ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection [*]

Cristian Grozea
Fraunhofer FIRST
IDA Group
Kekulestrasse 7,
12489 Berlin, Germany
cristian.grozea@first.fraunhofer.de

Christian Gehl
Fraunhofer FIRST
IDA Group
Kekulestrasse 7,
12489 Berlin, Germany
christian.gehl@first.fraunhofer.de

Marius Popescu
University of Bucharest
Faculty of Mathematics and Computer Science
Academiei 14, Sect. 1,
Bucharest, Romania
popescunmarius@gmail.com

Abstract: In this paper we describe a new general plagiarism detection method, that we used in our winning entry to the 1st International Competition on Plagiarism Detection, the external plagiarism detection task, which assumes the source documents are available. In the first phase of our method, a matrix of kernel values is computed, which gives a similarity value based on n-grams between each source and each suspicious document. In the second phase, each promising pair is further investigated, in order to extract the precise positions and lengths of the subtexts that have been copied and maybe obfuscated – using *encoplot*, a novel linear time pairwise sequence matching technique. We solved the significant computational challenges arising from having to compare millions of document pairs by using a library developed by our group mainly for use in network security tools. The performance achieved is comparing more than 49 million pairs of documents in 12 hours on a single computer. The results in the challenge were very good, we outperformed all other methods.

Keywords: n-gram, plagiarism detection, network security, challenge

1 Introduction

Many methods have been developed for plagiarism detection, especially for the *external plagiarism analysis*, which consists in finding passages in the suspicious documents which have been plagiarized and the corresponding text passages in the source documents. Almost all these methods handle the text at word level. Various comparison units have been employed in plagiarism detection methods. Entire documents are compared in (Lyon, Barrett, and Malcolm, 2004). Sentences from suspicious documents are compared to sentences from reference documents in (Kang, Gelbukh, and Han, 2006). Mixed-length comparisons in which suspicious sentences are compared with entire reference documents were used in (Barrón-Cedeño and Rosso, 2009; Barrón-Cedeño, Rosso, and Benedí, 2009). Irrespective of the comparison unit used, all methods of plagiarism detection need a similarity measure to compare the text fragments corresponding to the comparison unit. Most similarity measures used in plagiarism detection are based on estimating the amount of common configurations of words. They differ by the configurations considered (n-grams, subsequences, etc.) or by what words are used in comparisons (only words from the text fragments, stemmed or not, synonyms from WordNet, etc.). In (Lyon, Barrett, and Malcolm, 2004) word trigrams are used to measure the similarity between texts. The authors based their choice of using word trigrams for plagiarism detection on the fact that the number of common word trigrams in two independently written texts (even if the text are on the same topic) must be low given the Zipfian distribution of words. Also in (Barrón-Cedeño and Rosso, 2009) it is reported that using word bigrams and trigrams led to best results in their experiments. In order to address the prob-

[*] partly supported from the BMBF project ReMIND (KZ 01-IS07007A) and CNCSIS PN2-Idei project 228

lem of *rewording* in plagiarism, **PPChecker** (Kang, Gelbukh, and Han, 2006) is based on a special designed similarity measure, that takes into account also the synonyms (obtained from the WordNet) of the words in the suspicious sentences. Some of the most elaborate similarity measures used in plagiarism detection are described in (Bao et al., 2003; Bao et al., 2004a; Bao et al., 2004b). These measures are derived from the *string kernel*, a kernel type successfully used in text categorization (Lodhi et al., 2002). The string kernel works at character level, although in (Bao et al., 2003; Bao et al., 2004a; Bao et al., 2004b) it is extended to work at word level, comparing two semantic sequences according to their common words and position information.

Using words is natural in text analysis tasks like text categorization (by topic), authorship identification and plagiarism detection. Perharps surprisingly, recent results proved that methods that handle the text at character level can also be very effective in text analysis tasks. In (Lodhi et al., 2002) string kernels were used for document categorization with very good results. Trying to explain why treating documents as symbol sequences and using string kernels obtained such good results the authors suppose that: "the [string] kernel is performing something similar to stemming, hence providing semantic links between words that the word kernel must view as distinct". String kernels were also successfully used in authorship identification (Sanderson and Guenter, 2006; Popescu and Dinu, 2007). A possible reason for the success of string kernels in authorship identification is given in (Popescu and Dinu, 2007): "the similarity of two strings as it is measured by string kernels reflects the similarity of the two texts as it is given by the short words (2-5 characters) which usually are function words, but also takes into account other morphemes like suffixes ('ing' for example) which also can be good indicators of the author's style"[1]

For plagiarism detection, the only approach that handles the text at character level that we are aware of is in (Bao, Lyon, and Lane, 2006), for Chinese, and there is justified by the difficulties of the Chinese language (word segmentation).

There is a strong connection between the research in NLP and the research in computer network security. In recent years, network security research started to approach the problem of detecting automatically unknown attacks as soon as they reach the targeted system. These attacks may follow the syntax but try to exploit the semantics of the network communication between the client and the server applications, in order to gain access over the attacked computer or at least to prevent it from working normally. The communication process defined by the application layer protocols – e.g. HTTP, FTP, RPC or IMAP – can also be considered as a text-based communication in an artificial language. The idea of payload analysis, which treats the data as sequences of bytes has been explored in detail (Kruegel, Toth, and Kirda, 2002; Wang and Stolfo, 2004; Rieck and Laskov, 2006; Wang, Parekh, and Stolfo, 2006; Rieck and Laskov, 2007). As the focus in this field shifted towards applying more advanced machine learning methods, generalizing the extraction and representation of the features has increased much the flexibility in defining similarity measures between sequential data, in a security context. The work (Rieck and Laskov, 2008) presents an efficient way to combine features extracted from byte sequences, e.g. *words* or n-grams with arbitrary n value, for a wide range of linear and non-linear similarity measures.

Graphics methods in comparing sequences have been used in many fields, mostly under the name *dotplot* – see (Maizel and Lenk, 1981) for one of the first uses in biology and (Church and Helfman, 1993) for uses in source text comparison. Whereas very attractive for exploratory data analysis, building this graphic is potentially quadratic in time and space. Also it tends to be noisy, by showing many irrelevant coincidences between the sequences compared. Even with these limitations, the method has been applied to source code, videos, music, protein and other biological sequences, with various ways to filter the noisy graphics and to handle the problem of the potential quadratic size. We improve on this technique by deriving our own, linear space, linear time technique, that we named the *encoplot*, short for "eN-gram COincidence PLOT". It is fully described in Section 2.3, with code in Appendix 1.

[1]the string kernel used in (Popescu and Dinu, 2007) takes into account substrings of length up to 5 characters.

distance function $\mathbf{d(x,y)}$	
Minkowski	$\sqrt[k]{\sum_{ng \in A_n} \lvert\phi_{ng}(x) - \phi_{ng}(y)\rvert^k}$
Canberra	$\sum_{ng \in A_n} \frac{\lvert\phi_{ng}(x) - \phi_{ng}(y)\rvert}{\phi_{ng}(x) + \phi_{ng}(y)}$
kernel function $\mathbf{k(x,y)}$	
linear kernel	$\sum_{ng \in A_n} \phi_{ng}(x) \cdot \phi_{ng}(y)$
RBF kernel	$exp(-\frac{\sum_{ng \in A_n} \lvert\lvert\phi_{ng}(x) - \phi_{ng}(y)\rvert\rvert^2}{2\sigma^2})$

Table 1: Distances and kernels functions for sequential data.

Our plagiarism detection method can be described as a combination of techniques from many fields: it is character n-gram based. It leverages a very efficient network security software to compute the matrices of kernel values. It uses the very fast *encoplot* algorithm and processes the *encoplot* data in a quantitative fashion to solve what can be seen as a rudimentary machine vision or a specialized 2-dimensional data clustering task, in order to identify the matching text passages for a given document pair, as explained thoroughly below.

In what follows, the dataset specifics and the time performance figures refer to the dataset of the 1^{st} International Competition on Plagiarism Detection, external plagiarism detection task (Webis at Bauhaus-Universität Weimar and NLEL at Universidad Politécnica de Valencia, 2009). The development corpus of this dataset contained about 7000 source documents and 7000 suspicious ones, with the plagiarism generated automatically with various degrees of obfuscation (permutations, words deleted, inserted or replaced by synonyms or antonyms) and annotated. The competition corpus had the same characteristics (different documents) and the annotation was missing.

2 Methods

Our approach consists of two main phases. In the first phase, a matrix of string kernel values is computed, which gives a similarity value between each source and each suspicious document. Then, for each source, the possible "destinations" (suspicious documents) are ranked based on their similarity level with the current source, in decreasing order. In the second phase, each promising pair is further investigated, in order to extract the precise positions and lengths of the subtexts that have been copied and maybe obfuscated by the random plagiarist. In the end we do a supplementary filtering that increases the precision with the price of decreasing the recall.

2.1 Selecting a kernel and computing the matrix of kernel values for a large set of documents

Based on the work of (Rieck and Laskov, 2008), a *C* library for sequential data, *libmindy*, has been implemented by our network security research group. It has been developed mainly for being used in building real-time network analysis tools at packet level, as part of network intrusion detection and prevention systems. It can map byte sequences to a vectorial n-gram representation, such that the similarity between two byte sequences can be expressed in terms of distance and kernel functions on those representations. The n-gram extraction set of feasible byte sequences is given by $A_n = \Sigma^n$, where Σ is the alphabet (in our case the whole ASCII–8 set). The n-gram embedding function ϕ for a byte sequence x is then defined as $\phi(x) = (\phi_{ng}(x))_{ng \in A_n}$ with $\phi_{ng}(x) = \text{emb}(x, ng)$, where the dimension of the vector $\phi(x)$ is $|A_n|$. The function $\text{emb}(x, ng)$ returns either the frequency, the count or the presence bit for a n-gram ng in x. With the embedding function ϕ fixed, one can compute a pairwise similarity value for the vectorial representations of two byte sequences. Table 1 presents a selection of the implemented distances and similarity measures that we could have used (where x and y are arbitrary byte sequences).

Experiments with a very small subset of only 5 documents and our previous experience in string kernels led us to use the linear kernel over a representation where every n-gram present is marked by 1 and every other is marked by 0 (ignoring thus the frequencies of the n-grams). The kernel was normalized, such as $K(x, x) = 1$ for any string x. For the length of the n-grams we used 16 characters. Although in our estimations 18 should have been better (closer to three times the average word length plus two separators), the speed-up of the software used can only be obtained up to n-grams of length 16, see below and Appendix 1 for details. Using windows of two to three words in plagiarism detection was found to be the best choice by (Lyon, Barrett, and Malcolm, 2004) and (Barrón-Cedeño and Rosso, 2009).

The computation of a matrix of kernel values with sizes as large as 7000 is computationally intensive. There are more than 49 million pairs of documents for which the kernel value has to be computed, in each of the two datasets, the development and the competition corpus, accounting for a total of more than 98 million pairs to consider. *libmindy* has had already a tool for building a (symmetric) kernel matrix for a set of documents. We extended this tool for being able to handle asymmetric matrices of kernel values, where the kernel values are computed for each $x \in X$ and $y \in Y$, where X and Y are two independent finite sets of files, not necessarily having the same cardinal. While the new tool could in principle perform the task fast enough, it would have needed an amount of RAM of about 400 GB for a kernel based on length 16 n-grams. To avoid this issue, we partitioned the matrix of kernel values in blocks of sizes up to 1000x1000 (1 million pairs in most blocks), which required only 8 to 10 GB of RAM for processing. Those 64 blocks per dataset we processed one after the other, but the processing of each block was fully parallelized on the 8 cores of the machine, as a result of internally distributing the tasks by the means of OpenMP programming. Processing a full dataset took 12 hours on the machine we used (Dell Precision T7400). Although we had access to a cluster, it offered only a 32-bit environment. This would have slowed the whole processing by a factor that would almost completely eliminated the advantage of having 8 to 12 times more computing cores, and this is why we decided to use a single multi-core computer.

2.2 Pruning of the pairs

If the total processing for one pair of documents (up to book length level) would only take one second, this would lead to a total computation time of more than three years! Even by successfully parallelizing this task and dividing the time by hopefully 8 (the number of computing cores), the time needed would have been more than 4 months. It was obvious that even with the matrix of kernel values computed, there is too much work in comparing the documents in each pair. Pruning was seen from the start as a requirement, the question was what effect will it have on limiting the performance that can be achieved. We have considered ranking the pairs such that the ones with most chances of corresponding to plagiarism come first. Ranking on the absolute values of the kernel proved to work worst. Ranking for each source the suspicious documents proved to provide a consistent 10% advantage over ranking for each suspicious document the sources. Therefore, given also the values that can be seen in the Figure 1, we decided to limit our effort to the first 51 most promising suspicious documents for each given source.

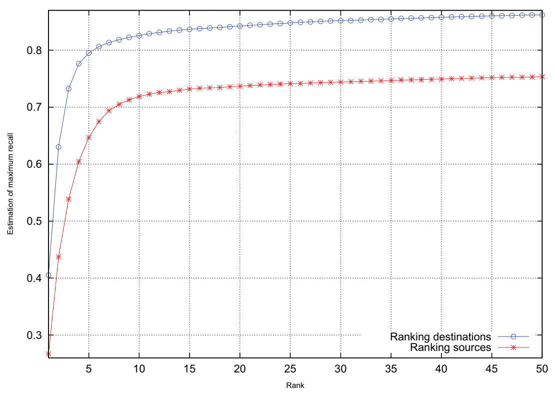

Figure 1: Maximum achievable recall for different pruning thresholds. Ranking the suspicious documents for each source leads consistently to better values than ranking the sources for each suspicious document.

2.3 Comparing two documents - The *encoplot*

With the maximum effort down to an estimate of about 100 hours, assuming spending in average a second per exhaustive document comparison (with the hope of reducing it to 12 hours by multicore parallelism), we proceeded to search for a way to identify what the documents have in common, if anything. Essential to this was the visualization of the coincidence pattern of n-grams between two documents. This is a scatter plot of a sublist of the positions where both texts have the same n-gram. We call this plot *encoplot*. Plots computed for pairs in the development corpus can be seen in Figures 2 and 3. All these plots use documents in the development dataset.

Related ideas (the “dotplot” graphs) exist about visualizing the n-grams that two texts (or sequences) share. The problem with those

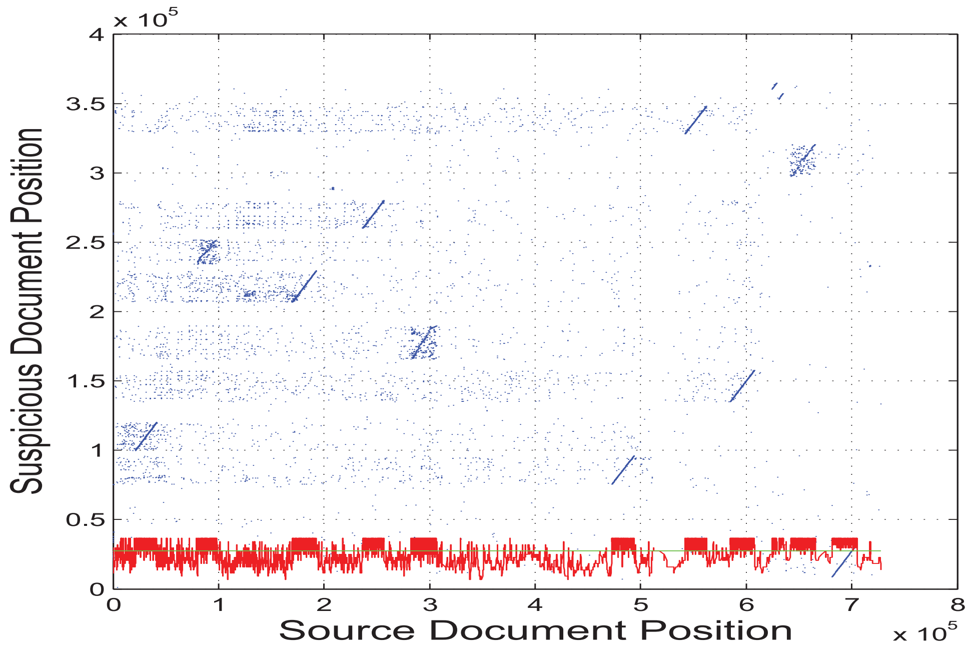

Figure 2: Encoplot for source #3094 and suspicious #9. Many plagiarism instances for the same document pair. The shattered look of some comes from higher obfuscation. In red, the local contiguity score, scaled.

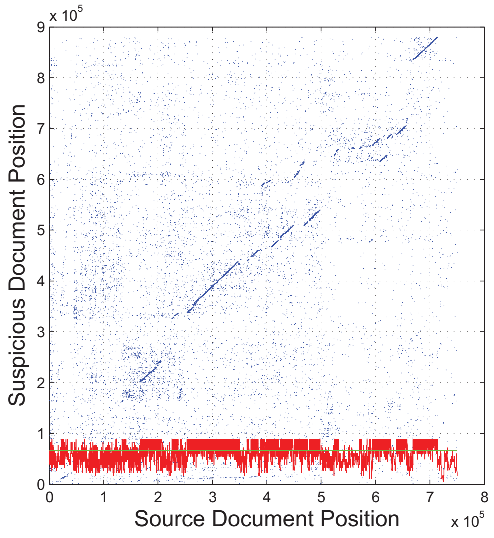

Figure 3: Encoplot for source #134 and suspicious #2499 – a real case of human (self) plagiarism.

is that the number of pairs can be quadratic in the size of the documents. For megabytes long texts, this can easily become computationally intractable. We solve this issue by limiting ourselves to a sublist that is guaranteed to be no longer than the shortest of the documents, and can be computed in linear time. The precise procedure we employed starts by sorting virtually the sets of n-grams for both documents to be compared. Then these ordered sets of n-grams are compared with a procedure that is derived from the procedure from merging two sorted lists. Every time the smallest elements of the two lists differ, the smallest of them is dropped, without producing any output. Every time the smallest elements of the lists are equal, the pair of positions on which this identical n-gram occurs is being collected by outputting it to the standard output. Code for this core procedure is given in Appendix 1. Please note that *encoplot* pairs the first instance of an n-gram in one document with the first instance of the same in the other document, the second one with the second one and so on – as opposed to the dotplot, wich pairs each instance with each instance.

2.4 Heuristics used for separating the copied subtexts

Once the *encoplot* data (the list of pairs of indexes) is obtained, it is sorted by the value of the first index in each pair, which corresponds to the position in source of the common n-gram. From this list a local "contiguity" score is derived by computing whether there is simultaneously a small jump on both indexes (sum of absolute jumps less than 4) when going from a pair to the next pair, followed by a smoothing by a convolution with a constant vector of length 16. The contiguity score for an *encoplot* is displayed in red in Figures 2 and 2. Then a Monte Carlo optimization procedure is called, not more than 30 times for each document pair, which in 10 attempts tries to find the largest group from the current *encoplot* data. The start of the group is decided randomly with uniform distribution over the list of available pairs, then the group is extended to left and right such that the average contiguity score stays above 0.5 and there are no jumps (skipped portions) longer than 512 in any 16 steps. After a group is obtained, it is checked to have an average contiguity score of over 0.75 and a length of at least 256 characters. If not, it is rejected as insignificant. If kept, it is projected to the dimension of the indexes that correspond to the suspicious document, and only the compact core of it is preserved. The compact core is obtained by sorting on the suspicious document axis and eliminating the outliers by starting from the group center and extending it to left and right while the skips are less than 256 positions. What remains is projected back onto the source document axis, obtaining thus an estimate of the indexes whose convex hull define the two subtexts corresponding to each other. This candidate of a plagiarism instance is checked once again, this time for a final length of at least 256, for not having shrinked to less than half with respect to the initial group length and for the two subtexts not having sizes too different (the absolute difference more than half of the mean of the two lengths). This subset of the *encoplot* data is removed, the plagiarism instance is outputted if all tests succeeded, and the procedure is repeated in the search for more groups. If the group found fails to satisfy the checks, it is deemed as a failure. At three consecutive failures the search is abandoned and the treatment of the pair of documents is considered completed. This decision may be risky, but accelerates substantially this phase, as on very complicated document pairs it can take minutes to completely examine an involved pair. On the other hand, for the actually unrelated documents this ends the investigation rapidly. Technically, we have accelerated this processing phase even more by running simultaneously up to 10 detailed examinations of document pairs at a time, trying to balance the processing power required and the disk latency.

3 Results

We combined the best *F-measure* – the harmonic mean of precision and recall – 0.6976 (the next competitor had 0.6192) with the best granularity – lack of fragmentation in detection of the plagiated passages – 1.0027 (the next best value was 1.0164), winning thus the competition.

4 Discussion and Conclusions

The first question is whether our choice to compare the documents in pairs was optimal. Indexing based methods could be faster, by eliminating the need for exhaustive pairwise comparison of documents in a large corpus. They function by first indexing the collection of source documents and then searching for parts of the suspicious documents in the index, as the system MOSS (Schleimer, Wilkerson, and Aiken, 2003) does. Such an inflexible approach cannot handle well obfuscation, as opposed to our approach. On the other hand, flexible matching is an always-current research topic in information retrieval systems (Navarro, 2001), and this eventually improves plagiarism detection as well. We think that, whereas needing more computational effort, our approach had the chance of producing better results. And, as a consequence of using highly optimized network analysis code, it did so in a reasonable time, even when run on a single contemporary computer, as opposed to a full cluster. One could say that it was closer to being optimal in terms of quality of the results, while still being acceptable in terms of running time.

A second question of interest is whether our values for the hyperparameters of the method are optimal for this dataset. The answer is probably no, but maybe not far from that. They have been chosen by educated guess guided by the exploratory data analysis, as opposed to blindly optimizing a cross-validation towards the best (over)fitting.

The third interesting issue is the claim of some experts that only the humans can have very good results at spotting plagiarism (Weber-Wulff, 2008). We think that, as far as the ethics is concerned, a human must look at the evidence before claiming a case as one of plagiarism. And of course, text understanding is still not within the reach of artificial intelligence yet. On the other hand, the claim that the only automatization in plagiarism detection should limit to using the one's favorite search engine and searching for paragraphs selected based on one's intuition is questionable. How would such an expert deal with 7000 documents up to a book length? How long would it take to process those by hand, even using a public search engine? How long does it take one to read 7000 works/books? The need for automatization seems evident, as it was to (Grozea, 2004) when he had to grade 400 projects from 60 students in less than 24 hours. Crowdsourcing could also be a possibility, but one needs very big crowds for that (optimally quadratic size, if using the same choice in the trade-off between speed and quality as we chose). Time is the key factor in plagiarism detection.

Given the very good results obtained by our method it is worth asking – and further investigating – whether using character n-grams offers any advantage over using word n-grams. First, let us note that our method uses n-grams of 16 characters which in average[2] correspond to word trigrams (the standard approach in plagiarism detection). It may seem that (on average) the same information is brought by 16 characters n-grams and word trigrams. What differentiates the two types of n-grams is in our opinion the fact that character n-grams favor long words over short ones, and when people copy text they do that for the content words of the copied text that tend to be longer than the functional words (stop words) which are short. For example: a common syntagmatic expression[3] like "as far as" will contribute with one word trigram, but with none character 16-gram. On the other hand, a sequence of content words (worth being copied) like "educated guess guided" will contribute again with only one word trigram, but with 6 character 16-grams.

Another item to discuss is how to balance precision and recall in automatic plagiarism detection systems. Given that a human is in many cases the final link in the chain that leads to the proof of plagiarism, the effort of that human must be spared as much as possible. The accuse of plagiarism is so strong, that it needs strong evidence. Both these aspects recommend to balance the precision and recall towards a high precision, even at the expense of lowering the recall. This is how we tuned our system's parameters, including but not limited to the last checking phase. Of course, accurate comparison of systems should take into account the entire precision-recall curve. By plotting on the same graph these curves for more systems, one could easily see where is the best performance region for each system and whether or not one of the systems is overall better than another system.

Related to the maximum achievable precision while keeping a fair recall is the issue of the documents independence and of the automatic plagiarism. The dataset contains plagiarism built automatically and randomly and only these borrowings between the source documents and the suspicious documents had to be found. But the documents were not independent enough: there are pairs of documents with the same or almost the same content, such as independent translations of "One Thousand and One Night" or several Bible editions, authors doing heavy reuse from their previous works (the so-called self-plagiarism). These are interesting in two ways: they are better examples of what the human plagiarism is, so spotting those as related is very good. On the other hand, this can be seen as unintended (by the organizers) plagiarism, so any such pair reported will actually lower the precision score.

A very interesting issue is the asymmetry of the ranking quality. Why is it 10% better to rank all suspicious documents for any fixed source instead of ranking all possible sources for every fixed suspicious document, as clearly seen in Figure 1? A possible source of this asymmetry is that while it was guaranteed for each suspicious document that the areas plagiated do not overlap, this was not the case for the source documents, where the areas plagiated could overlap. This asymmetry deserves more investigation, being one of the few glints of hope so far to tackling what could be the biggest open problem in automatic plagiarism detection, that is determining the direction of plagiarism in a pair of documents – being able to indicate with confidence which is the copy and which is the original.

To conclude, by combining advanced software engineering and effort-sparing heuristics tuned using the novel visualization technique *encoplot*, we have been able to achieve the top

[2]The average word length in the corpus is 5.2

[3]Frequently and systematically co-occurring lexical items.

placement in the final results, proving that the interaction of NLP researchers with networks security researchers can lead to high-performance NLP systems.

4.1 Acknowledgment

We thank Dr. Andreas Ziehe and the anonymous reviewers for the thorough review of our paper and for the useful suggestions.

References

Bao, Jun Peng, Caroline Lyon, and Peter C. R. Lane. 2006. Copy detection in chinese documents using ferret. *Language Resources and Evaluation*, 40(3-4):357–365.

Bao, Jun-Peng, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. 2003. Document copy detection based on kernel method. In *Proceedings of Natural Language Processing and Knowledge Engineering Conference (IEEE)*, pages 250–255.

Bao, Jun-Peng, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. 2004a. Finding plagiarism based on common semantic sequence model. In Qing Li, Guoren Wang, and Ling Feng, editors, *WAIM*, volume 3129 of *Lecture Notes in Computer Science*, pages 640–645. Springer.

Bao, Jun-Peng, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. 2004b. Semantic sequence kin: A method of document copy detection. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 529–538. Springer.

Barrón-Cedeño, Alberto and Paolo Rosso. 2009. On Automatic Plagiarism Detection based on n-grams Comparison. In Mohand Boughanem, Catherine Berrut, Josiane Mothe, and Chantal Soulé-Dupuy, editors, *ECIR 2009*, volume 5478 of *LNCS*, pages 696–700, Toulouse, France. Springer.

Barrón-Cedeño, Alberto, Paolo Rosso, and José-Miguel Benedí. 2009. Reducing the Plagiarism Detection Search Space on the Basis of the Kullback-Leibler Distance. In Alexander F. Gelbukh, editor, *CICLing 2009*, volume 5449 of *Lecture Notes in Computer Science*, pages 523–534, Mexico, Mexico. Springer.

Church, K.W. and J.I. Helfman. 1993. Dotplot: A program for exploring self-similarity in millions of lines of text and code. *Journal of Computational and Graphical Statistics*, pages 153–174.

Grozea, C. 2004. Plagiarism detection with state of the art compression programs. Report CDMTCS-247, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, August.

Kang, NamOh, Alexander F. Gelbukh, and Sang-Yong Han. 2006. Ppchecker: Plagiarism pattern checker in document copy detection. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *TSD*, volume 4188 of *Lecture Notes in Computer Science*, pages 661–667. Springer.

Kruegel, C., T. Toth, and E. Kirda. 2002. Service specific anomaly detection for network intrusion detection. In *Proc. of ACM Symposium on Applied Computing*, pages 201–208.

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Christopher J. C. H. Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Lyon, Caroline, Ruth Barrett, and James Malcolm. 2004. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. In *Plagiarism: Prevention, Practice and Policies Conference*.

Maizel, J.V. and R.P. Lenk. 1981. Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proceedings of the National Academy of Sciences*, 78(12):7665–7669.

Navarro, G. 2001. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1):31–88.

Popescu, Marius and Liviu P. Dinu. 2007. Kernel methods and string kernels for authorship identification: The federalist papers case. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP-07)*, Borovets, Bulgaria, September.

Rieck, K. and P. Laskov. 2008. Linear-time computation of similarity measures for sequential data. *The Journal of Machine Learning Research*, 9:23–48.

Rieck, Konrad and Pavel Laskov. 2006. Detecting unknown network attacks using language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proc. of 3rd DIMVA Conference*, LNCS, pages 74–90, July.

Rieck, Konrad and Pavel Laskov. 2007. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256.

Sanderson, Conrad and Simon Guenter. 2006. Short text authorship attribution via sequence kernels, markov chains and author unmasking: An investigation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 482–491, Sydney, Australia, July. Association for Computational Linguistics.

Schleimer, S., D.S. Wilkerson, and A. Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM New York, NY, USA.

Wang, K., J.J. Parekh, and S.J. Stolfo. 2006. Anagram: A content anomaly detector resistant to mimicry attack. pages 226–248.

Wang, K. and S.J. Stolfo. 2004. Anomalous payload-based network intrusion detection. pages 203–222.

Weber-Wulff, Debora. 2008. Softwaretest, http://plagiat.htw-berlin.de/software/2008/.

Webis at Bauhaus-Universität Weimar and NLEL at Universidad Politécnica de Valencia. 2009. PAN Plagiarism Corpus PAN-PC-09. http://www.webis.de/research/corpora. Martin Potthast, Andreas Eiselt, Benno Stein, Alberto Barrń Cedeño, and Paolo Rosso (editors).

A Appendix 1: Encoplot code

This appendix provides the listing of the implementation of the *encoplot* algorithm. At its core is a very fast implementation of the radix sort algorithm for virtually sorting the n-grams in a text without swapping any memory blocks. It is a specialization of the general radix sort algorithm. The key part is avoiding to recompute the frequencies at each step in the radix sort algorithm, and relying instead on updating those incrementally. Another key technical aspect is the use of the 128 bit unsigned integer type __uint128_t, possible with the gcc compiler on certain platforms, which allows for very good speeds up to n-grams of length 16, on 64-bit architectures, such as the common x86-64. The main code uses this virtual sorting of the n-grams sets to compute the *encoplot* data of two given files, a central part of our plagiarism detection method, as explained above.

```
//computes the encoplot data of a pair of files
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
typedef __uint128_t tngram;
//CrG rsort
#define fr(x,y)for(int x=0;x<y;x++)

int* index_rsort_ngrams(
        unsigned char *x, int l, int DEPTH){
int NN=l-DEPTH+1;if(NN>0){
unsigned char *pin=x+NN;
unsigned char *pout=x;
int *ix=(int*)malloc(NN*sizeof(int));
int *ox=(int*)malloc(NN*sizeof(int));
const int RANGE=256;
int counters[RANGE];int startpos[RANGE];
fr(i,NN)ix[i]=i;
//radix sort, the input is x,
// the output rank is ix
fr(k,RANGE)counters[k]=0;
fr(i,NN)counters[*(x+i)]++;
fr(j,DEPTH){int ofs=j;//low endian
        int sp=0;
        fr(k,RANGE){startpos[k]=sp;
                sp+=counters[k];}
        fr(i,NN){unsigned char c=x[ofs+ix[i]];
                ox[startpos[c]++]=ix[i];}
        memcpy(ix,ox,NN*sizeof(ix[0]));
        //update counters
        if(j<DEPTH-1){
counters[*pout++]--;counters[*pin++]++;}}
free(ox);return ix;}}
#define MAXBUFSIZ 8000123
unsigned char file1[MAXBUFSIZ];
unsigned char file2[MAXBUFSIZ];
int l1,l2;

inline tngram readat(
        const unsigned char *buf,int poz){
        return *(tngram *)(buf+poz);}

int main(int argc, char ** argv){
  int depth=sizeof(tngram);
  FILE *f1=fopen(argv[1],"rb");
  l1=fread(file1,1,MAXBUFSIZ,f1);fclose(f1);
  FILE *f2=fopen(argv[2],"rb");
  l2=fread(file2,1,MAXBUFSIZ,f2);fclose(f2);
//index the ngrams
int *ix1=index_rsort_ngrams(file1,l1,depth);
int *ix2=index_rsort_ngrams(file2,l2,depth);
int i1=0;int i2=0;//merge
tngram s1=readat(file1,ix1[i1]);
tngram s2=readat(file2,ix2[i2]);
l1-=(depth-1);l2-=(depth-1);
while(i1<l1 && i2<l2){
if(s1==s2){
        printf("%d %d\n",ix1[i1],ix2[i2]);
        i1++;if(i1<l1)s1=readat(file1,ix1[i1]);
        i2++;if(i2<l2)s2=readat(file2,ix2[i2]);}
else if(s1<s2){
        i1++;if(i1<l1)s1=readat(file1,ix1[i1]);}
else if(s2<s1){
        i2++;if(i2<l2)s2=readat(file2,ix2[i2]);}}
free(ix2);free(ix1);return 0;}
```