore, we assume in this approach that most of these matches occur coincidentally rather than through plagiarism.

Since the previous step associates the matches with a frequency value, we must no longer access the hash map that was used to determine the frequency. As the frequency values now apply to each match and the partial match frequencies will no longer be considered individually, this part is strategy independent. Accordingly, any chosen method of frequency determination can be combined with any chosen method of incorporating these frequencies into the similarity score. Because we use a token-based plagiarism detection approach, similarity is influenced by the number of matched tokens. In the general case, similarity can be calculated by the number of matched tokens in the submissions relative to the total number of tokens in the submission. In our approach, the number of matched tokens will be artificially increased or decreased, depending on the chosen weighting strategy. Artificially increasing the number of matched tokens leads to a higher similarity value, as a greater percentage of the submissions in the comparison appears to be identical. For each comparison, the algorithm calculates a similarity value. The examiners then review

this value to decide which submissions may be considered plagiarism. The similarity value cannot definitively determine whether plagiarism has occurred, but a higher value gives a first indication to take a closer look. The final decision must be made in each case by plagiarism examiners through a detailed comparison of the submissions. Nevertheless, we assume that the weighting approaches will cause plagiarized submissions to receive higher similarity values. Depending on the type of weighting function, frequent matches, which are assumed to contain the majority of the false-positives, can be weighted less, reducing the need for their separate consideration during the review.

To influence the similarity, we first calculate the relative frequency and normalize it to a range of 0 to 1. Where 0 corresponds to the highest frequency and 1 to the lowest frequency found. The relative frequencies are then mapped to a selected value range using a weighting function. At the lower bound (0), matches will keep their original length (number of tokens) after weighting. At the upper bound (1), matches will be extended up to twice their length. We choose this range because a match of moderate length, when doubled, can quickly account for a large portion of a submission. With a doubling of length, the influence of rare matches is assumed to already become substantial. The lower bound of 0, using the original length, ensures that frequent matches remain close to their unweighted length or unchanged, depending on the weighting strategy.

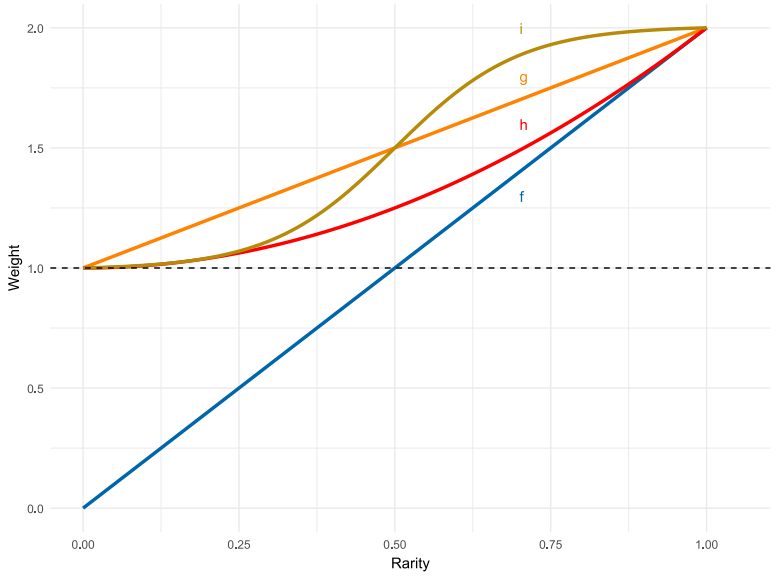
However, the aim of the weighting strategies considered is to make it easier to identify submissions with rare matches that are particularly significant but did not previously fall within the scope of suspicion. In the case of a proportional weighting strategy, frequent matches are less weighted by artificially reducing their length. A minimum length slightly above zero is maintained, ensuring that frequent matches are not entirely excluded. In general, similarity values above 100% are not possible. Therefore, they are limited to 100%. Logically, two submissions can not be more plagiarized than a complete copy of the entire submission.

Since the weighting of individual matches can have different effects on different datasets, and since the most effective weighting of rare matches cannot be determined without analyzing the actual data distribution, we will introduce a weighting factor. This factor determines how strongly the weighting is applied. It can take values between zero and one and can be specified during plagiarism evaluation. With a weighting factor of zero, the weighting strategy is not applied and the original similarity (without frequencies) is used. With a factor of one, the weighting is fully applied over the entire value range, up to doubling the match weighting. Intermediate values apply the weighting linearly and proportionally.

6.2 Methods for Weighting the Matches Based on Their Frequency

Once the statistics on match frequencies across the given submissions have been generated, we use them to influence the similarity score between the submissions. For this, the

frequencies are used as a base to weight the matches. There are various ways in which the frequencies can influence the similarity between two submissions. In this bachelor's thesis, we focus on four approaches. The approaches integrate the frequency data directly into a similarity score. Therefore, we first determine a relative frequency of all frequencies that occur across all matches. The relative frequency is then normalized to a range between 0 and 1. Then the relative frequencies can be compared to each other. We normalize these relative frequencies so that we can apply a weighting function. In this context, we refer to the normalized relative frequency as rarity to distinguish it from the absolute frequency. These rarities can then be weighted with different weighting functions to increase or decrease their influence on the similarity calculation:



blue proportinal weighting function, orange linear weight function,
red quadratic weighting function, yellow a sigmoid function.

Figure 6.1: Overview of the Different Weighting Functions Used

f) proportional weighting with $2 * rarity$, between $[0.01, 2]$, blue graph.

g) linear weighting with $1 + rarity$, between $[1, 2]$, orange graph.

h) quadratic weighting with $1 + rarity^2$, between $[1, 2]$, red graph.

i) sigmomid function with $1 + \frac{\frac{1}{1+e^{-10(rarity-0.5)}} - \frac{1}{1+e^{-10(0-0.5)}}}{\frac{1}{1+e^{-10(1-0.5)}} - \frac{1}{1+e^{-10(0-0.5)}}}$, yellow graph.

In these approaches, rare matches are given greater weight, which means that the similarity score increases when rare matches are found. This reflects the idea that rare matches may indicate plagiarism. Since, as previously mentioned, frequent matches are likely to be less indicative of plagiarism and more likely to occur due to the lack of common alternatives,

in proportional weighting we consider reducing their influence on the similarity score. In this case, rare matches will still be weighted more heavily, while frequent ones will contribute less. The other approaches only consider giving more weight to rare matches, to avoid reducing the similarity of comparisons that may contain plagiarism. Not decreasing the similarity should be considered since having a frequent match does not decrease the likelihood of plagiarism in that match.

6.3 Proportional Weighting

In proportional weighting, rare matches are weighted more strongly, while frequent matches are weighted less strongly. This allows rare matches to be more clearly distinguished from frequent ones. Even in cases where we find basecode or less meaningful structures in the frequency determination, we can address these with a lower weight. However, this approach may also lead to situations where the user does not notice, for example, if the basecode has been overlooked. Instead of many comparisons resulting in a very high similarity, these matches are weighted down. There is also the possibility that submissions with a large number of matches, which occur more frequently, may no longer be recognized as possible plagiarism. At the same time, it cannot be ruled out that matches that occur more frequently may also originate from plagiarism. In the implementation, frequent matches are weighted almost not at all, while rare matches are weighted up to twice as much. The matches are weighted using a linear weighting function:

$$weight = 2 * rarity$$

since the rarity is between 0 and 1.

The Figure 6.1 shows that, compared to the other weighting strategies, we obtain relatively small weights, since the curve stays below the others. Therefore, the focus of this weighting approach is to achieve a separation by stretching and compressing the length of a match. This has the advantage that potential false-positive matches are not, or only minimally, taken into account.

6.4 Linear Weighting

When increasing only the impact of matches, the value range of the weighting function is reduced to $[1, 2]$. The lower bound of 1 is used because the matches retain their original length as the minimum length. Therefore the similarity value of a comparison can only get higher than in the original evaluation. This is to ensure that potential plagiarism cannot suddenly go undetected. As a result, the weighting function also assigns higher weights to frequent and medium-frequency matches. Since the normalized relative frequency already lies within an interval between zero and one, we can obtain the weight function by just adding 1 to the rarity. Thus, we obtain:

$$weight = 1 + rarity$$

Nevertheless, frequent matches, which may be recognized as false-positives, are also weighted more strongly, which is not optimal, as these could otherwise be distinguished from the others using the frequency distribution. On the other hand, rare matches are weighted more strongly than before.

6.5 Quadratic Weighting

To reduce the weighting of frequent matches, we also consider a quadratic weighting function. This can be described as:

$$weight = 1 + rarity^2$$

Where the rarity again represents the normalized relative frequency of the match. In this case, frequent matches, which are, in principle, more likely to be false-positives or less meaningful, are weighted almost the same as before. However, medium-frequency matches are also weighted much less, which can lead to a situation where only a relatively small proportion of rare matches has a noticeable impact on increasing the similarity score of a comparison. Furthermore, the weighting function lies significantly below the linear weighting strategy for rare matches. This can be a disadvantage, as it may be desirable for matches that genuinely stand out due to their rarity to have a correspondingly strong influence.

6.6 Sigmoid Weighting

Because the quadratic rare-match weighting assigns lower weights to frequent matches, it also reduces the weight of rare matches compared to a linear weighting. However, we want to assign stronger weights to rare matches in order to achieve a more precise distinction. Therefore, we need a function that lies above the linear weighting function for rare matches. At the same time, we want to assign as little weight as possible to frequent matches, which are more likely to include false-positives and less informative results. To achieve this, the function should lie below the linear weighting function up to a certain point. A function that fulfills both requirements is the sigmoid function.

To obtain a sigmoid function that maps the domain $[0, 1]$ to the value range $[1, 2]$, the sigmoid function is transformed as follows: The original sigmoid function operates on a value range $[0, 1]$ with an inflection point at $(0, 0.5)$.

$$f(x) = \frac{1}{1 + e^{-x}}$$

To place the function symmetrically within our domain, we adjust it so that the inflection point moves to $(0.5, 1.5)$. To do this, we first shift the inflection point along the x-axis to $(0.5, 0.5)$.

$$g(x) = f(x - 0.5) = \frac{1}{1 + e^{-(x-0.5)}}$$

Since the sigmoid approaches 0 as $x \to -\infty$ and 1 as $x \to \infty$, the function will not pass precisely through the minimum $(0, 1)$ and maximum $(1, 2)$ points. However, we want the function to run very close to those points while staying within the range $[1, 2]$. To achieve this, we compress the function along the x-axis by a factor of 10.

$$h(x) = f\big(10(x - 0.5)\big) = \frac{1}{1 + e^{-10(x-0.5)}}$$

To ensure that the function later passes through the maximum point $(1, 2)$, we scale it by its function value at $x = 1$. This moves the function to the point $(1, 1)$. As a result, rare matches are weighted up to twice as much as in the other strategies. Because this makes the function no longer symmetric, common matches still receive a small weight, even if the base weight is 0. This distinguishes this strategy from the previous ones, where a weight of 0 always means that the weighting strategy was not applied.

$$i(x) = \frac{h(x)}{h(1)} = \frac{f\big(10(x - 0.5)\big)}{f\big(10(1 - 0.5)\big)} = \frac{\frac{1}{1+e^{-10(x-0.5)}}}{\frac{1}{1+e^{-5}}}$$

Since the function is still not in the correct value range, it is now shifted along the y-axis. This also allows us to scale it to the desired value range $[1, a]$. However, this factor drops out here, since we, like in the other strategies, have chosen the range $[1, 2]$.

$$j(x) = (a - 1) \cdot i(x) + 1 = (a - 1) \cdot \frac{h(x)}{h(1)} + 1 = (2 - 1) \cdot \frac{\frac{1}{1+e^{-10(x-0.5)}}}{\frac{1}{1+e^{-10(1-0.5)}}} + 1$$

Adjusted in this way, the sigmoid function is within the target range and fulfills both desired properties. Since we want the function to pass exactly through the point (0,1), not only to approach zero there, we normalize it by its value at $x = 0$.

$$k(x) = 1 + \frac{\frac{1}{1+e^{-10(x-0.5)}} - \frac{1}{1+e^{-10(0-0.5)}}}{\frac{1}{1+e^{-10(1-0.5)}} - \frac{1}{1+e^{-10(0-0.5)}}} \cdot (2 - 1)$$

Thus, we obtain $k(x)$ as the result, where x represents the normalized rarity in our case.

7 Implementation

This bachelor's thesis implements the various strategies presented to determine the frequencies of matches and influence the similarity score in JPlag. By integrating these strategies into JPlag, on one hand we demonstrate, that the concepts can be meaningfully incorporated into a token-based plagiarism detection tool. On the other hand, the proposed approaches can be evaluated through their concrete implementation in the evaluation, using datasets, and compared with each other.

Since we implement the strategies in JPlag, the exact implementation depends on the structure of JPlag and therefore would have to be adapted for other token-based plagiarism detection tools. The implementation is based on token types and is therefore largely independent of the programming language used in the submission. We chose JPlag for this bachelor's thesis because it is a widely used and well-established plagiarism detection tool. This choice allows us to obtain results that are as practice-oriented as possible. Since the part of JPlag where we integrate our implementation is written in Java, we use Java as the programming language for the implementation.

The implementation is structured in three steps. In the first step, the frequency distribution of the matches is created based on the selected strategy. The user has the option of specifying the strategy to be used as an input parameter, or JPlag utilizes a default strategy. In the second step, the frequency values are calculated and entered into the matches in order to facilitate the separation of the frequency determination strategy from the weighting strategy. In the third step, the weighting strategy is implemented, which can be either explicitly specified by an input parameter or JPlag utilizes a default. The following input parameters can specified: the strategy for determining frequencies, a minimum size of the subsequences to be considered, the weighting strategy, and a factor that determines how strongly the weighting strategy should influence the result. The minimum size of the subsequences considered in the matches can be chosen as an integer, while the weighting factor must be between 0 and 1.

We implement the strategies to record the frequency of the sequences using hash map-based counting. The exact definition of a sequence is dependent on the respective strategy: it can either be a complete match or a contiguous subsequence of it. The implementation follows the strategy design pattern, and each strategy is applied to all matches that were determined in the pairwise comparison of the submissions. In order to avoid passing the hash map reference in every call, we use two consumers. One only adds the keys to the map (without incrementing their count), while the other adds the given sequence and increases its frequency by one. As the key of the hash map, the sequence of token types is used. Using the token type is important because the token type is standardized across different

programming languages, ensuring consistency compared to using language-dependent tokens. The strategies can be iteratively invoked for all matches of all comparisons and build the frequency map.

When writing the frequencies for the matches, strategy-dependent approaches are required. Therefore, each strategy provides a method that determines the relevant keys and frequencies for each match and then writes the frequency into the match. On the one hand, it is useful for extensibility and clarity to link these methods with the strategies in the strategy pattern, since it is better expandable. Since we no longer need to access the hash map, and the frequency is recorded in the matches, we can separate the methods used for the third step from the previous ones. Since we have different weighting functions, we again use a strategy pattern to ease later extensions. The calculated and influenced similarity score is then finally handed over to the regular JPlag routine.

8 Evaluation

The evaluation tests the different approaches presented in in this thesis, as well as various combinations of them. The evaluation is done by applying the strategies to pre-defined datasets.

8.1 Methodology

We consider three datasets in total. Although the implementation of our strategies is language-independent, JPlag requires all submissions in a dataset to be written in the same programming language. For this reason, we have selected Java submissions for our evaluation, to ensure consistent, fair comparisons across all datasets. The focus of our evaluation is defined by a Goal-Question-Metric (GQM)[3] plan.

8.2 Datasets

We use three different datasets for the evaluation. Two of these come from the PROGpedia[11] collection: PROGpedia-19 and PROGpedia-56. The PROGpedia dataset[13] consists of a collection of submissions from introductory programming courses, including the corresponding tasks and student solutions. Each task may include multiple submissions from each participant in the course. Since multiple programming languages were available for the submissions inside the datasets, we restricted our evaluation to the Java submissions. This ensures comparability, as specific structures in the matches analyzed might otherwise occur with varying frequencies depending on the programming language used.

The PROGpedia-19 dataset also contains basecode that can be specified and filtered out in JPlag. In both PROGpedia datasets, we filter the non-final submissions so that only the submission with the highest version number is kept for each participant. This step is crucial: otherwise, we would risk including multiple highly similar submissions from the same participant. Since non-final submissions are typically incomplete but mostly identical to their successors, JPlag would probably detect numerous matches between versions of the same submission. In a realistic evaluation scenario, only the final versions of the contributions would be considered as a relevant input.

As a third dataset, we use submissions from a first-semester programming exercise sheet 3 task A, at KIT. Students were tasked to program a TicTacToe game. Their Java submissions are anonymized and represent a realistic application scenario.

For the calculation of frequency distributions, it is not initially necessary to distinguish between plagiarized and non-plagiarized submissions, as we aim to obtain an overall picture of the submissions. However, to analyze how the similarity score is influenced and how the different categories behave in comparison, we require labeled datasets. Since none of the datasets had with pre-existing plagiarism labels, two experienced plagiarism examiners categorized the submissions. For each dataset, threshold values were defined, determining at which similarity score a submission falls into one of three categories: *plagiarized*, *suspicious*, or *inconspicuous*. We will use the labels to analyze the effect of our results based on the different groups. It is important to note that not all individual cases are evaluated in detail. Therefore, it is possible that some submissions do not fully correspond to the category they were assigned.

The thresholds for categorization were defined as follows:

Dataset	Plagiarism Threshold	Suspicious Threshold	Inconspicuous
PROGpedia-19	0.74	0.60	< 0.60
PROGpedia-56	0.76	0.60	< 0.60
TicTacToe	0.71	0.51	< 0.51

Table 8.1: Threshold Values for Categorizing Submissions Across Datasets.

After defining these categories, we applied them to the datasets. The following table provides an overview of the distribution of comparisons among the categories once the criteria were applied:

Dataset / Category	Plagiarized	Suspicious	Inconspicuous	Total Submissions
PROGpedia-19	74	3	1354	1431
PROGpedia-56	92	26	2228	2369
TicTacToe	51	27	211497	211575

Table 8.2: Distribution of Plagiarism Categories Across the Datasets.

8.3 GQM Plan

The GQM Plan is intended to provide an overview of how the individual objectives are evaluated and assessed in the bachelor's thesis.

GQM Plan

G.1 Analyze the frequency and variety of rare matches for the purpose of evaluating their influence on token-based plagiarism detection with respect to their distribution across submissions.

Q.1.1 How widespread is the frequency distribution of rare matches?

M1.1.1 Consider a histogram, the standard deviation, and entropy of match frequencies.

These key figures provide an overview of how uneven matches are distributed and whether there are anomalies or clusters.

Q.1.2 Which influence does the inclusion of partial rare matches have on the distribution?

M.1.2.1 Compare the distributions using the metrics mean and median.

This evaluates whether the inclusion of partial matches results in significant differences in the match distribution. The approaches described in the concept section are compared with each other.

G.2 Analyze the impact of rare matches for the purpose of improving similarity measurement in JPlag with respect to increasing accuracy in distinguishing plagiarism from coincidence.

Q.2.1 Can the rarity of the matches be meaningfully integrated into the similarity score calculated by JPlag?

M.2.1.1 Difference in similarity scores between plagiarized and non-plagiarized pairs.

The aim is to see whether rare matches are systematically more strongly represented in plagiarism. And whether frequent matches are systematically not stronger represented in plagiarism pairs.

Q.2.2 How does the significance of the JPlag output differ depending on which integration approach is used?

M.2.2.1 Compare similarity score differences between plagiarized and non-plagiarized pairs for each integration method.

M.2.2.2 Compare similarity score differences based on the match frequency strategy used.

This examines the influence of the strategies used and compares how they perform against each other.

8.4 Frequency Determination

In this section, we examine the frequency distributions across the three datasets. This step allows us to identify notable differences between them, gain a clear impression of their overall characteristics, and assess whether the datasets contain a sufficient number of rare sequences to make later adjustments and weighting strategies meaningfully. Understanding the distribution of frequent, rare, and medium frequency sequences is essential, as it provides the foundation for interpreting the effects of our strategies.

Following this initial exploration, we shift our focus to a systematic comparison of the strategies themselves. The aim here is to determine whether individual approaches exhibit specific weaknesses or limitations that we can already observe at an early stage. Identifying these patterns is important, since it helps us evaluate which strategies are robust across different datasets and which may need further refinement.

For the evaluation, we deliberately set the minimum subsequence length parameter to its lowest meaningful value, i.e., the minimum match size of the GST algorithm. Choosing an even smaller threshold would not be reasonable, as matches that are too short are often coincidental and provide little reliable information, which is precisely why JPlag itself uses a minimum match size. Larger minimum values may later offer opportunities for fine-tuning, but starting with the smallest value ensures that we capture the highest possible impact of the strategies. At this threshold, the maximum number of substrings from the matches is taken into account, so that no information is lost and the contrasts between the strategies appear most clearly.

It should be taken into consideration that strategies remain functional even when no particularly rare or very frequent matches are detected. However, for the purpose of evaluation, we are especially interested in the cases where all three categories are represented: rare, frequent, and medium-frequency subsequences. Having this balance is crucial because it allows us to observe how weighting functions shift the similarity values of each category. By examining the changes, we can assess not only the overall impact on similarity scores.

8.4.1 Strategy A - Complete Matches

In the first step of our analysis, we focus on the complete matches strategy, in which only entire matches are considered. This approach is the most straightforward and intuitively understandable, as it remains closest to the original match definition used by JPlag. By counting only complete matches, we avoid introducing additional complexity through subsequence extraction while still obtaining meaningful insights.

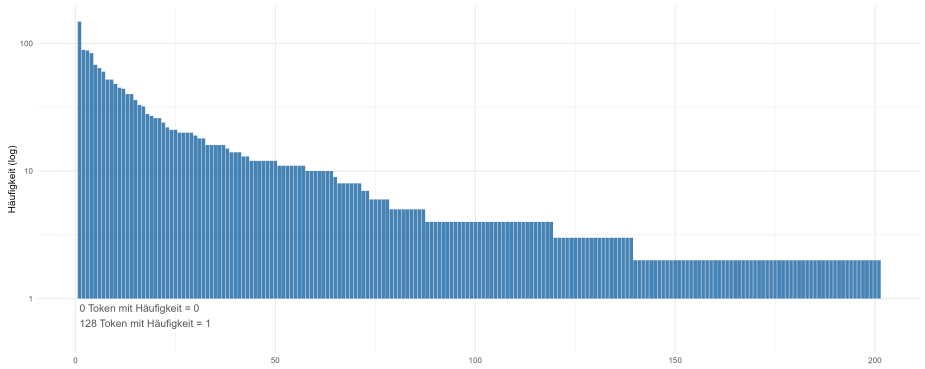


Figure 8.1: Complete Matches Strategy - PROGpedia-19 Frequency Distribution

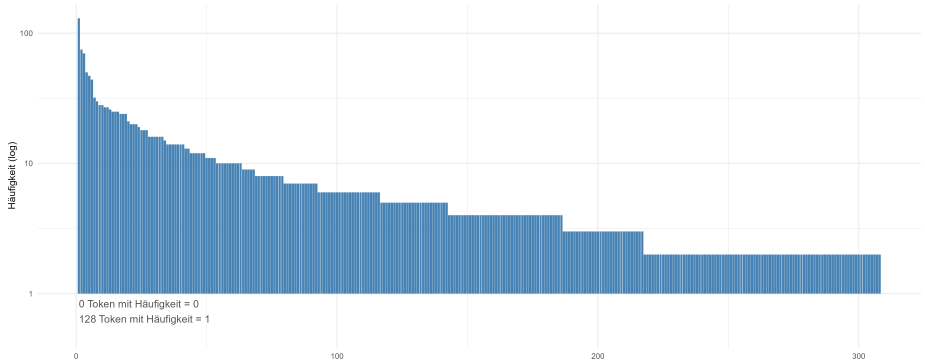


Figure 8.2: Complete Matches Strategy - PROGpedia-56 Frequency Distribution

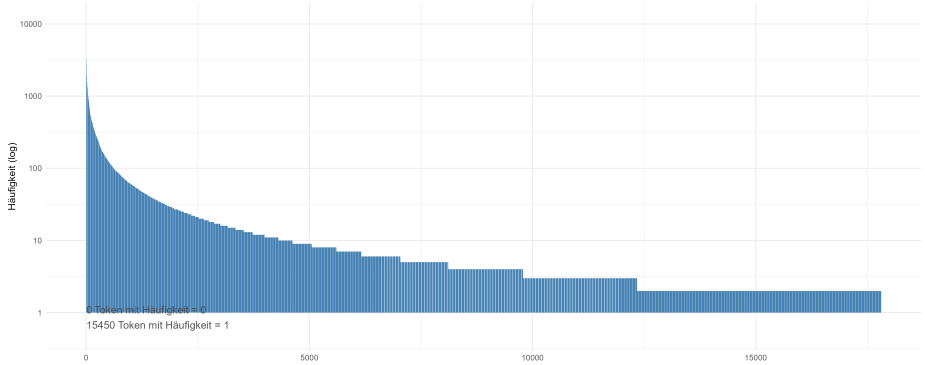


Figure 8.3: Complete Matches Strategy - TicTacToe Frequency Distribution

results: In the frequency distribution of the three datasets, we observe that there are matches with different frequencies in all three datasets. There are various matches with a frequency of one, which means that they are only found once in all comparisons. In addition, we find many matches that only occur three times. In all datasets, for the lower frequencies, occurrences that happen three times or less are over 50% of the matches. If we look at the other side of the distribution, we also find some frequent matches. In the PROGpedia datasets, we can find a match that has been found over 100 times. The frequency distribution of the TicTacToe dataset shows that the most frequent match has been found over 3,000 times. In general, we see a tendency that overall more matches are found that share the same low frequency, than matches that have a high frequency. In the PROGpedia datasets, the most frequent match is found a lot more often than the second frequent one. Although more matches with low frequencies are found, the frequency distributions do not have larger gaps where no match with a frequency in between has been found. But there are noticeably smaller gaps, which lead to gradations at the transitions between the individual probabilities of sequences, especially in rare matches. When looking at the larger TicTacToe dataset, the distribution of complete matches appears relatively well balanced, since highly frequent matches appear in various frequencies. We find examples of very frequent matches, moderately frequent ones, and truly rare ones.

Discussion. The distribution of complete matches across the three datasets already reveals several noteworthy patterns. On the one hand, we observe that this strategy identifies several rare matches, which means that matches occur only once across all comparisons. This finding is valuable, as it suggests that we detect matches that stand out from the rest of the data. These matches are those which we consider more likely to contain plagiarism, and therefore should be taken more strongly into account. Since we also need frequent matches to be able to recognize those matches as noticeable, it is helpful that more than half of the matches are found more than once. On the other hand, it is also very interesting to see that we found highly frequent matches. This supports the assumption that some matches are found often enough that we can assume that multiple, if not the majority, have been created by chance. Therefore, we consider these matches as less likely to indicate plagiarism.

The observed step pattern can mainly be attributed to the fact that if, for example, two people have copied the same passage from one person, the two instances of plagiarism are also recognized as plagiarism among themselves. This explains why most frequencies correspond to a number of $frequency = n(n-1)/2$, where n is the number of submissions with the same matching sequence. Values in between can only occur if a match appears multiple times in the same comparison.

Overall, no significant differences between the datasets can be observed in the complete matches strategy, apart from the expected effect of dataset size. The smaller datasets naturally produce fewer matches in total, and therefore also fewer rare matches. Nevertheless, even in the smaller PROGpedia datasets, complete matches already provide us with the essential variety, ranging from rare to frequent, that is necessary for meaningful similarity adjustments in later stages of the evaluation.

8.4.2 Strategy B - Contained Matches

The next step in our evaluation focuses on the contained matches strategy. Since this strategy inherently includes all complete matches, we find many overlaps with the previous results. However, it also introduces a considerable number of additional sequences, particularly with a frequency of zero. These are subsequences of matches that were recognized as sequences but do not correspond to any complete match.

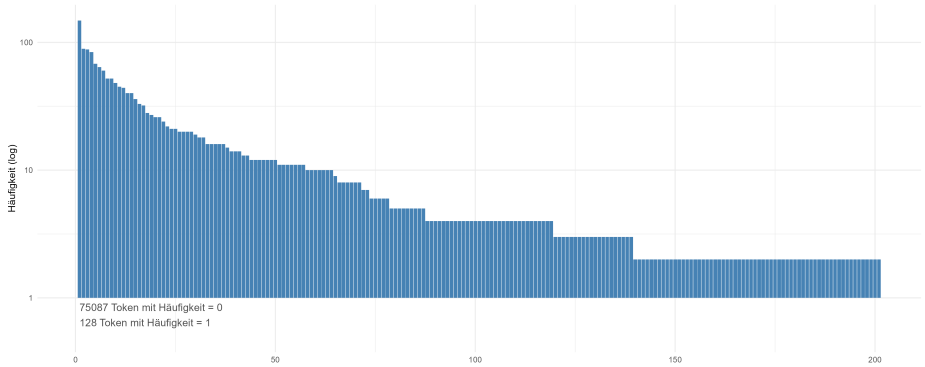


Figure 8.4: Contained Matches Strategy - PROGpedia-19 Frequency Distribution

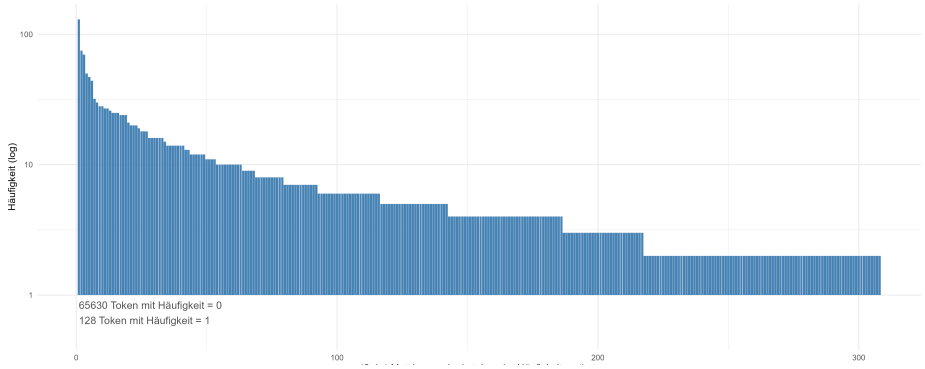


Figure 8.5: Contained Matches Strategy - PROGpedia-56 Frequency Distribution

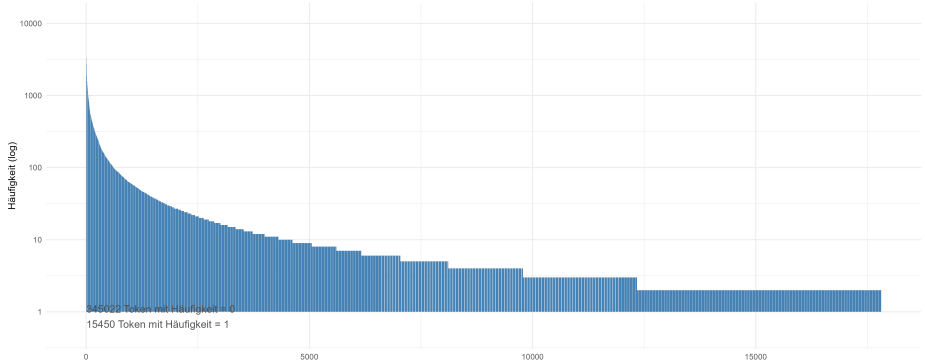


Figure 8.6: Contained Matches Strategy - TicTacToe Frequency Distribution

Result. As we can see, we obtain a large number of sequences that have a frequency of zero. The number of these sequences is particularly high in the PROGpedia datasets. There are more than 100 times as many sequences with a frequency of zero as all other frequencies combined. In the TicTacToe dataset, slightly less than half of all sequences have a frequency of zero. The PROGpedia datasets has sequences with frequencies of more than 100 occurrences, and the the TicTacToe dataset sequences with frequencies of more than 3000 occurrences. There are no larger, noticeable jumps between the frequencies, except for the most frequent matches in the PROGpedia datasets and a few sequences with higher frequencies in the TicTacToe dataset. Generally, we observe that the rarer the sequences, the more sequences with the same commonness are found.

Discussion. The inclusion of such zero frequency matches quickly becomes problematic. Because longer matches automatically generate many substrings, they appear artificially rare. If left unfiltered, this effect distorts the real distribution of frequencies and risks misrepresenting the actual structure of the data. For this reason, we exclude the zero frequency matches from the frequency analysis to maintain a realistic picture. The computational complexity increases substantially when considering submatches; we must consider every substring and each potential match requires an additional filtering step. This fact raises the question of whether the gain in information justifies the cost. The step pattern of the frequency distribution can be explained in the same way as for the complete matches strategy.

We find a large number of different frequently occurring sequences. The sequences with the lowest frequency are so rare that they only appear once. The most frequent sequences, on the other hand, occur often enough that we can assume that not all of them are the result of plagiarism. This means that this strategy also contains a frequency distribution that we can use to predict differences in the effects in the next step. However, the strategy has a disadvantage in that sequences with a frequency of zero must be filtered out, and that they are very common. The fact that the frequencies with occurrence zero are less often found in the larger TicTacToe dataset relative to the others shows that this strategy, even though it is significantly more computationally intensive, probably has a significantly greater impact on larger datasets.

8.4.3 Strategy C - Sub-Matches

In the sub-matches strategy, all substrings of a match are counted.

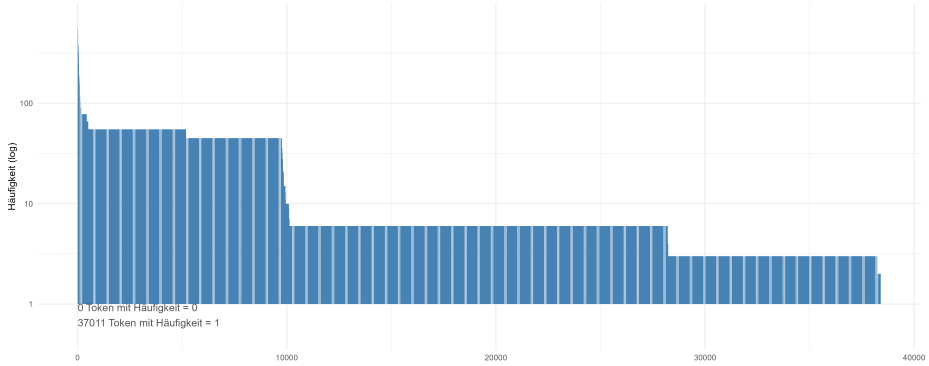


Figure 8.7: Sub-Matches Strategy - PROGpedia-19 Frequency Distribution

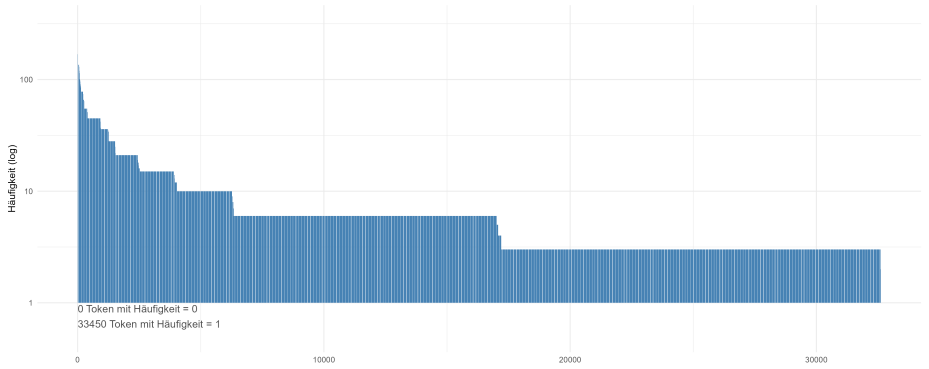


Figure 8.8: Sub-Matches Strategy - PROGpedia-56 Frequency Distribution

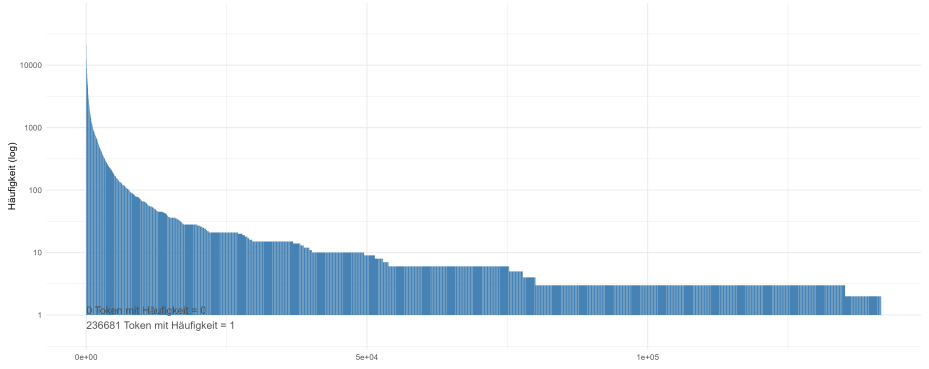


Figure 8.9: Sub-Matches Strategy - TicTacToe Frequency Distribution

Results. We observe that there is a large number of sequences with a frequency of 1. The number of sequences with a frequency of 1 in the PROGpedia datasets corresponds to about half of the sequences, in the larger TicTacToe dataset there are slightly fewer of them. The frequency with the most frequently found sequence is found differs between the three datasets: in the PROGpedia-19 dataset, it is more than 300, and in PROGpedia-56 it is slightly more than 100, and in the TicTacToe dataset it is more than 15,000. A step pattern of frequencies can be seen in all datasets. The most frequently occurring frequencies

seem to be distributed across a few values, with a few intermediate values in between. In all datasets, there is a strong tendency that the more frequently a sequence is found, the fewer other sequences with the same frequency are found. This can be observed in the TicTacToe dataset, in particular, from a frequency of 30 onward. From this point, the effect outweighs the step effect. However, when looking at the total number of sequences found, this proportion corresponds to only about 1% of the sequences considered.

Discussion. We can interpret the strong imprint of the step pattern as meaning that if matches coincide, their subsequences also coincide. If a sequence is found n times, its subsequences are also found at least n times. Since we consider all subsequences per match, we have several sequences per match that were found at least as often as the match itself. The fact that so many are always found with the same frequencies shows that we probably also find the submatches often with a frequency of a triangular number $frequency = n \cdot (n-1)/2$ at low frequencies. At higher frequencies, this assumption is not clearly evident from the data. This is because if a subsequence of a match is found even more frequently in other comparisons, or multiple times in the same comparison, intermediate values may arise. Especially in the TicTacToe dataset, where the step pattern dissolves above a certain frequency there are more different frequency values. The effect that fewer sequences with a high frequency were found can be interpreted in the way as meaning that either subsequences or short matches were found more frequently, since otherwise all subsequences would have the same or a higher frequency. By considering every match sequence and its subsequences there are no sequences with the default value of zero.

Overall, we observe frequent, moderately frequent, and rare sequences here as well, but comparatively more rare ones. This is not unusual, as we are looking at a large number of sequences overall. The only question is whether we are giving sufficient consideration to the comparatively few sequences with high frequencies, given that so many sequences occur relatively rarely.

Even if subsequences are found frequently, the subsequences with high frequencies may be too small and too few, so we may still obtain low frequencies for the entire match, especially for longer matches since we calculate an average across all subsequences. At first glance, this appears to be a clear disadvantage of this strategy.

8.4.4 Strategy D - Window-of-Matches

The window-of-matches strategy is characterized by the fact that it only considers sections of matches that are one specific size. This distinguishes it from the other strategies. All sections of a defined size from all matches are considered. This means that the difference in length between the matches is weighted less strongly.

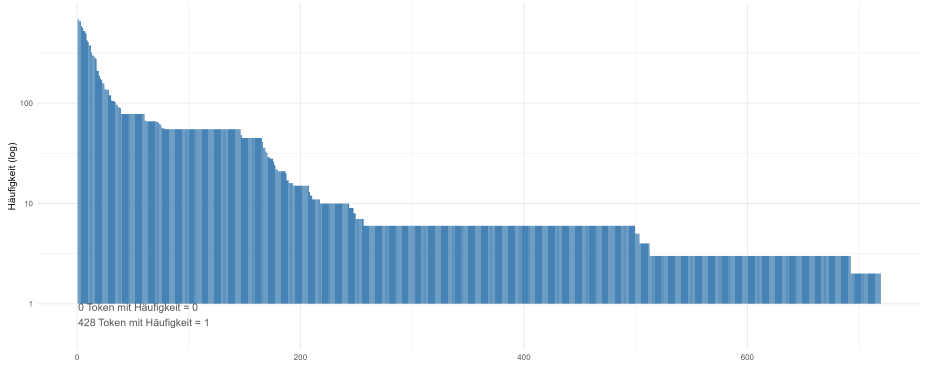


Figure 8.10: Window-of-Matches Strategy - PROGpedia-19 Frequency Distribution

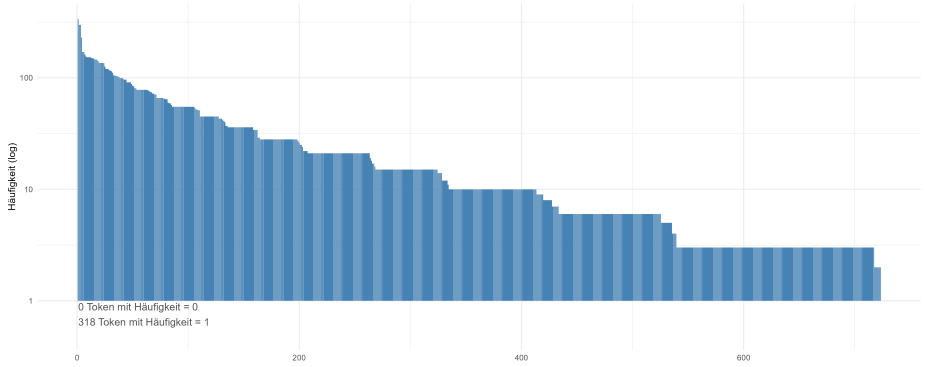


Figure 8.11: Window-of-Matches Strategy - PROGpedia-56 Frequency Distribution

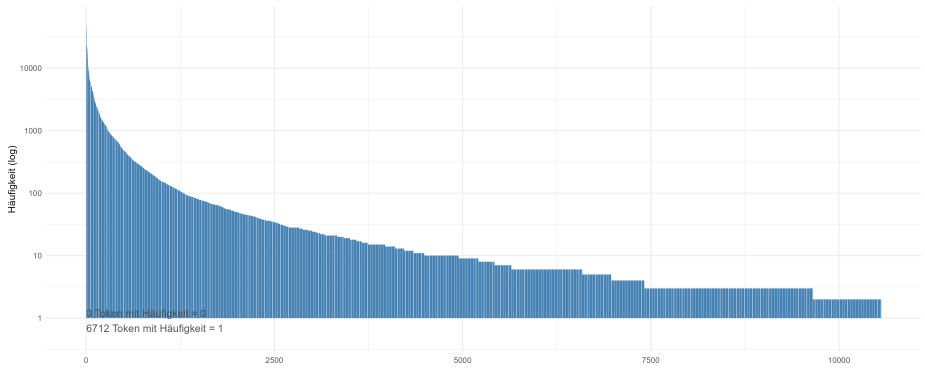


Figure 8.12: Window-of-Matches Strategy - TicTacToe Frequency Distribution

Result. We observe that the number of window sequences found only once in the PROGpedia-19 dataset and in the TicTacToe dataset is significantly less than half of the sequences, and in the PROGpedia-56 dataset it is even less than a third of the sequences. The maximum frequency of all three datasets differs, with the PROGpedia-56 dataset starting at just 300. The PROGpedia-19 dataset contains sequences that were found over 500 times, and in the large TicTacToe dataset, the most frequent sequence is found more than 30,000 times.

The frequency distribution of the datasets shows different characteristics when applying this strategy. In the PROGpedia-19 dataset, most of the sequences are divided into four distinct frequency ranges. The appearance is dominated by these levels, with the exception of the range of highest frequencies and a wider section of frequencies slightly below 10 to frequencies slightly above 30. The PROGpedia-56 dataset, on the other hand, displays multiple, smaller levels in its frequency distribution. The distribution levels steps begin at a frequency of 1 and rapidly become smaller as the frequency increases. Intermediate frequency levels can be observed between the levels. The step effect is least pronounced in the TicTacToe dataset. Except for a few steps at the low frequencies, there appear to be many intermediate steps here, so that no step shape is recognizable. The number of sequences with the same frequency tends to decrease with increasing frequency.

In general, we find rare, moderately rare, and frequent sequences in all three datasets using this strategy. The sequences seem to be relatively well distributed overall.

Discussion. The fact that we find many sequences multiple times indicates that if we choose a window size that is small enough, we can actually map some partial relationships using this strategy. Due to its structure, we should find parts of both: sequences contained within each other and sequences that match in sections, as long as they are at least the size of the window. Due to our chosen window size, this is the case for all of those relationships here. Since the distribution of frequencies appears relatively balanced overall, this strategy is promising for finding both false-positives and matches with a high probability of plagiarism. The fact that we find more rare sequences overall is strongly relativized by the fact that we also consider sequences that we have found with a high frequency, as often as they where found. In an initial assessment of the strategy, a relatively large number of matches is found. However, this also means that rare matches are more meaningful overall, as the individual window sequences carry more weight. Matches that are still very rare after the average has been calculated. At the same time, the frequent ones also carry a lot of weight, and we should be able to identify some sequences whose similarity is purely coincidental.

A notable observation is the considerable variation in the step characteristics across the datasets. A part of the effect can certainly be attributed to the different sizes of the datasets. Especially the PROGpedia-56 has a relatively large number of subsequences. The fact that we consider subsequences of window size, considers various possible overlaps of the sequences, resulting in a high variety of frequencies. While only considering subsequences of a fixed size, leads less significant effect of considering subsequences from the middle of a sequence more, since the borders of a sequence, with the size "window size - 1" are taken less into consideration.

8.4.5 Conclusion

If we look at our results from all strategies, we find both rare and more frequent sequences in all strategies. However, the frequencies and proportions of rare and frequent sequences vary between the different strategies. In the frequency distributions graphs, a step pattern can be observed in most strategies, which is particularly pronounced in the sub-match

strategy. Although steps can be seen in the contained matches strategy and the complete matches strategy, they are clearly less pronounced than in the sub-matches strategy. In the window-of-matches strategy, the step pattern becomes less pronounced as the dataset size increases.

However, the additional step of examining all subsequences and filtering out those who do not equal a match makes the contained matches strategy significantly more complex. The significant similarity in the resulting frequency distributions indicates that only a small number of additional contained matches are actually found. In summary, the contained matches strategy introduces substantial complexity while providing relatively little added value on our datasets. This raises the question whether the additional implementation effort is justified by the limited benefit gained from the few extra sequences identified.

In the sub-matches strategy, all matches and submatches that were also considered in the contained matches strategy are found. Consequently, the frequency distribution contains higher values. The increase in higher frequency values can be attributed to the fact that more subsequences are considered. It should be noted that the maximum frequency in the PROGpedia-56 dataset does not differ much, while it increases significantly in the others. The fact that the value in the PROGpedia-56 dataset does not increase as sharply as in the others is supposed to be caused by the fact that more different sequences were found in the matches.

The sub-matches strategy is also highly computationally complex, and as described in Section 5.5, has the disadvantage that individual frequencies may have little influence since many subsequences are considered. Given the small number of low frequencies found, this can lead to the match frequencies tending to be lower.

The windows of matches strategy differs from the approach of the other strategies in terms of computational complexity. It is less complex than the contained match or sub-match strategy, but it is slightly more complex than the complete matches strategy. Nevertheless, the windows sequences can take into account some of the subrelationships of sequences. We were able to observe that there are many window sequences with a high frequency found.

In conclusion, it can be said that the complete matches strategy and the windows of matches strategy provide the most information with a reasonable level of complexity.

8.5 Frequency Similarity

In the second part of the evaluation, we compare the different weighting strategies, discuss various options for selecting the corresponding weighting parameters, determine if we can identify a trend, and find parameter settings that produce particularly good results. We try to determine whether including more frequent and less frequent matches can lead to a higher probability of detecting plagiarism and thus improve plagiarism detection. Since we decide on the frequency of matches based on the strategy, we also look at the results for the different strategies in the following.

In order to determine whether we can actually detect plagiarism more effectively, we label the datasets as described in Section 8.2 and divide their submissions into the groups *plagiarized*, *conspicuous*, and *inconspicuous*. This allows us to observe how the similarity values of the individual groups relate to each other when weighting. In order to better detect plagiarism, it is desirable that the comparisons from the plagiarized group are more distinct from those from the inconspicuous group. The behavior of the conspicuous group is somewhat more difficult to evaluate, as we may have both plagiarism and non-plagiarism in this category.

In the following sections, we analyze the influence of the individual weighting functions and see how the different groups behave in relation to each other at different weights. In addition, we compare the mean and median differences for the similarity values of comparisons labeled *plagiarized* relative to those labeled as *inconspicuous*.

8.5.1 Proportional Weighting

In this evaluation section, we analyze the results of a proportional weighting function. This means that frequent matches are less heavily weighted and rare matches are more heavily weighted, as described in Section 6.3. The weighting function is considered with different parameter values. With a parameter value of 0, the weighting function is not included, allowing us to compare how the groups behave in relation to the original values.

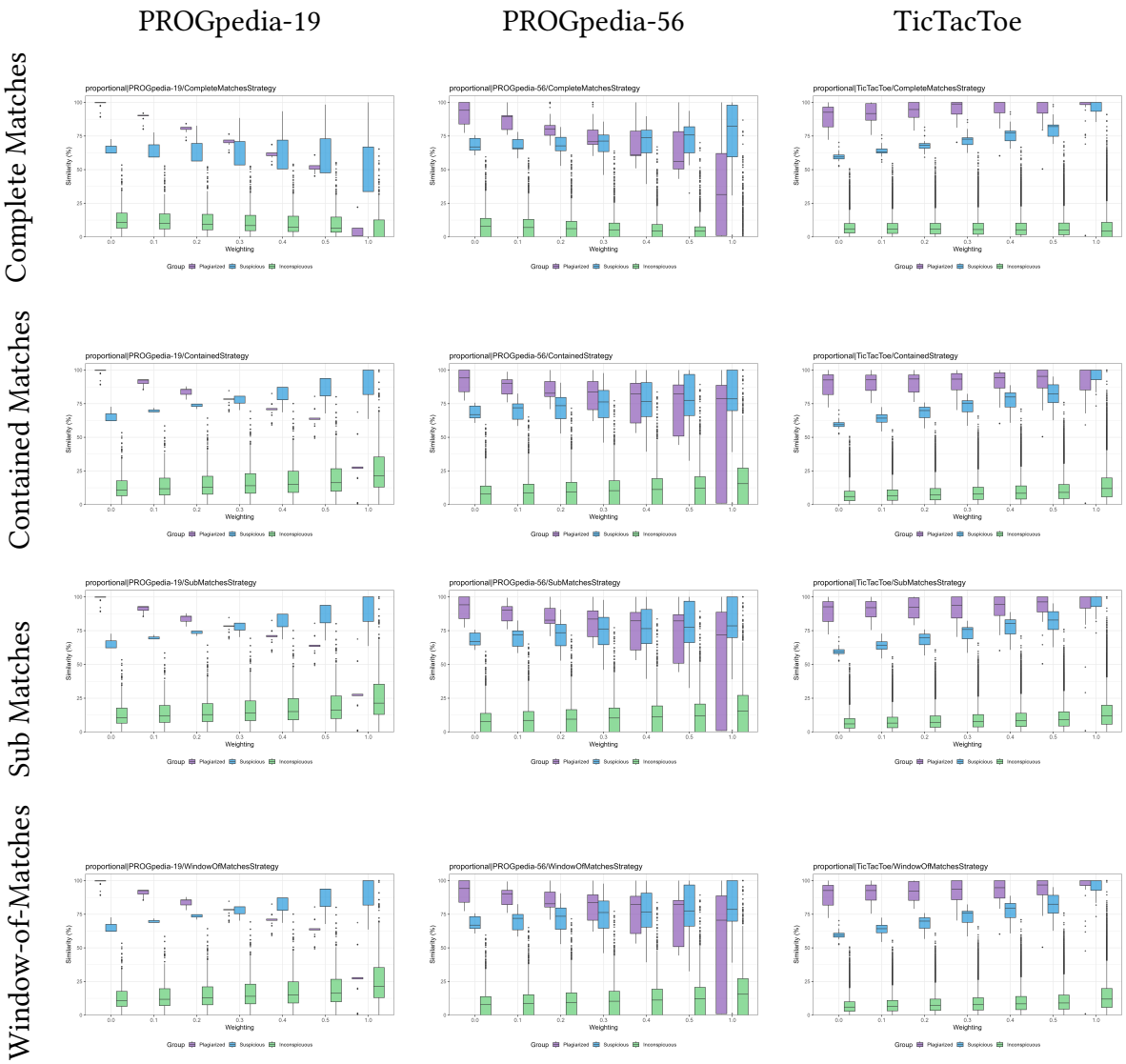


Figure 8.13: Proportional Weighting - Similarity Values by Category

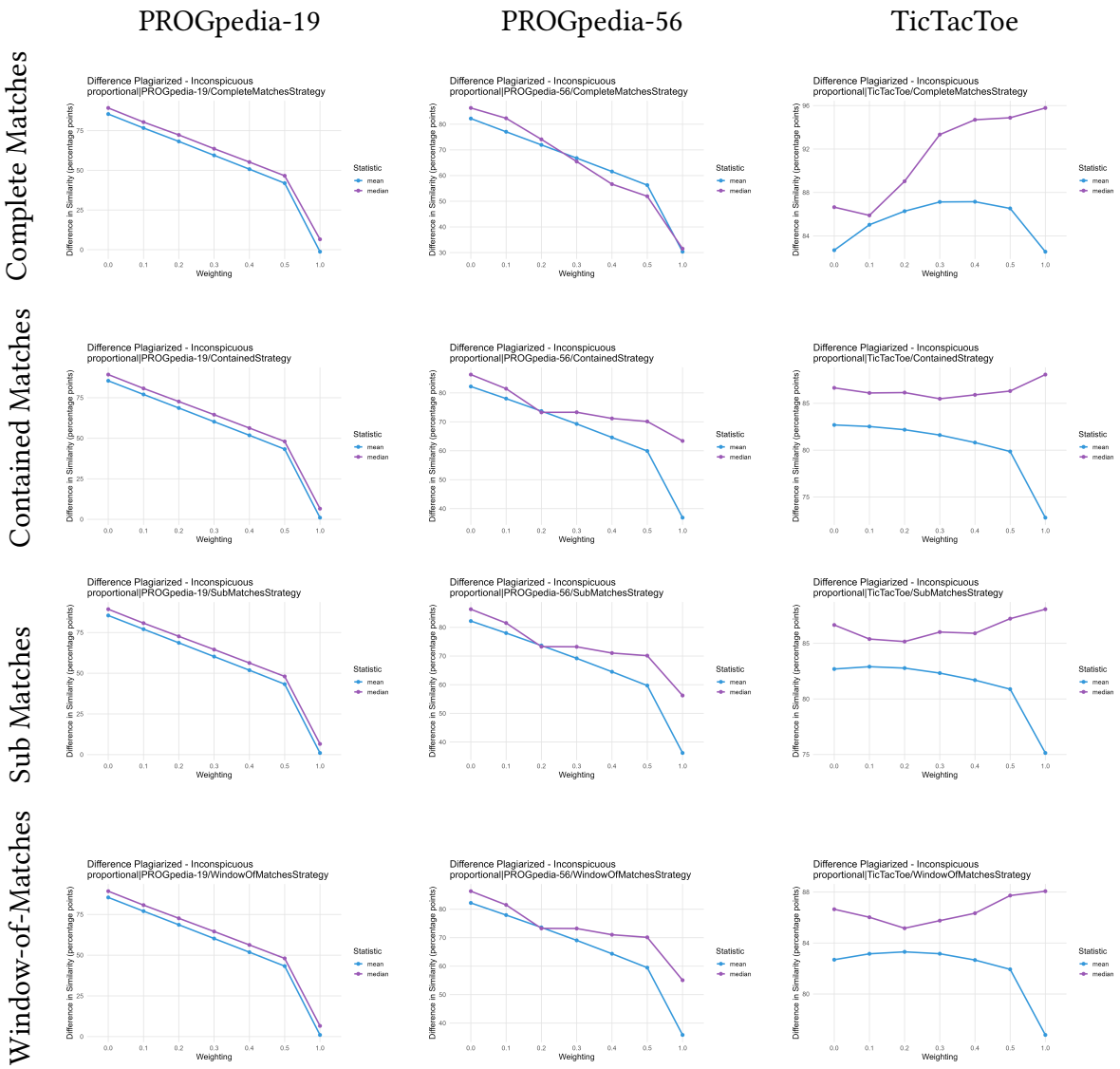


Figure 8.14: Proportional Weighting - Similarity Value Differences

Result. We can observe that, in the inconspicuous group, similarity values calculated using the complete matches strategy decrease across all three datasets as the weighting increases. For all other strategies, however, similarity values increase visibly. In the plagiarism group, similarity values decrease in both PROGpedia datasets. In PROGpedia-19, increasing weighting causes outliers to spread in both directions, whereas the TicTacToe dataset exhibits different behavior. Here, similarity values for the plagiarism group increase with higher weighting factors.

In the PROGpedia-56 dataset, we observe that the interquartile range broadens with higher weighting for all strategies. with a weighting of 1, the range covers more than 50% of the similarity scale. Except for the complete matches strategy in PROGpedia-19, a general trend is visible: the interquartile range of the flagged category generally achieves higher similarity values with increased weighting. In contrast, in PROGpedia-19 under the complete matches strategy, the interquartile range also tends to widen, but the majority of similarity values for the category are lower. Nevertheless, a large spread toward higher values is also observed.

Considering the differences between mean and median values for the plagiarism and inconspicuous categories, we see in PROGpedia-19 that all four strategies show the same trend. Both mean and median decrease approximately linearly with increasing weighting. PROGpedia-56 shows a similar downward trend in mean values across all strategies. However, the median behaves differently. In the complete matches strategy, it decreases more sharply than the mean but eventually converges to approximately zero in both PROGpedia datasets. In the other strategies, the median initially decreases with increasing weighting up to 0.2, even more sharply than in the complete matches strategy, then slightly increases at a weighting of 0.3, before decreasing again. The decrease is overall smaller than in PROGpedia-19, and at a weighting of 1, the difference between categories in the contained matches strategy remains above one percentage point and slightly below 60 percentage points in the other two strategies.

The TicTacToe dataset behaves differently. In all strategies, the difference of means between the two categories initially rises slightly and then declines more strongly. Except for the complete matches strategy, these curves are only slightly pronounced. In the window-of-matches strategy, the mean difference starts to decrease at a weighting of around 0.2, in the sub-match strategy at 0.1, and in the contained strategy, the decrease begins immediately. However, the difference declines much more slowly than in the other datasets. In the complete matches strategy, a pronounced curve is observed, with a difference of roughly five percentage points. The maximum occurs at a weighting between 0.3 and 0.4, and even at a weighting of 1, the difference remains as large as at a weighting of 0. In the other strategies, the mean difference at a weighting of 1 is noticeably smaller than at 0.

Regarding median differences in the TicTacToe dataset under the complete matches strategy, the difference increases significantly after initially decreasing slightly at a weighting of 0.1, with the incline becoming slightly less steep beyond a weighting of 0.3. Overall, the difference grows by up to 10 percentage points between weightings of 0 and 1. In the other strategies, the total difference also increases, but less sharply. In the window-of-matches strategy, the median difference decreases up to a weighting of 0.2. As in the sub-match and

contained matches strategies, the difference initially decreases with weighting up to 0.3 before rising again.

Discussion. The inconspicuous group shows a slight increase in similarity scores. At very high weightings, some of these submissions receive such high similarity scores that they exceed the plagiarism thresholds of the datasets. This tendency can occur in individual comparisons due to rare matches. However, since we assume that this group does not contain any plagiarism, this results in false-positives, especially when more than just a few comparisons are considered. Accordingly, this suggests that the weighting should not be set too high.

In contrast, similarity scores in the plagiarism group decrease sharply in the PROGpedia datasets. This tendency means that, starting from a weighting of approximately 0.4 in PROGpedia-56 and a slightly higher value in PROGpedia-19, some submissions may no longer be recognized as plagiarism by the user. The fact that both mean and median decrease uniformly in the difference in PROGpedia-19 indicates that the trend arises from overall group movement rather than the influence of individual extreme values. In PROGpedia-56, the median decreases significantly less than the mean for all strategies except complete matches strategy, starting from a weighting of 0.3. This suggests that the mean is influenced by a few outliers. Based on the boxplots, we can assume that many high outliers in the inconspicuous comparisons are largely responsible.

Since our goal is not only to ensure that plagiarism can still be detected despite weighting but also to improve plagiarism detection, the mean and median difference between the two groups should ideally increase. This is not the case here. Accordingly, the weighting strategy appears to provide no useful benefit for plagiarism detection in the two PROGpedia datasets.

In the TicTacToe dataset, similarity scores increase, but the mean decreases after a certain point, while the median continues to rise. This indicates that there are some outliers strongly influencing the trends of both plagiarism and inconspicuous comparisons. Except for the complete matches strategy at a weighting factor of around 0.15, there is no value where both mean and median differences rise. Even when the median rises, the mean decreases. This raises the question of whether the extreme values allow weighted similarities to provide an advantage or whether the outliers are too extreme. Many extreme values in the inconspicuous group result in false-positives at high weighting, and plagiarism submissions may also no longer be recognized as likely plagiarism by users at weightings of around 0.3 to 0.4 due to their low similarity scores. For this dataset, the weighting strategy could offer a benefit at low weightings by appropriately shifting submissions in the similarity score based on rare and frequent matches for plagiarism cases. However, as seen in the other two datasets, the thresholds at which a benefit occurs can shift quickly. It seems highly dataset-dependent whether a benefit arises. For most datasets and weighting functions, the trend tends to discourage plagiarism detection, and we risk generating false negatives in addition to false-positives, which would not occur without weighting. This effect is undesirable, as it not only fails to provide a measurable improvement but may actively worsen plagiarism detection.

8.5.2 Linear Weighting

In this weighing strategy, we prevent generating false-negatives. To do so, we will leave out the option to weight matches less, so comparisons have a minimum similarity value before applying the weighting. Rare matches will still be weighted more. The rarer the matches are, the more the similarity score of the comparison will increase. The value of the gradient is determined by a linear weighting function, as described in Section 6.4.

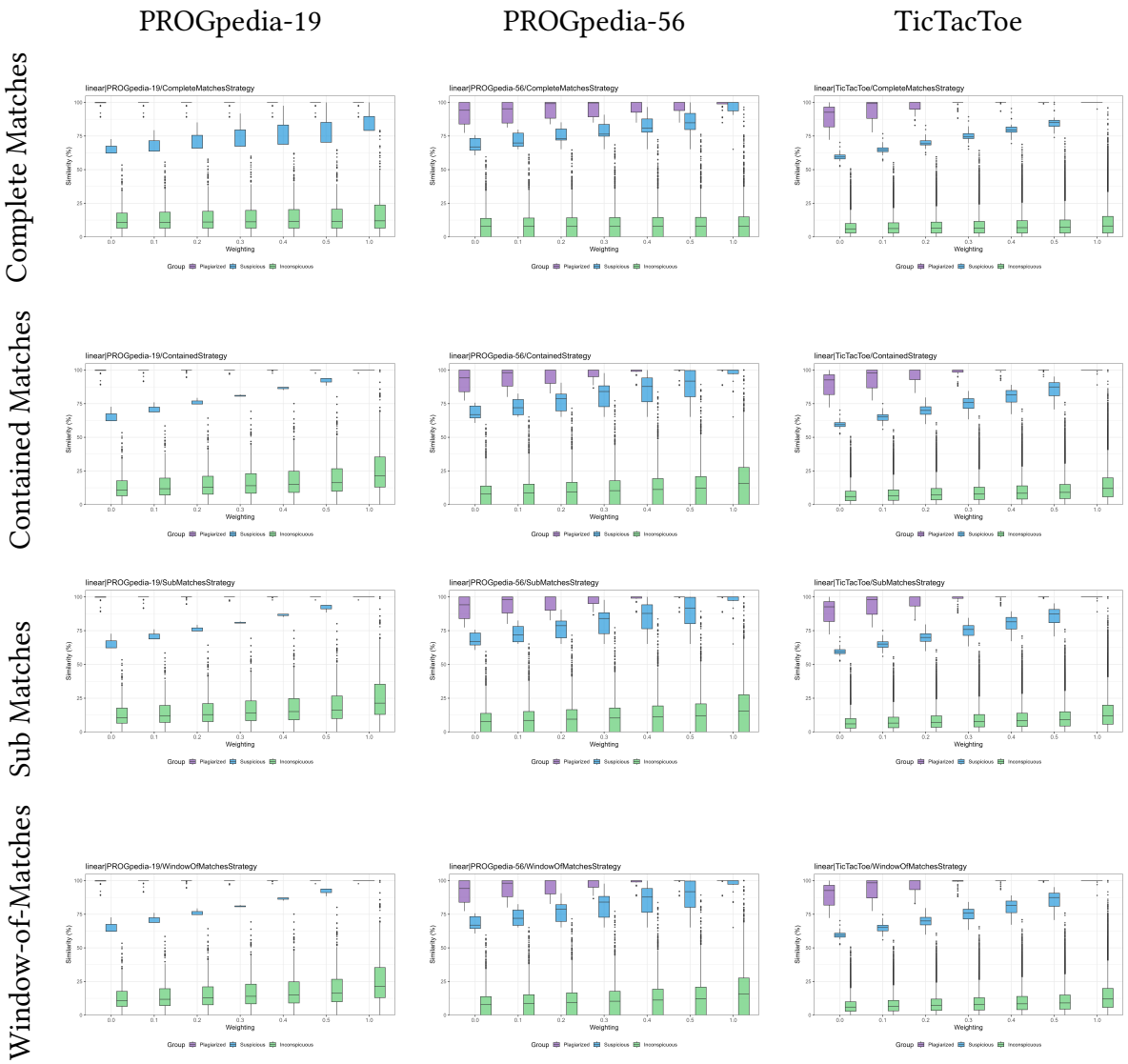


Figure 8.15: Linear Weighting - Similarity Values by Category

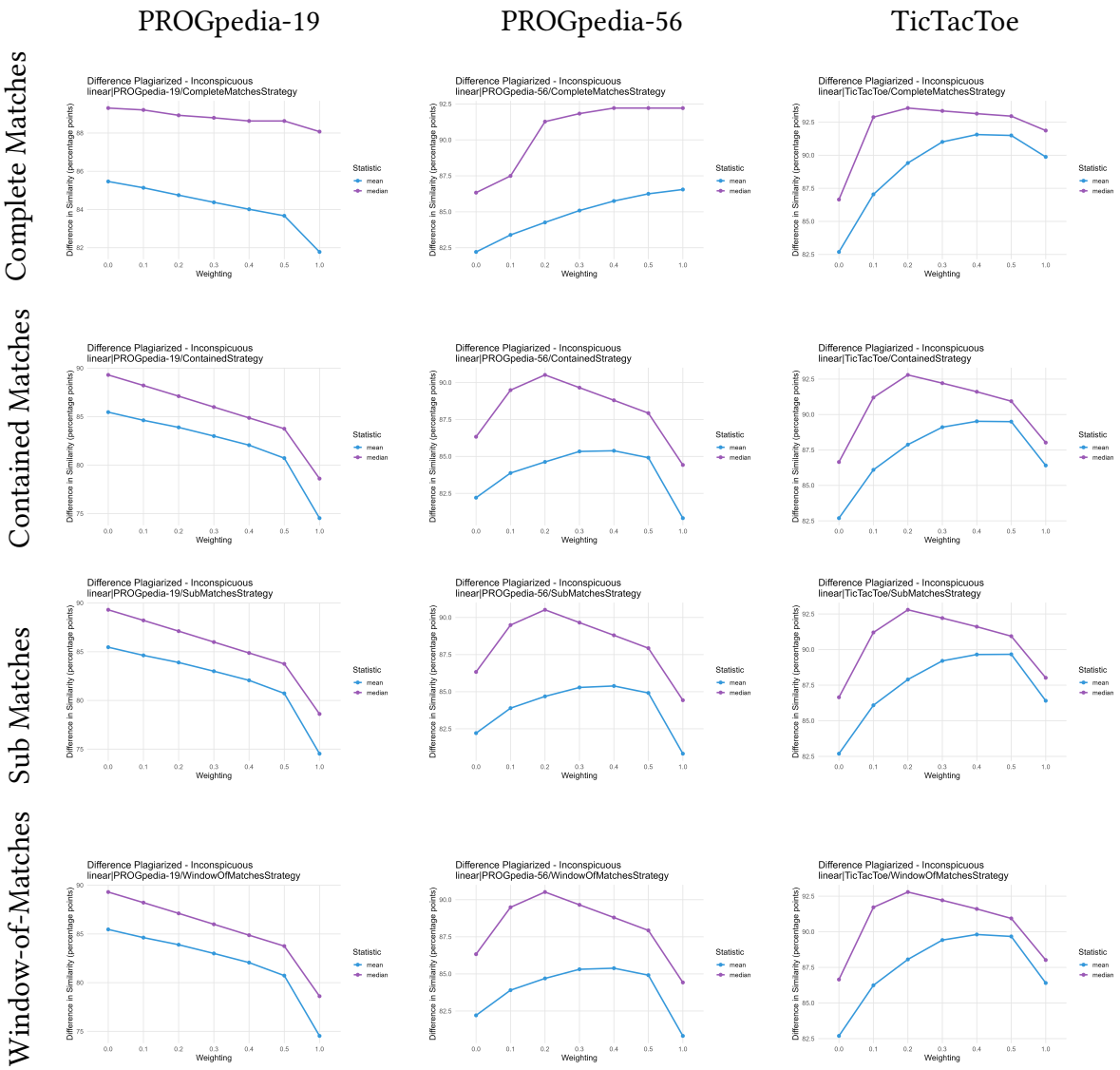


Figure 8.16: Linear Weighting - Similarity Value Differences

Result. With this weighting strategy, we can observe that the boxes in the boxplot only move upward with increasing influence of the weighting.

In the PROGpedia-19 dataset, where the similarity scores of the plagiarized comparisons start very high, the observed increase in this category is very small. In contrast, we can observe a noticeable increase of the interquartile range and an even higher increase if we observe the outliers. We can observe this trend in all strategies, with the weakest effect in the complete matches strategy. The graph of the mean and median shows a decline in the similarity scores. Again, in the complete matches strategy, the decline is much lower than in the other strategies.

The PROGpedia-56 dataset starts with lower values of the plagiarized comparisons. When applying the weighting factor the interquartile range increases and there is also a significant up-shift of the suspicious comparisons. When applying higher weighting factors there are some outliners in the lower range. In the mean values, there is a noticeable increase in the difference up to a weight of 0.4, before they decline; the same is happening more strongly in the median, with a peak at the weight of 0.2. The only exception is the complete matches strategy, where the values of the difference in the mean and median increase continuously. At the same time, there is a significant increase in the median difference between a weight of 0.1 and 0.2.

In the TicTacToe dataset, an increased weighting factor leads to a notable increase in the interquartile range of the plagiarized comparisons group, reaching a similarity of approximately 100% starting at a weighting factor of approximately 0.4. Starting from a weighting factor of approximately 0.3 there are there are some outliners in the lower range, slightly stronger than with the PROGpedia-56 dataset.

The inconspicuous group behaves similarly to the other datasets. In the mean and median values, there is an increase in the mean difference, up to a weighting factor of approximately 0.5. The median value difference of the plagiarized and inconspicuous group, shows a considerable increase, especially from weight 0 to 0.1 and then a gradual decrease after a weight of 0.2. Although the difference between the mean and median gets lower, there is still a noticeable increase against the values without weighting, i.e., the weighting factor set to zero.

Discussion. The observed increase in the difference in mean and median shows that applying a weighting factor actually improves the detection of plagiarism.

The observed decrease in the difference of the mean and medium values in the PROGpedia-19 dataset can be attributed to the fact that the comparisons in the plagiarized group already started with a very high similarity score. The weighting factor increases the similarity score, but the already high similarity score of the plagiarized comparisons reaches the range limit and cannot be raised further. This causes the values of the plagiarized and inconspicuous group to converge. When applying the weight factor to the frequency values of the complete matches strategy, the decrease is less than with the other strategies. This can be attributed to the fact that the similarity values of the inconspicuous group increase less than in the other strategies. There are no subsequences, and therefore also less rare sequences are

considered. This tendency becomes significantly smaller with an increasing dataset size. In the large TicTacToe dataset it becomes almost invisible.

In the other datasets we can see a strong incline in the mean and median values when applying a increasing weighting factor. Starting at a weighting factor of around 0.2 the median values start decreasing again. The mean values start to decrease at a weighting factor of around 0.4 with the PROGpedia-56 dataset and 0.5 in the TicTacToe dataset. In the PROGpedia-56 dataset the mean and median values get even lower than without a weighting factor, and in the TicTacToe dataset they stay above values without a weighting factor.

An exception is the complete matches strategy applied to the PROGpedia-56 dataset. The mostly increasing curves of the mean and median values can be attributed to the fact that the similarity values in the plagiarized group as well as in the inconspicuous group rise, but slower than with the other strategies and datasets. Therefore, the similarity values of the inconspicuous group do not reach the limit that fast when the weighting factor is applied. That the strong increase in the difference of the median value for weighting factors between 0.1 and 0.2 is not observable in the difference of the mean value, indicates that there are more outliers in the inconspicuous group that reduce the effect.

There is an improvement in the detection of plagiarized submissions, except for the PROGpedia-19 dataset. That the mean and median values start to decrease in the PROGpedia-56 dataset when using high weighting factors indicates that we most probably get the best effect when using low weighting factors.

Since we intend that the detection inside the plagiarized group distinguish more between the rare and more frequent matches, we will evaluate a quadratic weighting function that reduces the increase for rare matches in the next section.

8.5.3 Quadratic Weighting

In this section, we employ a quadratic weighting function to the similarity score. This function does not reduce the influence of the matches. Therefore, the comparisons can only increase in the similarity score. The slope of the quadratic weighting function 6is less than the slope of the linear weighting function. Therefore, frequent and medium frequent matches are rated less, as described in Section 6.5. Rare matches are also be less influenced by the quadratic weighting function.

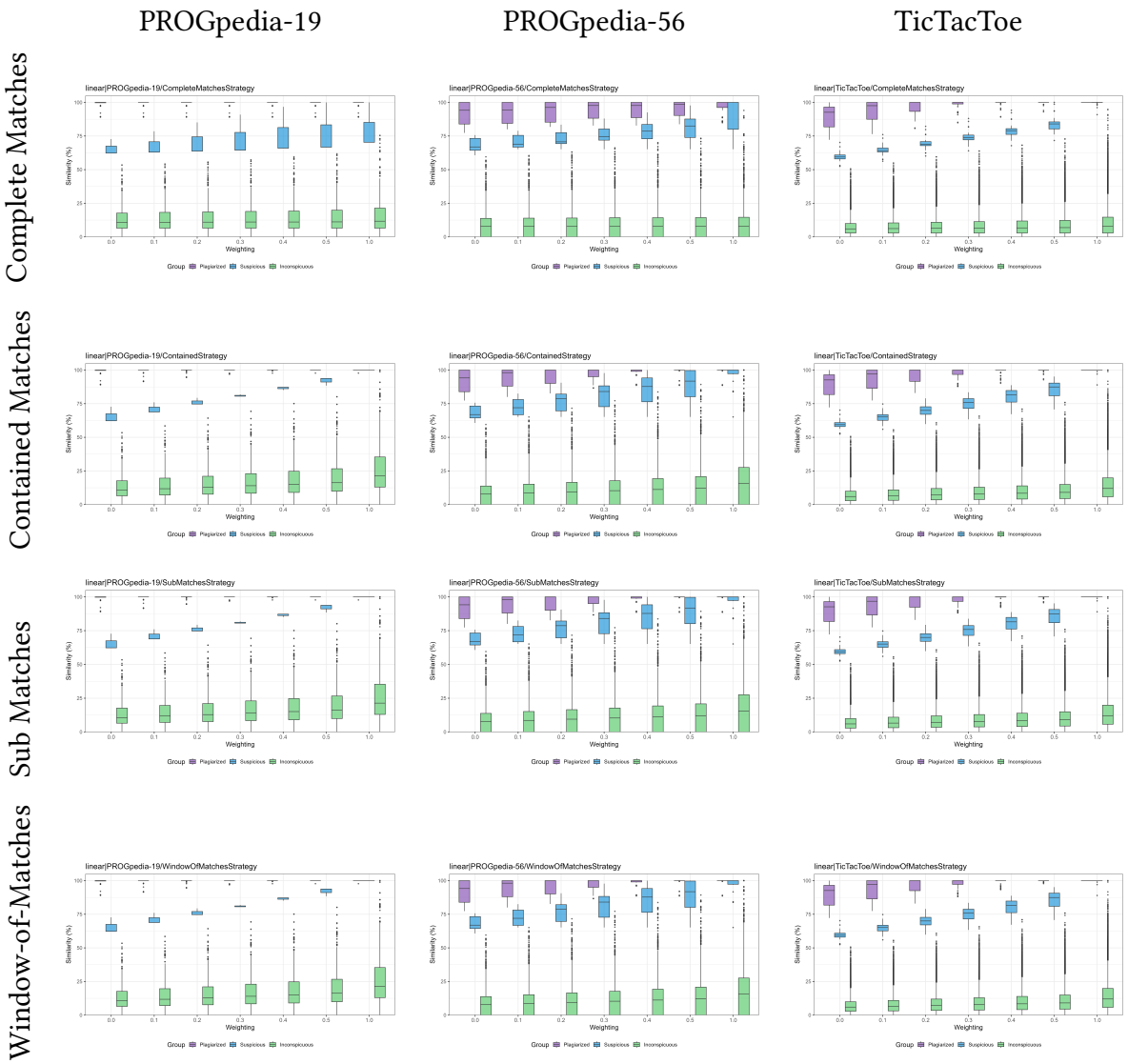


Figure 8.17: Quadratic Weighting - Similarity Values by Category

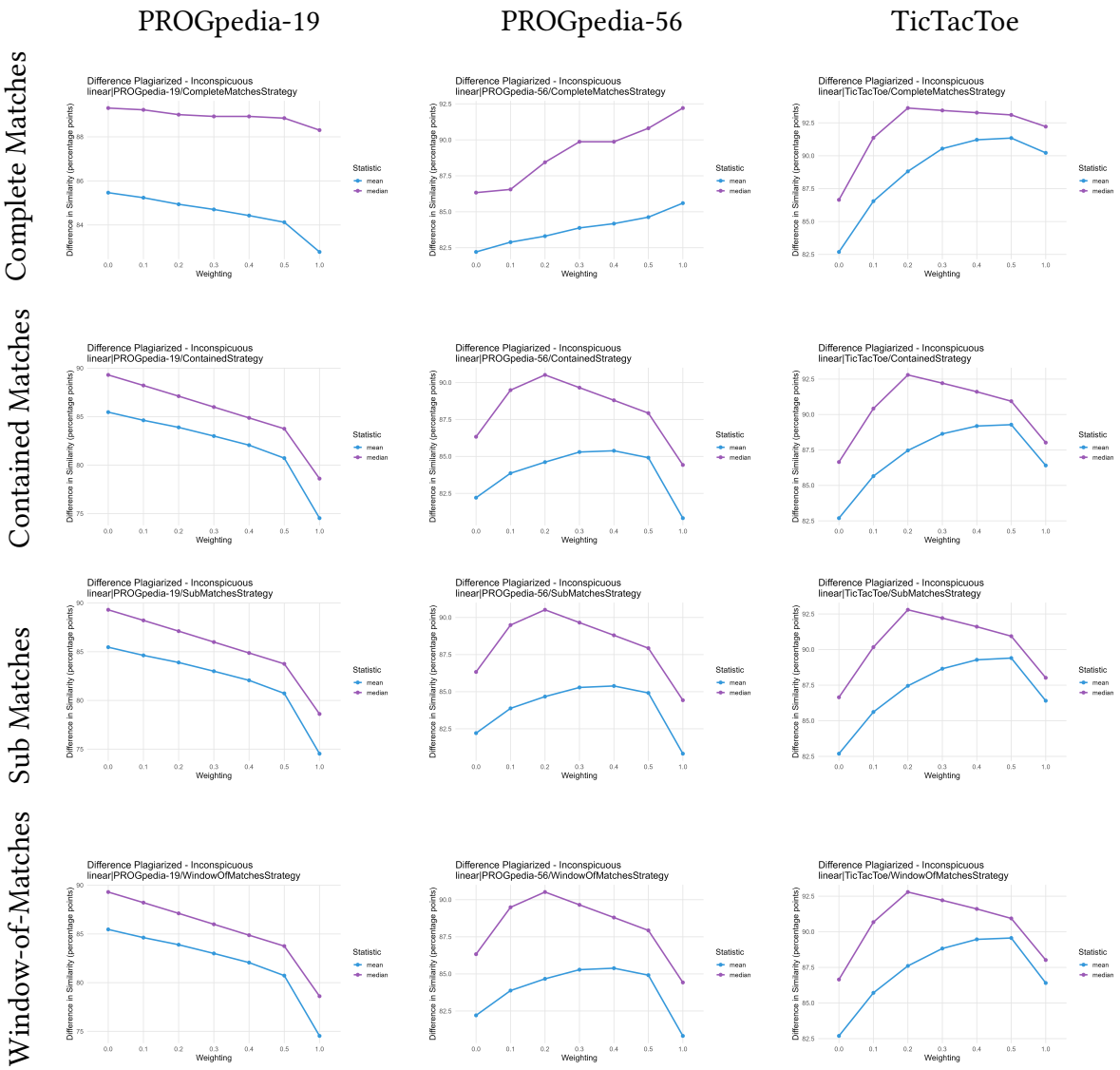


Figure 8.18: Quadratic Weighting - Similarity Value Differences

Result. A visual analysis of the graphs indicates an increase in similarity values across all datasets with an increase in the weighting factor. The comparisons reach a similarity score of 100%, starting from a weighting factor of approximately 0.4. The only exception is the complete matches strategy applied to the PROGpedia-56 dataset, where the similarity score of the plagiarized group increases less.

It can be seen that with a weighting factor of 1 even in the inconspicuous group the similarity value of few extremes reaches 100%, except in the complete matches strategy applied to the PROGpedia-19 dataset. When using a weighting factor of 1 on a PROGpedia-19 dataset some values in the inconspicuous group still reach a similarity score of around 75%. In the Progopedia-56 dataset in particular, we can observe that the range of the frequency distribution of similarity values increases with increasing weighting factor. Although the difference in the median for the complete matches strategy in the PROGpedia-19 dataset only decreases slightly and remains almost constant, the difference for all other strategies is in line with the mean difference values.

In the other two datasets, the mean and median differences first increase before decreasing again. The largest difference in the mean values is achieved with a weighting factor of 0.4 in PROGpedia-56 and 0.5 in the TicTacToe dataset. The highest median value is achieved with a weighting factor of 0.2. In the PROGpedia-56 dataset, except for the complete matches strategy, they fall below the original values again with a weighting factor of 1. In the TicTacToe dataset, even with a weighting factor of 1, the difference remains significantly above the original values (weighting factor = 0).

When applying the complete matches strategy on the PROGpedia-56, there is a consistently growing difference in the mean and median values of the PROGpedia-56 dataset. Between a weighting of 0.1 and 0.3, the difference in the median becomes particularly large, while between 0.3 and 0.4 it hardly increases at all. In general, the mean value increases almost evenly.

Discussion. The considerable increase in similarity scores we observe in Figure **??** for this weighting strategy leads to decreasing mean and median differences once the weighting factor becomes large enough. This effect can be attributed to the fact that many similarity values of the comparisons in the plagiarism group reach the upper limit of 100% and cannot increase further. At the same time, the values in the inconspicuous group continue to increase, since this group also contains matches and sequences with varying frequencies. The fact that the complete matches strategy shows a slightly less substantial increase in its similarity scores as the weight grows, indicates that this strategy handles both higher weights and a larger number of rare matches more robustly. In addition, it contains a lower risk of selecting an unsuitable weighting factor while still providing a positive effect on plagiarism detection.

Since the PROGpedia-19 dataset already starts with very high similarity scores, we observed a reduction in the differences between the mean and median. We can assume that similar behavior will appear in other datasets with high initial values. This assumption is supported by the fact that the other two datasets show the same trend as soon as most similarity scores in the plagiarism group reach 100%.

Because we do not want to evaluate the weighting function based on a single specific dataset, we should consider both cases: datasets containing many plagiarism submissions with very high similarity scores, and datasets where values are comparatively low. Overall, except for the PROGpedia-19 dataset, we observe an increase in mean and median differences in all datasets. This suggests that for moderate weighting factors, the approach provides added value for plagiarism detection. In the other two datasets, the median difference, with only one exception, reaches its maximum at a lower weighting factor than the mean difference. This gives us an approximate range between 0.2 and 0.45, where we likely achieve the highest benefit, as long as the similarity scores are not already very high at low weighting factors. In cases where they are, using a smaller weighting factor is advantageous, since the differences between mean and median decrease less. For unknown datasets, a weighting factor of around 0.2 or slightly higher is therefore likely to provide a positive effect without posing too high a risk of significant deterioration in plagiarism detection.

8.5.4 Sigmoid Weighting

In this section, we examine the effect of a sigmoid weighting function. We have adapted this weighting function to our goals as described in Section 6.6. The purpose is to minimize the similarity of likely false-positive comparisons, similar to the quadratic weighting function. At the same time, we want comparisons that contain large segments of rare matches to stand out clearly by assigning them a very high similarity score. In doing so, we combine the advantages of the weighting strategies considered so far.

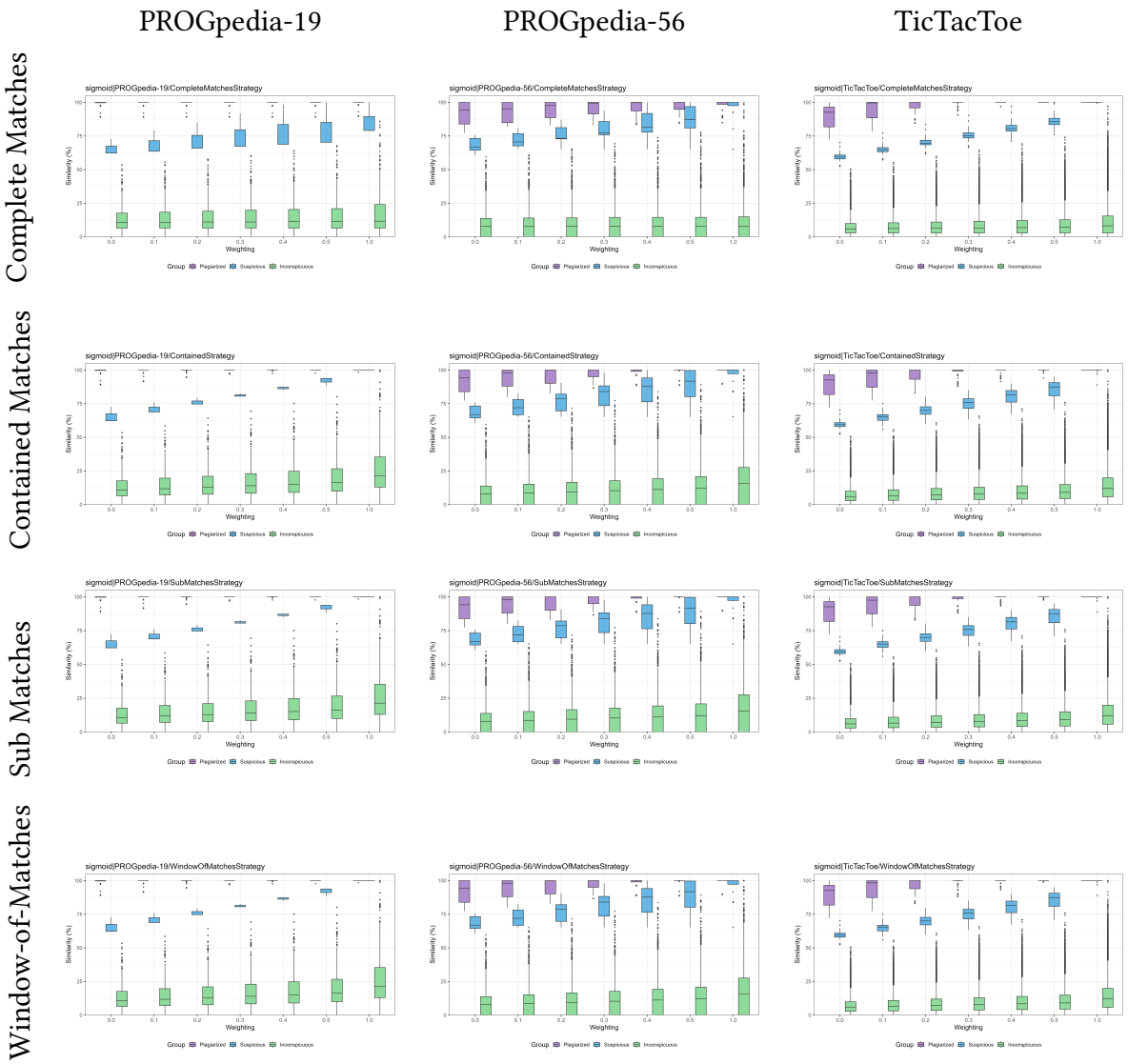


Figure 8.19: Sigmoid Weighting - Similarity Values by Category

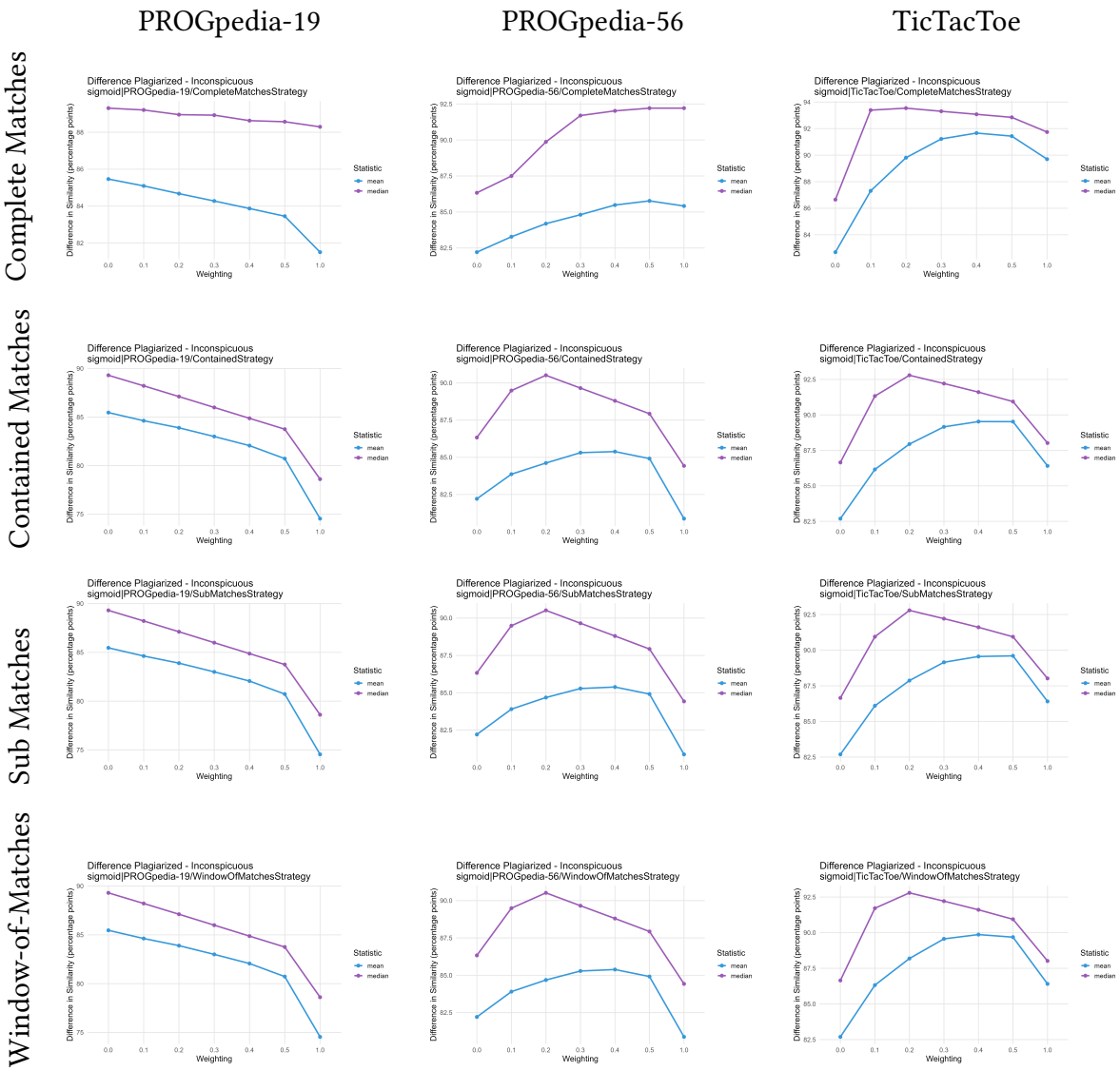


Figure 8.20: Sigmoid Weighting - Similarity Value Differences

Result. In the boxplot, we observe that all groups increase in similarity as the weight increases. In the PROGpedia-19 dataset, the comparisons in the plagiarism group already start with a very high similarity of approximately 100% in the median, whereas they start lower in the other datasets. When the weighting factor becomes large enough, the median and most values in the plagiarism groups also reach a similarity of 100%. In the PROGpedia-56 dataset, this generally happens at a weighting factor of 0.4, and in the TicTacToe dataset at a weighting factor of 0.5. In the PROGpedia-56 dataset, similarity values increase overall at the slowest rate. Even with a weighting factor of 0.4, the median is already at 100% similarity, but some values still have slightly lower similarity. The similarity values of all groups increase at roughly the same rate as the weighting factor increases.

As the dataset size increases, the maximum achieved mean and median difference also increase. In the complete matches strategy, the median difference slightly decreases in the PROGpedia-19 dataset. In contrast, in the TicTacToe dataset, the median difference first increases sharply at a weighting factor of 0.1, reaching almost 94 percentage points, before it slowly declines. The mean difference even increases up to a weighting factor of 0.4 before decreasing again. In the PROGpedia-56 dataset, the mean difference rises to a weighting factor of 0.5 before declining, while the median difference rises continuously, although the rate of increase slows noticeably at a weighting factor of 0.3.

In the other strategies, we see that in the PROGpedia-19 dataset, both the mean and median differences slowly decrease. In the PROGpedia-56 dataset, the differences first increase and then decrease; the mean reaches its peak at a weighting factor of 0.2, while the median reaches its peak at a weighting factor of 0.4. In the TicTacToe dataset, the boxplot in Figure 8.19 shows a similar trend. However, with increasing weighting, we obtain generally higher differences that do not drop below the value without applying the weighting function (weighting factor = 0). The maximum mean difference is sometimes slightly shifted here, and for the Contained matches and sub-matches strategies, the difference only reaches its peak at a weighting factor of 0.5.

The overall mean and median differences achieved are highest in the complete matches strategy, except for the PROGpedia-19 dataset, where the differences only decrease. The other strategies reach similar mean and median differences depending on the dataset.

Discussion. The fact that the mean and median differences begin to decrease again after a certain point can be explained by the similarity scores of the plagiarism group reaching 100% for a large portion of the comparisons once the weighting factor becomes sufficiently high. At that stage, the similarity can no longer increase. The substantial rise of the mean and median differences, especially in the Complete matches strategy, indicates that this strategy provides added value for plagiarism detection. Because of the downward trend at excessively high similarity scores, we can conclude that the weighting factor should not be set too high for an unknown dataset. At the same time, we want to achieve the most significant possible difference. Since the median tends to reach its peak already at a weighting factor of around 0.2, we could select 0.2 as a suitable value. However, the maximum of the mean difference appears later. While such extreme points may be dataset-dependent, we can observe in both the PROGpedia-56 and TicTacToe datasets that these extremes have a substantial impact,

causing the mean to fall clearly below the median. Therefore, we must assume that this behavior, including the occurrence of particularly high or low similarity scores, could also appear in comparisons from unknown datasets. Consequently, it would be reasonable to choose a weighting factor slightly above 0.2, for example, 0.25 or 0.3. Since in the complete matches strategy the PROGpedia-56 dataset shows a purely rising trend, and in the other two datasets the median differences decrease only very slightly, it may also be possible to select a somewhat higher weighting factor for this strategy.

Overall, we can observe a maximum increase of over nine percentage points in the mean difference and over six percentage points in the median difference. The fact that the highest values occur in the complete matches strategy shows that this strategy tends to achieve the most substantial effect in separation between plagiarized and inconspicuous labeled comparisons.

8.5.5 Conclusion

Overall, we find that with proportional weighting, we not only weight plagiarized submissions differently according to the rarity of their matches, but we also take a significant risk of failing to detect plagiarism and producing false-negatives that might have been detected without the additional weighting. Even though in the TicTacToe dataset, particularly in the complete matches strategy, a partially positive effect is observed, this can not be clearly confirmed in the other datasets. Instead, an opposite effect is observed in the other two datasets. Since we do not want to risk generating false negatives, we decided not to lower the minimum similarity further.

Overall, we observe similar trends in the other weighting strategies, albeit with different magnitudes. The results show that with the quadratic weighting function, we achieve the desired effect of similarity values increasing more slowly. This indicates that we find enough frequent matches such that the low weighting has a noticeable effect on frequent matches as similarity increases. However, this effect seems to impact comparisons from both the plagiarized and inconspicuous groups equally, as the differences in mean and median values are similarly high. When we apply the sigmoid weighting function instead to weight frequent matches less clearly, differently from rare ones, giving rare matches high importance, we see a significant increase in the differences of mean and median in the complete matches strategy. Although the similarity values of comparisons rise faster due to more substantial weighting of rare matches, this shows that in the complete matches strategy, we can improve plagiarism detection by deliberately giving rare matches high weight and frequent matches low weight. Plagiarized submissions thus appear, on average, to contain somewhat rarer matches than inconspicuous submissions. Unfortunately, we do not observe this as clearly in the other strategies. In those strategies, no large improvement in mean and median differences can be identified compared to the other weighting strategies. The fact that we observe a reduced increase in similarity values with increasing weighting in the quadratic weighting strategy shows that we achieve a change in the similarities of comparisons by giving frequent matches little weight. When we then weight medium-frequency matches somewhat more and rare matches significantly more in the sigmoid

function, we see that similarity values increase again. Since frequent matches are still weighted minimally, as in the quadratic weighting strategy, we can conclude that within the comparisons, a ranking emerges that is strongly influenced by the match frequencies in the comparisons. In conclusion, all weighting strategies except proportional weighting provide some value for plagiarism detection. To achieve a clear result, the weighting factor can initially be set as a rough guideline to approximately 0.2 to 0.3. The sigmoid weighting function, combined with the complete matches strategy, overall shows the most apparent separation between comparisons in the plagiarized and inconspicuous groups.

8.6 Threats to Validity

Internal validity: The datasets used are labeled by experienced plagiarism examiners based on a threshold. As a result, individual labeling errors cannot be completely ruled out. To reduce the risk comparisons that could not clearly be classified as *plagiarism* or *non-plagiarism* were labeled as *suspicious*. Therefore, the plagiarism and non-plagiarism groups likely contain very few mislabeled data, so the overall deviations in the evaluation can assumed to be low.

External validity: The evaluation is based on two PROGpedia datasets and one dataset from KIT. In total these contain 862 submissions, resulting in 370,791 pairwise comparisons. This large amount of data suggests that our findings should transfer reasonably well to other datasets. We only evaluated Java-based submissions. However, the token based analysis is language independent. The approach was designed to be dataset and language independent and the observed effects can be explained by general patterns, that are likely to occur under similar conditions also in other datasets and languages.

Construct validity: The underlying concept was developed independently of the datasets and can be further extended. We first examined four frequency-based strategies and then analyzed several weighting functions for each strategy. The design explicitly separates the frequency-based strategies from the Weighting functions to ease future extension. This ensures an easy extensibility of the concept.

We followed the GQM plan in our evaluation. We evaluated the output in a first evaluation to verify that our concept produces meaningful outputs. Then we refined the concept for the integration and verified the results. Our design choices were made consciously, based on the target to integrate them into JPlag, which makes us confident that we have already accounted for many relevant possibilities. It is possible that other strategies, data representations (e.g., alternatives to using a list of token types), or implementation methods could potentially improve efficiency or performance.

Reliability validity: We have described our algorithms and concepts in detail. We have created a reproduction package [5] with the source code, the evaluation data sets, and the evaluation scripts. This makes it easy to reproduce the results.

9 Conclusion

This thesis outlines how the frequency distribution of rare and frequent matches can vary and how these frequencies can be integrated into the similarity score using weighting functions. Several strategies that consider different relationships between matches and their subsequences are used to analyze the frequency distributions.

We observe that the sub-matches strategy includes many subsequences with identical frequencies. This effect is less pronounced, but also partially visible in the other strategies. The sub-matches strategy is computationally relatively complex, as it considers a large number of subsequences. The influence of an individual subsequence is reduced, because a large number of subsequences is included per match. Additionally, the frequency of subsequences in the match center has a disproportionately high impact which slightly distorts the intuition about what the frequency values represent.

In contrast, the window-of-matches strategy uses a single fixed window size. We select this size as small as possible, similar to the sub-matches strategy. This method is less computationally complex but still captures many of the relevant relationships.

The frequency distribution of the contained matches strategy is similar to the complete matches strategy. On the same time the complete matches strategy is considerably less computationally complex than the contained match strategy. Therefore, based on our data, the additional benefit of the contained matches strategy seems to be small.

All strategies are able to identify frequent, moderately frequent, and rare matches on all three datasets. However the complete matches strategy and the window-of-matches strategy both provide valuable insights at relatively low computational complexity. These strategies seem to provide the most practical benefit.

Several weighting functions are compared with varying strength (weighting factor), which determines the extent to which the weighting affects the similarity score.

In a first approach, we down-weight frequent matches and up-weighted rare ones, assuming that rare matches are more likely to result from plagiarism, while very frequent matches are often coincidental. However, this approach risks false-negatives because the similarity score may decrease even if some frequent matches still indicate plagiarism. For this reason, we classify the proportional weighting function as less suitable.

In the other approaches, we only increased the weights of matches. A linear weighting function provided an improvement for small weighting factors. However, the linear weighting function also up-weights frequent matches, which is not always desirable. In comparison, a

quadratic weighting function produces a smaller increase in the similarity score because it not only assigns lower weights to highly frequent matches but also lower weights overall.

When comparing the mean and median similarity values of plagiarism cases against inconspicuous comparisons, we observe that the separation increased with all weighting functions, except the proportional weighting function, for higher weighting factors. Using a sigmoid weighting function, which emphasizes rare matches further, we achieved the most significant mean and median difference in combination with the complete matches strategy. This weighting and frequency determination strategy combination has the most apparent separation between plagiarism and inconspicuous comparisons.

Except the proportional weighting function, all weighting functions already reach high similarity scores at moderate weighting factors. Since the most apparent separation occurs at weighting factors around 0.25, we recommend this as an initial reference value for further experiments.

In summary, we are able to identify rare and frequent matches with all strategies and finally determine two approaches with the highest practical values: the complete matches strategy and the window-of-matches strategy. Integrating frequency information into the similarity score through weighting functions proved to be beneficial. Overall, the complete matches strategy performs well, even when using higher weighting factors combined with the sigmoid weighting function. It delivers the best separation between plagiarism and inconspicuous comparisons.

10 Future Work

While this thesis has already explored and implemented several strategies of frequency-based plagiarism detection, there are still multiple promising directions for future research and development.

First, our evaluation was conducted on only three datasets in order to establish initial default parameters. Although these provide a meaningful starting point, additional datasets could help refine the parameters and frequency strategies further. Evaluating across a broader variety of sources may reveal patterns that differ from those observed here and could lead to improved parameter choices for practical use. In particular, future research would benefit from testing on datasets that are not only larger in scale but also explicitly labeled. Access to labeled data would allow for a more precise assessment of how well the strategies distinguish between plagiarism, suspicious submissions, and non-plagiarized work.

Second, we focused our experiments on Java submissions. Although the approach itself is designed to be language-independent, it is possible that submissions in other programming languages may exhibit different behavioral patterns. Future work should therefore extend the evaluation to multiple languages and adjust the thresholds accordingly to ensure that the strategies remain effective in different contexts.

Another promising approach is parameter optimization. In this bachelor's thesis, we selected default parameters for frequency distributions and weighting strategies based on well-founded estimates. With access to larger, more diverse, and explicitly labeled datasets, systematic optimization could further improve both accuracy and robustness.

In addition, runtime efficiency, although considered in our implementation, could further be enhanced. For example, leveraging parallel programming within JPlag may provide significant benefits when analyzing very large datasets, enabling faster and more scalable plagiarism detection.

Finally, future work could explore the integration of additional strategies that complement the ones presented here. Combining our current methods with alternative approaches may yield even stronger detection capabilities, broadening the practical applications of the system.

Bibliography

[1] Asim M. El Tahir Ali, Hussam M. Dahwa Abdulla, and Vaclav Snasel. “Overview and Comparison of Plagiarism Detection Tools”. In: *Proceedings of the Dateso 2011: Annual International Workshop on Databases, Texts, Specifications and Objects*. To read the full-text of this research, you can request a copy directly from the authors. Pisek, Czech Republic: VSB - Technical University of Ostrava, Apr. 2011.

[2] França B. Allyson et al. “Sherlock N-overlap: Invasive Normalization and Overlap Coefficient for the Similarity Analysis Between Source Code”. In: *IEEE Transactions on Computers* 68.5 (2019), pp. 740–751. DOI: 10.1109/TC.2018.2881449.

[3] A Anacleto, Teade Punter, and C Gresse von Wangenheim. “GQM-Handbook and Overview of GQM-plans”. In: *IESE Report* 8 (2003).

[4] J.A.W. Faidhi and S.K. Robinson. “An empirical approach for detecting program similarity and plagiarism within a university programming environment”. In: *Computers & Education* 11.1 (1987), pp. 11–19. ISSN: 0360-1315. DOI: https://doi.org/10.1016/0360-1315(87)90042-X. URL: https://www.sciencedirect.com/science/article/pii/036013158790042X.

[5] Elisabeth Hermann. *Reproduction Package for "Frequency-based Highlight Extraction in Software Plagiarism Detection"*. Version version 1. Zenodo, Aug. 2025. DOI: 10.5281/zenodo.17092494. URL: https://doi.org/10.5281/zenodo.17092494.

[6] Georgina Cosma; Mike Joy. “Towards a Definition of Source-Code Plagiarism”. In: *IEEE Transactions on Education* 51.2 (May 2008), pp. 195–200. ISSN: 0018-9359. DOI: 10.1109/TE.2007.906776. URL: https://doi.org/10.1109/TE.2007.906776.

[7] Mike Joy and Michael Luck. “Plagiarism in Programming Assignments”. In: *IEEE Transactions on Education* 42.2 (May 1999), p. 129.

[8] Oscar Karnalim. “TF-IDF-Inspired Detection for Cross-Language Source Code Plagiarism and Collusion”. In: *Computer Science* 21.1 (2020), pp. 47–65. DOI: 10.7494/csci.2020.21.1.3389. URL: https://doi.org/10.7494/csci.2020.21.1.3389.

[9] Oscar Karnalim, Simon, and William Chivers. “Preprocessing for Source Code Similarity Detection in Introductory Programming”. In: *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. Koli Calling ’20. Koli, Finland: Association for Computing Machinery, 2020. ISBN: 9781450389211. DOI: 10.1145/3428029.3428065. URL: https://doi.org/10.1145/3428029.3428065.

[10] Tomáš Kučečka. “Obfuscating plagiarism detection: vulnerabilities and solutions”. In: *CompSysTech ’11: Proceedings of the 12th International Conference on Computer Systems and Technologies*. ACM, 2011, pp. 423–428. DOI: 10.1145/2023607.2023678. URL: https://doi.org/10.1145/2023607.2023678.

[11] Cynthia Kustanto and Inggriani Liem. "Automatic Source Code Plagiarism Detection". In: *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*. 2009, pp. 481–486. DOI: 10.1109/SNPD.2009.62.

[12] Robin Maisch, Nathan Hagel, and Alexander Bartel. "Towards Robust Plagiarism Detection in Programming Education: Introducing Tolerant Token Matching Techniques to Counter Novel Obfuscation Methods". In: *Proceedings of the 6th European Conference on Software Engineering Education*. ECSEE '25. Association for Computing Machinery, 2025, pp. 11–19. ISBN: 9798400712821. DOI: 10.1145/3723010.3723019. URL: https://doi.org/10.1145/3723010.3723019.

[13] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. *PROGpedia*. Version 1.0.1. Zenodo, Dec. 2022. DOI: 10.5281/zenodo.7449056. URL: https://doi.org/10.5281/zenodo.7449056.

[14] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. "Finding Plagiarisms among a Set of Programs with JPlag". In: *Journal of Universal Computer Science* 8.11 (2002). Submitted: 30/3/00, Accepted: 17/4/02, Appeared: 28/11/02, pp. 1016–1038. URL: https://www.jucs.org/jucs_8_11/finding_plagiarisms_among_a.

[15] Timur Sağlam et al. "Detecting Automatic Software Plagiarism via Token Sequence Normalization". In: *Proceedings of the 46th International Conference on Software Engineering, ICSE '24, Lissabon, April 14-20*. 46th International Conference on Software Engineering. ICSE 2024 (Lissabon, Portugal, Apr. 14–20, 2024). 46.23.03; LK 01. Institute of Electrical and Electronics Engineers (IEEE), 2024, pp. 1–13. ISBN: 979-8-4007-0217-4. DOI: 10.1145/3597503.3639192.

[16] Saul David Schleimer, Daniel S. Wilkerson, and Alex Aiken. "Winnowing: Local Algorithms for Document Fingerprinting". In: *ACM Transactions on Information and System Security* (2003). DOI: 10.1145/872757.872770. URL: https://doi.org/10.1145/872757.872770.