Hashing and Merging Heuristics for Text Reuse Detection

Notebook for PAN at CLEF-2014

Faisal Alvi[1], Mark Stevenson[2], Paul Clough[2]

[1]King Fahd University of Petroleum & Minerals, Saudi Arabia,
[2]University of Sheffield, United Kingdom.
alvif@kfupm.edu.sa, mark.stevenson@sheffield.ac.uk,
p.d.clough@sheffield.ac.uk

Abstract This paper describes a joint software entry by King Fahd University of Petroleum & Minerals and the University of Sheffield for the text-alignment task at PAN-2014. We employ the three steps of seeding, extension and filtering for text alignment. For seeding we use character n-grams with a variant of Rabin-Karp Algorithm for multiple pattern search. We then use an elaborate merging mechanism with several cases to combine the individually found seeds. A short filtering step is then used to remove extraneous passages. This approach scored $plagdet$ scores of 0.65954 and 0.73416 on test corpora 2 and 3 during the final test run.

1 Introduction

Text Reuse Detection has been a part of plagiarism detection at the PAN Competition since inception as text alignment [5,6] in 2012-14 and as extrinsic plagiarism detection earlier. For the last year i.e., 2013, the text alignment task consisted of finding passages of reused text in five different types of obfuscated document pairs: no plagiarism, no obfuscation, random obfuscation, translation obfuscation and summary obfuscation. Since training corpus for PAN-2014 text alignment is the same as in 2013, it could be inferred that similar obfuscation types could be in use in 2014.

Several approaches have been employed by participants for text alignment including character n-grams, word n-grams and their variants, along with various similarity metrics. Three distinct stages in participants' approaches have been identified in PAN Overview paper for 2013 [6]: (1) seeding, (2) extension, and (3) filtering.

For our current year's (2014) entry, our submitted software also consists of these three stages of seeding, extension/merging and filtering. In the seeding stage we use character *n*-grams after post-processing and apply a variant of the Rabin-Karp Algorithm [2] for multiple pattern search [3,4] to find matching seeds. We then classify pairs of matching seeds within source and suspicious documents into four classes: containment, overlap, near-disjoint and far-disjoint. We then use case-by-case merging to combine pairs of various classes. Finally, we apply filtering to remove short passages in order to exclude false positives. Our approach has shown mid-range results for the 2013 and 2014 training and test corpora.

2 The Approach

As outlined in the first section our approach consists of 3 phases: seeding, extension and filtering along with some preprocessing. An overview of the resulting software appears in the block diagram as shown in figure 1. The language of the software was java.

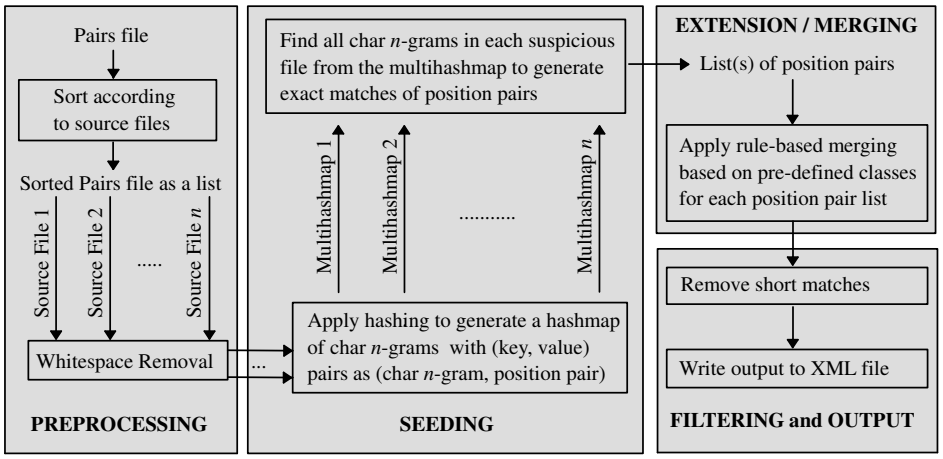

Figure 1. Block Diagram of the Submitted Software

2.1 Preprocessing

Initially the 'pairs' file is sorted according to the source file by file number. Since one source document may be linked to several suspicious documents, a single hashing of source document is repeatedly used for finding matches in corresponding suspicious documents. Furthermore, during hashing, all punctuation, whitespace and non-printable characters are removed. This is done as two perfectly reused strings (one in the source document and other in the suspicious one) could go undetected because of an unequal amount of spaces, punctuation and/or other characters.

2.2 Seeding

Seeding is defined as "*Given a suspicious document and a source document, matches (so-called "seeds") between the two documents are identified using some seed heuristic.... By coming up with as many reasonable seeds as possible, the subsequent step of extending them into aligned passages becomes a lot easier*". [6]

For seeding, or identifying exact matches of character n-grams between two documents, we use Rabin-Karp Algorithm [2] for multiple pattern search [3,4]. This is done by first placing all character n-grams (for a particular value of n) into a multiple valued hashtable, and then finding matches for character n-grams of the suspicious document

from the hashtable. Due to hashing, the search time of this is reduced from O(*source size* $\times$ *susp size*) to O(*source size* + *susp size*) - a substantial saving over naïve string search. However, we employ several optimizations and simplifications to the Rabin-Karp Algorithm in java as follows:

1. We use java's built-in `java.util.hashmap` collection, which in-turn utilises java's built-in `hashCode()` method as a hash function, that ensures a unique hash value for each unique character n-gram.
2. Since several strings are repeated in a document at different locations, use of a single-valued hashtable will inevitably lead to collisions - hence we use a multiple valued hashtable called a multihashmap to store these, as shown in figure 2. A multihashmap can be created by some modifications to the original hashmap provided in java. For example, the character 20-gram "`symptomsofarrhythmia`" appears several times in a source document and it is recorded each time with new locations as a value.
3. For each *n*-gram its location is stored as a 3-tuple (*start location*, *end location*, *size*) in the multihashmap. Some book-keeping is done to keep track of whitespace and other ignored characters in the location(s), while the *size* parameter keeps track of actual size.

Once the multihashmap for a particular source document is created, we make a run of each suspicious file to find the matching character n-grams. The output is a list of pairs of 3-tuples: one from the source file and the other from the suspicious file.

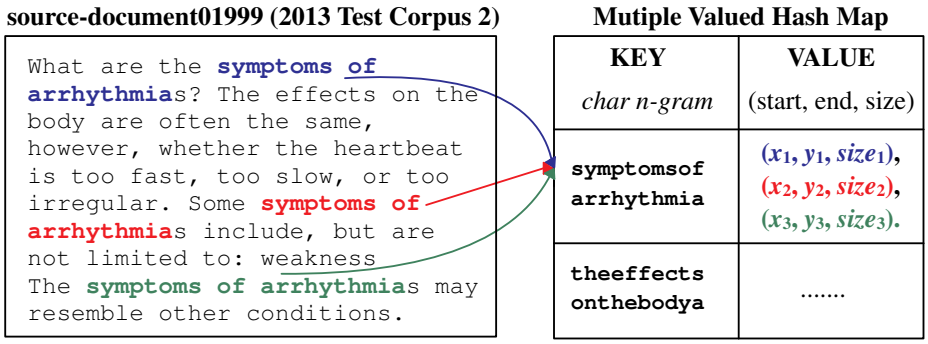

Figure 2. Collision Management in Multiple Valued Hashmap

For example after the algorithm is run on a source file with the corresponding suspicious file, the output would be a list of the form:

List of 3-tuple pairs = $\{(x_1, y_1, s_1) \rightarrow (a_1, b_1, s'_1), (x_2, y_2, s_2) \rightarrow (a_2, b_2, s'_2), \ldots\}$

In the list above, the entry $(x_1, y_1, s_1) \rightarrow (a_1, b_1, s'_1)$ means that the text in the source file from position x_1 to y_1 of size s_1 is aligned with the text in the suspicious file from location a_1 to b_1 of size s'_1. It may be noted that $y_1 > x_1$ and $b_1 > a_1$, furthermore $s_1 \leq y_1 - x_1$, $s'_1 \leq b_1 - a_1$.

2.3 Extension

Once the list of matching pairs is generated between a pair of source and suspicious documents, the next step is the extension or merging of these matches into contiguous, aligned passages. This is done by combining seeds or matches that satisfy certain criteria both in the source and suspicious documents into larger matches. We identify four distinct categories of matches as follows based on their vicinity towards each other.

Consider two matching pairs $(x_1, y_1, s_1) \rightarrow (a_1, b_1, s'_1)$, $(x_2, y_2, s_2) \rightarrow (a_2, b_2, s'_2)$ where (x_1, y_1, s_1) indicates text from position x_1 to position y_1 of size s_1 in the source document, and (a_1, b_1, s'_1) indicates the corresponding text from position a_1 to b_1 of size s'_1 in the suspicious document. In order to merge these pairs in each of the source and suspicious documents, we identify four categories of matches as follows:

1. **Containment**: Containment is the relationship between two matches within the same document when the text-positions of one match are fully contained within the text-positions of the other. More formally, the match (x_2, y_2, s_2) is contained within (x_1, y_1, s_1) if $x_2 \geq x_1$, $y_2 \leq y_1$ and the size $s_1 \geq s_2$.
2. **Overlap**: Two matches (x_1, y_1, s_1) and (x_2, y_2, s_2) overlap if some, but not all text-positions of the first one are contained within the text-positions of second match. More formally, the match (x_1, y_1, s_1) overlaps (x_2, y_2, s_2) if $y_2 \geq y_1 \geq x_2 \geq x_1$ i.e., x_2 lies between x_1 and y_1, y_1 lies between x_2 and y_2.
3. **Near-Disjoint:** Two matches are near-disjoint, if they are close enough to be combined into a single chunk of text, i.e. separated by a small number of characters, but have no overlapping text-positions. More formally if (x_1, y_1, s_1) and (x_2, y_2, s_2) are two matches such that $0 \leq x_2 - y_1 \leq gap$, where the parameter `gap` depends on the particular features of the training corpus, then the matches may be categorized as near-disjoint.
4. **Far-Disjoint**: Two matches (x_1, y_1, s_1) and (x_2, y_2, s_2) are far-disjoint if they are separated by such a large number of characters that they cannot be adequately merged into a single chunk of text. In this case $x_2 - y_1 > gap$.

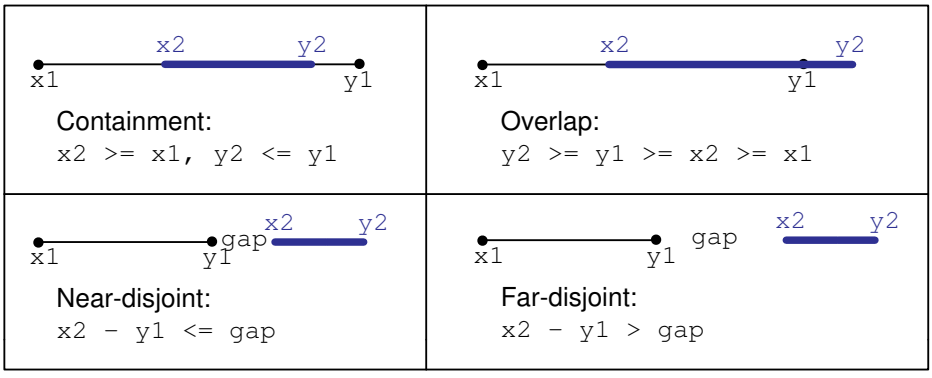

Figure 3. Classes of Matching pairs

Strategy for Merging The overall strategy for merging is based on the idea that two matching pairs that have either one of the containment, overlap or near-disjoint relationship can be merged towards forming a larger 3-tuple. 3-tuples that are far-disjoint cannot be merged. Furthermore, if two mapping pairs of 3-tuples have a different relationship in the source and suspicious documents then these tuples may be combined according to their respective categories. However, in a particular case of 3-tuples, i.e., when the 3-tuples have a near-disjoint relationship in one document and overlap or containment relationship in the other document, then no merging may be carried out. This is because it is a likely case of *term-repetition*, a situation in which a term is repeated within one of the documents, but it may mistakenly map to a large portion of text in the other document.

Figure 3 shows the four categories, while Table 1 gives a case-by-case listing of the merging strategies that we used for merging.

2.4 Filtering

After seeding and merging, the next step is filtering. For PAN-2014 training corpus, it was observed that short passages, typically less than 200 characters resulted in a lot of false positives. Hence all passages which were less than 200 characters in the source document aligned with passages less than 100 characters in the suspicious document were removed. The final output was then written to the XML files.

Relationship in Source $(x_1, y_1, s_1), (x_2, y_2, s_2)$	Relationship in Suspicious $(a_1, b_1, s'_1), (a_2, b_2, s'_2)$	Replacement Action Replacement 3-tuple
Containment: (x_1, y_1, s_1) contains (x_2, y_2, s_2)	Containment	$(x_1, y_1, s_1) \rightarrow (a_1, b_1, s'_1)$
	Overlap	$(x_1, y_1, s_1) \rightarrow (a_1, b_2, b_2 - a_1)$
	Near-disjoint	No change (Term Repetition likely)
	Far-disjoint	Merging not possible
Overlap: (x_1, y_1, s_1) overlaps (x_2, y_2, s_2)	Containment	$(x_1, y_2, y_2 - x_1) \rightarrow (a_1, b_1, s'_1)$
	Overlap	$(x_1, y_2, y_2 - x_1) \rightarrow (a_1, b_2, b_2 - a_1)$
	Near-disjoint	No change (Term Repetition likely)
	Far-disjoint	Merging not possible
Near-disjoint: $x_2 - y_1 \leq gap$	Containment	No change (Term Repetition likely)
	Overlap	No change (Term Repetition likely)
	Near-disjoint	$(x_1, y_2, y_2 - x_1) \rightarrow (a_1, b_2, b_2 - a_1)$
	Far-disjoint	Merging not possible
Far-disjoint $x_2 - y_1 > gap$	Containment	Merging not possible
	Overlap	
	Near-disjoint	
	Far-disjoint	

Table 1. Strategies for Merging Various Cases

3 Results

3.1 Setup

The first step before testing was to select the values of the parameters n and gap for character-n-grams and the gap size. Since this was a first time participation for us, one of the goals of our approach was to ensure at least a 90% precision while getting a high value of recall. After testing through a range of values, n = 20, gap = 200 was found to be the most suitable since, this set of values ensured at least 90% precision on all training corpora, while giving the highest values for the overall plagdet score and recall.

3.2 Training Corpora

We tested our approach on three corpora [7] for training: PAN-2014 training corpus, PAN-2013 test corpus 1 and PAN-2013 test corpus 2. Figure 4 gives a graphical representation of the results on the three corpora. The scores ranges obtained were as follows:

(a) Overall plagdet scores: 0.6 – 0.7 for each corpus.
(b) No-plagiarism scores: a full plagdet score of 1.0 for each corpus,
(c) No-obfuscation scores: > 0.9 score for each corpus,
(d) Random obfuscation scores: 0.4 – 0.5 for each corpus,
(e) Translation obfuscation scores: 0.5 – 0.6 for each corpus,
(f) Summary obfuscation scores: < 0.1 for each corpus.

These results imply that our approach based on hashing and merging heuristics gives very good plagdet scores for no plagiarism and no obfuscation cases, mid-range scores for random and translation obfuscations, and marginal scores for summary obfuscation.

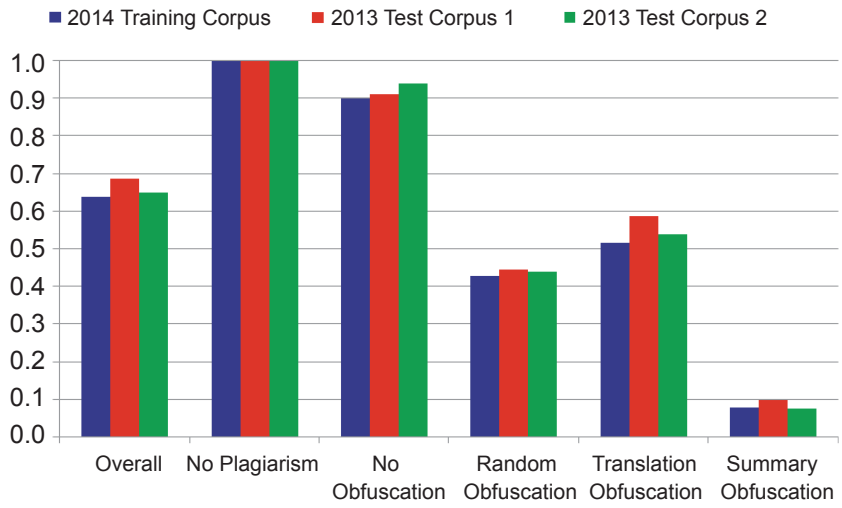

Figure 4. Results on Training Corpora

3.3 Test Corpora

For the final testing, the final software was uploaded on 30/April/2014 and test runs on all three test corpora were done on 16/May/2014. No errors were found during the runs, and hence exactly a single run was done on each of the test corpora. Results of these tests are available and are listed in Table 2. It can be observed from table 2 that the scores are inline with those obtained for the training corpora earlier.

	Test Corpus 2	Test Corpus 3
Plagdet	0.65954	0.73416
Precision	0.93375	0.90081
Recall	0.55068	0.67283
Granularity	1.07111	1.06943
Runtime	4m 57s	4m 17s
Documents	5185	4800

Table 2. Results on Test Corpora

4 Conclusion and Future Work

In this work, we used hashing and merging heuristics on character n-grams for text reuse detection. Our approach scored a mid-range performance on PAN-2014 test corpora. From the training corpora, it was obvious that while the approach scored very well in no plagiarism and no obfuscation types, there was room for improvement for translation and random obfuscations. Furthermore, a completely new approach may need to be devised for the marginal summary obfuscation scores.

Several directions of future work arise from here. For the current approach, a better detection with improved recall can be achieved if a more fine-grained merging mechanism is adopted with further sub-cases. Likewise, an improved granularity would also help in improvising detection score, by confining matches to only the necessary number.

On a different level, we can devise new approaches by using a variety of mechanisms such as character skip grams [1] or other natural language processing mechanisms for a more indepth processing. This could provide new insights into text reuse detection on random, translation and summary obfuscations.

5 Acknowledgement

Faisal Alvi would like to acknowledge the support provided for this work by the Deanship of Scientific Research at King Fahd University of Petroleum & Minerals (KFUPM) under Research Grant RG-1113-1 & 2.

References

1. Järvelin, A., Järvelin, A., Järvelin, K.: s-grams: Defining Generalized n-grams for Information Retrieval. Information Processing Management 43(4), 1005–1019 (Jul 2007)
2. Karp, R.M., Rabin, M.: Efficient Randomized Pattern-Matching Algorithms. IBM Journal of Research and Development 31(2), 249–260 (March 1987)
3. Moraru, I., Andersen, D.G.: Exact Pattern Matching with Feed-forward Bloom Filters. Journal of Experimental Algorithmics 17, 3.4:3.1–3.4:3.18 (Sep 2012)
4. Muth, R., Manber, U.: Approximate Multiple Strings Search. In: 7th Annual Symposium, Combinatorial Pattern Matching. pp. 75–86. Lecture Notes in Computer Science, Springer (1996)
5. Potthast, M., Gollub, T., Hagen, M., Graßegger, J., Kiesel, J., Michel, M., Oberländer, A., Tippmann, M., Barrón-Cedeño, A., Gupta, P., Rosso, P., Stein, B.: Overview of the 4th International Competition on Plagiarism Detection. In: Forner, P., Karlgren, J., Womser-Hacker, C. (eds.) Working Notes Papers of the CLEF 2012 Evaluation Labs (Sep 2012)
6. Potthast, M., Gollub, T., Hagen, M., Tippmann, M., Kiesel, J., Rosso, P., Stamatatos, E., Stein, B.: Overview of the 5th International Competition on Plagiarism Detection. In: Forner, P., Navigli, R., Tufis, D. (eds.) Working Notes Papers of the CLEF 2013 Evaluation Labs (Sep 2013)
7. Potthast, M., Stein, B., Barrón-Cedeño, A., Rosso, P.: An Evaluation Framework for Plagiarism Detection. In: 23rd International Conference on Computational Linguistics (COLING '10). pp. 997–1005 (Aug 2010)