A TWO-STEP IN-CLASS SOURCE CODE PLAGIARISM DETECTION
METHOD UTILIZING IMPROVED CM ALGORITHM AND SIM

ASAKO OHNO\(^1\) AND HAJIME MURAOT\(^2\)

\(^1\)Department of Life Design
Shijonawate Gakuen Junior College
4-10-25, Hojo, Daito, Osaka 574-0011, Japan
asako.ohno@mulabo.org

\(^2\)Graduate School of Intercultural Studies
Kobe University
1-2-1, Tsurukabuto, Nada 657-8501, Japan
murao@i.cla.kobe-u.ac.jp

Received May 2010; revised October 2010

ABSTRACT. Source code plagiarism is becoming one of the most serious problems in academia. There have been many proposed methods that attempt to detect source code plagiarism in programming classes. Most of them extract algorithmic features from the source code and measure the similarity between them. These methods show high levels of accuracy in evaluation experiments, and however, it is concerning that the similarity detected by the methods might not be caused by plagiarism. As a result, we propose a method called the CM Algorithm, which utilizes a student’s coding style, the way the student writes source code, to check whether the source code submitted by the student was produced by him/her. In this paper, we propose a combined method that measures the similarity between source codes by using SIM \(^7\), one of the well-known in-class source code plagiarism detection systems, and then checks the outputs of SIM against our improved CM Algorithm. The new method is expected to reduce false positives in plagiarism detection systems. This paper also gives a detailed explanation of the improved CM Algorithm, which assumes fluctuations in the source code produced by a student’s coding style.

Keywords: Coding style, In-class source code plagiarism, Coding model, Hidden Markov model

1. Introduction. World-wide Internet access enables students to find many examples of programming source code, and so it is not difficult for these to be a source of plagiarism. Also, since it is becoming more common for students to edit or submit exercises electronically, it has become easier for a student to copy another student’s work and misrepresent it as his/her own work. Recently, a number of methods have been proposed to solve one of the serious and growing problems in academia, namely source code plagiarism. The most popular approach is to measure structural, algorithmic, or metric-based similarities of the source code \([1, 2, 3]\). This type of method has been successfully utilized in industrial fields, such as source code refactoring. Many methods make tokenization and normalization as a preprocessing step to represent the source code by using graphs \([4, 5, 6]\) or token sequences \([7, 8, 9, 10, 11]\), and then make pair-wise comparisons. Other methods generate feature vectors, which are quantitative representations of source code based on such features such as size, complexity, or a number of specific types of tokens contained in the source code \([12, 13]\). In another example, D’Souza \([14]\) developed a tool that finds similar contents that might be plagiarized from the Internet. Source codes produced as exercises in programming classes, hereinafter called in-class source codes, generally have
certain properties, including: 1) they are often too short to extract sufficient algorithmic features, and 2) they are obviously similar to each other, because they are produced for the same purpose. In these cases, it is difficult to distinguish between plagiarism and coincidental similarity [1]. It is also difficult to detect plagiarism if a student asks someone to ghost-write his/her exercise or makes a copy from the Internet, because existing methods of detection require both the original and the copied source code. Thus, existing methods need to include another step to check if detected similarities are caused by plagiarism or by pure coincidence. There are a number of studies that refer to this point. Engels [13] noted that current methods ignore important cues that teachers use when visually scanning two assignments for signs of plagiarism. Ji et al. [15] made an evolutionary analysis of homogeneous source code in order to improve the accuracy and reliability in existing plagiarism detection algorithms. We propose the CM Algorithm [16], which represents superficial features extracted from text data as parameters of HMM-based stochastic models, called coding models. We have applied the CM Algorithm to source code plagiarism detection. Mozgovoy [17] noted that any of the existing methods can be cheated if the algorithm is known by students, and that using a combination of these methods might improve the robustness of plagiarism detection systems. We quantified students’ coding styles that show how a student writes source code, and utilized these features to identify the author of the code. The system that implements the CM Algorithm showed that was able to identify differences between source codes produced by different authors, even if the algorithms in the source codes were quite similar [16]. Originally, we proposed the method as an alternative to existing methods. However, we found that it might be more effective to judge plagiarism by two different measures: the similarity based on algorithmic features, and the similarity based on the author’s coding style. This paper introduces our new, combined method that utilizes the CM Algorithm to check the outputs of SIM: one of the proposed methods that have shown high accuracy when measuring algorithmic similarity between source codes. In this way, we obtained outputs based on a reliable similarity measure, which enabled us to judge if the similarity is caused by a plagiarism or coincidence. In this paper, we provide a step-by-step explanation of our method in Section 2. Section 3 gives an explanation of the improved methodology, and Section 4 reports the results of our evaluation experiments and discusses how to combine our method with an existing method, called SIM. Section 5 concludes the paper.

2. CM Algorithm.

2.1. An overview of CM algorithm. In our method, we extract superficial features from a number of examples of source code produced by the same student, and quantify the features by parameters of a number of stochastic models, called coding models. Detailed definitions including mathematical expressions of the method can be seen in [16]. As shown in Figure 1, we use a number of coding models to represent the coding style of a student (currently 14 models). Each of the coding models represents a part of the student’s coding style as its internal parameters. We input 14 sets of token sub-sequences generated from the student’s source code into the corresponding coding models, and calculate the observation probabilities. For each of the models, we calculate an average of the observation probabilities and treat this as an output of the coding model. Then, we calculate an average of the 14 average output probabilities to obtain the final output of the student’s coding models. This value represents the percentage of the source code that is produced by the student. We use this value to identify the author of the code. The method consists of the following 3 steps: preparation, training and author identification.
2.2. Preparation. In the first step of preparation, we define the coding style. The type of coding style depends on what kind of superficial feature we want to quantify, for example, naming conventions of variables, description conventions of comments, or the probability of inserting a "(single space) after specific symbols, such as "{". Here, we focused on an occurrence of the following six kinds of tokens: "single space" (described as "sp1"), "2-lettered space" ("sp2"), "3-lettered space" ("sp3"), "4-lettered space" ("sp4"), "tab" ("tab") and "linefeed" ("rtn"), which we call individuality tokens, as adjacent tokens for the specific type of tokens named base-point tokens. We defined 14 groups of symbols as observed values for base-point tokens, such as the opening brace ("{" described as "obr"), colon (":" ) or arithmetic operator ("+", "-", etc.). Other tokens are treated as other tokens (variables, etc., "els"). Following the above definitions, we defined a simple coding model, as shown in Figure 2. In this model, the coding style is simply defined as the probability that one of the individual tokens, or nothing, was observed before and after the base-point token "{". To represent one’s coding style, we need to make 13 other coding models for other base-point tokens, such as ",",";", and so on. The coding model is based on the Hidden Markov Model (HMM) [18, 19]. HMM is a stochastic model used for various applications, such as face recognition [20], speech recognition [21], or machine-fault prevention [22]. A model, M, consists of a finite set of unobservable states lined up in order, according to the Markov Process. Each state holds the probability distribution of the occurrence of the symbols that appear in the model’s output, called an observation sequence, O. We train M by inputting sequential data and updating the internal parameters of M: initial probabilities, observation probabilities, and transition probabilities between states. To identify an author, we input O to M and calculate $M(O)$, the probability that $O$ is observed from $M$.

2.3. Training. To enable a set of coding models to learn a student $A_a$’s coding style, we use a number of examples of source code that were definitely produced by $A_a$ as input data. Generally, it is difficult for teachers to collect hundreds of examples of source code as learning data from students. To extract richer information to use as a student’s coding style from a small number of examples of source code, we generate 14 different kinds of coding models that represent different types of information. Therefore, each coding model only represents a part of a student’s coding style. As a preprocessing step, we normalize and tokenize the source code to transform it into a number of token sub-sequences.
apply 14 different kinds of preprocessing to one source code example and obtain 14 sets of token sub-sequences as input data for each of the coding models. Now we have a set of 14 coding models $M$ representing the coding style of $A$.

2.4. An author identification using the coding models. In our method, we define a student $\alpha$ as a source of token sequences.

$$\alpha = \{1 \leq \alpha \leq N_\alpha\} \quad (1 \leq \alpha \leq N_\alpha)$$

where, $N_\alpha$ is the number of students and $1 \leq N_\alpha$. $D_\alpha^\beta$ is the source code produced by $\alpha$. Here, $D_\alpha^\beta$ is defined as a set of token sequences that are constituents of the source code generated from $\alpha$.

$$D_\alpha^\beta = \{1 \leq D_\alpha^\beta\} \quad (1 \leq \beta \leq N_\beta)$$

where, $N_\beta$ is the number of source code examples and $1 \leq N_\beta$. We generate $X_\alpha^\beta$ consisting of $N_\alpha^\beta$ token sub-sequences from $D_\alpha^\beta$.

$$X_\alpha^\beta \in D_\alpha^\beta$$

$$X_\alpha^\beta = \{x_{n,\alpha,\beta}\} \quad (1 \leq n \leq N_\alpha)$$

where, $x_{n,\alpha,\beta}$ is a token sub-sequence that is one of the constituents of $X_\alpha^\beta$. We train $M_\alpha$ as a set of 14 coding models, each of which represents the coding style related to 14 kinds of base-point tokens to represent the coding style of $A_\alpha$ by inputting $X_\alpha^\beta$ and updating the parameters of each of the models.

$$M_\alpha = \{M_1^\alpha, M_2^\alpha, \ldots, M_{14}^\alpha\}$$

Prior to identifying an author, we have a coding model $M_\alpha$ for student $A_\alpha$ that learns the coding style of $A_\alpha$ by inputting a number of source code example that were definitely produced by $A_\alpha$. We determine whether $A_\alpha$ had produced $D_\alpha^\beta$ by calculating $M_\alpha (X_\alpha^\beta)$, which is the average of the likelihood $P(M_\alpha, x_{n,\alpha,\beta})$.

Here, we represent $x_{n,\alpha,\beta}$ as a sequence of observation probabilities $O_{1:T_n^\alpha,\beta}$ according to the following definition:

$$O_{1:T_n^\alpha,\beta} = \{O_t | 1 \leq t \leq T_n^\alpha,\beta, O_t = P(\sigma_{k,t}|S_t = \{s_i\})\}$$
A TWO-STEP IN-CLASS SOURCE CODE PLAGIARISM DETECTION METHOD

Figure 3. The procedure to calculate the final output from the coding models in the improved version of CM Algorithm, which includes fluctuations in one’s description as a part of coding style. The final outputs are used as similarity measures of coding styles.

$M_a (X_{\beta'}^a)$, which is utilized for author identification, is thus calculated by the following equations:

$$M_a (X_{\beta'}^a) = \frac{1}{N_{a^{\beta'}^a}} \sum_{n=1}^{N_{a^{\beta'}^a}} P(M_a, x_n^{a, \beta'})$$

where,

$$P(M_a, x_n^{a, \beta'}) = \sum_{alls} \prod_{i=1}^{T_{n}^{a, \beta'}} \pi_i b_{S_t}(O_t) \prod_{t=2}^{L} a(S_{t-1}, S_t)b_{S_t}(O_t).$$

3. An Improvement of the CM Algorithm; Including Fluctuations in One’s Description as a Part of Coding Style. In our study, we assume that a student has his/her own coding style that is different from someone else’s. We also assume that there is no guarantee that a student always writes sentences in source code in exactly the same way. Therefore, in the CM Algorithm, we represent a student’s coding style by adjusting parameters of the student’s coding models. In this way, we are able to observe token sequences that are generated stochastically from a certain range of generation patterns, can also observe different token sequences from one coding model. In other words, a coding style is a stochastic generation pattern of token sequences. However, in this way, we only specify that a coding style contains fluctuations. We cannot see how much fluctuation there is in a coding style. We consider a fluctuation of description as a part of one’s coding style by individually examining the calculated observation probabilities for each of the token sub-sequences, as shown in Figure 3. Thus, in this paper, we improved a part of CM Algorithm to allow us to quantify the degree of fluctuation. In the previous version of the method, we calculated an average observation probability of a coding model, but in this improved system, we do not. Instead, we divide the range of observation probabilities into 21 classes, calculate the frequencies, and generate feature vectors consisting of the frequency of each of the observation probabilities. There are 14 coding models, and each model outputs 21 dimensional vectors. We connect 14 vectors...
together to form an 84-dimensional vector that is a final output of a set of coding models. We input the source code used as learning data for $M_\alpha$ to $M_\alpha$ again, and calculate the correct set of observation probabilities for $M_\alpha$ to represent the coding style of $A_\alpha$. We determine if the source code belongs to student $A_\alpha$ by using the output probabilities. To do this, we input a final project submitted by student $A_\alpha$ to $M_\alpha$, and compare the project’s observation probabilities with the previously obtained correct observation probabilities. If the distance between the two sets of observation probabilities are close, we assume that the coding style extracted from the final project is similar to that of student $A_\alpha$.

4. Experiments.

4.1. Evaluation experiment of the improved version of the CM algorithm. Here, we evaluate the performance of the improved method. We gave 20 Java exercises to 12 authors, represented as $A_\alpha$ ($1 \leq \alpha \leq 12$) and asked them to produce 20 sets of Java source code according to the instructions in each of the exercises. In this way, we obtained $20 \times 12$ source code examples that contained no plagiarism. We used 19 of the 20 source code examples, represented as $D_\beta$ ($D_1^1, D_2^1, \ldots, D_{19}^1, D_1^2, \ldots, D_{19}^2$), as learning data for the coding models $M_\alpha$.

$$M_\alpha = \{M_i \mid 1 \leq i \leq N_\alpha\} \quad (9)$$

$N_\alpha = 12$ is the number of authors.

$$M_i = \{M_j^i \mid 1 \leq j \leq 14\} \quad (10)$$

where, $M_i$ is a set of 14 coding models representing the coding style of $A_i$. We treat $D_{20}^\alpha$ as the source code submitted by its author as the final project. We train $M_\alpha$ by using $D_{1,19}^\alpha$ ($D_1^1, D_2^2, \ldots, D_{19}^2$) as learning data. We input the learning data to $M_\alpha$ again in order to calculate the observation probabilities, and treat them as the correct outputs of $M_\alpha$ that correctly represent the coding style of $A_\alpha$. Then, we input $D_{20}^\alpha$ to $M_\alpha$, and compare the obtained output to the correct output for $M_\alpha$. We generate a 21-dimensional feature vector from the percentages of observation probabilities at each class, for each of the following source code examples: learning data for $M_{19}$, which are $D_{19}^\alpha$, and the final submitted project $D_{20}^\alpha$. Then, we calculate the Euclidean distance between the vectors of the learning data and each of the other vectors. We calculate distances for each of the 14 coding models and use an average of them as the distance for coding style $A_\alpha$. If the calculated distance is short, we assume that the coding style extracted from the final project is close to the correct coding style of $A_\alpha$, represented by $M_\alpha$.

Table 1 shows the distances between each of the correct coding styles represented by $M_\alpha$ and the coding styles calculated from $D_{20}^\alpha$. To identify an author, we first input a source code, for example, $D_{20}^1$ to $M_1, M_2, \ldots, M_{12}$, as indicated in Table 1. In this way, we compare the coding style of $A_1$, extracted from $D_{20}^1$ to the coding styles of $A_1, \ldots, A_{12}$, represented by $M_1, M_2, \ldots, M_{12}$, respectively. The resulting distances were 0.2390($M_1$), 0.3923($M_2$), 0.2749($M_3$), 0.3325($M_4$), 0.4442($M_5$), 0.2516($M_6$), 0.3506($M_7$), 0.2595($M_8$), 0.2562($M_9$), 0.4064($M_{10}$), 0.3655($M_{11}$) and 0.5074($M_{12}$). The coding model $M_\alpha$ that showed the smallest value represents the closest coding style to that of $A_\alpha$. Thus, we deduce that $D_{20}^1$ was produced by author $A_1$. Even though in some cases the distance between the coding style of $D_{20}^3$ and $M_3$ was not the smallest value, for example, for the author of $D_{20}^3$, in most cases, the coding models for each of the authors did show the smallest values for the final projects produced by the corresponding authors’ coding styles. Therefore, we assumed that the overall results were good enough to say the improved method showed high accuracy when identifying an author.
4.2. A discussion of a two-stepped source code plagiarism detection method using a combination of the CM algorithm and SIM.

4.2.1. Overview. In this paper, we propose a new method for measuring the similarity between source code examples that is specialized for in-class source code plagiarism detection. SIM [7] is one of the popular similarity measuring tools and is used for plagiarism detection in academia. In the method, we present here, we use SIM to measure the similarity between source code examples. We then use the improved CM Algorithm to identify the author of the code and so check whether the similarity is caused by plagiarism. In this way, we obtain a highly accurate measurement of source code similarity, and so prevent false positives that can occur due to the nature of in-class source code.

4.2.2. Procedure. We begin with a data set of a set of source code examples that contain no plagiarism, and then carry out plagiarism detection using the following two steps: (1) measure the similarity between source code examples using SIM, and (2) examine whether the detected similarity is caused by plagiarism by using the tool implementing the CM Algorithm.

To generate a data set, we asked five authors, represented as $A_{\alpha}$ (1 \leq \alpha \leq 5) to produce five sets of Java source code each. After measuring the similarity between the sets of source code by using SIM, we identify the author by using the CM Algorithm to check the SIM outputs. The procedure is almost the same as that of the previous experiment. We use four of the five source code sets, represented as $D_{\alpha}^a; \alpha = 1, 2, 3, 4$ as learning data for the coding models, $M_{\alpha}$. We treat $D_{\alpha}^a; \alpha = 1, 2, 3, 4$ as the ones submitted as the final projects. We have $M_{\alpha}$ learn the coding style of $A_{\alpha}$ by inputting $D_{\alpha}^a$ as learning data. Then, we again input the learning data to $M_{\alpha}$ in order to calculate the observation probabilities that represent the correct coding style for $A_{\alpha}$. We input $D_{\alpha}^a$ to $M_{\alpha}$, and compare the obtained observation probabilities to the correct observation probabilities of the coding style, $A_{\alpha}$. Then, we derive 21-dimensional feature vectors, which consist of the percentages of the observation probabilities in each class, for each of the following source code sets: learning data for $M_{\alpha}$ ($D_{\alpha}^a$) and $D_{\alpha}^a$. Then, we calculate the Euclidean distance between the vectors of the learning data and each of the remaining source code sets. If the calculated distance is short, we assume that the coding style of the final project $D_{\alpha}^a$ is similar to the correct coding style for $A_{\alpha}$, represented by $M_{j}^a$ (1 \leq j \leq 14), meaning the $j$-th coding model of $M_{\alpha}$. We calculate distances for each of the 14 coding models, $M_{j}^a$, and use an average of these distances as the distance from the coding style $A_{\alpha}$. To identify an author, we first input a source code to all the coding models. For example, we
Table 2. Results of comparison between \(D_5^1\), \(D_5^2\), \ldots, \(D_5^5\) and \(D_5^1\), \(D_5^2\), \ldots, \(D_5^5\) calculated by SIM. The similarity values in the table are not symmetrical, since the outputs of the alignment algorithm differ depending on the order of a pair of data.

<table>
<thead>
<tr>
<th></th>
<th>(D_5^1)</th>
<th>(D_5^2)</th>
<th>(D_5^3)</th>
<th>(D_5^4)</th>
<th>(D_5^5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D_5^1)</td>
<td>74.5%</td>
<td>51.5%</td>
<td>52.5%</td>
<td>81.0%</td>
<td></td>
</tr>
<tr>
<td>(D_5^2)</td>
<td>64.0%</td>
<td>53.0%</td>
<td>59.0%</td>
<td>66.0%</td>
<td></td>
</tr>
<tr>
<td>(D_5^3)</td>
<td>48.0%</td>
<td>60.0%</td>
<td>51.0%</td>
<td>46.5%</td>
<td></td>
</tr>
<tr>
<td>(D_5^4)</td>
<td>47.0%</td>
<td>68.5%</td>
<td>52.0%</td>
<td>48.5%</td>
<td></td>
</tr>
<tr>
<td>(D_5^5)</td>
<td>71.5%</td>
<td>67.5%</td>
<td>44.5%</td>
<td>51.0%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Quantitative similarities: distances between each of the coding styles represented by \(M_\alpha\) and \(D_\alpha^\beta\)

<table>
<thead>
<tr>
<th></th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>(M_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D_5^1)</td>
<td>0.1149</td>
<td>0.3770</td>
<td>0.2693</td>
<td>0.3233</td>
<td>0.1988</td>
</tr>
<tr>
<td>(D_5^2)</td>
<td>0.3139</td>
<td>0.1938</td>
<td>0.4353</td>
<td>0.2770</td>
<td>0.3477</td>
</tr>
<tr>
<td>(D_5^3)</td>
<td>0.3298</td>
<td>0.3314</td>
<td>0.1100</td>
<td>0.6016</td>
<td>0.2485</td>
</tr>
<tr>
<td>(D_5^4)</td>
<td>0.5043</td>
<td>0.4030</td>
<td>0.5216</td>
<td>0.0853</td>
<td>0.5984</td>
</tr>
<tr>
<td>(D_5^5)</td>
<td>0.0878</td>
<td>0.2865</td>
<td>0.2372</td>
<td>0.3826</td>
<td>0.0596</td>
</tr>
</tbody>
</table>

input \(D_5^1\) to \(M_1\), \(M_2\), \(M_3\), \(M_4\) and \(M_5\), respectively. In this way, we compare the coding style extracted from \(D_5^1\) with the correct coding style of \(A_1\), \(A_2\), \ldots, \(A_5\), respectively.

4.2.3. Results of the first step: a similarity measurement using SIM. SIM is based on an alignment algorithm, one of dynamic algorithms popular in bioinformatics. It outputs percentages of shared sub-sequences in a pair of source code sets.

Table 2 shows the similarity values between \(D_5^5\) outputted from SIM. For example, in the second row of the table, we have no value in the first cell and, starting from the second cell, we have 74.5% for \(D_5^2\), 51.5% for \(D_5^3\), 52.5% for \(D_5^4\) and 81.0% for \(D_5^5\). The values indicate the similarity when comparing \(D_5^1\) to \(D_5^2\), \ldots, \(D_5^5\). Since it is of little use comparing source code to itself, there are five cells that have diagonal lines instead of similarity values. Since SIM makes unidirectional comparisons, the results in the table are not symmetrical. Here, we treat values over 70% as a high similarity value. In the case of the comparison of \(D_5^1\) with \(D_5^2\), \ldots, \(D_5^5\), we see that \(D_5^2\) and \(D_5^5\) show a high similarity. Another case that shows a high similarity is that of \(D_5^5\) with \(D_5^1\), which has a similarity value of 71.5%. Overall, SIM detected high similarities between \(D_5^1\) and \(D_5^2\), \(D_5^1\) and \(D_5^5\), and \(D_5^5\) and \(D_5^1\). The code sets \(D_5^1\) and \(D_5^5\) showed an especially high similarity value.

4.2.4. Results of the second step: author identification by using the CM algorithm. To examine the cause of high similarity values for certain pairs of source codes, we check the outputs of the CM algorithm to determine whether the corresponding source code sets had been produced by the same authors. Table 3 shows the distance values between each of the correct coding styles for the five authors, represented as \(M_1\), \(M_2\), \ldots, \(M_5\) in the first row, and the coding styles calculated from \(D_5^\alpha\), represented as \(D_5^1\), \(D_5^2\), \ldots, \(D_5^5\) in the first row.
column. For example, the cell in the second row and the second column of the table has a distance value of 0.1149 between the coding style from $D_5^1$ and the correct coding style of $A_1$, represented by his/her coding model, $M_1$. The other values in the same row contain the values $0.3770(M_2), 0.2693(M_3), 0.3233(M_4)$ and $0.1998(M_5)$. To estimate the author of a given source code, we simply find the coding model that shares the smallest distance value with the source code. In this case, 0.1149 for $D_5^1$ and $M_1$ is the smallest, and so we assume that there is high possibility that $D_5^1$ had been produced by $A_1$. The rest of the results in the table show the same tendency as this case. In this way, we confirmed that all of the five source code sets had been produced by their true authors. Since we were sure that $D_5^1, D_5^2, \ldots, D_5^5$ had been produced by the true authors: $A_1, A_2, \ldots, A_5$, the author identification results were correct. In the previous step of the experiment, we observed some pairs of source code that shared high similarity values. However, from the author identification results, all five of the source code sets had been produced by their true authors. Therefore, we assumed that the similarities calculated by SIM were not due to plagiarism, and so treated them as false positives. In this way, we can reduce the risk of obtaining false positives caused by the similarity measure based on algorithmic similarity, which we discussed in Section 1. The greatest difference between these two algorithms is that the CM algorithm measures similarity based on the coding style of the authors who produced the source code, while SIM measures similarity based on the algorithmic features of the source code. Therefore, the combination of those two algorithms gives us information on two different aspects of similarities, which makes plagiarism detection significantly different. We expect this to be a novel methodology specialized for in-class source code plagiarism detection.

4.2.5. To obtain more detailed information for author identification. In this paper, we use the distance between a pair of 21-dimensional vectors to identify an author. If a teacher wants to see more detailed information on the similarity between two coding styles, he/she can check the constituents of the vectors. In other words, he/she can check the values of the observation probabilities that are output from each of the coding models $M_j$ and classified into 21 classes. Figure 4 shows the observation probabilities obtained from 3 of 14 coding models, $M_a$. This figure consists of three graphs that provide the results for the three coding models. The numbers on the $x$-axis shared among the three graphs are observation probability values classified into 21 classes. The percentages on the $y$-axis indicate the amounts of observation probabilities for each of the classes. We use percentages rather than numbers to indicate the amount of obtained observation probabilities, because the number of observation probabilities differs depending on the number or the length of the source code sets. The correct observation probabilities of $A_\alpha$ are “$D_{14}^\alpha$”, represented as the bar graphs with the diagonal lines. Here, “$A_\alpha$”, shown as the black bar graphs, are the resulting observation probabilities when we input $D_5^\alpha$ to $M_a$. We can see that there is a similar tendency in the distribution of the obtained observation probabilities between $A_\alpha$ and $D_{14}^\alpha$. In this way, we can examine in detail the similarity between the correct output and the output for a final project, using the information provided from each of the coding models $M_j^\alpha$, when needed.

5. Conclusion. We proposed a two-step similarity measuring method specialized for in-class source code plagiarism detection. Many existing methods show high accuracy when measuring similarity between source code sets. However, the algorithms of in-class source codes are often very similar. Thus, as other researchers have noted, there is a possibility that the output from existing methods contain false positives, and there is no method to check whether the similarity is caused by plagiarism or by coincidence. In this paper, we
discussed a novel methodology that first uses SIM, one of the well-known existing methods that uses algorithmic features to measure similarity, and then uses our CM Algorithm to check if the source codes had been produced by its original author. The information provided from our new method is useful for a teacher to judge if the detected similarity between source code sets was the result of plagiarism. In this way, we expect to reduce the risk of obtaining false positives caused by algorithmic-based similarity measures and the nature of in-class source code. From the results of our experiments, we found that there were some cases in which the similar source codes detected by SIM had been produced by different authors. In these cases, the detected similarity seemed to be a false positive caused by the nature of the in-class source code. However, if we see that two sets of source code have a high similarity, as detected by SIM, and we also observe that the coding style in the source code sets is similar, there is a high probability that the source code has been plagiarized. Another contribution of this paper is that we have improved our proposed method by considering the fluctuation of descriptions in the learning data. From the results of an evaluation experiment, we confirmed that the improved version of the CM Algorithm identified authors of source code that were short in length and contained considerable amounts of fluctuation.

Acknowledgment. This work is partially supported by Grant-in-Aid for Young Scientists (Start-up) 21800085.
REFERENCES


