Olympiads in Informatics, 2022, Vol. 16, 89–106
© 2022 *IOI, Vilnius University*
DOI: 10.15388/ioi.2022.08

Detecting Plagiarism as Out-of-distribution Samples for Large-scale Programming Contests

Runfan WU, Aohui LV, Qiyang ZHAO
SKLSDE and SCSE, Beihang University
e-mail: {alralr, luaohui, zhaoqy}@buaa.edu.cn

Abstract. In competitive programming, standard solutions for easy tasks are usually simple and shorter, making submissions more convergent both in idea and texts. The huge difference in submission diversity between easy and hard tasks, brings inescapable challenges to plagiarism judging by means of similarity thresholding. In this paper, by drawing the strong data support from the China National Olympiads in Informatics (NOI), we study the statistical characteristics of submission similarities for tasks of wide range of difficulty degrees. Finally, we propose a new adaptive method to detect submission plagiarism as out-of-distribution samples, together with a large-scale challenge dataset of competitive programming submission plagiarism detection. Our method is shown to be of higher accuracy and robustness, thus feasible and reliable for large-scale competitive programming contests.

Keywords: competitive programming, plagiarism detection, out-of-distribution.

1. Introduction

Programming contests are competitive programming design events, where contestants need to finish source codes fulfilling various resource consumption restrictions, and are expected to make submissions correct with their best effort (Halim *et al.*, 2013). Because of the advantages of objectivity, straightforwardness and relative unbiasedness, programming contests are widely adopted for qualifications and assessments related to computer algorithms.

Cheating is a troublesome issue in competitive contests, including directly plagiarizing the source codes or copying ideas from other contestants. Usually, cheaters are lack of thorough understanding of the plagiarized codes or ideas, making their submissions full of segments which are almost identical to original ones. Therefore, plagiarism detection is a reliable approach to revealing source code cheating. Automatic tools, mainly specifically-designed softwares, are usually adopted in plagiarism detection.

The present situation regarding plagiarism calls for a more accurate, robust, and adaptive plagiarism detection system. Many teachers against source code plagiarism find themselves overwhelmed by the recent surge of the admission counts of computer-related majors, which they barely manage with overreliance on automated plagiarism detection tools (Roberts *et al.*, 2018). For easier tasks with short and less diverse standard solutions, contestants are more inclined to finish similar submissions of identical algorithms and data structures. Most submissions would be highly similar to each other for these tasks, whereas the situations for hard tasks are totally different. This brings a great challenge to traditional plagiarism detection methods based on similarity thresholding (Freire *et al.*, 2007). Furthermore, the open-sourcing of common plagiarism detection tools[1] enables cheaters to crack and evade plagiarism detection.

Cheating codes are usually like stitched monsters -swallowed ideas, code segments migrated from others, exhibiting exceptionally high similarity with original submissions which are plagiarized. Therefore, source code plagiarism detection can be regarded as recognizing out-of-distribution samples, which aligns with the objective of outlier detection (Ruff *et al.*, 2021). We propose to answer current setbacks in plagiarism detection by taking similarity distributions of submissions into account. The main contributions of our research are:

- We propose a robust, accurate, and adaptive source code plagiarism detection method with sufficient accuracy.
- We build a highly automatic plagiarism detection platform for porting related algorithms, supporting batch manual inspection of suspicious code pairs under a predetermined plagiarism filtering ratio.
- We employ a plagiarism detection dataset based on real-world, large-scale data from the Certified Software Professional (CSP) programming contest[2] and including a variety of problem designs and contestant code styles.
- We verify the performance of our method with the OI dataset and discover that our method outperforms conventional plagiarism detection methods.

The structure of this paper is as follows. Section 1 provides an introduction. Section 2 describes the background of this paper and the related works. Section 3 discusses the details and implementation of our method. Section 4 presents the experimental findings. Section 5 concludes this paper and offers outlooks.

2. Backgrounds

In a programming contest, tasks are usually designed to be of various difficulty degrees and distinct skill coverages, thus to examine contestants comprehensively. On the other hand, contestants might be much different in problem-solving ways and capabilities,

[1] Refer to Table 2.1 for details.

[2] The CSP programming contest, part of the qualification process for the NOI, is regarded as part of the Olympiad in Informatics (OI)in China. An overview of the CSP submission dataset is in Table 4.1.

and have distinct coding styles. It makes the distributions of similarities between pairs of submissions varying dramatically across tasks and contestant groups. It is extremely hard to designate a global threshold for all situations in conventional plagiarism detection methods. Implementing an adaptive plagiarism detection method could potentially alleviate the issue of varying distribution, thus significantly increase the reliability and efficiency of plagiarism detection.

With the trends of open-sourcing, most existing plagiarism detection tools have either released the original source, or been re-implemented by third parties. Table 2.1 summarizes the situation. Since those plagiarism detection tools are easy to access, cheaters are able to develop cheating skills in a trial-and-error mode to evade plagiarism detection. On the contra try, when detecting plagiarism as out-of-distribution samples, it is dependent on all submissions which are untouchable for cheaters during contests, thus cheaters cannot crack the detection scheme easily as before.

There are mainly two streams of plagiarism detection methods: intrinsic detection and extrinsic detection (Foltỳnek *et al.*, 2019).

2.1. *Intrinsic Plagiarism Detection*

Intrinsic detection links source codes with their authorships, capturing the lack of stylistic distinction stemming from plagiarism. There are two approaches to intrinsic detection with an identical final step (Bandara and Wijayarathna, 2011). One is to straightforwardly decide the author of every source code by the maximum likelihood principle. The other is to partition the approximated distribution of source code features. The common final step is to search for the unmatchedness of the predicted and claimed authorships.

Intrinsic detection has a solid foundation on probability theory and decent interpretability. However, it is not practical in programming contests, where an extreme number of contestants each submit few source codes. The first is prone to the confusion of authors, while the second requires an impractical granularity of partition.

Table 2.1

Source code availability of common plagiarism detection systems

System	Implementation	Source code URL
MOSS (Schleimer *et al.*, 2003)	Third-Party	`https://github.com/agranya99/MOSS-winnowing-seqMatcher`
SIM (Gitchell and Tran, 1999)	Official	`https://dickgrune.com/Programs/similarity tester/`
YAP (Wise, 1996)	Third-Party	`https://github.com/zymk9/YAPDS`
JPlag (Prechelt *et al.*, 2000)	Official	`https://github.com/jplag/jplag`

2.2. *Extrinsic Plagiarism Detection*

Extrinsic detection mainly focuses the relation between one source code or source code pair to the others, instead of the relation between source codes and their authorships. It is more frequently used in programming contests. Different types of extrinsic detection can be characterized by the feature extraction method.

Text comparison-based methods. The main aim is to detect repeating character sequences or any of the derived features, which is complexified text comparison. A relatively representative method is local finger printing algorithms (LFA), which are employed by MOSS and JPlag (Schleimer *et al.*, 2003; Prechelt *et al.*, 2000). Typically an LFA extracts the positionally independent features of every window in the original strings, which partly provides robustness against code fragment repositioning. These methods make a hasty assumption that all edits do not change local features radically, but it is not always true across all scenarios.

Classification-Based methods. Given a source code pair, the state of plagiarism could be codified as two classes. Adding intermediate classes tends to ease the classification of borderline samples. Viewing source codes as character sequences, Arwin and Tahaghoghi (2006) propose using general classifiers for the problem after extracting source code features with a recurrent neural network (RNN). These methods oversimplify group wise relations into pairwise labels, thus are unable to deal with group plagiarisms without modifications.

Outlier detection-based methods. Outlier detection is the process of determining samples distant from its distribution. Such methods assume in-distribution source code pairs as innocent and out-of-distribution ones as suspicious and necessary for further investigation. There are five general steps of source code plagiarism detection based on outlier detection (Foltỳnek *et al.*, 2019). Fig. 2.1 illustrates the definition of out-of-distribution samples[3]. This type of methods are outlined as follows.

2.2.1. *Outline of Outlier Detection-Based Methods*

Data preprocessing. Preprocessing is likely needed before the main steps to remove the portions of source codes that is relatively irrelevant to plagiarism detection. Kamalim and Chivers (2020) acknowledge the need of tokenization for reducing factors not determinative of semantics. Wise (1996) proposes lowercasing all tokens relatively early. Đurić and Gašević (2013) regards high frequency tokens removable, for they barely contribute to source code distinguishability despite providing syntactic conformance.

Feature extraction. There are two major basic ideas for feature extraction. One is to calculate the distance of the feature vectors for every source code. Yasaswi *et al.* (2017) uses RNN, viewing source codes as character sequences. Freire *et al.* (2007) consider

[3] The blue and red points indicates respectively the ordinary samples and outliers, and the circles denote group boundaries.

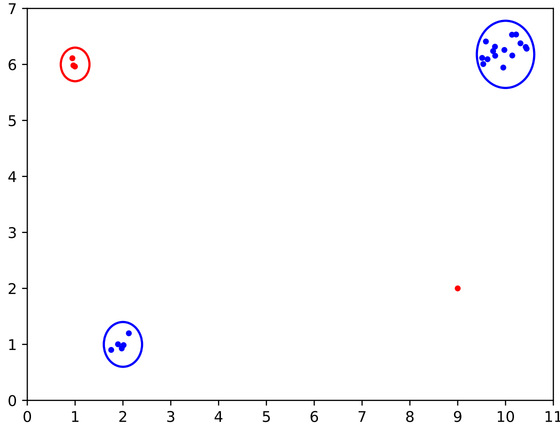

Fig. 2.1. Illustration of out-of-distribution samples and sample groups.

source codes as token streams, extracting the frequencies of each token type as the features. The intermediate representation (IR) (Rabbani and Karnalim, 2017) or the generated machine code (Arwin and Tahaghoghi, 2006) of the source codes could also be treated as regular ones.

The other is to calculate a similarity metric for each source code pair after extracting pairwise features. Freire *et al.* suggest to measure the growth rate of the informational entropy when the two source codes in question are concatenated (Freire *et al.*, 2007). In the perspective of string editing, local repetitive substrings (Karp and Rabin, 1987) and local positional independent features (Prechelt *et al.*, 2000; Schleimer *et al.*, 2003) can also derive the pairwise similarity metric.

Distribution approximation. A similarity matrix could be constructed by the similarity values of each source code pair. Considering each column of the similarity matrix as feature vectors, we can detect the outliers by the approximation of the feature distribution, using support vector machines (SVM) (Suthaharan, 2016) or sparse auto encoders (Ng *et al.*, 2011).

The similarity matrix can also be regarded as an adjacency matrix of a graph, on which an implicit distribution approximation might be performed using graph algorithms to find abnormal nodes or edges. For example, graph embedding is able to transform graph nodes into their vector representations. Frequently used graph embedding algorithms include multidimensional scaling (MDS) (Cox and Cox, 2008), Node2Vec (Grover and Leskovec, 2016), structural deep network embedding (SDNE) (Wang *et al.*, 2016), etc. Fig. 2.2 depicts feature extraction and distribution approximation in conjunction.

Suspicious code pair filtering. For the convenient and easily interpretable quantification of the possibility of plagiarism, many existing methods employ a single suspiciousness index for each source code pair (Devore-McDonald and Berger, 2020; Freire *et al.*, 2007; Ajmal *et al.*, 2013; Yasaswi *et al.*, 2017; Sulistiani and Karnalim, 2019;

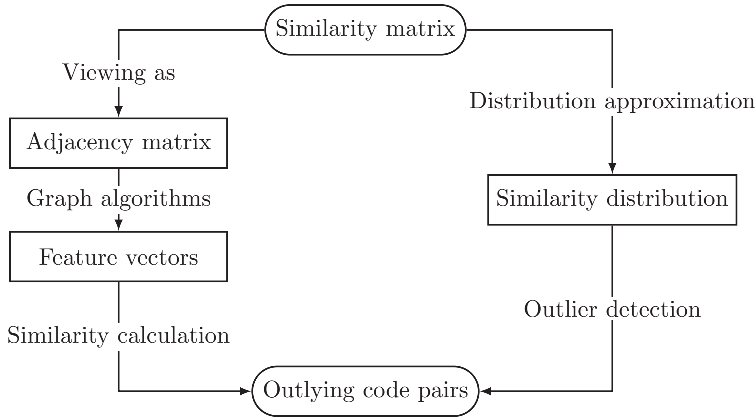

Fig. 2.2. Feature extraction and distribution approximation.

Jiffriya *et al.*, 2014). With the aid of the suspicious index, human operators are able to filter the most relevant pairs for manual inspection (Devore-McDonald and Berger, 2020; Freire *et al.*, 2007). The suspiciousness indices are typically computed from the feature vectors of the individual source codes using vector similarity functions, such as the Euclidean distance (Ajmal *et al.*, 2013; Yasaswi *et al.*, 2017) and cosine similarity (Rahutomo *et al.*, 2012; Sulistiani and Karnalim, 2019; Jiffriya *et al.*, 2014).

Checking and evaluation. Manual inspection results are generally regarded as the reference for determining the status of plagiarism. The commonly used method is to inspect the source code pairs whose ranks of the suspiciousness index are within a previously chosen ratio, then calculate the precision, recall, and F1 score using both the predictions and the manual inspection results (Yasaswi *et al.*, 2017; Flores *et al.*, 2014; Lee *et al.*, 2012).

3. Our Method

3.1. *Data Cleaning*

Data cleaning is the removal of the factors with little relevancy to plagiarism from the submissions. In our method, there are four data cleaning steps executed in succession.

Deletion of irregular submissions. Irregular submissions are those with excessive line count or line width, which typically contains a nonsensical paragraph repeated verbatim innumerably. Fig. 3.1 exhibits an example. The efficiency of plagiarism detection system will be critically impared unless those submissions are removed.

Deletion of submissions with insufficient line counts. Typically those submissions is either a framework or a program that could handle only the cases dispensed with

```
1     #include <stdio.h>
2
3     int main(int argc, char **argv) {
...       /* parts related to problem solving */
49
50        printf("hello␣world\n");
51        printf("hello␣world\n");
...       /* repetitive content */
10000     printf("hello␣world\n");
10001
10002     return 0;
10003 }
```

Fig. 3.1. An example of irregular submissions.

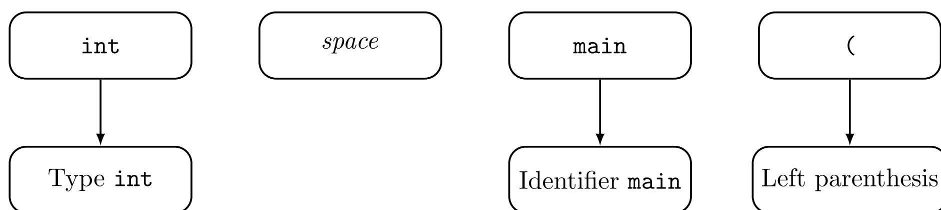

Fig. 3.2. Illustration of tokenization.

the original program description, indicating that the contestant was unable to discover a valid idea. The submissions of both types are extremely inadequate, rendering plagiarism detection on them unnecessary. We also include empty submissions in this step.

Tokenization. Tokenization is to transform the source code from its string form to the token stream form according to the programming language syntax, removing the influences of the factors less relevant to plagiarism. The token stream is the input of the next step, with additional properties preserved, such as the type and function names. Fig. 3.2 illustrates the current tokenization process.

3.2. *Feature Extraction*

We use greedy string tiling (GST) algorithm, which is a widely used algorithm for string similarity (Prechelt *et al.*, 2000). The steps of the GST algorithm are in Fig. 3.3. An optimization of this algorithm is to precalculate the rolling hashes (Karp and Rabin, 1987) of every window of length M on both strings.

Algorithm 1: The GST algorithm

Input: Strings $a = a_1a_2\cdots a_m, b = b_1b_2\cdots b_n$, minimal length of valid common substrings M
Output: Similarity s

```
tiles, a', b' ← {}, 00···0 (×m), 00···0 (×n);
do
    maxmatch, matches ← M, {};
    for 1 ≤ i ≤ m do
        if a'_i = 0 then
            continue;
        end
        for 1 ≤ j ≤ n do
            if a'_j = 0 then
                continue;
            end
            k ← 0;
            while a_{i+k} = b_{j+k} and a'_{i+k} = b'_{j+k} = 0 do
                k ← k + 1;
            end
            if k = maxmatch then
                matches ← matches ∪ {(i, j, k)};
            end
            else if k > maxmatch then
                maxmatch, matches ← k, matches ∪ {(i, j, k)};
            end
        end
    end
    for (i, j, k) ∈ matches do
        for 0 ≤ k ≤ maxmatch − 1 do
            a'_{i+k}, b'_{j+k} ← 1, 1;
        end
    end
    tiles ← tiles ∪ matches;
while maxmatch > M;
s ← 0;
for tile ∈ tiles do
    s ← s + |tile|;
end
s ← 2s/(m+n);
return s;
```

Fig. 3.3. Algorithmic steps for the GST algorithm.

3.3. *Approximation of Distribution*

We choose graph embedding as the approach for this step, extracting a feature vector for every valid source code. There are two graph embedding algorithms to be used.

Multidimensional scaling (MDS). MDS (Cox and Cox, 2008) is a dimensionality reduction method. It calculates the dimensionally reduced feature matrix $\mathbf{X}'$ from the symmetric pairwise distance matrix $\mathbf{D}$, which is derived from the unknown feature matrix $\mathbf{X}$. If we regard $\mathbf{D}$ as a weighted adjacency matrix, $\mathbf{X}'$ can be viewed as the embedding of the corresponding graph, effectively converting it to a graph embedding algorithm.

Suppose the dimensionalities of the original and dimensionally reduced feature vectors are m and k ($1 \leq k < m$) respectively. We denote $\mathbf{X}$ and $\mathbf{X}'$ as:

$$\begin{aligned} \mathbf{X} &\stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{pmatrix} \\ \mathbf{X}' &\stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{x}'_1 & \boldsymbol{x}'_2 & \cdots & \boldsymbol{x}'_n \end{pmatrix} \end{aligned} \tag{3.1}$$

The exact steps of metric MDS are below.

We define $\boldsymbol{e} \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$. Consider the $n \times n$ centering matrix

$$\mathbf{H} \stackrel{\text{def}}{=} \mathbf{I} - \frac{1}{n}\boldsymbol{e}^{\mathrm{T}}\boldsymbol{e} \tag{3.2}$$

For any $n \times n$ matrix $\mathbf{A}$, it is easy to prove that being left or right multiplied by $\mathbf{H}$ is equivalent to subtracting from each row or column of $\mathbf{A}$ its average respectively. We define the $n \times n$ matrix

$$\mathbf{Z} \stackrel{\text{def}}{=} \begin{pmatrix} \|\boldsymbol{x}_1\|^2 & \|\boldsymbol{x}_2\|^2 & \cdots & \|\boldsymbol{x}_n\|^2 \end{pmatrix}^{\mathrm{T}} \boldsymbol{e} \tag{3.3}$$

Where $\|\cdot\|^2$ is the length of a vector. From (3.3) we can deduce the matrix algebraic definition of $\mathbf{D}$:

$$\mathbf{D} \stackrel{\text{def}}{=} \mathbf{Z} - 2\mathbf{X}^{\mathrm{T}}\mathbf{X} + \mathbf{Z}^{\mathrm{T}} \tag{3.4}$$

According to (3.3), (3.4) and the centering property and symmetry of $\mathbf{H}$, we define

$$\mathbf{B} \stackrel{\text{def}}{=} -\frac{1}{2}\mathbf{HDH} = (\mathbf{XH})^{\mathrm{T}}(\mathbf{XH}) \tag{3.5}$$

It is apparent that $\mathbf{B}$ is an inner product matrix. Note that $\mathbf{B}$ is thus positive semidefinite. Therefore, we could obtain another form of $\mathbf{B}$ by singular value decomposition (SVD) and then the closed form of $\mathbf{X}$:

$$\begin{aligned}\mathbf{B} &= \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}}\\ \mathbf{X} &= \mathbf{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\end{aligned} \tag{3.6}$$

Where $\mathbf{U}$ is a $n \times n$ orthogonal matrix, $\boldsymbol{\Sigma}$ is a diagonal matrix containing all singular values of $\mathbf{B}$, and $\boldsymbol{\Sigma}^{\frac{1}{2}}$ is the elementwise square root of $\boldsymbol{\Sigma}$. The ultimate goal of MDS is to reduce the dimensionality of the feature vectors from m to k while maximally preserving the pairwise distances, which could be represented as the following optimization problem:

$$\begin{cases}\arg\min_{\operatorname{rank}\mathbf{X}'\leq k} \left\|\mathbf{B} - (\mathbf{X}')^{\mathrm{T}}\mathbf{X}'\right\|_{\mathrm{F}}^{2}\\ \sum_{i=1}^{n}\boldsymbol{x}_i = \mathbf{0}\end{cases} \tag{3.7}$$

Where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. According to the low-rank approximation property of SVD, a closed solution of $\mathbf{X}'$ is

$$\mathbf{X}' = \mathbf{V}_k\boldsymbol{\Sigma}_k^{\frac{1}{2}} \tag{3.8}$$

Where $\mathbf{V}_k$ is the first k rows of $\mathbf{V}$ and $\boldsymbol{\Sigma}_k^{\frac{1}{2}}$ is a $k \times k$ diagonal matrix formed by the largest k diagonal elements of $\boldsymbol{\Sigma}$.

Nonmetric MDS does not employ an Euclidean distance matrix, necessitating a solution based on optimization algorithms. The ultimate goal is fundamentally identical to metric MDS, and the problem can be generally framed as the following optimization problem:

$$\begin{cases}\arg\min_{\mathbf{X}'} \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 - d_{ij}\right)\\ \sum_{i=1}^{n}\boldsymbol{x}_i = \mathbf{0}\end{cases} \tag{3.9}$$

An analytical solution typically does not exist, thus requiring numeric calculation. Generally the distance calculation on the solution is Euclidean and the distance matrix $\mathbf{D}$ is still nonnegative, symmetric, and has zeroes as its diagonal elements.

AttentionWalk. AttentionWalk (Abu-El-Haija *et al.*, 2018) is a fast graph embedding algorithm based on random walks, introducing adaptability on the parameters such as random walk step count.

Given a simple graph of n nodes whose adjacency matrix is $\mathbf{A}$. The node are numbered from 1 to n. The edge weights are in the interval [0,1]. The maximal step count is C. We intend to embed the nodes into an m-dimensional vector space and m is even. The embedding is denoted as $\mathbf{Y} \stackrel{\text{def}}{=} \begin{pmatrix}\mathbf{L} & \mathbf{R}^{\mathrm{T}}\end{pmatrix}$ where $\mathbf{L}$ and $\mathbf{R}$ are $n \times \frac{m}{2}$ and $\frac{m}{2} \times n$ matrices respectively. We define the reconstructed adjacency matrix $\hat{\mathbf{A}} \stackrel{\text{def}}{=} \mathbf{L}\mathbf{R}$.

An initial configuration $\mathbf{S}$ is provided as an $n \times n$ diagonal matrix, where each diagonal value is the number of random walks starting from the correspondingly num-

bered node. As an algorithm based on random walks, we attempt to discover the expectation matrix $\mathbf{E}$ where Eij equals to the expected count of the random walks from node i to node j. Considering a single step from node u,

$$\mathcal{P}\left(\text{choose node } v \mid \text{currently at node } u\right) = \frac{A_{uv}}{\sum_{w=1}^{n} A_{uw}} \tag{3.10}$$

Random walks are Markov processes. Therefore, according to (3.10), after row normalization $\mathbf{A}$ is converted to a transition matrix $\mathbf{T}$ covering one random walk step. Using the Markov property, we know $\left(\mathbf{T}^k\right)ij$ $(1 \leq k \leq C)$ is the probability of a certain walk being a k-step one from node i to node j.

We define the probability vector $\boldsymbol{q} \overset{\text{def}}{=} \begin{pmatrix} q_1 & q_2 & \cdots & q_C \end{pmatrix}$, where qk represents the probability of a certain walk to have k steps. According to Bayes theorem, the closed form of the probability matrix $\mathbf{P}$ where Pij is the probability of a certain walk being from node i to node j is:

$$\mathbf{P} = \sum_{k=1}^{C} q_k \mathbf{T}^k \tag{3.11}$$

Then it is easy to deduce the closed form of $\mathbf{E}$ with (3.11):

$$\mathbf{E} = \mathbf{SP} = \mathbf{S} \sum_{k=1}^{C} q_k \mathbf{T}^k \tag{3.12}$$

The loss function is similar to cross-entropy loss, respectively treating $\mathbf{E}$ and $\mathbf{I}$ $[\mathbf{A} = 0]$ similarly to the positive and negative labels ina binary classification problem. The final optimization problem is

$$\underset{\mathbf{L},\mathbf{R},\boldsymbol{q}'}{\arg\min} \left\{ -\left\| \mathbf{E} \circ \log \sigma\left(\hat{\mathbf{A}}\right) + \mathrm{I}\left[\mathbf{A} = \mathbf{0}\right] \circ \log\left[1 - \sigma\left(\hat{\mathbf{A}}\right)\right] \right\|_1 + \beta \left\|\boldsymbol{q}'\right\|^2 + \gamma \left\|\hat{\mathbf{A}}\right\|_{\mathrm{F}}^2 \right\} \tag{3.13}$$

Where the probability vector $\boldsymbol{q}$ in (3.12) needed for the closed form of $\mathbf{E}$ is calculated by the parameter $\boldsymbol{q}'$ using the softmax function. $\sigma(\cdot)$ and $\circ$ represent the sigmoid activation function and the elementwise product respectively. $\mathbf{I}\,[\cdot]$ is the Iverson notation applied elementwise, yielding 1 when the condition in the given position holds and 0 otherwise. $\|\cdot\|_1$ denotes the 1-norm of a matrix, *i.e.* the mean of the absolute values of all elements. β and γ control the regularization strengths of $\boldsymbol{q}$ and $\hat{\mathbf{A}}$ respectively.

According to (3.12) and (3.13), the closed and differentiable form of the loss function exists. Asa result, the embeddings can be obtained by solving the optimization problem described in (3.13), typically using gradient descent-based methods.

3.4. *Suspicious Pair Filtering*

The suspicious indices for each source code pair can be calculated by the pairwise similarity values of the feature vectors obtained in the previous step. Given two feature vectors $\boldsymbol{x}_1 = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1k} \end{pmatrix}$ and $\boldsymbol{x}_2 = \begin{pmatrix} x_{21} & x_{22} & \cdots & x_{2k} \end{pmatrix}$, their Euclidean distance is

$$\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \tag{3.14}$$

While their cosine similarity is

$$\cos\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \frac{\boldsymbol{x}_1 \cdot \boldsymbol{x}_2}{\|\boldsymbol{x}_1\| \|\boldsymbol{x}_2\|} \tag{3.15}$$

Where $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$ denotes the angle $\leq \frac{\pi}{2}$ between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Then we filter the source code pairs for the suspicious ones within a certain rank of similarity for manual inspection.

In practical settings, among the suspicious ones, the pairwise string similarity values obtained in the feature extraction step could be another metric to further filter out the source code pairs unnecessary for manual inspection. We can consider within the suspicious ones that also have a string similarity value above a certain low threshold, because actual plagiarizing source code pairs are hardly free of textual similarity.

3.5. *Evaluation*

There are several challenges of evaluating the results of our method. Plagiarizing source code pairs are generally very rare in a submission group if they do exist, and the groups with plagiarizing pairs also tends to be uncommon. The suspiciousness indices are not portable across different methods, rendering the plagiarism state of a pair binary. Determining plagiarism needs manual inspection besides automatic filtering, while it is difficult to ingrain domain knowledge into computers.

The evaluation metric intends to compare the consistency between manual inspection results and the current ones. Therefore, it should consider the plagiarizing pairs alone as they have far more significance and treat them as equal. It is also supposed to be based on the normalized ranks of the suspiciousness indices for a general performance comparison across groups. Ideally it might favor greatly the smaller ranks and penalizes the larger ranks less severely, because smaller ranks indicate larger consistency with manual results and generally larger ranks could greatly fluctuate.

Based on the above criteria, we propose an evaluation metric named geometric mean of normalized ranks (GMNR), which satisfies the above mentioned properties:

$$\mathrm{GMNR} \stackrel{\mathrm{def}}{=} \sqrt[n]{\prod_{i=1}^{n} \frac{r_i}{N}} \tag{3.16}$$

Where N and n are the total and suspicious pair count in a particular group respectively. We consider only the pairs formed by two different source codes that are not removed after data cleaning, and treat the elements of the pairs as exchangeable. r_i $(1 \leq i \leq n)$ are the 1-based ranks of the suspiciousness indices of each plagiarizing pair. It is apparent that a value of GMNR is on (0,1] and a smaller GMNR indicates greater overall consistency with manual results.

4. Experiments

4.1. *Environment*

All experiments are conducted on a computer with 32 GB of RAM. The Python version is 3.9.9. We employ a GTX 2080 Ti for the training and evaluation of the models.

4.2. *Dataset*

We test our method on submissions for the second round of the Certified Software Professional programming contest in 2021. The contest is organized by the China Computer Federation (CCF) and has junior and senior groups that assess programming skills of middle and high school students respectively. It is a onsite contest hold distributedly in provinces. There are two rounds typically in early Octobers and early Novembers, and only contestants passing the first round can participate in the second.

The CSP submission dataset consists of submissions from 25 participating provinces in the second round of CSP 2021. Table 4.1 provides an overview of the dataset. Every participant may submit multiple times for each problem, and only the last submission is

Table 4.1
Overview of the CSP submission dataset

Property	Junior group	Senior group
Problems included	`candy, fruit, network, sort`	`airport, bracket, palin, traffic`
Number of contestants	15073	10644
Number of submissions	52147	35582
Programming language allowed	C, C++	C, C++
Total file size	39.2 MB	44.2 MB

Table 4.2
Parameters of the graph embedding algorithms

Algorithm	Parameter	Value
AttentionWalk	Embedding dimension	512
	Training epochs	50000
	Attention vector length	20
	Walk count	80
	L2 regularization strength	0.01
	Learning rate	5e-5
	GMNR calculation interval	1000
MDS	Dimension of new vector space	Largest power of 2 ≤ group count

rated afterwards and given scores. All types of sensitive information involving personal privacy, such as contestant names and schools, are removed.

We use the method described in Section 3, and conduct experiments with both MDS and AttentionWalk as the graph embedding algorithm. We view all submissions from each province and each task as a submission group, for plagiarism across provinces is practically impossible. For every group, we calculate the minimal GMNR during training only if manual inspection had found any plagiarizing source code pairs, as we intend to compare the results of our method with manual inspection results.

The training parameters for the graph embedding algorithms are in Table 4.2. For AttentionWalk, we use the Adam optimizer. We also apply L2 regularization on the adjacency matrix reconstruction $\hat{\mathbf{A}}$. The GMNR is calculated every 1000 epochs to find the minimal one. For MDS, as the dimension of the embedding vectors cannot exceed that of the original features, adopting an embedding dimension of the largest possible power of 2 provides an adequate trade-off between accuracy and practicality.

4.3. *Results on the CSP Submission Dataset*

All results are even rounded and have four significant digits, unless otherwise noted.

4.3.1. *Junior Group*

Table 4.3 and Table 4.4 show the results on the junior group of the CSP submission dataset using MDS and AttentionWalk respectively. Only provinces of plagiarizing code pairs are shown in tables, and empty cells denote submission groups without known plagiarizing code pairs.

From the tables, our method is robust against different difficulty and skill coverage combinations, and accurately identifies plagiarizing code pairs. On the submission groups of `fruit` our method has an overall lower performance, possibly due to the ability of several ready-made approaches to this problem to obtain a nearly full score, which rarely happens to other problems.

Table 4.3
GMNRs using MDS on the junior group, CSP 2021

Province	`candy`	`fruit`	`network`	`sort`
Anhui	5.581e-2	3.822e-1	2.378e-1	3.334e-1
Beijing	1.162e-1	2.537e-2		2.978e-2
Guangdong		3.070e-1		
Guangxi	2.167e-1	3.453e-1		2.345e-2
Hunan	3.496e-1			
Jiangsu		2.366e-1		
Sichuan		1.515e-1		
Shandong	3.265e-3	3.190e-1		2.737e-1
Shanghai	1.624e-1			
Shannxi		4.081e-2		
Shanxi	1.589e-1			1.046e-2
Tianjin	3.676e-1	1.975e-1		2.412e-1
Xinjiang				3.928e-1
Yunnan				4.452e-1
Zhejiang	3.128e-1			

Table 4.4
GMNRs using AttentionWalk on the junior group, CSP 2021

Province	`candy`	`fruit`	`network`	`sort`
Anhui	4.733e-7	5.960e-4	2.394e-4	1.571e-5
Beijing	3.145e-5	7.556e-4		2.011e-5
Guangdong		6.687e-5		
Guangxi	6.656e-6	1.308e-5		1.253e-4
Hunan	1.629e-6			
Jiangsu		3.040e-6		
Sichuan		8.215e-5		
Shandong	6.488e-7	1.387e-4		1.355e-5
Shanghai	5.348e-5			
Shannxi		3.564e-5		
Shanxi	1.919e-5			1.702e-4
Tianjin	7.652e-5	6.724e-4		8.290e-4
Xinjiang				1.057e-3
Yunnan				8.340e-5
Zhejiang	6.567e-6			

4.3.2. *Senior Group*

The results of our algorithm are in Table 4.5 and Table 4.6, with MDS and AttentionWalk respectively. Only provinces of plagiarizing code pairs are shown in tables. Empty cells mean no plagiarism detected in these groups, the same as in Section 4.3.1.

Our method also performs accurately with robustness across all submission groups.

Table 4.5
GMNRs using MDS on the senior group, CSP 2021

Province	`airport`	`bracket`	`palin`	`traffic`
Chongqing			4.794e-1	
Hubei		1.132e-1		
Jiangsu	5.286e-1			
Jiangxi			1.521e-1	
Sichuan		2.934e-1		
Tianjin		6.330e-2		
Zhejiang	4.120e-1	3.298e-1	4.333e-1	

Table 4.6
GMNRs using AttentionWalk on the senior group, CSP 2021

Province	`airport`	`bracket`	`palin`	`traffic`
Chongqing			5.018e-5	
Hubei		4.808e-4		
Jiangsu	8.037e-4			
Jiangxi			1.277e-4	
Sichuan		1.508e-5		
Tianjin		4.267e-4		
Zhejiang	4.349e-6	2.858e-6	3.993e-3	

4.4. *Results against Mossad*

We test our method against the Mossad approach[4] to plagiarism detection evasion (Devore-McDonald and Berger, 2020). After applying our method, the rank of the pair of the original and the mutation are consistently below 10 in the 5 groups tested. To the best of our knowledge, our method is the first practical countermeasure against Mossad.

5. Conclusions

We propose an adaptive source code detection method offering robustness and accuracy comparable to conventional methods. We eliminate thresholds in the core parts of our method, easing manual inspection while enhancing adaptability. Real-World tests on the OI dataset indicate the its practicality when faced with the challenges of the varied submission groups and similarity distributions. Almost all known plagiarizing code pairs

[4] Mossad mutates the original submission by inserting repetitive statements and uses `gcc -O3` to determine the semantic equivalence. We insert pre-existing lines instead, as C++ parsing is complex, and choose the first generated mutation with a similarity value by the GST algorithm below 0.4.

have low ranks of suspiciousness index regardless of whether they are syntactically or semantically similar.

Plagiarism detection based on graph embedding can serve as an overlay upon traditional methods, facilitating the transition to adaptive, grey-box algorithms. However, graph embedding lacks sufficient capture of the high-level semantics of the source codes as well as other nuances. More advanced graph algorithms, such as graph neural networks (GNN) might be researched and employed to alleviate this problem.

6. Acknowledgement

This work is supported by State Key Laboratory of Software Development Environment, Beihang University under Grant No. SKLSDE-2022ZX-09.

References

Abu-El-Haija, S., Perozzi, B., Al-Rfou, R.,Alemi, A.A. (2018). Watch your step: Learning node embeddings via graph attention. *Advances in Neural Information Processing Systems*, 31.

Ajmal, O., Missen, M.S., Hashmat, T., Moosa, M., Ali, T. (2013). Eplag: A two layer source code plagiarism detection system. In: *Eighth International Conference on Digital Information Management (ICDIM 2013)* (pp. 256–261).

Arwin, C., Tahaghoghi, S.M. (2006). Plagiarism detection across programming languages. In: *Proceedings of the 29th Australasian Computer Science Conference-Volume 48* (pp. 277–286).

Bandara, U., Wijayarathna, G. (2011).A machine learning based tool for source code plagiarism detection. *International Journal of Machine Learning and Computing* , 1(4), 337.

Cox, M.A., Cox, T.F. (2008). Multidimensional scaling. In: *Handbook of Data Visualization*. Springer, pp. 315–347.

Devore-McDonald, B., Berger, E.D. (2020). Mossad: Defeating software plagiarism detection. *Proceedings of the ACM on Programming Languages*, *4*(OOPSLA), 1–28.

Flores, E., Rosso,P., Moreno, L., Villatoro-Tello, E. (2014). On the detection of source code re-use. In: *Proceedings of the Forum for Information Retrieval Evaluation* (pp. 21–30).

Foltýnek, T., Meuschke, N., Gipp, B. (2019). Academic plagiarism detection: a systematic literature review. *ACM Computing Surveys (CSUR)*, 52(6), 1–42.

Freire, M., Cebrían, M., Del Rosal, E. (2007). Ac: An integrated source code plagiarism detection environment. *arXiv preprint cs.IT/0703136* .

Gitchell, D., Tran, N. (1999). Sim: a utility for detecting similarity in computer programs. *ACM Sigcse Bulletin*, 31(1), 266–270.

Grover, A., Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 855–864).

Halim, S., Halim, F., Skiena, S.S., Revilla, M.A. (2013). *Competitive Programming 3*. Citeseer.

Jiffriya, M., Jahan, M.A., Ragel, R.G. (2014). Plagiarism detection on electronic text based assignments using vector space model. In: *7th International Conference on Information and Automation for Sustainability* (pp. 1–5).

Karnalim, O., Chivers, W. (2020). Preprocessing for source code similarity detection in introductory programming. In: *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (pp. 1–10).

Karp, R. M., Rabin, M.O. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development* , 31(2), 249–260.

Lee, Y.-J., Lim, J.-S., Ji, J.-H., Cho, H.-G., Woo, G. (2012). Plagiarism detection among source codes using adaptive methods. *KSII Transactions on Internet and Information Systems (TIIS)*, 6(6), 1627–1648.

Ng, A., *et al.* (2011). Sparse autoencoder. *CS294A Lecture notes* , 72(2011), 1–19.

Prechelt, L., Malpohl, G., Philippsen, M. (2000). *Jplag: Finding Plagiarisms among a Set of Programs*. Citeseer.
Rabbani, F.S., Karnalim, O. (2017). Detecting source code plagiarism on. net programming languages using low-level representation and adaptive local alignment. *Journal of Information and Organizational Sciences*, (1), 105–123.
Rahutomo, F., Kitasuka, T., Aritsugi, M. (2012). Semantic cosine similarity. In: *The 7th International Student Conference on Advanced Science and Technology Icast* (Vol. 4, p. 1).
Roberts, E., Camp, T., Culler, D., Isbell, C., Tims, J. (2018). Rising cs enrollments: Meeting the challenges. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 539–540).
Ruff, L., Kauffmann, J.R., Vandermeulen, R.A., Montavon, G., Samek, W., Kloft, M., . . . Müller, K.-R. (2021). A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*.
Schleimer, S., Wilkerson, D. S., Aiken, A. (2003). Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (pp. 76–85).
Sulistiani, L., Karnalim, O. (2019). Es-plag: Efficient and sensitive source code plagiarism detection tool for academic environment. *Computer Applications in Engineering Education* , 27(1), 166–182.
Suthaharan, S. (2016). Support vector machine. In: *Machine learning models and algorithms for big data classification* (pp. 207–235). Springer.
Đurić, Z., Gašević, D. (2013). A source code similarity system for plagiarism detection. *The Computer Journal*, 56(1), 70–86.
Wang, D., Cui, P., Zhu, W. (2016). Structural deep network embedding. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1225–1234).
Wise, M.J. (1996). Yap3: Improved detection of similarities in computer program and other texts. In: *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education* (pp. 130–134).
Yasaswi, J., Kailash, S., Chilupuri, A., Purini, S., Jawahar, C. (2017). Unsupervised learning based approach for plagiarism detection in programming assignments. In: *Proceedings of the 10th Innovations in Software Engineering Conference* (pp. 117–121).

R. Wu has graduated from Beihang University with a bachelor degree in Computer Science and Engineering, and is currently pursuing his Master degree in Beihang. He is involved in the plagiarism detection and technical supporting of the China National Olympiads in Informatics (NOI). His current research interests include computer vision and discrete mathematics.

A. Lv is involved in software development and technical supporting of the China National Olympiads in Informatics (NOI). He graduated from Taiyuan University of Technology with a Bachelor degree in Mathematics, and is currently pursuing his Master degree in Beihang.

Q. Zhao is currently the vice chairman of the scientific committee of the China National Olympiads in Informatics (NOI). He is a lecturer of computer science in Beihang University, working on computer vision and deep learning.