A Dataset of Clone References with Gaps

Hiroaki Murakami, Yoshiki Higo, and Shinji Kusumoto
Osaka University
1-5 Yamadaoka, Suita, Osaka, Japan
{h-murakm,higo,kusumoto}@ist.osaka-u.ac.jp

ABSTRACT

This paper introduces a new dataset of clone references, which is a set of correct clones consisting of their locational information with their gapped lines. Bellon's dataset is one of widely used clone datasets. Bellon's dataset contains many clone references, thus the dataset is useful for comparing accuracies among clone detectors. However, Bellon's dataset does not have locational information of gapped lines. Thus, Bellon's benchmark does not evaluate some Type-3 clones correctly. In order to resolve the problem, we added locational information of gapped lines to Bellon's dataset. The new dataset is available at "`http://sdl.ist.osaka-u.ac.jp/~h-murakm/2014_clone_references_with_gaps/`".

This paper also shows some examples that the new dataset and Bellon's dataset yield different evaluation results. Moreover, we report an experimental result that compares Bellon's dataset and the new dataset by using three clone detectors that can detect Type-3 clones. Finally, we conclude that the new dataset can evaluate Type-3 clones more correctly than Bellon's dataset.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; E.5 [**Data**]: Files

General Terms

Design, Measurement

Keywords

Code clone, Software maintenance, Dataset

1. INTRODUCTION

Many software systems have code clones. Code clones are identical or similar code fragments to one another in source code. Recent research revealed that some code clones have negative impacts on software maintenance [2][3]. In order to find code clones in software systems, many clone detectors have been developed. Since each clone detector has its own definition of code clones, different code clones are detected by different clone detectors from the same source code [7].

Bellon et al. compared six clone detectors in order to reveal characteristics of them [1]. He made references of code clones, and compared accuracies among clone detectors. The references represent locational information of code clones with file name and start/end lines. In his benchmark, code clones are categorized into the following three types.

Type-1 is an exact copy without modifications (except for white space and comments).

Type-2 is a syntactically identical copy; only variable, type, or function identifiers were changed.

Type-3 is a copy with further modifications; statements were changed, added, or removed.

Programmers often make some changes to code fragments after cloning [5]. Moreover, cloned fragments often evolve differently from the original fragments [3]. These facts indicate that there often exists some gaps between the original code fragments and pasted fragments. Thus, it is important to find Type-3 clones for software maintenance.

However, Bellon's Type-3 clones do not have locational information of gapped lines. In other words, Bellon's references represent code clones only with information of line numbers. We do not consider that gapped lines in code clones should be regarded as code clones. Thus, Bellon's benchmark does not evaluate some Type-3 clones correctly. Therefore, we remade the references of code clones with locational information of gapped lines. Moreover, we conducted experiments to indicate differences between Bellon's references and the new references by using three clone detectors that can detect Type-3 clones.

Contributions of this paper are as following.

- We added locational information of gapped lines to Bellon's references of code clones and made it public.
- It is revealed that our dataset produces different results from Bellon's dataset.

2. BELLON BENCHMARK

Bellon's benchmark is one of the most famous benchmark in clone community. He compared six clone detectors from the perspective of accuracy and performance. He made the benchmark with the following steps.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597133

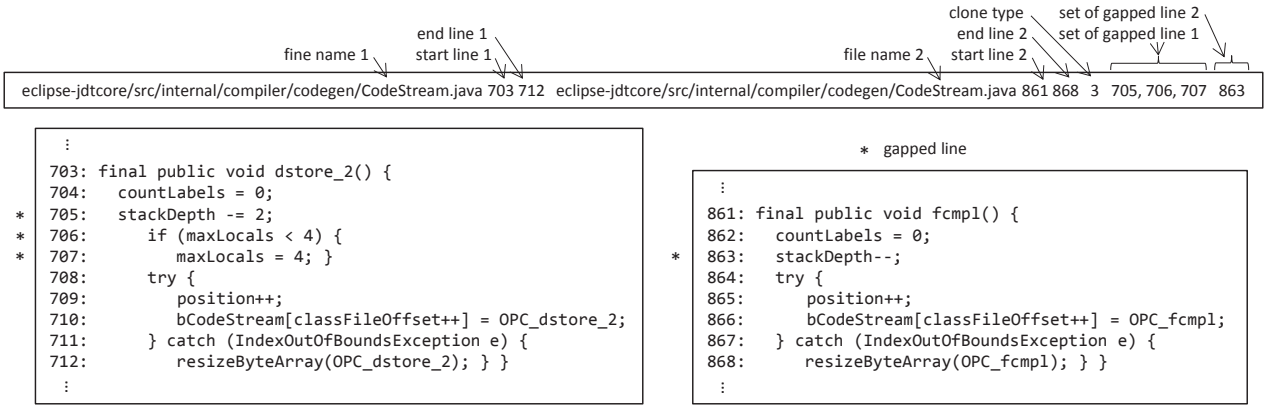


Figure 1: An example of the new dataset and correspondent source files

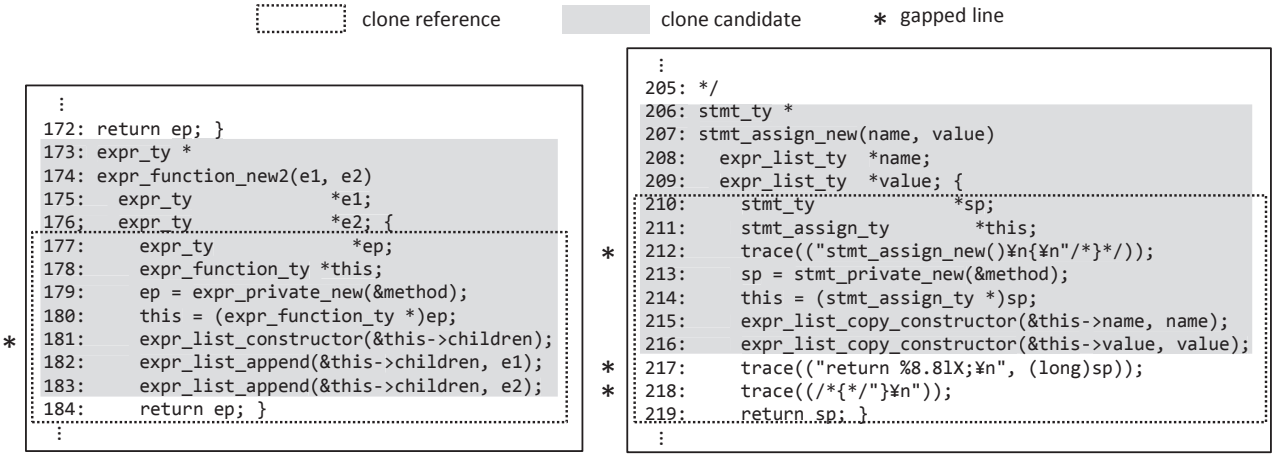


Figure 2: An example of reference-candidate match with gapped line

Step-1: Bellon selected eight software systems as target systems, and six clone detectors as target detectors.

Step-2: He asked the developers of the clone detectors to detect code clones from the software systems. Then the developers sent the locational information of the detected code clones to Bellon.

Step-3: 2% of the code clones sent from the developers were randomly selected, then he checked each of them manually whether it was actually code clone or not.

In the remainder of this paper, we use the following terms.
Clone candidates: code clones found by clone detectors.
Clone references: code clones judged by human manually.

In the Bellon's benchmark, *ok* and *good* value are defined. For definitions of *ok* and *good* value, refer to the literature [1]. These two values decide whether every *clone candidate* matches any of *clone references* or not. The *ok* value means intuitively the overlapping ratio of a *clone candidate* and a *clone reference*. The more *ok* value increases, the more the overlapping part of a *clone candidate* and a *clone reference* becomes large. Meanwhile, the *good* value is much more restrictive for a candidate-reference match.

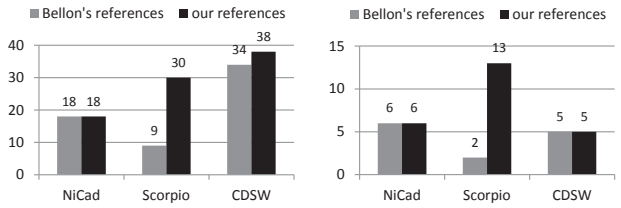
3. DATASET

We made a new dataset by adding locational information of gapped lines to Bellon's dataset. The new dataset is available at our website[1]. Furthermore, we put file format of our *clone references* on the same website. Fig. 1 illustrates an example of our *clone reference* and the correspondent source files. The left source file has three gapped lines (lines 705, 706 and 707), and the right one has a single gapped line (line 863). If our *clone references* are used in accuracy evaluation, the evaluation can take into account gapped lines in code clones.

On the other hand, Bellon's *clone references* represents code clones with file name and start/end lines. Thus, Bellon's benchmark does not evaluate some Type-3 clones correctly. If our *clone references* are used in accuracy evaluation, *ok* and *good* value may be changed from the case of Bellon's *clone references*. Our *clone references* can evaluate Type-3 clones more correctly than Bellon's *clone references* because our *clone references* introduce information of gapped lines into evaluation of code clones.

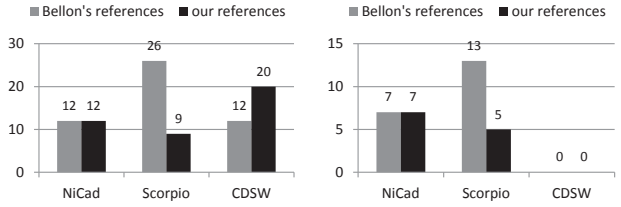
Herein, we explain the case that *ok* value gets changed. Fig. 2 shows that there are two clone pairs. One clone

[1] `http://sdl.ist.osaka-u.ac.jp/~h-murakm/2014_clone_references_with_gaps/`



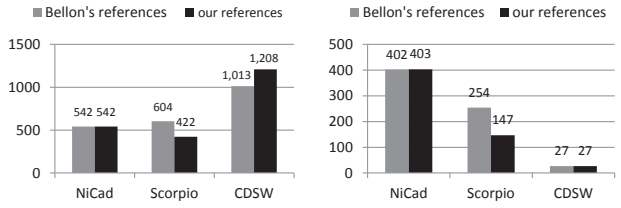
(a) *ok* (b) *good*

Figure 3: Experimental results for `netbeans`



(a) *ok* (b) *good*

Figure 4: Experimental results for `ant`



(a) *ok* (b) *good*

Figure 5: Experimental results for `jdtcore`

pair is one of *clone references*, and the other is one of *clone candidates* that our clone detector CDSW [6] detected. If *ok* value is calculated for this example by using Bellon's *clone references*, *ok* value is obtained as follows:

$$\begin{aligned} ok &= min(max(\frac{7}{8}, \frac{7}{11}), max(\frac{7}{10}, \frac{7}{11})) \\ &= 0.7 \end{aligned}$$

On the other hand, in the case of using our *clone references*, *ok* value is obtained as follows:

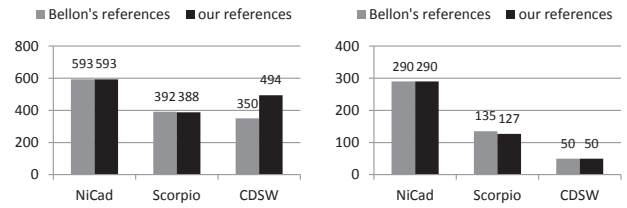
$$\begin{aligned} ok &= min(max(\frac{6}{7}, \frac{6}{10}), max(\frac{6}{7}, \frac{6}{10})) \\ &\fallingdotseq 0.857 \end{aligned}$$

In this way, *ok* value is changed if our *clone references* are used. Accordingly, the evaluation scale (e.g. *recall*, *precision* and *F-measure*) are might be changed.

4. EXPERIMENT

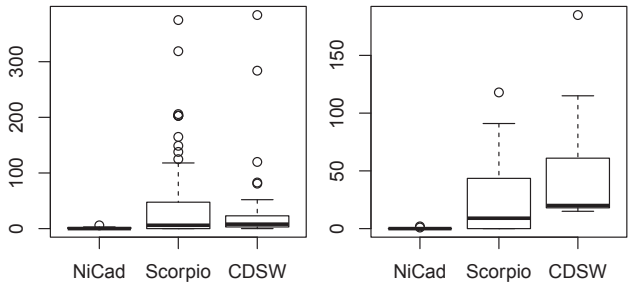
The purpose of this experiment is to reveal differences of evaluation results yielded by Bellon's *clone references* and the new *clone references*. In this experiment, we compared the number of detected *clone references* based on a threshold. We used 0.7 as the threshold, which was the same value used in Bellon's benchmark.

In this experiment, we chose the clone detectors shown in Table 1 as targets for the comparison. All of them can detect



(a) *ok* (b) *good*

Figure 6: Experimental results for `swing`



(a) `netbeans` (b) `ant`

Figure 7: Variabilities of gapped lines

Type-3 clones. All the clone detectors were used in their default configurations. We used four Java systems shown in Table 2 as targets software systems. The Java systems were also used in Bellon's experiments.

Fig. 3, Fig. 4, Fig. 5 and Fig. 6 show experimental results for each target software system. Each graph represents the number of detected *clone references* whose *ok* and *good* values exceed the threshold. In the case of NiCad, the number of the detected *clone references* was hardly changed. CDSW increased the number of the detected *clone references* in half the cases. Scorpio produced interesting results. By using our *clone references* for evaluation, the number of the detected *clone references* was increased in the case of `netbeans`. On the other hand, in the case of the others, the number of the detected *clone references* was decreased.

Fig. 7 shows variabilities of gapped lines in *clone candidates* among the target clone detectors for `netbeans` and `ant`. NiCad has narrow variability, meanwhile Scorpio and CDSW have wider one. Fig. 3, Fig. 4 and Fig. 7 reveal that our *clone references* has a strong effect on clone detectors

Table 1: Clone detectors used in this experiment

Developer	Clone detector	Detection method
Roy	NiCad [8]	text-based
Higo	Scorpio [4]	PDG-based
Murakami	CDSW [6]	statement-based

Table 2: Target software systems

Name	Language	# Files	Lines of code
`netbeans`	Java	101	14,360
`ant`	Java	178	34,744
`jdtcore`	Java	741	147,634
`swing`	Java	538	204,037

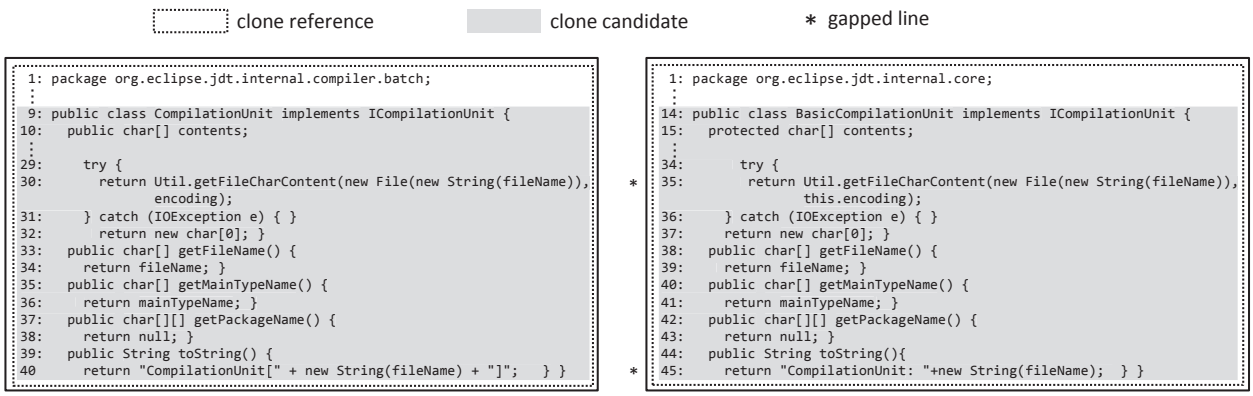


Figure 8: A *clone reference* that becomes to be undetected when our *clone references* are used for evaluation

that detect Type-3 code clones having many gapped lines.

Fig. 8 shows one of examples of *clone references* that becomes to be undetected when our *clone reference* is used for evaluation. The left source file has 40 lines of *clone reference* (lines 1-40), and 32 lines of *clone candidate* (lines 9-40), and 2 gapped lines (lines 30, 40). The right source file has 45 lines of *clone reference* (lines 1-45), and 32 lines of *clone candidate* (lines 14-45), and 2 gapped lines (lines 35, 45). In this case, *good* value was calculated as follows when Bellon's *clone references* were used.

$$\begin{aligned} good &= min(\frac{32}{40}, \frac{32}{45}) \\ &\fallingdotseq 0.711 \end{aligned}$$

On the other hand, *good* value is calculated as follows when our *clone references* are used.

$$\begin{aligned} good &= min(\frac{30}{38}, \frac{30}{43}) \\ &\fallingdotseq 0.697 \end{aligned}$$

In this case, *good* value fell below 0.7 when our *clone references* were used. Thus, these *clone candidates* did not match any of *clone references* in *good* value evaluation. In this way, not all Type-3 clones increase their *ok*/*good* values when our *clone references* are used.

The experimental result is summarized as follows. All the target clone detectors achieved different results. Some detectors yielded almost unchanged results, and others made changes significantly. These results rely on the amount of gapped lines in code clones. Thus, an evaluation using the locational information of gapped lines will be a new evaluation of Type-3 clones.

5. CONCLUSION

This paper introduces a new dataset containing locational information of code clones with their gapped lines. Furthermore, we explained the example that the evaluation results are changed when our dataset was used. Lastly, the experiment proved that our dataset achieved different outcomes from Bellon's dataset by using some clone detectors. We hope that our dataset will diversify an evaluation of Type-3 clones.

In the future, we are going to conduct strictly experiments using the new dataset. If our dataset is used for evaluating Type-3 clones, more accurate results would be obtained.

6. ACKNOWLEDGMENTS

This study was supported by Grants-in-Aid for Scientific Research (S) (25220003), Grant-in-Aid for Exploratory Research (24650011) from the Japan Society for the Promotion of Science, and Grant-in-Aid for Young Scientists (A) (24680002) from the Ministry of Education, Culture, Sports, Science and Technology.

7. REFERENCES

[1] S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Trans. on Software Engineering*, 31(10):804–818, October 2007.

[2] N. Bettenburg, W. Shang, W. M. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan. An empirical study on inconsistent changes to code clones at release level. In *Proc. of the 16th Working Conference on Reverse Engineering*, pages 85–94, October 2009.

[3] N. Göde and R. Koschke. Frequency and risks of changes to clones. In *Proc. of the 33rd International Conference on Software Engineering*, pages 311–320, May 2011.

[4] Y. Higo and S. Kusumoto. Code clone detection on specialized pdgs with heuristics. In *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, pages 75–84, March 2011.

[5] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Proc. of the 3rd International Symposium on Empirical Software Engineering*, pages 83–92, August 2004.

[6] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto. Gapped code clone detection with lightweight source code analysis. In *Proc. of the 21st International Conference on Program Comprehension*, pages 93–102, May 2013.

[7] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *Technical Report No. 2007-541, Queen's University*, 2007.

[8] C. K. Roy and J. R. Cordy. Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proc. of the 16th International Conference on Program Comprehension*, pages 172–181, June 2008.