Scaling Up Software Birthmarks Using Fuzzy Hashing

Takehiro Tsuzaki
Graduate School of Kyoto Sangyo University, Japan

Teruaki Yamamoto, Haruaki Tamada, and
Kyoto Sangyo University, Japan

Akito Monden
Okayama University, Japan

ABSTRACT

To detect the software theft, software birthmarks have been proposed. Software birthmark systems extract software birthmarks, which are native characteristics of software, from binary programs, and compare them by computing the similarity between birthmarks. This paper proposes a new procedure for scaling up the birthmark systems. While conventional birthmark systems are composed of the birthmark extraction phase and the birthmark comparison phase, the proposed method adds two new phases between extraction and comparison, namely, compression phase, which employs fuzzy hashing, and pre-comparison phase, which aims to increase distinction property of birthmarks. The proposed method enables us to reduce the required time in the comparison phase, so that it can be applied to detect software theft among many larger scale software products. From an experimental evaluation, we found that the proposed method significantly reduces the comparison time, and keeps the distinction performance, which is one of the important properties of the birthmark. Also, the preservation performance is acceptable when the threshold value is properly set.

Keywords: Software Birthmark, Fuzzy Hashing, Preprocessing,

I. INTRODUCTION

Till today, software theft has been causing serious damage to software industry. From the BSA global software survey 2016[1], 39% of software installed on computers in the world is not properly licensed. Also, violation of open source software (OSS) licenses, such as GPL[2], by unexpected and unaware reuse of OSS source code[3] has now become a serious problem for both software companies and OSS developers [Monden et al., 2011]. Software birthmark methods have been proposed against such software theft to enable us to detect the theft [Tamada et al., 2004], [Tamada et al., 2005]. A software birthmark is a set of characteristics which a program originally possesses. It is extracted from a binary code and used to evaluate the similarity between one program and another (extraction and comparison phases). Various types of birthmarks have been proposed, each focusing on different characteristics in a program. Different extraction methods and comparison methods have also been defined for each type of birthmark and have been evaluated according to those definitions.

[1] `http://globalstudy.bsa.org/2016/downloads/studies/BSA_GSS_US.pdf`
[2] `http://www.gnu.org/licenses/licenses.en.html`
[3] `http://www.itmedia.co.jp/news/0210/18/njbt_06.html`

Software birthmarks are designed to search of large amounts of software to detect suspected copies; hence, their use requires high-speed, large-volume software repository searches. However, the software birthmark has the one essential problem in the practical use case. That is, the scale of the target software was not assumed. Figure 1 illustrates the problem in use of the software birthmarks. The developer can examine for detecting the copy of p_0 from the target set programs p_1 to p_n. However, the many unchecked programs are still existing in the Internet. The most important issue of the software theft is to detect suspected copies. The programs p_{n+1} to p_{n+m} in the figure 1 are never investigated because memory constraints, vast amount of time consumed for comparison, and the enormous computational complexity. However, almost programs are innocent and quite different. Therefore, to detect the software theft requires more simple and casual algorithm for huger target set.

Therefore, we proposed the method for the software birthmark procedure to narrow the defendants with compressing birthmark information and simplifies comparison algorithms. Figure 2 shows the difference of the conventional and the proposed birthmark procedures. Form figure 2, we insert two phases, compression phase and pre-comparison phase, between the conventional phases. The compression phase compresses the birthmark information for the next phase. The pre-comparison phase compares compressed birthmarks by simple algorithm and computes similarity. Then, remains of the pre-comparison are still defendants, then, the remains are the inputs for the comparison phase.

This remainder of this paper is organized as following. Section II reviews the related works. Section III describes the proposed method and illustrates the novel procedure of the birthmark system. Section IV represents the empirical studies of our method. Section V shows conclusion and some future works.

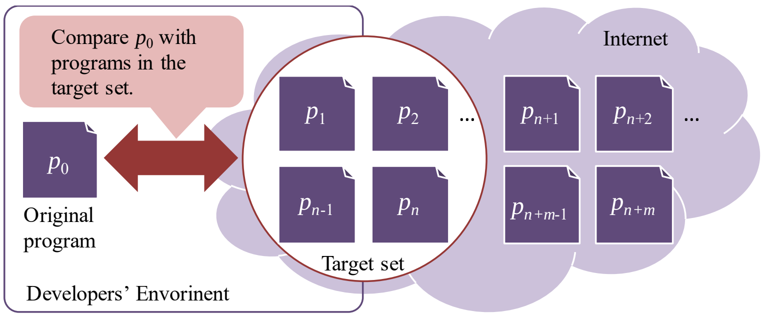

Figure 1. The problem of the current birthmark system

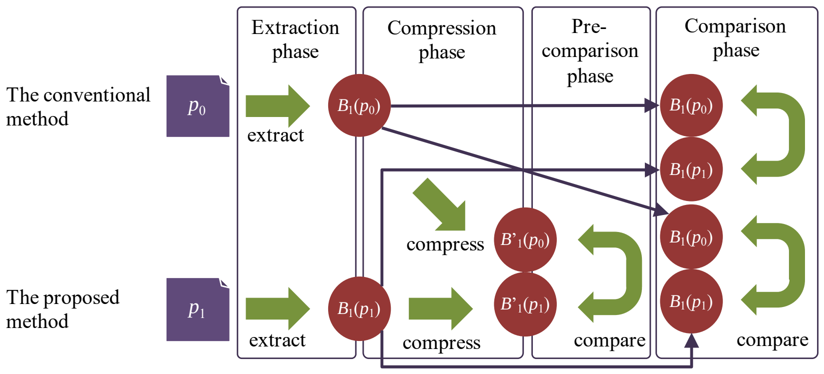

Figure 2. The difference between the conventional method and the proposed method

II. RELATED WORKS

Birthmarks are a concept that was proposed by Tamada et al. as a method for detecting software theft [Tamada et al., 2004], [Tamada et al., 2005]. Characteristics unique to a program that are contained in the program's binary code are extracted as birthmark information and used to measure similarity. Unlike software watermarks, there is no need for prior information embedding; characteristics unique to the program are taken from the compiled binary and defined as the birthmark. Several different types of birthmark that focus on a different program characteristics have been proposed [Chan et al., 2012], [Choi et al., 2009], [Jhi et al., 2011], [McMillan et al., 2012], [Schuler et al., 2007], [Park. et al., 2008].

Birthmarks ordinarily extract elements by examining characteristics in the program. The extracted elements are maintained in a structure such as a set, sequence or graph, and the similarity computation method is defined in accordance with the structure used. For example, the Jaccard index [Schuler et al., 2007], Dice's coefficient [Choi et al., 2009] or cosine similarity [McMillan et al., 2012] are often used for sets, and the longest common subsequence [Jhi et al., 2011], [Park. et al., 2008] is used for sequences. Methods for graphs are more complex; however, methods using graph isomorphism have been proposed [Chan et al., 2012], [il Lim et al., 2008].

The number of birthmark elements can become massive, depending on the birthmark type and program length. An increasingly large number of birthmark elements leads to a corresponding increase in the cost of birthmark comparison and time needed for the similarity computation. Accordingly, the complexity of the similarity computation method and a number of birthmark elements affects the search time.

However, although many different definitions have been proposed for birthmark methods, these birthmark methods do not compete with each other. The greater the number of birthmarks that are defined, the greater the number of clues enabling theft detection. Longer check times resulting from the greater number of birthmarks used can be reduced by the proposed method.

III. THE PROPOSED METHOD

A. The Definition of Birthmarks

Before describing the proposed method, we review the definition of birthmarks by Tamada et al. Birthmarks are defined as follows [Tamada et al., 2004], [Tamada et al., 2005].

Definition 1 (*Birthmark*): Let p and q be given programs. Further, let $f(p)$ be a set of characteristics extracted from p by a given method f. If the conditions below are met, $f(p)$ is said to be a birthmark of p.

Condition 1: Set $f(p)$ can only be obtained from program p.

Condition 2: If q is a copy of p, then $f(p) = f(q)$

Condition 1 implies that a birthmark is information necessary for running p, rather than supplementary program information. In other words, a birthmark does not require supplementary information in the manner of a software watermark. Condition 2 indicates that the same birthmark can be obtained from a copied program. The contrapositive of this condition is that if birthmarks $f(p)$ and $f(q)$ are different, q is not a copy of p.

Two properties known as preservation and distinction should also ideally be satisfied.

Property 1 (*Preservation*): For a p' obtained by an arbitrary equivalent transformation of p, $f(p) = f(p')$ is satisfied.

Property 2 (*Distinction*): When programs p and q that develop independently, $f(p) \neq f(q)$ is satisfied.

Preservation indicates a birthmark's resistance to various types of attacks. Distinction indicates that programs created completely independently can be differentiated even if their specifications are the same. Because birthmarks that completely satisfy both these properties are difficult to create, in practice, birthmark strength must be set appropriately at the user's discretion.

B. Overview of the Proposed Method

The ultimate aim of a birthmark is to enable theft detection, not to prove cases of theft. Birthmarks therefore need to be able to detect programs that resemble the comparison program amid a large amount of software. Hence, a costly birthmark similarity computation that increases the search time itself is undesirable. False negatives (undetected copies) are much more of a problem than false positives (incorrectly detected innocent programs) because false positives need to be proven as cases of theft by other subsequent methods, and the error will be uncovered during that process.

Given these considerations, we looked at transforming birthmarks into short data sequences, and then using the data obtained to compute similarity from a simple algorithm. Because we replaced the conventional algorithm with a simple method, a rise in the false positive rate is expected. However, because the method can check for suspected copies over a much larger amount of software than previously possible, this makes it easier to narrow down searches for suspected copies. This paper focuses on the hash function for the compression method, since the hash function dramatically reduce the data length.

Although birthmark extraction is another time-intensive process, this paper does not cover the topic of extraction time reduction, as birthmark extraction can be executed in advance and the extraction results stored in a database.

C. The Key Idea

The key idea of the proposed method is to convert a birthmark obtained by a conventional method to a hash function and to then use the obtained hash values to compute similarity. Cryptographic hash functions such as MD5 or SHA2 cannot be used to compute similarity for this process, as the output values will be completely different if the input values are even slightly different. Cryptographic hash functions can be used for identification, but not for similarity computation. Hence, to compute similarity, we use fuzzy hashing, also known as context-triggered piecewise hashing (CTPH), which was developed as a method for identifying fairly different data strings [Kornblum, 2006].

Although birthmarks are never deleted, they can sometimes be attacked in order to transform them into different birthmarks. Because trivial differences in birthmarks could lead to large differences in the proposed method, we defined two preprocessing steps before compression phase to enable resistance to attacks —*sort* and *memoization*.

The sort preprocessing step sorts the birthmark elements before hash calculation. For example, when the birthmark is handled as a sequence, large changes can result from reordering that sequence. For birthmarks extracted from individual Java methods such as k-gram birthmarks [Myles and Collberg, 2005], reordering the sequence of Java methods greatly changes the birthmark. While there are conventional methods that define the comparison method in a way that can handle Java method reordering, the simple comparison method used by the proposed method is not resistant to such reordering. We therefore sorted birthmark elements in advance before hash calculation.

The memoization preprocessing step encodes birthmark elements as numerical values instead of character strings before applying fuzzy hashing. This preprocessing step is required to satisfy the distinction property of birthmarks, because similar strings yield similar fuzzy hash values but in general similar strings does not indicate (functionally) similar programs. For example, the Used Class (UC) birthmark [Tamada et al., 2004], [Tamada et al., 2005] denotes a series of classes used by the target program (class).

Supposing that the target program uses "`String`" and "`System`" classes, then "`java.lang.String`" and "`java.lang.System`" are included in the UC birthmark. Although "`java.lang.String`" and "`java.lang.System`" are functionally different, their strings are similar because both contain "`java.lang`" portion. To make these classes dissimilar, the memoization preprocess maps these strings to different numerical values.

D. Comparison Method of the Proposed Method

This section describes the method used to compute the similarity of two compressed data obtained by the proposed method. Because we need to use a simple algorithm for comparing the compressed data to keep the computational complexity low, we use edit distance. Edit distance is a metric that defines the distance between two data strings and is defined as the number of addition, deletion, or replacement operations needed to transform one string into another. The similarity of two compressed data is hence defined as follows:

Definition 2 (*Edit distance-based similarity*): Let d represent a data stream and $l(d)$ represent its length. Let $\mathrm{ed}(d_1, d_2)$ represent the edit distance between d_1 and d_2. The similarity between d_1 and d_2 is: $1 - \frac{\mathrm{ed}(d_1, d_2)}{\max(l(d_1), l(d_2))}$.

As in conventional birthmark methods, we express similarity as a number between 0 and 1, with similarity increasing as the value approaches 1 and decreasing as it approaches 0.

E. The Framework of the Proposed Method

Figure 3 shows the proposed framework alongside the framework used by the conventional method. The conventional method starts by extracting birthmarks from both the software to be checked and the software to be searched. The similarity of these birthmarks is then calculated. The proposed framework also compresses obtained birthmarks to hashes. The hashes are

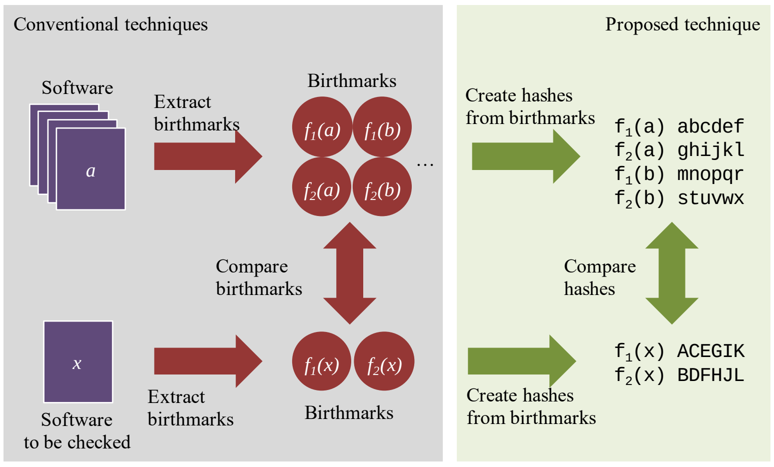

Figure 3. Overview of the conventional and proposed method

compared by comparing their edit distance [Levenshtein, 1966].

This framework is designed to enable faster checking while preserving the features of previously proposed birthmarks. Conventional birthmark comparison processes are often complex; using edit distance to compare hash data simplifies the process. The use of hash data should result in shorter data lengths than in existing birthmarks. A single element in a birthmark can be complex and require a long time for element comparison, even though its data length is short, resulting in a long total comparison time. Creating hashes can eliminate the complexity of birthmark elements.

Note that the proposed method creates hashes from any birthmark information, and hence can be used with any previously proposed birthmark method.

IV. EXPERIMENTAL EVALUATION

A. The Overview of the Evaluations

The proposed method was evaluated from the following four perspectives:

1) Change in number of elements,
2) Comparison time,
3) Distinction performance, and
4) Preservation performance.

Because the proposed method finds programs similar to a comparison program amid a large amount of software, it needs to perform that comparison faster than the conventional method. We therefore measured the reduction in the number of elements affecting comparison time. To evaluate performance, we also measured the reduction in comparison time. In addition, we measured distinction and preservation, two properties that birthmarks should satisfy.

We selected three types of birthmark for testing: k-grams of opcode sequences [Myles and Collberg, 2005] (set-based), UCs [Tamada et al., 2004], [Tamada et al., 2005] (used classes; sequence-based), and WSPs [Lim et al., 2008] (weighted stack patterns; graph-based). We used ssdeep[4] for fuzzy hashing, Stigmata[5] for birthmark extraction, and Python to implement the proposed method.

As described in Section III-C, the proposed method has two preprocessing steps. We tested the proposed method for all combinations of steps (a total of four different test patterns), to compute four sets of hash values.

B. Change in number of elements

Birthmarks generally consist of multiple elements. Because birthmark comparison requires element-to-element comparison, comparison time increases as the number of elements increases. The proposed method transforms the birthmarks into data strings. Although the data strings also need to be compared, they should be shorter than the original birthmarks. This test determined how much shorter they became.

As test data, we collected the latest products from the repository of the Apache Software Foundation[6], and used the 3,786 JAR files we obtained as the programs to be checked. They contained a total of 567,691 classes.

---

[4] `http://ssdeep.sourceforge.net/`

[5] `http://github.com/tamada/stigmata/`

[6] `http://www.apache.org/`

Table 1. Minimum, maximum, and average element count/hash lengths

		Conventional method	The proposed method			
			Neither	Memoization only	Sort only	Both
2-gram	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	1,140.00	101.00	101.00	101.00	101.00
	Ave.	47.26	54.37	49.32	58.90	50.48
3-gram	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	3,669.00	102.00	101.00	102.00	102.00
	Ave.	76.56	64.84	52.79	65.46	53.05
4-gram	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	8,178.00	102.00	102.00	102.00	102.00
	Ave.	99.36	67.22	53.38	66.71	47.48
5-gram	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	16,949.00	102.00	102.00	102.00	102.00
	Ave.	114.58	67.53	53.15	67.32	53.18
6-gram	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	32,111.00	102.00	102.00	102.00	102.00
	Ave.	124.83	67.17	52.56	67.46	52.73
UC	Min.	0.00	4.00	4.00	4.00	4.00
	Max.	407.00	102.00	101.00	102.00	101.00
	Ave.	7.86	55.34	17.18	55.34	17.18
WSP	Min.	1.00	4.00	4.00	4.00	4.00
	Max.	364,007.00	102.00	102.00	102.00	102.00
	Ave.	53.10	67.39	40.34	67.31	39.85

Table 1 shows the minima, maxima, and averages for both the element counts (conventional method) and hash lengths (proposed method) obtained for each of the birthmark methods tested. The first column shows the results of extracting birth- marks from individual classes using the conventional method. The next four columns show results obtained using each of the four test patterns of the proposed method — "Neither" indicates that neither of the two preprocessing steps were performed, "Memoization only" that only memoization was performed, "Sort only" that only sort was performed, and "Both" that both of the preprocessing steps were performed.

Figure 4 shows the distribution of birthmark element counts obtained by the conventional method. Figure 5 shows the data string length distribution obtained by the proposed method. In each graph, the horizontal axis represents the range of element counts (data string lengths), and the vertical axis represents the corresponding frequency (number of instances) on a log scale.

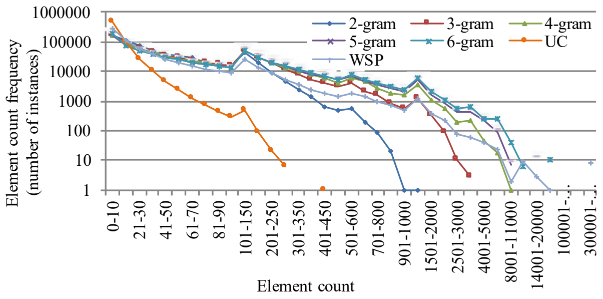

Figure 4. Distribution of birthmark element counts obtained by the conventional method

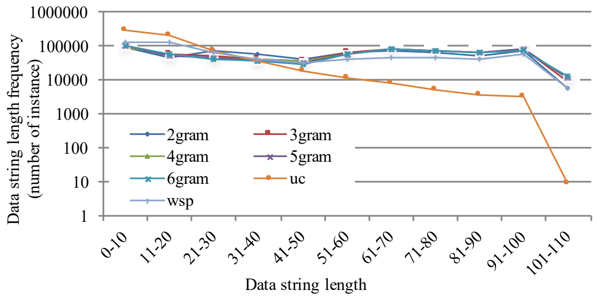

Figure 5. Distribution of data string lengths obtained by the proposed method (with sort and memoization preprocessing)

Table 2. Time required for comparison (s)

		Groovy	H2	Velocity
Number of classes		3,086	627	270
Number of comparison		4,763,241	196,878	36,585
Conventional method	2-gram	104.10	2.71	0.74
	3-gram	136.30	6.31	0.85
	4-gram	177.06	10.97	1.18
	5-gram	186.03	13.51	1.49
	6-gram	209.17	15.22	1.37
	UC	72.35	1.17	0.50
	WSP	N/A	1,936.67	70.78
Proposed method	2-gram	57.17	3.18	0.51
	3-gram	64.90	3.25	0.54
	4-gram	69.07	3.31	0.56
	5-gram	66.77	3.28	0.56
	6-gram	67.38	3.32	0.59
	UC	32.43	1.64	0.44
	WSP	57.26	3.02	0.50

Because of space constraints, the only graph shown for the proposed method is the result obtained when both the sort and memoization preprocesses were performed. As shown, the conventional method generated many birthmarks with small numbers of elements, but also some with extremely large numbers of elements. The proposed method generated data string lengths of a nearly uniform distribution and a maximum length of just over 100. Its top data string length range was 100 to 110, but the occurrence frequency for this range was low, possibly because the maximum data string length was 102, making this range smaller than the others.

C. Comparison Time

The proposed method should also reduce comparison time. This section describes the testing we performed to determine how much comparison time was reduced. Conventional method comparison times were measured using the method set for each of the evaluated birthmark methods. We used three libraries for comparison: Groovy 2.4.0 Beta 2[7], H2 1.4.182[8], and Velocity 1.7[9]. Proposed method comparison times were measured using the four test patterns previously described, taking the average of the results obtained. This is because the four test patterns only resulted in marginally different hash lengths and did not generate major differences in comparison time. The execution environment was a MacBook Air PC running on the OS X Yosemite operating system, with a 1.7 GHz Intel Core i7 CPU and 8 GB of memory.

The result is shown in Table 2. The comparison time for the conventional method using WSP birthmarks and the Groovy library has been omitted because the comparison did not complete in a realistic amount of time (6 h).

The Velocity library has the smallest number of classes. For the UC birthmarks, there was nearly no difference between the comparison times of the proposed and conventional methods. UC birthmarks are a very simple type of birthmark that use the Jaccard index as the comparison method. Because they have a small number of elements (as indicated in Section IV-B), they gain little advantage from the size reduction provided by the proposed method.

[7] `http://www.groovy-lang.org/`

[8] `http://www.h2database.com/`

[9] `http://velocity.apache.org/`

Table 3. Distinction rate list

		Conventional method	The proposed method			
			Neither	Memoization only	Sort only	Both
Groovy	2-gram	65.86%	91.10%	93.04%	04.04%	95.69%
	3-gram	89.63%	91.96%	96.92%	96.61%	97.01%
	4-gram	93.52%	94.41%	96.80%	96.90%	97.46%
	5-gram	94.06%	95.75%	97.13%	96.59%	97.43%
	6-gram	95.54%	96.89%	97.40%	97.38%	97.41%
	UC	92.45%	93.88%	98.30%	93.88%	98.30%
	WSP	N/A	94.65%	97.48%	95.96%	97.51%
H2	2-gram	57.89%	99.96%	99.97%	99.96%	99.96%
	3-gram	99.40%	99.96%	99.99%	99.97%	99.97%
	4-gram	99.31%	99.98%	99.99%	99.99%	99.99%
	5-gram	99.30%	99.98%	99.99%	99.99%	99.99%
	6-gram	99.28%	99.98%	99.99%	99.99%	99.99%
	UC	96.82%	96.19%	99.20%	96.19%	99.20%
	WSP	91.72%	99.96%	99.97%	99.96%	99.97%
Velocity	2-gram	69.04%	99.82%	99.80%	99.79%	99.84%
	3-gram	96.81%	99.79%	99.80%	99.83%	99.85%
	4-gram	96.56%	99.80%	99.78%	99.84%	99.87%
	5-gram	96.31%	99.75%	99.78%	99.86%	99.86%
	6-gram	96.31%	99.75%	99.82%	99.86%	99.85%
	UC	99.17%	99.12%	99.49%	99.12%	99.49%
	WSP	91.43%	99.72%	97.03%	99.78%	97.01%

In contrast, the proposed method's comparison time for WSP birthmarks was about 141 times faster than the conventional method, even for the Velocity library, which had the shortest comparison times. Because WSP birthmarks are represented by graphs and use a complex comparison method, we found the proposed method provides a major reduction in comparison time.

D. Distinction performance evaluation

This section describes the test to determine how much the birthmarks transformed by the proposed method possessed the property of distinction. Because similar classes will usually not exist in a single product, we checked the class data used in Section IV-C to determine whether similar items were found. If similar classes exist, they are integrated by refactoring, but small classes can sometimes be coincidentally similar, often preventing checks using birthmarks. Even for the proposed method, short hash lengths are not likely to produce correct results, so we excluded hash lengths of five or less from the comparison.

Table 3 lists the distinction rates we obtained. Using a past standard as our reference [Schuler et al., 2007], we set a similarity threshold of 0.75, and evaluated results higher than this value as distinction failures. The rates of successfully distinguished pairs are shown as percentages of the whole for each library and birthmark type. The results show that the proposed method had distinction rates that were above 90% in every category, higher than the rates obtained by the conventional method in every category except for the UC birthmarks with the Velocity library. For the H2 and Velocity libraries, there was almost no difference in the distinction rates of the proposed method, regardless of whether preprocessing was performed. For the Groovy library, we found that performing preprocessing produced better results.

We show the results of our distinction rate evaluation graphically by graphing similarity frequencies instead of the distinction rates themselves. Figure 6 shows the results for 2-gram birthmarks with the Groovy library (which had the lowest average distinction rate), while Figure 7 shows the results for UC birthmarks with the Velocity library (which had the highest average

distinction rate). In each graph, the horizontal axis represents similarity and the vertical axis represents the corresponding similarity frequency. The lines in each graph show the frequencies for the conventional method and for each of the four test patterns of the proposed method. Figure 6 shows that relative to the conventional method, the proposed method had lower numbers of results with high levels of similarity.

E. Preservation performance evaluation

We also evaluated the proposed method's preservation property using ProGuard[10] to obfuscate files from the H2 library and comparing their classes before and after obfuscation. The default ProGuard obfuscation process consists of name obfuscation, deletion of unneeded Java method calls, and optimization.

Table 4 shows the comparison results. The values labeled "Min." and "Max." are the minimum and maximum similarity values generated by comparing results before and after obfuscation. The "Avg." values are the average similarity values obtained for all the comparisons in each category. "Rate (0.75)" and "Rate (0.5)" show the frequency (percent) of pairs with similarity values of over 0.75 and 0.5, respectively.

The "Rate (0.75)" values show that the proposed method does not possess the property of preservation at the 0.75 similarity threshold. However, as shown by the "Rate (0.5)" values, preservation can be assured if the threshold is lowered to 0.5. We found neither preprocessing step contributed to preservation. When memoization was performed, the maxi- mum similarity only exceeded 0.5 for UC birthmarks, likely because memoization reduced the differences among the original birthmark elements, resulting in them being greatly affected by obfuscation. Similarly, we found that performing sort also greatly reduced preservation for all birthmarks except 2-gram, WSP, and UC. Because UC birthmarks are already defined as sorted, the sort preprocessing had no effect on them. WSP birthmarks were also nearly unaffected by the sort preprocessing, but the preservation of 2-gram birthmarks was improved by the sort preprocessing.

These findings indicate that the proposed method offers no benefit for some types of birthmark. They also indicate that our preprocessing steps are unnecessary. However, we only evaluated preservation in response to ProGuard obfuscation, and preservation in response to other obfuscation processes needs to be evaluated.

V. CONCLUSION

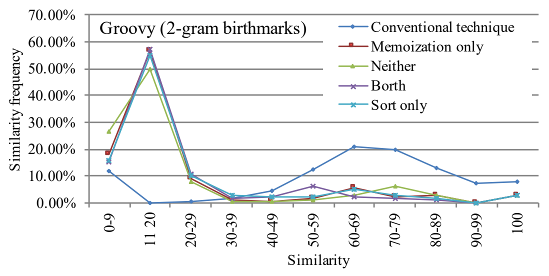

Figure 6. Groovy comparison result for 2-gram birthmarks

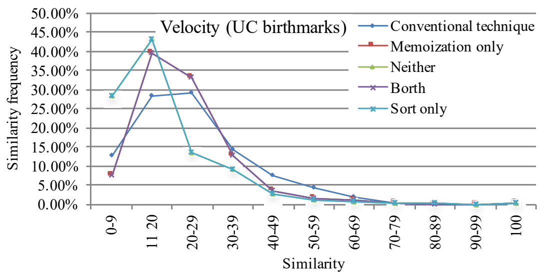

Figure 7. Velocity comparison result for UC birthmarks

[10] `http://proguard.sourceforge.net/`

Table 4. Preservation performance evaluation results

		Min	Max	Avg.	Rate (0.75)	Rate (0.5)
2-gram	Conventional	0.00	1.00	0.96	96.76%	98.92%
	Neither	0.06	1.00	0.60	31.60%	62.62%
	Memoization only	0.04	0.44	0.12	0.00%	0.00%
	Sort only	0.08	1.00	0.63	33.33%	69.17%
	Both	0.05	0.38	0.12	0.00%	0.00%
3-gram	Conventional	0.00	1.00	0.91	94.05%	98.38%
	Neither	0.07	1.00	0.62	36.61%	66.67%
	Memoization only	0.03	0.33	0.11	0.00%	0.00%
	Sort only	0.05	1.00	0.54	24.47%	46.44%
	Both	0.03	0.43	0.11	0.00%	0.00%
4-gram	Conventional	0.00	1.00	0.87	89.73%	97.12%
	Neither	0.10	1.00	0.61	35.10%	64.90%
	Memoization only	0.06	0.44	0.11	0.00%	0.00%
	Sort only	0.07	1.00	0.50	20.59%	43.33%
	Both	0.04	0.33	0.11	0.00%	0.00%
5-gram	Conventional	0.00	1.00	0.82	80.36%	95.50%
	Neither	0.06	1.00	0.61	32.68%	67.72%
	Memoization only	0.03	0.38	0.11	0.00%	0.00%
	Sort only	0.08	1.00	0.47	16.54%	37.60%
	Both	0.04	0.39	0.11	0.00%	0.00%
6-gram	Conventional	0.00	1.00	0.79	71.71%	93.69%
	Neither	0.06	1.00	0.61	33.20%	66.40%
	Memoization only	0.04	0.50	0.11	0.00%	0.00%
	Sort only	0.07	1.00	0.44	15.02%	34.39%
	Both	0.03	0.44	0.11	0.00%	0.00%
UC	Conventional	0.08	1.00	0.98	97.48%	98.74%
	Neither	0.06	1.00	0.97	94.12%	97.61%
	Memoization only	0.05	0.71	0.28	0.00%	1.85%
	Sort only	0.06	1.00	0.97	94.12%	97.61%
	Both	0.05	0.71	0.28	0.00%	0.00%
WSP	Conventional	0.00	1.00	0.96	96.76%	98.92%
	Neither	0.03	1.00	0.61	38.73%	65.13%
	Memoization only	0.03	0.50	0.14	0.00%	0.00%
	Sort only	0.02	1.00	0.59	31.79%	64.74%
	Both	0.01	0.44	0.14	0.00%	0.00%

This paper proposed two phases for the birthmark system that uses fuzzy hashing to enable faster comparison and is designed for use with existing birthmark methods. The compression phase defines two preprocessing steps, sort and memoization. We evaluated the proposed method on three existing birthmark types. The results show that the proposed method can improve comparison speed and preserve the property of distinction that birthmarks possess. However, we found it greatly reduces preservation, so its use would require a lower similarity threshold or similar adjustment.

The following areas need to be covered in future studies: refining the proposed method to improve preservation, evaluating its preservation in response to other obfuscation methods, and evaluating its use with other birthmark extraction methods.

REFERENCES

[Chan et al., 2012] Chan, P., Hui, L., and Yiu, S. (2012). Heap graph based software theft detection. IEEE Transactions on Information Forensics and Security, 8:101–110.

[Choi et al., 2009] Choi, S., Park, H., il Lim, H., and Han, T. (2009). A static API birthmark for windows binary executables. Journal of Systems and Software, 82(5):862–873.

[il Lim et al., 2008] il Lim, H., Park, H., Choi, S., and Han, T. (2008). Detecting theft of Java applications via a static birthmark based on weighted stack patterns. IEICE Transactions on Information and System, E91-D(9):2323–2332.

[Jhi et al., 2011] Jhi, Y.-C., Wang, X., Jia, X., Zhu, S., Liu, P., and Wu, D. (2011). Value-based program characterization and its application to software plagiarism detection. In Proc. the 33rd International Conference on Software Engineering (ICSE 2011), pages 756–765.

[Kornblum, 2006] Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. Journal Digital Investigation: The International Journal of Digital Forensics & Incident Response, 3:91– 97.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8):707–710.

[McMillan et al., 2012] McMillan, C., Grechanik, M., and Poshyvanyk, D. (2012). Detecting similar software applications. In Proc. the 34th International Conference on Software Engineering (ICSE 2012), pages 364–374.

[Monden et al., 2011] Monden, A., Okahara, S., Manabe, Y., and Matsumoto, K. (2011). Guilty or not guilty. Using cline metrics to determine open source licensing violations. IEEE Software, 28(2), pages 42–47.

[Myles and Collberg, 2005] Myles, G. and Collberg, C. (2005). K-gram based software birthmarks. In Proc. the 20th Annual ACM Symposium on Applied Computing, pages 314–318.

[Park. et al., 2008] Park., H., il Lim, H., Choi, S., and Han, T. (2008). A static Java birthmark based on operand stack behaviors. In International Conference on Information Security and Assurance (ISA 2008), pages 133– 136.

[Schuler et al., 2007] Schuler, D., Dallmeier, V., and Lindig, C. (2007). A dynamic birthmark for Java. In Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), pages 274– 283.

[Tamada et al., 2004] Tamada, H., Nakamura, M., Monden, A., and Matsumoto, K. (2004). Design and evaluation of birthmarks for detecting theft of Java programs. In Proc. IASTED International Conference on Software Engineering (IASTED SE 2004), pages 569–575. (Innsbruck, Austria).

[Tamada et al., 2005] Tamada, H., Nakamura, M., Monden, A., and Matsumoto, K. (2005). Java birthmarks —detecting the software theft —. IEICE Transactions on Information and System, E88-D(9):2148–2158.