Journal of King Saud University – Computer and Information Sciences 31 (2019) 327–334

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

IR-based technique for linearizing abstract method invocation in plagiarism-suspected source code pair

Oscar Karnalim *

Faculty of Information Technology, Maranatha Christian University, Indonesia

ARTICLE INFO

Article history:
Received 20 December 2017
Revised 16 January 2018
Accepted 31 January 2018
Available online 2 February 2018

Keywords:
Abstract method linearization
Source code plagiarism
Software engineering
Information retrieval

ABSTRACT

According to several works, low-level approach is an effective and efficient solution for detecting source code plagiarism. Instead of relying on source code tokens, it compares the executable form of given code; that form only contains semantic-preserving tokens and is resistant to various plagiarism attacks. However, to our knowledge, an issue about statically linearizing abstract method (i.e. replacing each abstract method invocation with its respective invoked method content without considering invocation semantic) has not been handled comprehensively. Such issue, at some extent, will generate inaccurate plagiarism detection result when handling object-oriented source codes. This paper aims to solve such issue locally per plagiarism-suspected pair. It will generate all possible linearization pair alternatives and select the correct one through IR-based similarity. According to our evaluation regarding the reversed number of mismatched token, the number of false positive, and the number of process, proposed technique is more effective and efficient when compared to state-of-the-art and combinatoric technique. In addition, it is also observed that each IR mechanism used in proposed technique has its own exclusive benefit for selecting the correct linearized forms.

© 2018 The Author. Production and hosting by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Low-Level structure-based Approach (LLA) is one of source code plagiarism detection approaches that relies on executable form to determine similarity. Instead of comparing source code token sequences directly (Karnalim, 2016a), it compiles both source codes and quantifies similarity degree based on resulted low-level token sequences. According to several works (Karnalim, 2016a, 2017a, b; Rabbani and Karnalim, 2017), this approach outperforms a popular approach that relies on source code token sequence similarity in terms of effectiveness and efficiency.

In the matter of scope, some LLAs (Karnalim, 2016a, 2017a, b) are able to handle object-oriented codes; they consider object-oriented aspects such as class, method, and attributes. However, some object-oriented issues are still handled naively, resulting inaccurate plagiarism detection result. One of the issues is about linearizing abstract method invocation: without considering the whole program semantic, each abstract method invocation should be linearized while some of them may be derived to more than one alternative due to polymorphism.

This paper proposes a technique for solving aforementioned issue. Different with previous techniques (Karnalim, 2016a, 2017a, b), our proposed technique keeps the semantic of linearized abstract method invocation. Our proposed technique works for each plagiarism-suspected pair in twofold. First of all, it will enlist all possible linearization alternatives in combinatoric manner. Afterward, it will select the correct linearization pair based on the highest information-retrieval-based similarity degree.

2. Related works

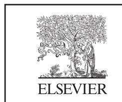
Source code plagiarism in an activity of using existing work without properly citing the original author (Cosma and Joy, 2008). It is an emerging issue in Computer Science (CS) major due to high submission frequency (Kustanto and Liem, 2009) and detection difficulty (Rabbani and Karnalim, 2017). Hence, to handle the issue, several source code plagiarism detection approaches have been proposed (Lancaster and Culwin, 2004).

Generally speaking, there are three kinds of approach for detecting source code plagiarism: attribute-based, structure-

* Address: Prof. Drg. Surya Sumantri Street No. 65, Bandung, West Java 40164, Indonesia.

E-mail address: oscar.karnalim@it.maranatha.edu

Peer review under responsibility of King Saud University.



Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.01.012
1319-1578/© 2018 The Author. Production and hosting by Elsevier B.V. on behalf of King Saud University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

based, and hybrid approach (Al-Khanjari et al., 2010). First, attribute-based approach determines similarity based on source code attributes (e.g. the number of branching and line of code). To determine plagiarism-suspected pairs, this approach is usually featured with either Information Retrieval (IR) (Flores et al., 2016; Ganguly et al., 2017; Cosma and Joy, 2012) or machine learning (Ohmann and Rahal, 2015; Bandara and Wijayarathna, 2011) technique. Second, structure-based approach determine similarity based on source code structure; each source code is converted to intermediate representation and compared to each other to determine plagiarism-suspected pairs. Third, hybrid approach combines the former two approaches to enhance either the effectiveness (El Bachir Menai and Al-Hassoun, 2010; Engels et al., 2007) or efficiency (Burrows and Tahaghoghi, 2007; Mozgovoy et al., 2007).

Among these approaches, structure-based approach is the most widely-used approach in the domain of source code plagiarism detection. This approach has been used in numerous studies (Duric and Gasevic, 2013; Karnalim, 2017a, c; Lim et al., 2011; Fu et al., 2017; Zhao et al., 2015; Brixtel et al., 2010) while some of the studies are about the development of major source code plagiarism detection system (Prechelt et al., 2002; Schleimer et al., 2003; Wise, 1996). We believe that such phenomenon is natural since structure-based approach is more effective than attribute-based approach (Duric and Gasevic, 2013; Prechelt et al., 2002) and hybrid approach is impractical to be implemented (it requires the implementation of both attribute-based and structure-based approach at once).

Low-Level structure-based Approach (LLA) is a structure-based approach which defines plagiarism-suspected pair based on the similarity of low-level token sequence (i.e. executable file's token sequence) (Karnalim, 2016). Despite its rare practical use considering only few studies discuss it, several works (Karnalim, 2016a, 2017a, b; Rabbani and Karnalim, 2017) prove that LLA is more effective and efficient than a popular approach that relies on source code token sequence similarity. It is true that utilizing LLA limits the input to compilable source codes only. However, we would argue that such limitation is not a big issue; most undergraduate students are encouraged to submit compilable source codes. Moreover, if there is a need to handle the uncompilable ones, it is possible to use source-code-token-sequence-based approach as an alternative in addition to LLA (Rabbani and Karnalim, 2017).

When classified based on their target programming language, existing LLAs can be classified into two categories:.NET-targeted and Java-targeted LLA..NET-targeted LLA is focused on.NET programming languages such as C# by relying on Common Intermediate Language (CIL) to determine similarity (Rabbani and Karnalim, 2017). To date, there are three existing works about .NET-targeted LLA (Juričić, 2011; Juričić et al., 2011; Rabbani and Karnalim, 2017) where the most salient difference between them is about applied similarity algorithm. Levenstein distance, Rabin-Karp Greedy-String-Tiling, and adaptive local alignment are applied in Juričić (2011), Juričić et al. (2011), and Rabbani and Karnalim (2017) respectively. On the other, Java-targeted LLA is focused on Java programming language by relying on Bytecode to determine similarity (Karnalim, 2016a). There are four existing works about Java-targeted LLA:

- A work proposed in Ji et al. (2008). It is the 1st LLA that targets Java programming language.
- A work proposed in Karnalim (2016a). It is extended from Ji et al. (2008) by incorporating several additional features such as instruction generalization, instruction interpretation, and recursion handling.
- A work proposed in Karnalim (2017a). It is a successor of Karnalim (2016a) with three additional features: flow-based token weighting, argument removal heuristic, and invoked method removal.
- A work proposed in Karnalim (2017a). It is a successor of Karnalim (2016a) with a naive mechanism to linearize abstract method invocation.

Since LLA utilizes adapted string matching algorithm to determine similarity, it is natural for each LLA to linearize given executables to token sequences before comparison. Some of them (Juričić, 2011; Juričić et al., 2011; Rabbani and Karnalim, 2017) linearize it naively by considering the whole executable's tokens as a long sequence while the others (Karnalim, 2016a, 2017a, b; Ji et al., 2008) linearize it locally per method. We would argue that the latter linearization technique will generate more accurate result since it considers more contextual features.

While linearizing given executables to token sequences locally per method, each method invocation will be replaced with its respective invoked-method's content to provide more accurate result. It will raises an issue when an abstract method invocation is being linearized: each abstract method invocation may have more than one invoked-method's contents due to polymorphism. It is true that applying pseudo-execution toward given executables may solve the issue. However, we would argue that applying such execution is considerably impractical; providing program input (which is required for pseudo-execution) for each programming task is inconvenient. Further, processing time for conducting pseudo-execution may be high for complex source codes; its processing time is proportional to how the program works in real environment.

Several attempts have been done for linearizing abstract method invocation in LLA. The simplest technique is to ignore any abstract method invocations (Karnalim, 2016a, 2017b). Another technique is to concatenate all linearization alternatives as a replacement of abstract method invocation (Karnalim, 2017a). It is true that both techniques, at some extent, solve the issue about linearizing abstract method invocation. However, we would argue that their approach may generate misleading plagiarism detection result; they indirectly change the semantic of given token sequences.

Information Retrieval is a technique to obtain relevant information from a collection based on given information need (Croft et al., 2010). It is frequently used to mitigate processing time while maintaining resulted accuracy. From Software Engineering perspective, it has been used in several tasks. First, it mitigates the number of compared-to-be source codes for plagiarism detection (Burrows and Tahaghoghi, 2007; Mozgovoy et al., 2007). Second, it retrieves relevant software artifacts (e.g. source code Stolee et al., 2016; Vinayakarao et al., 2017; Karnalim, 2018; Lu et al., 2015 and executable file Karnalim and Mandala, 2014; Karnalim, 2016b) in a no time. Third, it enables software reuse by either providing relevant information (Thummalapenta and Xie, 2007; Lemos et al., 2007; Ye and Fischer, 2002) or source code examples about how to use API or libraries (Holmes and Murphy, 2005; Sindhgatta, 2006; Niu et al., 2017). Fourth, it enables user to keep track about developed software (Borg et al., 2014).

3. Methodology

In response to the gap defined on related works, an effective linearization technique which holds the semantic of token sequence is proposed. At first, it will generate all possible linearization pair alternatives in combinatoric manner regarding derived methods' content per abstract method invocation (one method content per invocation generates one alternative). Later the best alternative will be heuristically defined using Information Retrieval (IR) mech-

anism. Each of those phases is called as generation and selection phase respectively.

Generation phase generates all possible linearization pair alternatives. These alternatives are derived in combinatoric manner by propagating given token sequences in regard to replacing abstract method invocation with their respective derived methods' content. For efficiency reason, this phase will accept all method-scoped low-level token sequence pairs from both original and plagiarized code at once. In general, this phase consists of four sub-phases (see Fig. 1):

1. For each code (either original or plagiarized code), abstract methods with their respective derived methods will be enlisted using method proposed in Karnalim (2017a). Such method will remodel class/interface relation as a graph, sort them in ascending order using Topological sort (Sedgewick and Wayne, 2011), and enlist derived methods for each abstract method from sorted class/interface entities (starting from the first to the last entity). Further example of this sub-phase can be seen in Karnalim (2017a).
2. Each abstract method invocation in declared methods will be replaced with the content of its derived method. If an abstract method has more than one derived method, given sequence will be propagated to several sequence alternatives where each alternative will be exclusively assigned with the content of one derived method. For instance, suppose we have a sequence with two abstract method invocations; the first one has two derived methods while the latter one has three derived methods. Given sequence will be propagated to 2×3 sequence alternatives since each invocation will propagate given sequence or sequence alternative regarding to its number of derived methods.
3. Index-like representation for each linearization sequence alternative will be generated. It stores all tokens with their occurrence frequency in key-value format. The representation resulted from this sub-phase will be passed to selection phase for mitigating comparison time.
4. Each sequence alternative will be paired with each alternative from another token sequence. It will generate $N * M$ linearization pair alternatives; N and M refer to the number of alternative for the first and second sequence respectively. For instance, if a sequence has 6 sequence alternatives and another sequence has 4 alternatives, it will generate 6×4 pair alternatives that will be passed to selection phase.

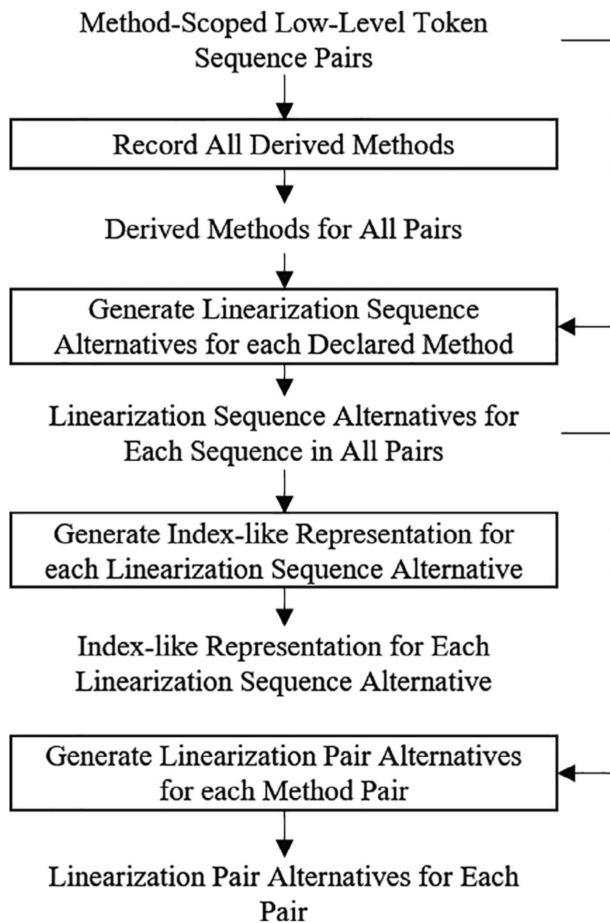


Fig. 1. Generation phase from proposed methodology.

Selection phase determines which linearization pair alternative is the correct one. Different with generation phase, it is conducted locally per method pair. In general, this phase consists of two sub-phases (see Fig. 2):

1. Similarity degree for each linearization pair alternative will be measured using IR mechanism; one sequence from the pair will act as a query toward another sequence based on their index-like representation. Two IR mechanisms are proposed in this work: smoothed boolean and vector space model retrieval. Smoothed boolean retrieval refers to boolean retrieval (Croft et al., 2010) that returns approximate similarity degree instead of the boolean one. Such similarity is resulted from (1); A and B are distinct tokens from both sequences. On the other hand, vector space model retrieval refers to a widely-used retrieval model which determine similarity based on occurrence proportion (Croft et al., 2010).
2. Correct linearization pair will be selected based on the highest IR-based similarity degree. If such degree is resulted from two or more alternatives, the first-measured pair alternative will be selected as the result.

$$SIM(A,B) = 2 * |AnB|/(|A| + |B|) \quad (1)$$

It is important to note that our proposed technique can also be applied for solving similar issue on source code level; with some modifications, it can linearize abstract method invocation from source code token sequence. Further, it can also be used to accuse plagiarism; linearized token sequences can be used to address which source code parts are suspected as plagiarism fragments.

4. Evaluation

4.1. Evaluation preliminaries

Dataset used in this evaluation consists of 261 method-scoped source code pairs. Each of these pairs is exclusively assigned to either controlled and empirical dataset based on its implemented plagiarism attack.

Controlled dataset exploits the characteristic of proposed approach. It consists of 24 main method pairs which plagiarism attack theoretically favors proposed approach in terms of effectiveness and efficiency. To provide clearer analysis, original code for each pair is defined as simple as possible. On the first half cases, original code is a main method which only prints *Hello World* once. On the others, original code is a plagiarized form from the first half cases. Six plagiarism attacks are considered in this dataset:

- Replacing statements with a Standard Method Invocation (SMI). Plagiarized code is generated by replacing a *Hello World* print statement with a method invocation; the content of given

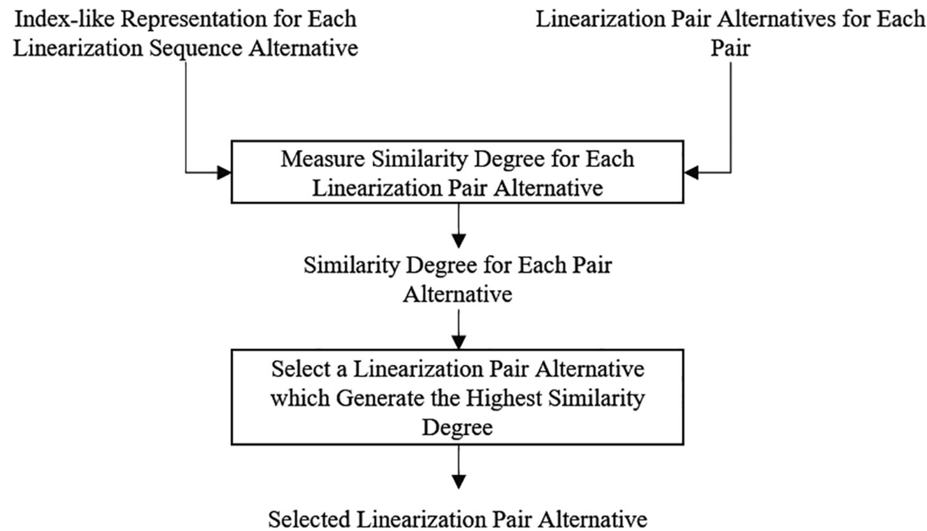


Fig. 2. Selection phase from proposed methodology.

method is a print statement. This attack will generate 1 case (C01).

- Replacing statements with an Abstract Method Invocation (AMI). Plagiarized code is generated by replacing a *Hello World* print statement with an abstract method invocation; given abstract method has been derived to a concrete method that has a print statement. This attack will generate 1 pair (C02).
- AMI where given abstract method is defined in Depth structure (AMID). Plagiarized code is generated in similar fashion as in AMI except that given abstract method has been passed to N classes. This attack will generate 4 pairs (C03-C06). Each pair will be assigned with a unique N, starting from 2 to 5 respectively.
- AMI where given abstract method is defined in Breadth structure (AMIB). Plagiarized code is generated in similar fashion as in AMI except that given abstract method has been derived to N concrete methods and only one of them has the correct print statement. This attack will generate 4 pairs (C07-C10). Each pair will be assigned with a unique N, starting from 2 to 5 respectively.
- AMI where given abstract method is defined in Depth and Breadth structure (AMIDB). Plagiarized code is generated in similar fashion as in AMID except that each passing generates 2 derived classes. This attack will generate 2 pairs (C11 and C12). Each pair will be assigned with a unique N, starting from 1 to 2 respectively. N will work in similar manner as in AMID.
- Verbatim Copy (VC). Plagiarism code is copied exactly as it is from original code. The original code itself will be taken from plagiarized code generated by previous attacks (C01-C12) respectively. This attack will generate 12 pairs (C13-C24).

Different with controlled dataset, empirical dataset represents real plagiarism cases; each plagiarized method is generated as a part of removing a design pattern from a code without changing its flow (an implementation of advanced plagiarism attack). It consists of 237 method pairs that are enlisted from dataset proposed in Karnalim (2017a).

Four approaches will be considered in our evaluation. All of them are derived from unweighted LLA proposed in Karnalim (2017b) by adding an abstract method linearization mechanism at the end of its extraction phase. The first two approaches use our proposed technique; each approach is exclusively assigned with one proposed IR mechanism. These approaches are labeled as Smoothed-boolean-Retrieval Combinatoric approach (SRC) and Vector-space-model-Retrieval Combinatoric approach (VRC) respectively. On the other, the last two approaches are the baselines to evaluate the impact of our proposed approach. These approaches are labeled as Naive Approach (NA) and Combinatoric Approach (CA). First, NA uses technique proposed in Karnalim (2017a). It concatenates the content of all derived methods as a replacement of an abstract method invocation. To our knowledge, this approach is the only work which addresses exactly-similar problem as ours. Second, CA uses only generation phase from our proposed technique. To determine correct linearization pair, it utilizes in-depth comparison phase on Karnalim (2017b) (which is normally used after the correct linearization pair is selected); a pair with the highest similarity degree will be considered as the correct one. This approach is used to measure the benefit of selection phase from our proposed technique.

All approaches will be evaluated in regard to three metrics: the reversed number of mismatched token, the number of false positive, and the number of process. First, the Reversed number of Mismatched Token (RMT) measures how far a particular approach affects resulted similarity degree. It is calculated based on (2) where *MT(A,B)* refers to the number of mismatched tokens. This metric has been commonly used in related works about LLA (Karnalim, 2016a, 2017a, b) for measuring effectiveness (despite the use of different terminology); higher RMT refers to higher similarity degree.

$$RMT(A,B) = MT(A,B) * -1 \tag{2}$$

Second, the number of false positive measures how many incorrect linearization pair alternatives that are resulted from a particular approach. NA merges all alternatives as one big sequence which is neither true nor false positive. Hence, it is expected that NA is excluded from evaluation regarding to this metric.

Last, the number of process measures how much time a particular approach will take for completing its task. It will be calculated according to a simplified Time-Complexity-like Equation (TCE) for linearizing abstract methods in one method-scoped low-level pair and comparing selected pair to determine similarity. The latter phase is involved for fairness purpose; as it is known that the latter phase is exclusively used by CA to determine the correct linearization pair. It is true that execution time can be calculated empirically. However, we would argue that such approach is not suitable for our dataset; each method only generate a small number of process and empirical time measurement may be inaccurate for that kind of method.

TCE for involved approaches are defined as follows:

- TCE for NA (see (3)) is defined based on TCE for comparing two token sequences using RK-GST algorithm in (4); *lim(A)* and *lim (B)* are linearized token sequences generated by NA.
- TCE for CA (see (5)) is defined in similar manner as in NA except that it tries all linearization pair alternatives instead of concatenating tokens to two long sequences; N refers to the number of linearization pair alternatives.
- TCE for SRC (see (6)) is defined by summing TCE for three core steps: indexing, retrieving, and comparing. First, indexing step converts each linearization sequence alternative to index form; its TCE can be seen in (7) where *na* and *nb* refer to the number of linearization sequence alternative for both sequences respectively. Second, retrieving step selects the correct linearization pair alternative using smoothed boolean retrieval; its TCE (which is based on time complexity to perform such retrieval) can be seen in (8) where *n* refers to the number of linearization pair alternative. Third, comparison step is aimed to compare two token sequences from correct linearization pair alternative (i.e. A_c and B_c).
- TCE for VRC (see (9)) is defined in similar manner as in SRC except that its retrieving step is based on vector space model. TCE for VRC's retrieving step can be seen in (10) where *n* refers to the number of linearization pair alternative.

$$TN(A,B) = T_S(lin(A), lin(B)) \quad (3)$$

$$T_S(A,B) = (max(|A|,|B|))^3 \quad (4)$$

$$TC(A,B) = \sum_{i=1}^{n} T_S(A_i, B_i) \quad (5)$$

$$TS(A,B) = T_I(A,B) + TS_R(A.B) + T_S(A_c, B_c) \quad (6)$$

$$T_I(A,B) = \sum_{i=1}^{na} |A_i| + \sum_{i=1}^{nb} |B_i| \quad (7)$$

$$TS_R(A,B) = \sum_{i=1}^{n} max(|A_i|,|B_i|) \quad (8)$$

$$TV(A,B) = T_I(A,B) + TV_R(A.B) + T_S(A_c, B_c) \quad (9)$$

$$TV_R(A,B) = \sum_{i=1}^{n} (|A_i| + |B_i| + min(|A_i|,|B_i|)) \quad (10)$$

According to the fact that there will be numerous results from empirical dataset due to its high number of cases, empirical results will be analyzed only based on rank distribution instead of real values. For each case, involved approaches will be ranked according to targeted evaluation metric in a particular order (descending order for the reversed number of mismatched token and ascending order for other two metrics). If several approaches generate similar metric, then these approaches will be assigned with same rank.

4.2. Evaluation regarding the reversed number of mismatched token

According to resulted RMT on controlled dataset (see Fig. 3), it is clear that no approaches generate zero RMT. Further observation shows that original and plagiarized code on these cases are not exactly similar to each other; plagiarized code either employs additional instructions or removes some instructions from original code. It is important to note that such phenomenon is also applied

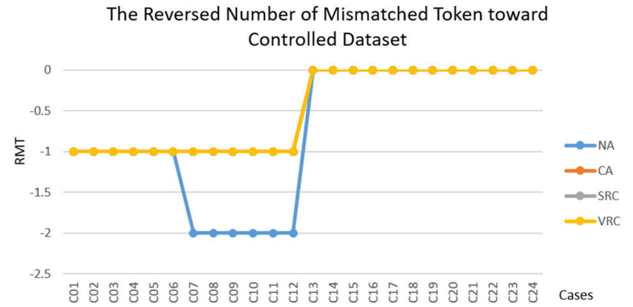


Fig. 3. The reversed number of mismatched token toward controlled dataset.

on empirical dataset. However, it is not explicitly shown on Fig. 4 since given figure does not show resulted RMT directly.

Among involved approaches, NA is the least effective approach in terms of RMT. As seen on Fig. 3, it generates lower RMT than other approaches on C07-C12. Further, it even gets the fewest number of 1st rank on empirical dataset according to Fig. 4. Such ineffectiveness is caused by NA's linearization technique; it concatenates unrelated method content to given sequence, reducing resulted RMT. In contrast, CA is the most effective approach in terms of RMT. It is always assigned with the highest RMT for controlled dataset (see Fig. 3) and gets the most number of 1st rank on empirical dataset (see Fig. 4). Such finding is natural since CA tries all possible combinations in order to generate the most accurate result.

As seen in Fig. 4, it is interesting to see that CA is assigned as the 2nd rank on some cases from empirical dataset; it is coincidentally outperformed by NA as a result of changing the semantic of original token sequences. However, this finding cannot be claimed as CA's drawback; different with NA, CA holds sequence semantic while linearizing method invocation.

IR-based similarity, at some extent, can be used as a replacement of sequence-based minimum matching similarity (Prechelt et al., 2002) (i.e. an in-depth comparison algorithm used by Karnalim, 2017c) to select the correct linearization pair; SRC and VRC (which utilize IR-based similarity) generate similar RMT as CA (which utilizes sequence-based minimum matching similarity) on most cases from controlled and empirical dataset (see Figs. 3 and 4).

When compared to each other, IR-based approaches show similar characteristics on most cases (see Figs. 3 and 4). They only differ on two conditions: when both token sequences share similar token proportion and when the tokens from a sequence is a subset of another sequence. The former condition will favor VRC (which uses vector space model retrieval) while the latter one will favor SRC (which uses smoothed boolean retrieval).

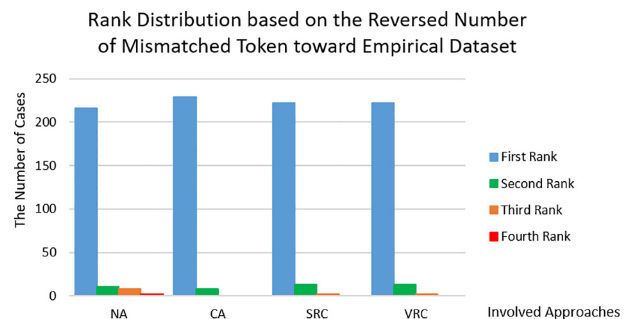


Fig. 4. Rank distribution based on the reversed number of mismatched token toward empirical dataset.

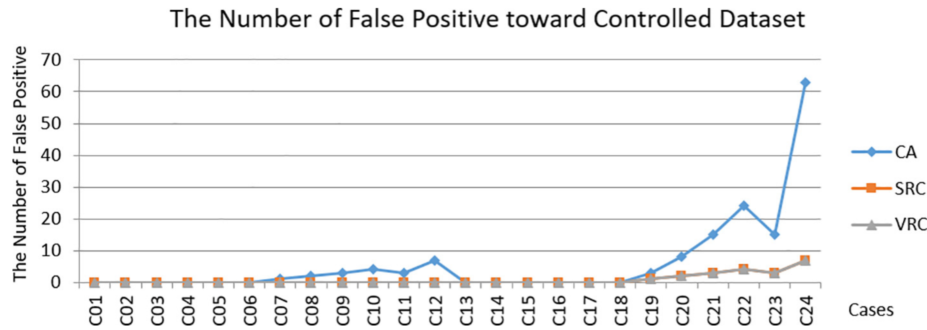


Fig. 5. The number of false positive toward controlled dataset.

4.3. Evaluation regarding the number of false positive

Since involved approaches do not consider the semantic of abstract method invocation, it is expected that they generate false positives when comparing two identical sequences with more than one linearization pair alternative; all linearization pair alternatives will be assigned with the highest possible similarity degree regardless of used similarity mechanism. Fig. 5 shows such issue on C19-C24.

Among involved approaches, CA generates the largest number of false positive; it generates all alternatives without providing a proper heuristic to select the correct one. Fig. 5 explicitly shows such issue on C07-C12 and C19-C24. Further, such issue is also shown on Fig. 6 where CA is assigned with the fewest number of the 1st rank. CA is only able to generate no false positives on cases with one linearization pair alternative; numerous number of 1st rank assigned to CA on empirical dataset (see Fig. 6) are resulted from such phenomenon. It is important to note that CA's number of false positives can become extremely large. For instance, on a case about abstract factory's main method from empirical dataset, CA generates 34.012.223 false positives while SRC and VRC (i.e. other approaches) only generate 11.665 and 1.297 false positives respectively.

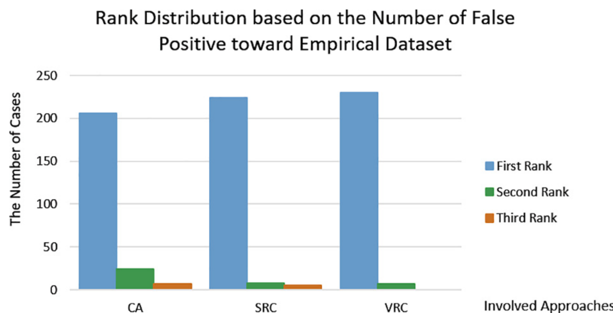


Fig. 6. Rank distribution based on the inversed number of false positive toward empirical dataset.

Different with CA, IR-based approaches (i.e. SRC and VRC) generates lower number of false positive; it can be clearly seen on Figs. 5 and 6. Their IR technique is able to distinguish correct pair from other alternatives as long as that pair generates the highest IR-based similarity degree. Nevertheless, it is important to note that IR-based approaches are not always able to select the correct linearization pair alternative; their IR-based similarity does not consider token order while such order may be the only thing that could distinguish correct linearization pair. Manual observation toward the result of empirical dataset shows that such issue occurs on 14 and 15 cases respectively for SRC and VRC.

When compared with SRC, VRC is assigned as the 1st rank on more empirical cases (see Fig. 6); on some cases, correct linearization pair is only distinguishable when token proportion (i.e. VRC's exclusive feature) .

4.4. Evaluation regarding the number of process

Despite its simple TCE, NA generates a large number of process on cases that involve numerous derived methods; it generates long compared-to-be token sequences as a result of naive linearization, resulting more processes on comparison phase. This drawback can be explicitly seen on C07-C12 from Fig. 7; NA generates the highest number of process. However, further observation toward both controlled and empirical dataset shows that the impact of such drawback is insignificant; NA's resulted number of process is still comparable to IR-based approaches (i.e. SRC and VRC) on most cases.

Similar with NA, CA also generates a large number of process on cases that involve numerous derived methods. However, the cause of such issue is the existence of enormous generated pair alternatives instead of long compared-to-be token sequences. CA's comparison will be executed for each alternative once, resulting $M \times N$ comparisons where each comparison takes $(max(|A|, |B|))^3$ processes (M and N are the number of alternatives for both sequences while A and B are compared-to-be token sequences for each pair alternative). This drawback can be explicitly seen

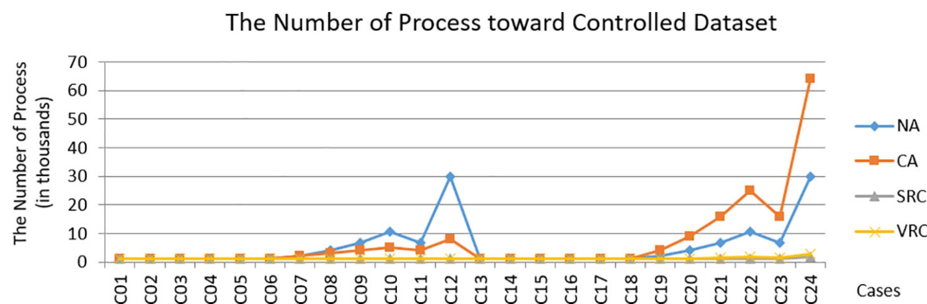


Fig. 7. The number of process toward controlled dataset.

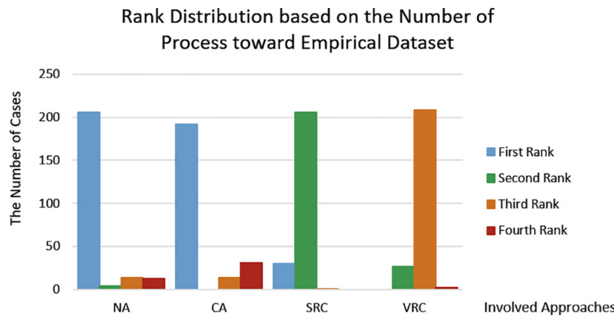


Fig. 8. Rank distribution based on the inversed number of process toward empirical dataset.

on C19-C24 from Fig. 7 where the highest number of process is generated by CA. Further, it is also shown on a case about abstract factory's main method from empirical dataset; CA takes 193 trillions processes while other approaches take at most 1.63 billions processes.

Despite their drawback showed on Fig. 7, NA and CA are faster than IR-based approaches when handling cases with only one linearization pair alternative; both approaches do not involve alternative selection phase, resulting fewer processes. This benefit can be explicitly seen from the result of empirical dataset given on Fig. 8; both NA and CA are assigned with the 1st rank on most cases since these cases are featured with only one pair alternative each.

When compared to SRC, VRC takes the largest number of process; it generates slightly more processes on controlled dataset (see Fig. 7) and is assigned with lower rank on empirical dataset (see Fig. 8). Such finding is natural since VRC's TCE has higher complexity than SRC's.

When perceived from real execution time toward the whole empirical dataset, CA takes the longest processing time (47 h), followed by NA (2 h), VRC (5 min), and SRC (5 min). Hence, it can be stated that CA is the most impractical approach while VRC and SRC are the most practical ones.

5. Conclusion and future work

In this paper, an IR-based technique for linearizing abstract method invocation in plagiarism-suspected source code pairs has been proposed. According to our evaluation, four findings can be deducted. Firstly, proposed technique leads to more accurate plagiarism result. It generates higher reverse number of mismatched token (i.e. an accuracy measurement) than state-of-the-art technique (Karnalim, 2017a). Further, it does hold the semantic of token sequence and it is as effective as a combinatoric technique (which tries all possible linearization pair alternatives) on most cases. Secondly, proposed technique is effective to remove false positives; only few false positives share similar or higher IR-based similarity degree than the correct one. Thirdly, proposed technique is more efficient in terms of processing time on cases with numerous derived methods. Finally, each IR mechanism has its own benefit for linearizing abstract method.

For future work, we plan to strengthen our findings by evaluating proposed approach on real plagiarism cases taken from object-oriented programming course. In addition, we also plan to incorporate attribute-based approach to enhance the accuracy of proposed approach.

6. Threats to validity

In general, two threats of validity should be considered toward the result of this work. First, this result cannot be generalized to all attacks regarding abstract method in object-oriented environment. We try to mitigate this threat by enlisting possible attacks in controlled dataset. Further, we also use source codes with advanced object-oriented techniques to exploit more possible attacks in empirical dataset. However, it is still possible that some attacks are not covered in our datasets. Second, since our proposed technique has not been used in real environment, its characteristics may be different in real environment. We try to mitigate this threat by utilizing evaluation metrics related to real environment. However, it is still possible that external factors from real environment affect the characteristics of proposed technique.

Acknowledgement

This work was supported by Maranatha Christian University, Indonesia.

References

Al-Khanjari, Z.A., Fiaidhi, J.A., Al-Hinai, R.A., Kutti, N.S., 2010. PlagDetect: a Java programming plagiarism detection tool. ACM Inroads 1 (4), 66. https://doi.org/10.1145/1869746.1869766. URL http://dl.acm.org/citation.cfm?doid=1869746.1869766.

Bandara, U., Wijayarathna, G., 2011. A machine learning based tool for source code plagiarism detection. Int. J. Mach. Learn. Comput. 1 (4), 337–343.

Borg, M., Runeson, P., Ardö, A., 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Softw. Eng. 19 (6), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y. URL http://link.springer.com/10.1007/s10664-013-9255-y.

Brixtel, R., Fontaine, M., Lesner, B., Bazin, C., Robbes, R., 2010. Language-independent clone detection applied to plagiarism detection. In: 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation. IEEE, pp. 77–86. https://doi.org/10.1109/SCAM.2010.19. URL http://ieeexplore.ieee.org/document/5601829/.

Burrows, S., Tahaghoghi, S.M.M., Zobel, J., 2007. Efficient plagiarism detection for large code repositories. Softw.: Pract. Exp. 37 (2), 151–175. https://doi.org/10.1002/spe.750. URL http://doi.wiley.com/10.1002/spe.750.

Cosma, G., Joy, M., 2008. Towards a definition of source-code plagiarism. IEEE Trans. Edu. 51 (2), 195–200. https://doi.org/10.1109/TE.2007.906776. URL http://ieeexplore.ieee.org/document/4455461/.

Cosma, G., Joy, M., 2012. An approach to source-code plagiarism detection and investigation using latent semantic analysis. IEEE Trans. Comput. 61 (3), 379–394. https://doi.org/10.1109/TC.2011.223. URL http://ieeexplore.ieee.org/document/6086533/.

Croft, W.B., Metzler, D., Strohman, T., 2010. Search Engines: Information Retrieval in Practice. Addison-Wesley.

Duric, Z., Gasevic, D., 2013. A source code similarity system for plagiarism detection. Comput. J. 56 (1), 70–86. https://doi.org/10.1093/comjnl/bxs018. URL https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/bxs018..

El Bachir Menai, M., Al-Hassoun, N.S., 2010. Similarity detection in Java programming assignments. In: 2010 5th International Conference on Computer Science & Education. IEEE, pp. 356–361. https://doi.org/10.1109/ICCSE.2010.5593613. URL http://ieeexplore.ieee.org/document/5593613/.

Engels, S., Lakshmanan, V., Craig, M., Engels, S., Lakshmanan, V., Craig, M., 2007. Plagiarism detection using feature-based neural networks. ACM SIGCSE Bull. 39 (1), 34. https://doi.org/10.1145/1227504.1227324. URL http://portal.acm.org/citation.cfm?doid=1227504.1227324.

Flores, E., Moreno, L., Rosso, P., 2016. Detecting source code re-use with ensemble models. In: Proceedings of the 4th Spanish Conference on Information Retrieval – CERI '16. ACM Press, New York, New York, USA, pp. 1–7. https://doi.org/10.1145/2934732.2934738. URL http://dl.acm.org/citation.cfm?doid=2934732.2934738.

Fu, D., Xu, Y., Yu, H., Yang, B., 2017. WASTK: a weighted abstract syntax tree kernel method for source code plagiarism detection. Sci. Program. 2017, 1–8. https://doi.org/10.1155/2017/7809047. URL https://www.hindawi.com/journals/sp/2017/7809047/.

Ganguly, D., Jones, G.J.F., Ramírez-de-la Cruz, A., Ramírez-de-la Rosa, G., Villatoro-Tello, E., 2017. Retrieving and classifying instances of source code plagiarism. Inf. Retrieval J., 1–23 https://doi.org/10.1007/s10791-017-9313-y. URL http://link.springer.com/10.1007/s10791-017-9313-y.

Holmes, R., Murphy, G., 2005. Using structural context to recommend source code examples. In: Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. IEEE, pp. 117–125. https://doi.org/10.1109/ICSE.2005.1553554. URL http://ieeexplore.ieee.org/document/1553554/.

Ji, J.-H., Woo, G., Cho, H.-G., 2008. A plagiarism detection technique for java program using bytecode analysis. In: 2008 Third International Conference on Convergence and Hybrid Information Technology. IEEE, pp. 1092–1098. https://doi.org/10.1109/ICCIT.2008.267. URL http://ieeexplore.ieee.org/document/4682179/.

Juričić, V. 2011. Detecting source code similarity using low-level languages. In: 33rd International Conference on Information Technology Interfaces, Dubrovnik. http://ieeexplore.ieee.org/document/5974090/.

Juričić, V., Jurić, T., Tkalec, M., 2011. Performance evaluation of plagiarism detection method based on the intermediate language. The Future Inf. Sci., 355

Karnalim, O., 2016a. Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. In: The 10th International Conference on Information & Communication Technology and Systems (ICTS). IEEE, pp. 63–68. https://doi.org/10.1109/ICTS.2016.7910274. http://ieeexplore.ieee.org/document/7910274/.

Karnalim, O. 2016b. Extended vector space model with semantic relatedness on java archive search engine. Jurnal Teknik Informatika dan Sistem Informasi 1 (2). URL http://jutisi.maranatha.edu/index.php/jutisi/article/view/372.

Karnalim, O., 2017a. An abstract method linearization for detecting source code plagiarism in object-oriented environment. In: The 8th International Conference on Software Engineering and Service Science (ICSESS). IEEE, Beijing, 2017.

Karnalim, O. 2017b. A low-level structure-based approach for detecting source code plagiarism. IAENG Int. J. Comput. Sci. 44 (4). URL http://www.iaeng.org/IJCS/issues_v44/issue_4/IJCS44_411.pdf.

Karnalim, O. 2017c. Python source code plagiarism attacks on introductory programming course assignments. Themes Sci. Technol. Edu. 10 (1). URL http://earthlab.uoi.gr/theste/index.php/theste/article/view/237.

Karnalim, O. 2018. Language-agnostic source code retrieval using keyword & identifier lexical pattern. Int. J. Softw. Eng. Comput. Syst. (IJSECS) 4 (1).

Karnalim, O., Mandala, R., 2014. Java archives search engine using byte code as information source. In: 2014 International Conference on Data and Software Engineering (ICODSE). IEEE, pp. 1–6. https://doi.org/10.1109/ICODSE.2014.7062660. URL http://ieeexplore.ieee.org/document/7062660/.

Kustanto, C., Liem, I., 2009. Automatic source code plagiarism detection. In: 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing. IEEE, pp. 481–486. https://doi.org/10.1109/SNPD.2009.62. URLhttp://ieeexplore.ieee.org/document/5286623/.

Lancaster, T., Culwin, F., 2004. A comparison of source code plagiarism detection engines. Comput. Sci. Edu. 14 (2), 101–112. https://doi.org/10.1080/08993400412331363843. URL http://www.tandfonline.com/doi/abs/10.1080/08993400412331363843.

Lemos, O.A.L., Bajracharya, S.K., Ossher, J., Morla, R.S., Masiero, P.C., Baldi, P., Lopes, C.V., 2007. Using test-cases to search and reuse source code. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering – ASE '07. ACM Press, New York, USA, pp. 525. https://doi.org/10.1145/1321631.1321726. URL http://portal.acm.org/citation.cfm?doid=1321631.1321726.

Lim, J.-S., Ji, J.-H., Cho, H.-G., Woo, G., 2011. Plagiarism detection among source codes using adaptive local alignment of keywords. In: Proceedings of the 5th International Confernece on Ubiquitous Information Management and Communication – ICUIMC '11. ACM Press, New York, USA, pp. 1. https://doi.org/10.1145/1968613.1968643. URL http://portal.acm.org/citation.cfm?doid=1968613.1968643.

Lu Meili, Sun, X., Wang, S., Lo, D., Duan Yucong, 2015. Query expansion via WordNet for effective code search. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, pp. 545–549. https://doi.org/10.1109/SANER.2015.7081874. URL http://ieeexplore.ieee.org/document/7081874/.

Mozgovoy, M., Karakovskiyz, S., Klyuev, V., 2007. Fast and reliable plagiarism detection system. In: 2007 37th Annual Frontiers in Education Conference. IEEE, pp. S4H-11–S4H-14. https://doi.org/10.1109/FIE.2007.4417860. URL http://ieeexplore.ieee.org/document/4417860/.

Niu, H., Keivanloo, I., Zou, Y., 2017. Learning to rank code examples for code search engines. Empirical Softw. Eng. 22 (1), 259–291. https://doi.org/10.1007/s10664-015-9421-5. URL http://link.springer.com/10.1007/s10664-015-9421-5.

Ohmann, T., Rahal, I., 2015. Efficient clustering-based source code plagiarism detection using PIY. Knowl. Inf. Syst. 43 (2), 445–472. https://doi.org/10.1007/s10115-014-0742-2. URL http://link.springer.com/10.1007/s10115-014-0742-2.

Prechelt, L., Malpohl, G., Philippsen, M., 2002. Finding plagiarisms among a set of programs with JPlag. J. Universal Comput. Sci. 8 (11), 1016–1038. URL http://jucs.org/jucs{_}811/finding{_}plagiarisms{_}among{_}a/Prechelt{_}L.pdf.

Rabbani, F.S., Karnalim, O., 2017. Detecting source code plagiarism on .NET programming languages using low-level representation and adaptive local alignment. J. Inf. Org. Sci. 41 (1), 105–123. URL https://jios.foi.hr/index.php/jios/article/view/1086.

Schleimer, S., Wilkerson, D.S., Aiken, A., 2003. Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD International Conference on on Management of Data – SIGMOD '03. ACM Press, New York, USA, p. 76. https://doi.org/10.1145/872757.872770. URL http://portal.acm.org/citation.cfm?doid=872757.872770.

Sedgewick, R., Wayne, K., 2011. Algorithms. Addison-Wesley. URL https://www.pearson.com/us/higher-education/program/Sedgewick-Algorithms-4th-Edition/PGM100869.html.

Sindhgatta, R., 2006. Using an information retrieval system to retrieve source code samples. In: Proceeding of the 28th International Conference on Software Engineering – ICSE '06. ACM Press, New York, USA, pp. 905. https://doi.org/10.1145/1134285.1134448. URL http://portal.acm.org/citation.cfm?doid=1134285.1134448.

Stolee, K.T., Elbaum, S., Dwyer, M.B., 2016. Code search with input/output queries: generalizing, ranking, and assessment. J. Syst. Softw. 116, 35–48.

Thummalapenta, S., Xie, T., 2007. Parseweb: a programmer assistant for reusing open source code on the web. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 204–213.

Vinayakarao, V., Sarma, A., Purandare, R., Jain, S., Jain, S., 2017. ANNE: improving source code search using entity retrieval approach. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining – WSDM '17. ACM Press, New York, USA, pp. 211–220. https://doi.org/10.1145/3018661.3018691. URL http://dl.acm.org/citation.cfm?doid=3018661.3018691.

Wise, M.J., 1996. YAP3: improved detection of similarities in computer program and other texts. In: Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education – SIGCSE '96, vol. 28. ACM Press, New York, USA, pp. 130–134. https://doi.org/10.1145/236452.236525. URL http://portal.acm.org/citation.cfm?doid=236452.236525.

Ye, Y., Fischer, G., 2002. Supporting reuse by delivering task-relevant and personalized information. In: Proceedings of the 24th International Conference on Software Engineering – ICSE '02. ACM Press, New York, USA, pp. 513. https://doi.org/10.1145/581339.581402. URL http://portal.acm.org/citation.cfm?doid=581339.581402.

Zhao, J., Xia, K., Fu, Y., Cui, B., 2015. An AST-based code plagiarism detection algorithm. In: 2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA). IEEE, pp. 178–182. https://doi.org/10.1109/BWCCA.2015.52. URL http://ieeexplore.ieee.org/document/7424821/.