A Picture Is Worth a Thousand Words: Code Clone Detection Based on Image Similarity

Chaiyong Ragkhitwetsagul, Jens Krinke
CREST, University College London
London, UK

Bruno Marnette
Prodo.ai
London, UK

***Abstract*—This paper introduces a new code clone detection technique based on image similarity. The technique captures visual perception of code seen by humans in an IDE by applying syntax highlighting and images conversion on raw source code text. We compared two similarity measures, Jaccard and earth mover's distance (EMD) for our image-based code clone detection technique. Jaccard similarity offered better detection performance than EMD. The F1 score of our technique on detecting Java clones with pervasive code modifications is comparable to five well-known code clone detectors: CCFinderX, Deckard, iClones, NiCad, and Simian. A Gaussian blur filter is chosen as a normalisation technique for type-2 and type-3 clones. We found that blurring code images before similarity computation resulted in higher precision and recall. The detection performance after including the blur filter increased by 1 to 6 percent. The manual investigation of clone pairs in three software systems revealed that our technique, while it missed some of the true clones, could also detect additional true clone pairs missed by NiCad.**

I. Introduction

In this study, we aim to explore a new way of detecting code clones based on their image similarity. Code clone detection techniques in research mainly involve analysing source code text directly [5], [20], or its derivative abstract representation, such as tokens [17], [22], [9], [4], abstract syntax trees [2], [8], or graphs [12], [11]. Nevertheless, the first impression programmers face when looking at source code in IDE during development are visual images of code. To manually look for clones in their software projects, one roughly scans the code by looking for similar shape and layout of two source code fragments. Once a similar piece of code is found visually, one can perform a more fine-grained checking of the source code text, its syntax, or its behaviours. We follow that intuition of using visual similarity between source code images to look for clones. By representing a source code fragment as an image, the code consists of *pixels* (i.e. dots) encoding colour, grey-scale or RGB values.

Performing code similarity computation based on images has two major benefits. First, it is the finest-grained level of matching which captures similarity based on distribution of pixels in source code images. Two cloned code fragments will have the majority of their code pixels aligned, and vice versa. A code clone detector that compares source code text needs to perform identifier renaming in order to detect type-2 clones [20]. In our approach, although data types or identifiers are renamed, they can still be partially matched at pixel level. We found that our technique is capable of detecting type-1, type-2 and some type-3 clones. Second, comparing code based on their images opens a whole new area of code normalisation. Instead of renaming variables or converting source code text into tokens to normalise the effects of code modifications, we can adopt techniques from image processing and computer vision and normalise their image representation instead. For example, in this paper, we exploit the scale-space representation techniques [10], [13] which compares images on different structural levels by applying image blurring filters. The findings show that normalising code images using a Gaussian blur filter can improve clone detection performance by 1 to 6 percent. Third, by representing source code with images, it lays a foundation to deep learning techniques, which are well-established in the area of computer vision, to classify images of code fragments as cloned or non-cloned pairs.

This paper makes the following primary contributions:

1. An exploratory study of image-based code clone detection. To the best of our knowledge, we are the first to try comparing clones using images. We implemented a tool, called *Vincent*, based on the proposed image-based code clone detection framework and evaluated it on two data sets[1]. We found that the technique gives a decent performance, comparable to four well-known code clone detectors.

2. An investigation of using Gaussian blur as a code normalisation technique. We evaluated the effectiveness of applying blurring filter to code images before performing clone detection and observed that it modestly increased the detection's precision and recall.

II. Methodology

A. Image-based Code Clone Detection Framework

The framework of our image-based clone detection is depicted in Figure 1. We divide the clone detection process into 4 steps including Pre-processing, Image Conversion, Image Processing, and Similarity Measurement.

In Step 1, given a of Java source code file within a software system, the source code is prepared for image comparison by going through the preprocessing steps as the following. We parse the Java file to extract methods. Comments are removed from the methods and pretty printing is applied to the code. The pretty-printed methods are then converted into HTML

[1]The Vincent tool, its results, and the manually validated clone pairs can be found at https://ucl-crest.github.io/iwsc2018-vincent-web

To be published in the Proceedings 12th International Workshop on Software Clones, 2018 in Campobasso, Italy.

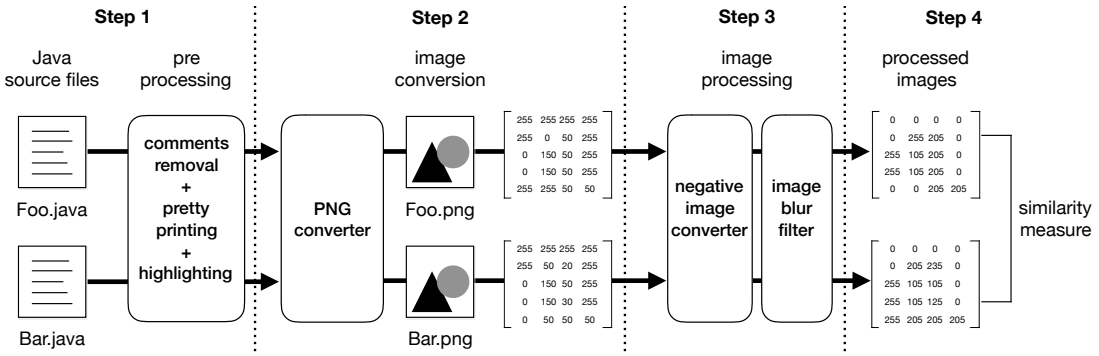


Fig. 1: Image-based code clone detection framework

documents of highlighted Java code. We add highlights to Java source code because it adds visual cues into the source code image by giving different colours or different font weights to keywords, packages, data types, class or function names, identifiers, and number/string literals. In Step 2, each method is converted into a Portable Network Graphics (PNG) image. Each image is read into memory as a 2-dimensional matrix of size *m* times *n* containing a value ranging from 0 to 255, representing an 8-bit grey-scale image of the original image. We compute a grey-scale value for each pixel by averaging over the red, green, and blue (RGB) colour channels.

In Step 3, we convert the image into a negative image, i.e. reversing the colour value, to ease the similarity detection process. As displayed in Figure 2a, after the negative image conversion, the pixels containing source code are either grey or white, while the background, non-source-code pixels, are completely dark. This means we can easily count the number of pixels containing source code by the number of non-zero elements in the matrix. In this step, image filters can be added to the source code image before performing similarity detection. In this paper, we investigated the effect of adding a Gaussian blur filter to the image (Section II-C) and measured it effects to the detection performance. Lastly, in Step 4, two matrices of processed images are compared for similarity (Section II-B). In order to locate clones in a software system, we perform a pairwise comparison of every method pair.

B. Image Similarity

We consider two similarity measures for image comparison in this study: Jaccard and Earth Mover's Distance.

1) Jaccard similarity (L0-norm): In order to calculate Jaccard similarity, we define the difference of two code images based on zero norm as follows. Given two images A and B as matrices of size $m \times n$, the amount of difference between the two images, i.e. diff, is a count of non-zero element in the element-wise difference matrix D derived from A and B. For example, given a pair of source code images `BubbleSortLong1` and `BubbleSortFloat2` as depicted in Figure 2a and Figure 2b, their difference is visualised in Figure 2c. Then, the diff value is a count of non-zero pixels within the red-boxed areas in the image.

$$D = A - B$$
$$\mathrm{diff} = \#(d_{ij} \in D | d_{ij} \neq 0)$$

To bound the distance score between zero and one, we normalise the distance with sum, the area containing source code text in the two images. This code area represents by non-zero elements in the element-wise sum matrix S derived from A and B. Again, given a pair of source code images `BubbleSortLong1` and `BubbleSortFloat2` (Figure 2a and Figure 2b), the value of sum is the number of non-zero pixels within the blue-boxed areas in Figure 2d.

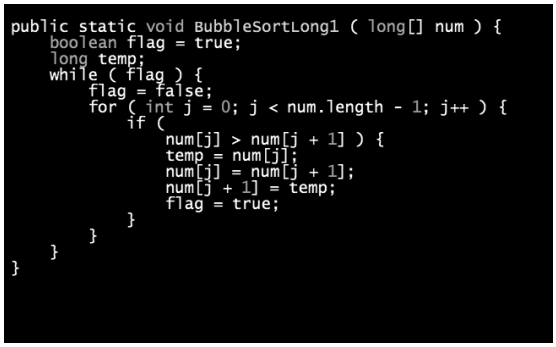
$$S = A + B$$
$$\mathrm{sum} = \#(s_{ij} \in S | s_{ij} \neq 0)$$

We specifically choose the code areas instead of total number of pixels in an image because of a large area of non-code background. Taking the non-code areas into account while computing the distance will result in a tiny distance value because they are large and always matched. The normalised $\mathrm{distance}$ is then calculated from the diff and the sum value. The similarity is an inverse of the distance.

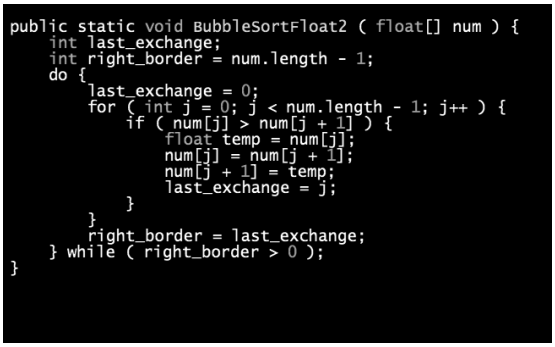
$$\mathrm{distance} = \frac{\mathrm{diff}}{\mathrm{sum}}$$
$$\mathrm{similarity} = 1 - \mathrm{distance}$$

This similarity measure can be considered as Jaccard similarity on images because it quantifies how many pixels are similar between the two images compared to the number of all source code pixels (intersection over union).

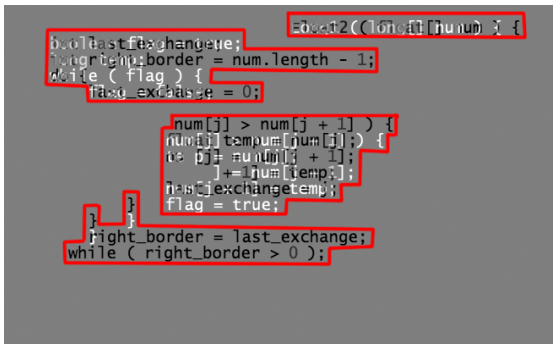
2) The Earth Mover's Distance (EMD): The earth mover's distance (EMD) is a metric that treats image comparison as a transportation problem by finding the minimal cost to transform one distribution into the other. Given signatures of two images (e.g. bins of histogram or texture), one signature is considered as distributions of earth mass spread in one space and another signature is a collection of holes in the same space [21]. EMD measures the least amount of "work" to fill the holes with earth based on a pre-defined "ground distance" between the location of the earth and the hole. From previous studies, EMD is used for document similarity measure based



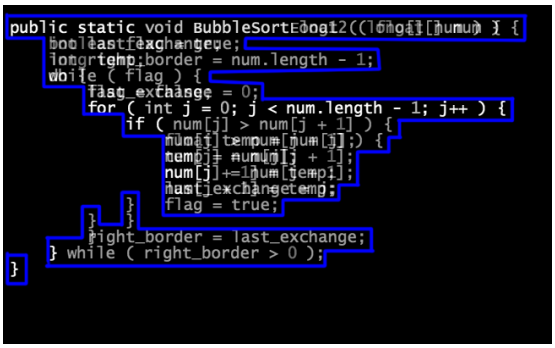
(a) The first method, `BubbleSortLong1`



(b) The second method, `BubbleSortFloat2`



(c) Difference between the two images



(d) Sum of the two images

(e) The first method after adding a Gaussian blur filter (radius=3)

(f) The second method after adding a Gaussian blur filter (radius=3)

Fig. 2: Source code images during the image processing step

on subtopics in the documents [25] and image retrieval based on colours and textures [21].

In this paper, we use EMD to measure distance of two source code images based on distribution of grey-scale colour in source code pixels. We create an n-dimensional signature for each image by dividing an image into n equal rectangles (i.e. regions). For each part, we use a sum of grey-scale values as the weight. The $n \times n$ ground distance matrix of two signature S_1 and S_2 contains the distance between each elements in S_1 to the other element in S_2. The ground distance can be any distance such as Manhattan distance, Euclidean distance, or a custom distance measure. In order to bound the EMD distance between zero and one, we normalise the value with the max EMD distance, i.e. the distance between pure black and pure white images, (denoted $\mathrm{EMD}_{\max}$). Again, we compute this value only from the regions that contain source code pixels. Our EMD similarity is an inverse of the distance:

$$\text{similarity} = 1 - \frac{\mathrm{EMD}(P, Q)}{\mathrm{EMD}_{\max}}$$

C. Image Filters as a Code Normalisation Technique

In text- or token-based code clone detection, one can apply normalisation to the code before performing similarity detection in order to handle clones with modifications. Examples of normalisation techniques include pretty-printing [20], variable renaming [20], [9], [17], and n-gram generation [23].

In image-based clone detection, we are allowed to use a new approach for code normalisation which is not previously possible in the text- or token-based methods. In this paper, we consider using "image filters" as code normalisation techniques. This is similar to the scale-space representation technique in image processing and computer vision [10], [13]

TABLE I: Configurations of Vincent

Parameter	Values	Default	Description
minsize	[0, n]	10	minimum clone size
maxsize	[minsize, n]	100	maximum clone size
similarity	jaccard/emd	jaccard	similarity measure
threshold	[0, 1]	0.25, 0.9985	similarity threshold
blur	on/off	off	add blur to images

which tries to detect similar images with different structures at different scales. We choose a Gaussian blur image filter [14] to normalise the source code images before detection. In our context, blurring code images increases fuzziness in the similarity computation and helps to handle detection of clones with changes in keywords, data types or identifier names, i.e. type-2 clones. Gaussian blur filter works by applying a Gaussian averaging operator to each pixel at coordinate (x, y) in the image. The level of blurriness is controlled by the radius (σ) value representing a radial distance (a square of size $\sigma \times \sigma$ pixels having the target pixel at the centre) that the Gaussian operator will have effects on. Choosing a larger σ value results in a more blurry image than a small value. The code images of `BubbleSortLong1` and `BubbleSortFloat2` after applying a Gaussian blur filter with a radius size of 3 are shown in Figure 2e and Figure 2f respectively. The two processed images are now ineligible and only shapes of the two source code text remain.

D. Vincent: an image-based code clone detector

According to the framework presented in Section II-A, we implemented **Vincent**, an imaged-based code clone detector. Vincent is a method-level code clone detector and currently supports Java. Vincent employs the *JavaParser* library [24] to extract methods from a Java source file and the *highlight* tool [1] to highlight Java source code. Image conversion from HTML to PNG is performed using Python imgkit [7] and the Gaussian blur image filter is done by the *Pillow* Python imaging library [3].

Vincent compare PNG images of size 300 $\times$ 300 pixels. From an empirical analysis, we found that processing source code image at 300 by 300 pixels offers a good balance between the accuracy and speed of detection. The image processing and Jaccard distance is computed using Python's *numpy* package. The earth mover's distance is based on *PyEMD* Python wrapper of Ofir Pele and Michael Werman's C++ implementation [15], [16]. During the EMD computation, we divide an image into 36 squares of size 50 by 50 pixels and create a signature of 36 numbers containing a sum of grey-scale values from the 2,500 pixels within the square. We compute ground distance based on a Euclidean distance between two elements from the two signatures.

Vincent reports clone pairs in two formats: comma-separated value file (CSV), and the XML-based General Clone Format (GCF) [26] file. The tool offers five configurable parameters as explained in Table I. *minsize* is the minimum clone lines (default 10), *maxsize* is the maximum clone lines (default 100), *similarity* is the image distance measure (Jaccard or EMD), *threshold* is the cut-off similarity threshold between cloned/non-cloned pairs (default 0.25 for Jaccard and 0.9985 for EMD),

TABLE II: The generated data set

Data set	Files	#Comparisons	Positives	Negatives
generated	100	10,000	1,000	9,000

TABLE III: Open source software projects

Project	Version	Files	SLOC
JUnit	4.13	203	9,777
JFreeChart	1.5.0	644	96,711
Tomcat	9.0	1,688	241,924

E. Data Sets

We employed two data sets in our evaluation as displayed in Table II. The first data set, called the *generated* data set, is used in our previous study of comparing 30 code similarity analysers [19]. It contains 100 Java source code files with pervasive code modifications. Pervasive modifications are code modifications that are applied to the code globally across the whole file, and contain several changes made one after another. These modifications are found in software plagiarism, code cloning, and code refactoring. The generated data set provides a complete clone ground truth and allows us to measure accuracy, precision, and recall of a given tool. The 100 Java source files forms 1,000 true, and 9,000 false clone pairs at a file level. The data set is constructed using source code and bytecode obfuscators, compiler and decompilers. Hence, it includes challenging clone pairs with modifications at both syntactic (variable renaming, formatting changes, equivalent statement replacements), and semantic level (compiled and decompiled clones with totally different source code).

The second data set consists of three well-known Java open source systems used in another clone study [18]: *JUnit v.4.13*, *JFreeChart v.1.5.0*, and *Apache Tomcat v.9.0*. We removed test classes from the data set to avoid generating too many trivial clones usually found in testing methods. The size of the three systems after removing test code are shown in Table III.

F. Research Questions

We performed an experiment to answer the following research questions.

RQ1 Clone Detection Performance: *How is the precision and recall of our image-based code clone detection compared to other tools?* We evaluated precision, recall, and F1-score of Vincent compared to three clone detectors on the generated data set containing clones with pervasive code modifications.

RQ2 Code Normalisation with Gaussian Blur Filter: *To what extent blurring images increase the detection accuracy?* We compared the performance of Vincent before and after adding image blurring normalisation.

RQ3 Manual Clone analysis: *What are clones that uniquely found and missed by image-based technique?* We

compared the clones found by Vincent and NiCad on three real-world software systems and manually studied the clones that were only reported by Vincent and NiCad.

G. Experimental Design

To measure the tool's performance on locating clones (RQ1, RQ2), Vincent was executed using the default configurations against the generated data set in a file-to-file pairwise comparison manner. This resulted in performing 10,000 comparisons and reporting 10,000 similarity values. Using the ground truth, we could compute score such as precision, recall, accuracy, and F1 score, based on the number of true (TP, TN), and false (FP, FN) results. Moreover, we also search for Vincent's optimal configurations. With the presence of the groundtruth, we varied the tool's parameters and searched for the similarity threshold that gave the highest F1 score.

To qualitatively study clones detected by Vincent (RQ3), we executed the tool on the three open source systems and compared the reported clones with NiCad. NiCad is a text-based clone detector which can detect clone from type-1 to type-3. It preprocesses source code before detecting clones by using pretty-printing, variable renaming, and code abstraction. We chose NiCad because it has been used in several clone studies [19], [26], [22] and it reports clones at method-level, similar to Vincent. Both Vincent and NiCad were configured with the default configurations. We compared clones reported by Vincent and NiCad, and randomly sampled distinct clone pairs that were solely reported by each tool and manually looked at them.

III. Results and Discussion

We performed the experiment on an Ubuntu 16.04.1 machine with two 3.40 GHz processors and 8 GB of RAM. The answers to the research questions are discussed below.

RQ1: Clone Detection Performance

Table IV shows the results from running two versions of Vincent, with Jaccard similarity (denoted Vincent-Jaccard) and with EMD similarity (denoted Vincent-EMD), and other five widely-used code clone detectors including *CCFinderX* [9], *Deckard* [8], *iClones* [4], *NiCad* [20], and *Simian* [5] on the generated data set. For the top half of the table, every tool was configured with its default configurations and the similarity threshold (T) was chosen to retrieve the optimal F1 score. For the bottom half of the table, every tool was tuned to offer its optimal performance based on their F1 score by varying their parameters and similarity threshold.

Vincent-Jaccard gives a comparable results to other clone detectors and was ranked the third with an F1 score of 0.5172 behind the highest ranked tool, Deckard (0.6837) and CCFinderX (0.5772). The third tool is NiCad (0.4369), followed by Simian (0.4195) and iClones (0.4109). Vincent-EMD is ranked the last with an F1 score of 0.3869. We performed a manual checking of false positive and negative clone pairs reported by Vincent-EMD to gain insight on its poor performance. We found that EMD similarity reported false positives from non-cloned source code files with similar layout. Since our EMD is computed from the distributions of grey-scale colour values in 36 non-overlapped regions in the images, it is a coarse-grained similarity measurement. In the case of pervasive code modifications, we found that a finer-grained Jaccard similarity which compares images pixel-by-pixel provides superior results.

Nevertheless, after tuning for the optimal performance, we observed that Vincent-Jaccard and Vincent-EMD were ranked the 6th and 7th while the best performing tool was CCFinderX (0.9760). The F1 score of Vincent-Jaccard (0.6268) was slightly lower than the 5th tool, iClones (0.6345). This finding shows that while other clone detectors were sensitive to configuration change, our image-based clone detection technique obtains only small improvement from parameter tuning.

To answer RQ1, we empirically evaluated the image-based code clone detection technique on code clones with pervasive modifications. We observed that the technique, with its default configurations, reported clones with higher F1 score than iClones, NiCad and Simian. However, our technique offered only tiny improvement of F1 score after parameter tuning and was ranked the last with its optimal configurations. Jaccard similarity is more suitable similarity measure for image-based clone detection than EMD similarity.

RQ2 Code Normalisation with Gaussian Blur Filter

Table V presents the performance of Vincent-Jaccard after adding a Gaussian blur filter to code images before detection. We evaluated 5 different level of blurring radius: 1, 3, 10, 20, and 30. We observed that small blurring radius did harm the detection performance by reporting an F1 score of 0.4993 which is lower than the default configurations without blurring (0.5172). However, after increasing the radius to 3, 10, and 20, the Gaussian blur filter boosted the tool's F1 score to 0.5206 (1% increment), 0.5393 (4% increment), and 0.5478 (6% increment) respectively. Nonetheless, with the radius of 30, the F1 score stopped increasing and dropped to 0.5433.

To answer RQ2, we found that applying a Gaussian blur filter help to increase the image-based clone detection precision, recall, and F1 score. The size of blurring radius also affected the tool's performance. The blurring radius of 20 gave the highest F1 score, followed by a radius of 30, 10, and 3.

RQ3: Manual Clone Analysis

Clones in the three software systems reported by NiCad and Vincent are presented in Table VI. NiCad reported 7; 2,282; 901 clone pairs for JUnit, JFreeChart, and Tomcat respectively. Vincent reported more clones of 9 clone pairs for JUnit, 9,435 clone pairs for JFreeChart, and 1,864 pairs for Tomcat. The execution of Vincent was dramatically higher than NiCad due to several steps of preprocessing and image similarity comparison. NiCad completed the analysis of JUnit, JFreeChart, and Tomcat in 1 second, 6 seconds, and 7 seconds respectively. It took Vincent 1 minutes, 1 hour and 16 minutes, and 5 hours 31 minutes to complete the same task.

TABLE IV: Vincent performance compared to other clone detectors on the *generated* data set

Tool	Configurations	T	TP	FP	TN	FN	Precision	Recall	Accuracy	F1
Default configurations										
CCFinderX	b=50, t=12	8	417	28	8,972	583	0.9371	0.4170	0.9389	0.5772
Deckard	mintoken=50, stride=inf, similarity=1.0	5	536	32	8,968	464	0.9437	0.5360	0.9504	0.6837
iClones	minblock=20, minclone=100	0	181	0	4,200	519	1.0000	0.2586	0.8941	0.4109
NiCad	UPI=0.30, minline=10	7	284	16	8,984	716	0.9467	0.2840	0.9268	0.4369
Simian	threshold=6	0	276	40	8960	724	0.8734	0.2760	0.9236	0.4195
Vincent-Jaccard	threshold=0.25	6	376	78	8,922	624	0.8282	0.3760	0.9298	0.5172
Vincent-EMD	threshold=0.9985	7	260	84	8,916	740	0.7558	0.2600	0.9176	0.3869
Optimised configurations										
CCFinderX	b=5, t=11	36	976	24	8,976	24	0.9952	0.9760	0.9760	0.9760
Deckard	mintoken=30, stride=2, similarity=0.95	17	773	44	8,956	227	0.9729	0.9461	0.7730	0.8509
iClones	minblock=10, minclone=50	0	342	36	4,164	358	0.9196	0.9048	0.4886	0.6345
NiCad	UPI=0.50, minline=8 rename=blind, abstract=literal	38	654	38	8,962	346	0.9616	0.9451	0.6540	0.7730
Simian	threshold=4, ignoreVariableNames	5	835	150	8,850	165	0.9685	0.8477	0.8350	0.8413
Vincent-Jaccard	threshold=0.15, GBRadius=20	16	492	77	9123	509	0.8647	0.4915	0.9426	0.6268
Vincent-EMD	threshold=0.9975, GBRadius=20	22	339	405	8,595	661	0.4556	0.3390	0.8934	0.3888

TABLE V: F1 scores of Vincent-Jaccard (threshold=0.25) with Gaussian blur filter

Blur radius	N/A	1	3	10	20	30
F1 score	0.5172	0.4993	0.5206	0.5393	0.5487	0.5433

We compared the clone pairs reported by the two tools and categorised them into three groups: (1) *Common* are clone pairs reported by both tools, (2) *NiCad only* are the pairs solely reported by NiCad, and (3) *Vincent only* are clone pairs reported only by Vincent. The number of clone pairs in the three groups are shown in Table VI. For JUnit, there were 3 common pairs, 4 pairs only reported by NiCad, and 5 pairs only reported by Vincent. For JFreeChart, 1,284 pairs were common, 998 pairs were reported only by NiCad, and 8,151 pairs were reported only by Vincent. Lastly, for Tomcat, common pairs contained 604 pairs. There were 297 and 1,260 NiCad-only and Vincent-only pairs respectively.

We manually checked a sample of clone pairs from the three systems in order to gain insights on the characteristics of code clones that were detected by either Vincent or NiCad. The results of JUnit were feasible for a complete manual investigation so we checked all the 3 NiCad-only and 4 Vincent-only pairs. However, the number of distinct clone pairs in JFreeChart and Tomcat were large. Thus, we applied a random sampling of 100 clone pairs for the manual checking. Half of them (50 pairs) were NiCad-only and the other half (50 pairs) were Vincent-only pairs. The first author took a role of the investigator and performed the clone investigation. He validated if a given clone pair candidate was a true clone pair or not and also studied why they were only reported by a single tool. The manual investigation results are also explained in Table VI. For JUnit, the 3 distinct clone pairs reported by NiCad and the 5 distinct Vincent pairs were all true positive. For JFreeChart, the sampled 50 NiCad-only candidate pairs contained 50 true clone pairs and no false clone pair. The true clone pairs included 33 type-3 pairs, and 17 type-2 pairs. On the other hand, the sample 50 Vincent-only candidate pairs consisted of 45 true and 5 false pairs. The true clone pairs included 42 type-3 pairs and 3 type-2 pairs. For Tomcat, the sampled 50 NiCad-only candidate pairs contained 50 true positive (27 type-3 and 23 type-2 pairs) and no false positive pair. The 50 Vincent-only pairs consisted of 43 true positive (27 type-3 and 16 type-2 clones) and 7 false positive pairs. Most of Vincent's false clone pairs were caused by methods with first few lines being similar or identical while the rest were different. Several NiCad-only pairs were type-3 clones with relocated statements which were difficult to detect using our technique. On the contrary, Vincent-only pairs were clones with similar structure but had low similarity when compared using NiCad's algorithm. All the manually validated clone pairs can be found on the study website[2].

To answer RQ3, we observed that the image-based clone detection technique could detect the same clones that were reported by the text-based clone detector, NiCad, in three real-world Java software systems. Moreover, the manual investigation of sampled clone pairs showed that our technique could report true clone pairs that were missed by NiCad but also suffered from some false positives.

IV. Threats to Validity

The evaluation of the tools' performance is based on a data set of Java code with pervasive modifications. Although the data set contains several types of code modifications covering the use cases of code cloning, software plagiarism, and refactoring, it may not be generalised to all cloned code and to other languages besides Java. We performed a comparison of Vincent to five clone detectors based on their default and optimised configurations. Although we searched for a wide range of parameter values, we might not cover all possible configurations of the tools. Lastly, the manual investigation is performed by only the first author. Although it was carefully executed, it might still be subject to human errors.

[2]The study website is at https://ucl-crest.github.io/iwsc2018-vincent-web

TABLE VI: Comparison of clones found by NiCad and Vincent (default configurations)

System	NiCad		Vincent		Common	NiCad only						Vincent only					
						All	Manual					All	Manual				
	Time	Clones	Time	Clones			TP	T3	T2	T1	FP		TP	T3	T2	T1	FP
JUnit	1s	7	1m	9	4	3	3	3	0	0	0	5	5	5	0	0	0
JFreeChart	7s	2,282	1h 16m	9,435	1,284	998	50	33	17	0	0	8,151	45	42	3	0	5
Tomcat	16s	901	5h 31m	1,864	604	297	50	27	23	0	0	1,260	43	27	16	0	7

V. Related Work

Several clone detectors, both for source code and binary code, have been introduced in the literature. Many of them exploit string comparison techniques such as Simian [5], and NiCad [20]. Other tools transform source code into an intermediate representation to measure similarity. Examples are token-based clone detection tools, e.g. SourcererCC [22], CCFinderX [9], JPlag [17], and iClones [4], AST-based clone detection tools, CloneDR [2] and Deckard [8], and graph-based clone detection tools [12], [11]. In this paper, we introduce a new representation of source code by using images.

Code normalisation enhances similarity of cloned code fragments by changing their layouts, identifier names, or statements into a standard format or by transforming the code into another representation. Token-based clone detectors convert source code text into token streams [9]. Tree-based tool converts a program into abstract syntax trees [2], [8]. Compilation/decompilation can also be used as a code normalisation in Java [19]. NiCad [20] uses TXL with pretty printing as a code normaliser. In this study, we applied a Gaussian blur filter as a code normaliser before comparing two code images.

There are studies on measuring document similarity based on their images [6], [21]. Hu et al. [6] compute visual similarity of two document images based on a feature set, called interval encoding, which is more accurate than computing distances between blocks in the images (e.g. Manhattan distance). Rubner et al. [21] used EMD for image retrieval tasks based on colour and texture. In this paper, we deployed EMD as a code image similarity measure based on grey-scale colour distributions in image subregions.

VI. Conclusion

This paper presents an image-based code clone detection technique and a tool called Vincent. The tool compares code fragments based on their visual representation, which resembles how programmers manually looks for clones. We showed that Jaccard similarity is preferred over the earth mover's distance as a similarity measure for our technique. Moreover, we found that applying a Gaussian blur filter to source code images before performing clone detection can increase an F1 score of Vincent by 6 percent. The performance of Vincent on a data set of code clones with pervasive code modifications was comparable to CCFinderX, Deckard, iClones, NiCad, and Simian. A manual comparison of clones from three software projects reported by Vincent and NiCad showed that the two tools, while agreed on some clone pairs, also discovered additional clone pairs that were missed by another tool.

References

[1] S. Andre. Highlight 3.41. http://www.andre-simon.de, 2017. Online; access 17-Dec-2017.

[2] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *ICSM '98*, volume 98, pages 368–377, 1998.

[3] A. Clark. Pillow: ImageFilter module. http://pillow.readthedocs.io/en/latest/reference/ImageFilter.html, 2017. Online; access 17-Dec-2017.

[4] N. Göde and R. Koschke. Incremental clone detection. In *CSMR'09*, pages 219–228, 2009.

[5] S. Harris. Simian – similarity analyser, version 2.4. http://www.harukizaemon.com/simian/, 2015. Accessed: 2016-02-14.

[6] J. H. J. Hu, R. Kashi, and G. Wilfong. Document image layout comparison and classification. pages 285–288, 1999.

[7] K. Jia. Python imgkit. http://www.andre-simon.de, 2017. Online; access 17-Dec-2017.

[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. DECKARD: Scalable and accurate tree-based detection of code clones. In *ICSE'07*, pages 96–105, 2007.

[9] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *Transactions on Software Engineering*, 28(7):654–670, 2002.

[10] J. J. Koenderink. The structure of images. *Biological Cybernetics*, 50(5):363–370, Aug 1984.

[11] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *SAS'01*, pages 40–56, 2001.

[12] J. Krinke. Identifying similar code with program dependence graphs. In *WCRE '01*, 2001.

[13] T. Lindeberg. Scale-space theory: a basic tool for analyzing structures at different scales. 21(1):225–270, 1994.

[14] M. S. Nixon and A. S. Aguado. *Feature Extraction and Image Processing*. 2002.

[15] O. Pele and M. Werman. A linear time histogram metric for improved sift matching. In *ECCV 2008*, pages 495–508. Springer, October 2008.

[16] O. Pele and M. Werman. Fast and robust earth mover's distances. In *ICCV '09*, pages 460–467. IEEE, September 2009.

[17] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002.

[18] C. Ragkhitwetsagul and J. Krinke. Using compilation / decompilation to enhance clone detection. In *IWSC'17*, 2017.

[19] C. Ragkhitwetsagul, J. Krinke, and D. Clark. A comparison of code similarity analysers. *Empirical Software Engineering*, 2017.

[20] C. K. Roy and J. R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *ICPC'08*, pages 172–181, 2008.

[21] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 40(2):99–121, 2000.

[22] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes. SourcererCC: Scaling code clone detection to big-code. In *ICSE '16*, pages 1157–1168, 2016.

[23] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *SIGMOD '03*, page 76, 2003.

[24] D. van Bruggen. JavaParser. http://javaparser.org, 2017. Online; access 17-Dec-2017.

[25] X. Wan. A novel document similarity measure based on earth mover's distance. *Information Sciences*, 177(18):3718–3730, 2007.

[26] T. Wang, M. Harman, Y. Jia, and J. Krinke. Searching for better configurations: A rigorous approach to clone evaluation. In *FSE' 13*, pages 455–465, 2013.