Clone Detection Method Selection Based on Bandit Algorithm: a Preliminary Analysis

Takeru Tabo
Kindai University
Higashi-osaka, Japan

Takuto Kudo
Kindai University
Higashi-osaka, Japan
1910370079g@kindai.ac.jp

Masateru Tsunoda
Kindai University
Higashi-osaka, Japan
tsunoda@info.kindai.ac.jp

Amjed Tahir
Massey University
Palmerston North, New Zealand
a.tahir@massey.ac.nz

Kwabena Ebo Bennin
Wageningen UR
Wageningen, Netherlands
kwabena.bennin@wur.nl

Koji Toda
Fukuoka Institute of Technology
Fukuoka, Japan
toda@fit.ac.jp

Keitaro Nakasai
Kagoshima College, NIT
Kirishima, Japan
nakasai@kagoshima-ct.ac.jp

Akito Monden
Okayama University
Okayama, Japan
monden@okayama-u.ac.jp

***Abstract*—Various clone detection methods have been proposed in the past, with results varying depending on the combination of the methods and parameters setting used (i.e., configurations). To help with the selection of a suitable clone detection configuration, we propose Bandit Algorithm (BA) approach that can help in evaluating the configuration used dynamically while using detection methods. Our preliminary analysis showed that our approach is able to identify the best configurations from four used code clone detection methods.**

Keywords—online optimization, multi-armed bandit, external validity, duplicated code

I. Introduction

Copy past existing code is a very common practice in software development [1], which can lead to code clones. There are many purposes to modify the clones (e.g., refactoring and bug fixing) [1]. The recommendation is to remove clones from the code to avoid potential problems. The standard process followed to modify cloned code is as follows:

1. Code clones are detected by tools.
2. Developers judge whether they should modify the code, if yes,
3. The clones are modified or removed.

Different clone detection methods have been proposed in the past. To help with the selection of a suitable method, a previous study [1] compared the accuracy of different detection methods. The accuracy of the detection methods usually depends on the parameter settings of the method (i.e., configurations).

To select clone detection tool configuration, considering its accuracy, we propose to collect the evaluation of detected clones done by developers. Before this process takes place, we used several tools to detect code clones. On step 1, a clone candidate detected by a tool is shown to a developer. When a developer decided that a clone is valid, the developer will rate the tool which showed the candidate as "valid" (e.g., rate it by clicking a thumbs-up icon). On the other hand, when the developer judges the clone as not valid, then they will rate it as "invalid." This evaluation procedure is similar to the rating used in many recommender systems. According to the accumulation of the evaluation, the best possible configuration will then be identified.

To speed up this selection process, we integrate Bandit algorithm (BA) to the evaluation process. BA is often explained through an analogy with slot machines (they are referred to as **arms**). Assume that a player has 100 coins to bet on several arms, and the player wants to maximize their reward. BA may suggest that the player to bet only one coin on each arm to seek the best chances. Intuitively speaking, our approach regards different configurations as arms, and the evaluation of each configuration by developers are the rewards.

II. Selection of Detection Tool Configuration by BA

To apply our approach, n clone detection tool configuration (n is a natural number) are selected at first. It is recommended to set n to six configurations or less, as BA performs best when the number of arms is set to six [2]. Next, clone candidates are detected using the different configurations. The candidates shown by each configuration are treated as arms (with n arms are created). For instance, in Fig. 1, ac1, ac2, ac3..., and bc1, bc2, bc3... are the clone candidates, spread across two arms.

After the above preparation, the developers iteratively evaluate whether code candidates are valid clones m times (m is a natural number). BA seeks the best configuration as follows:

N1. Select an arm based on the average reward of arms.
N2. A developer input the evaluation whether a clone candidate shown by the selected arm is valid (or not) to the system.
N3. Recalculate the average reward of each arm, based on the evaluation on step N2.

Our method replaces step 1 and 2 as step N1 and N2. Step N3 is newly added after step 3. We show a concrete example of behavior of our method from the first to third iterations. Initially, the average reward of all arms is set to zero. Hence, for the first iteration in Fig. 1, arm A is selected randomly in step N1. In step N2, a developer evaluates weather a clone candidate ac1 is a valid clone. In step N3, the reward is then set to -1, since the developer does not regard it as "valid." Based on the evaluation, we calculate the average reward for each arm. It shows the bottom row of each table in Fig. 1. For the second iteration, arm B is selected in step N1, because the average reward of arm B is higher than A. In step N2, the developer regards a clone candidate bc1 as "valid." We then perform step 3. In step N3, based on the evaluation, the reward is set to 1, since the developer regards it as "valid." Likewise, the third iteration is performed, and arm B is selected based on the average reward.

As explained above, there are many valid reasons for modifying/removing clones, and depending on the purpose, the method used might differ [1]. Thus, it can be difficult to judge the validity of candidates in step N2 without a human judgement, though visualization tools such as CCFinderX (www.ccfinder.net) are useful in reducing the effort required.

III. Preliminary Analysis

We conducted a preliminary analysis on our approach based on the result reported in Section 5.5.1 of study [1]. The study generated a dataset that includes 10,000 clone candidates, with 10% of those candidates are true clones. We performed the analysis, assuming that developers evaluate the true clones as valid in step N2. The study ranked 30 tool configurations, based on F-score. We selected the first, 10th, 20th, 30th ranked configurations from the list, as shown in Table I. That is, four arms were created. On each arm, the probability of appearance of valid clone candidates is the same as the precision shown in Table 1. In the analysis, we set the following research questions:

RQ1. Can our approach select the best available configuration?
RQ2. How many times should developers evaluate the configuration candidates in order to identify the best one?
RQ3. To what extent does the accuracy degrades by the evaluation?

Our proposed approach requires developers' evaluation. RQ2 can help in quantifying the level of evaluation required by developers. Candidates by non-best configurations are shown to developers in step N2, and that might degrade the detection accuracy. RQ3 is set to clarify the extent of the degradation.

For our BA setup, we used ε-greedy (ε = 0.1, 0.2 and 0.3), UCB (upper confidence bounds), and TS (Thompson sampling). The calculation of average rewards in step N3 differs among algorithms. The order of clone candidates on each arm could affect arm selection by BA. For instance, when the order is bc2, bc1, bc3... on arm B, then arm B might not be selected. To suppress influence of the order, we randomly changed the order 20 times. As the baseline, we compared BA with the case when the configurations are randomly selected.

In the analysis, 200 iterations of BA were performed. We analyzed how many times the best possible configuration (i.e., ccfx based on study [1]) was selected, picking up successive 10 iterations within the 200 iterations, to answer RQ1. We call the number of the selection "% of the best config selection". For instance, when ccfx is selected seven times from the 91st to 100th iterations, we regarded the percentage as 70% at 100th iteration (it was calculated based on the moving average). In Fig. 2, when TS is used, the percentage increases to more than 90% after the 150th iterations. Hence, to answer RQ1, it is possible to use BA to select the best configurations from a number of available ones. The random selection theoretically selects the best arm at 25% probability (in the presence of four arms).

Note that our approach does not calculate F-scores as we do not really know the number of false-negative in Fig. 1. Nevertheless, the approach selected the best possible configuration. This is because, in general, if the precision is high, F-score tend to be high as well (in Table 1). We plan to consider F-score in our future work.

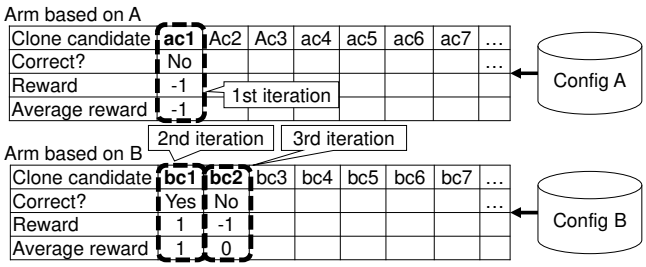

Fig. 1. Selection of detection tool configuration based on BA.

TABLE I. Used Configurations and their Performance [1]

Tool	Settings	Precision	Recall	F-score	Rank
Ccfx	b=5, t=11	0.98	0.98	0.98	1
ncd-zlib	N/A	0.88	0.79	0.84	10
7zncd-Deflate	mx=7	0.87	0.79	0.82	20
Bsdifa	N/A	0.68	0.42	0.52	30

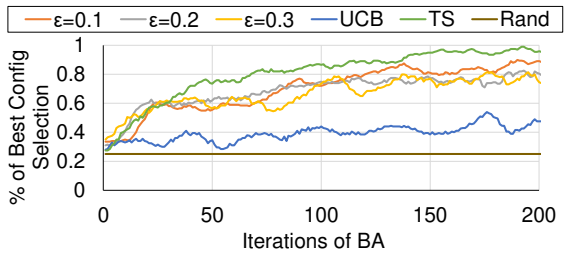

Fig. 2. Relationship between iterations of BA and % of best config selection.

Based on Fig. 2, the answer of RQ2 is, developers will need a minimum of 150 evaluations in order to identify and select the best possible configuration. When there are multiple developers working together (e.g., five), each developer should evaluate clone candidates 30 times to identify the best configuration.

In answering RQ3, we focused on the precision of BA. Note that we can only evaluate positive results (i.e., precision), since the actual number of false-negative cannot be known as explained above. When the number of evaluations is 150 and TS is used, the precision of BA was 0.89. The accuracy of the random selection becomes 0.85, which is average precision of the four arms. Therefore, although the accuracy of our method was lower than the highest configurations (i.e., ccfx), it was still higher than the second one (i.e., ncd-zlib) and the baseline (i.e., random selection).

Acknowledgment

This research is partially supported by the Japan Society for the Promotion of Science (No.21K11840 and No. 20H05706).

References

[1] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," Empirical Software Engineering, vol.23, pp.2464-2519, 2018.

[2] M. Tsunoda, A. Monden, K. Toda, A. Tahir, K. Bennin, K. Nakasai, M. Nagura, and K. Matsumoto, "Using Bandit Algorithms for Selecting Feature Reduction Techniques in Software Defect Prediction," Proc. Int. Conf. on Mining Software Repositories (MSR), pp.670-681, 2022.