Syntactic Similarity of Web Documents

Álvaro R. Pereira Jr   Nivio Ziviani
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil
{alvaro, nivio}@dcc.ufmg.br

Abstract

This paper presents and compares two methods for evaluating the syntactic similarity between documents. The first method uses the Patricia tree, constructed from the original document, and the similarity is computed searching the text of each candidate document in the tree. The second method uses shingles concept to obtain the similarity measure for every document pairs, and each shingle from the original document is inserted in a hash table, where shingles of each candidate document are searched. Given an original document and some candidates, two methods find documents that have some similarity relationship with the original document. Experimental results were obtained by using a plagiarized documents generator system, from 900 documents collected from the Web. Considering the arithmetic average of the absolute differences between the expected and obtained similarity, the algorithm that uses shingles obtained a performance of 4.13% *and the algorithm that uses Patricia tree a performance of* 7.50%.

1. Introduction

The problem of detecting web documents that have some similarity degree with a given input document has been studied in several contexts: search engines avoid indexing similar documents in their document bases, people wish to find documents that originated an input text, or even detect plagiarism between several documents obtained from the Web, among others.

This work is part of a system called “Copy Detection Mechanism of Web Documents”. It consists of a system that search and match similar documents to an input document. From a text document whose “possible predecessor” we wish to find, the system searches the Web for similarity candidate documents and then calculate the similarity degree of these documents with the document informed by the user. The word “similarity” will be used to indicate the “similarity degree” between “candidate documents” and the input document, called here “plagiarized document”. Figure 1 shows the main steps performed by the system, as follows:

1. The user presents the input document that he wishes to find similar documents;
2. The input document is converted to ASCII format;
3. The parsing of input document is performed to obtain a “fingerprint” that will be used in the meta-search step;
4. The meta-search is performed in different search engines using the document fingerprint;
5. Several candidate documents are returned and converted to ASCII format;
6. The similarity between the original document and each candidate is calculated and returned to the user.

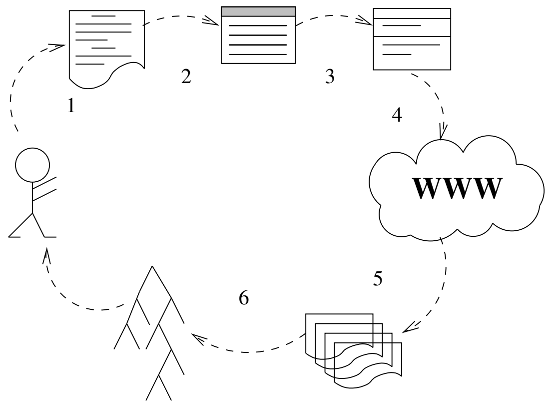

Figure 1. copy detection mechanism of web documents

This paper presents two methods for detecting and evaluating the syntactic similarity between documents: Patricia tree and shingles. Given a plagiarized and several candidate documents, the two methods allow to find documents with some similarity relationship with the plagiarized document.

The first method uses the Patricia tree [11]. The Patricia tree is constructed over the plagiarized document and the candidate documents have their contents searched on the tree, which allows detect occurrences of long similar passages in the plagiarized document. The Patricia tree construction algorithm can be found in [1, 16] and has time complexity O($n \log n$). A quadratic algorithm was proposed in [2] for secondary memory. In [15] a linear algorithm is proposed.

The second method uses the “shingles” concept [4] for measuring syntactic similarity between each candidate document and the plagiarized, compared in pairs. The total number of shingles present and not present in each pair of documents is used to calculate the pair similarity.

The aim of this work is to study the similarity between a given document and a set of candidate documents, which represents step 6 of the system “Copy Detection Mechanism of Web Documents”. Experimental results were obtained using a set of documents generated by a plagiarized document generator system. The similarity of a generated document was compared with each candidate document used in its composition.

2. Inserting and Searching SiStrings in the Patricia Tree

Patricia tree (Practical Algorithm To Retrieve Information Coded In Alphanumeric) algorithm was presented in [11]. It is a binary digital tree where individual bits from the key are used to decide the branch that should be followed. A bit “zero” will indicate a branch to the left sub-tree and a bit “one” will indicate a branch to the right sub-tree. Each tree internal node contains an integer that indicates which bit of the query might be analyzed for branching. The external nodes store key values [16, 1].

A semi-infinite string, SiString, is a subsequence of characters from the text document, taken from a given starting point but going on as necessary to the right. It was also used in [10], as a data structure called suffix arrays.

Figure 2 shows the SiStrings for “uma rosa é uma rosa.”, considering each character as indexing points, and Figure 3 shows for the same example the SiStrings considering the beginning of each term as indexing points.

In the following we show an example of a Patricia tree for the SiStrings of Figure 3. For the understanding of the process, we take the binary extended ASCII code of the first character of every different term of the example: *é* = 11101001, *r* = 01110010 and *u* = 01110101. We also consider the code of the character that indicates the end of the text, in this case “.” = 00101110 and the space character

uma rosa é uma rosa.
ma rosa é uma rosa.
a rosa é uma rosa.
rosa é uma rosa.
......
osa.
sa.
a.

Figure 2. SiStrings with indexing points being each character

1. uma rosa é uma rosa.
2. rosa é uma rosa.
3. é uma rosa.
4. uma rosa.
5. rosa.

Figure 3. SiStrings with indexing points being the begin of the terms

“ ” = 00100000. Figure 4 shows the Patricia tree for the SiStrings of the text “uma rosa é uma rosa.”. The internal nodes contain a number that represents the order of the bit that does not match the SiStrings that appear in that subtree. The external nodes have pointers to places on the text.

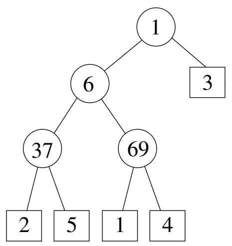

Figure 4. Patricia Tree using SiStrings

As characters *r* and *u* differ from character *é* in the first bit, the tree root node has the value 1. Going to the right subtree, we find a leaf node pointing to the single SiString that begins with the character *é*. Going to the left sub-tree, we find that the bit that differs these characters is the sixth, and the number 6 appears in the next internal node. From this point, going to the left sub-tree, the number 37 appears in internal node indicating that SiStrings 2 and 5 differ themselves only in their thirty seventh bit. This can be verified by noticing that SiStrings have 4 equal characters. Following, the SiString 2 has a space and the SiString 5 has an indicative of end of the text. Comparing the set of bits that represents these two characters, we verify that they differ themselves by the fifth bit. Thus, the number 37 in the node indicates that they have 4 equal characters ($4 * 8 = 32$) and in the fifth character they differ themselves by the fifth bit ($32 + 5 = 37$).

Figure 5 presents the algorithm. We consider the beginning of each term as indexing points. Meaning of each variable is:

D_p – plagiarized document;
D_{ci} – candidate document i;
len_p – number of characters of plagiarized document;
SiS – set of SiStrings from plagiarized document;
N_{ci} – number of characters matched for candidate document i;
S_{ci} – Similarity counter for candidate i;
S_{ti} – Total similarity between candidate document i and the plagiarized document;

```
1. read Dp and each Dci;
2. lenp = Length(Dp);
3. SiS = GenerateSiStrings(Dp);
4. for each SiSi generated:
      InsertPatricia(SiSi);
5. for each Dci:
      while (Dci != EOF):
            Nci = SearchPatricia(Dci);
            if (Nci > 15):
                  Sci = Sci + Nci;
            Dci = Dci + Nci;
      Sti = Sci/lenp;
      return Sti;
```

Figure 5. Patricia tree algorithm

Each candidate document is searched in the Patricia tree obtained for the plagiarized document. The similarity counter S_{ci} is incremented only when $N_{ci} > 15$. This is the way of eliminating "false matches" in the answer set. Other values were tried but 15 gave the best results. The new text for searching is taken by $D_{ci} = D_{ci} + N_{ci}$.

The total similarity $S_{ti} = S_{ci}/len_p$ indicates how much of candidate document i is present in the plagiarized document.

As an example, consider D_p = "uma rosa é uma rosa é uma rosa." and D_{ci} = "nunca uma rosa é uma rosa é uma violeta.". The algorithm starts reading the two documents and storing the length of D_p, which is 31 characters long, in len_p. Next the SiStrings of D_p are inserted in the Patricia tree. From this point, the algorithm computes Sci, by searching each D_{ci} in the Patricia tree obtained for D_p. The result of this search, given in number of characters, is stored in N_{ci}. If this number is greater than fifteen, Sci is incremented by N_{ci}. The new D_{ci} will be the remaining of the SiString that it was not found in the search. In the example, the first six searches return values less than 15 for N_{ci}, since the term "nunca " is not found in the tree. The next SiString in D_{ci} is "uma rosa é uma rosa é uma violeta.". The search for this SiString returns $N_{ci} = 26$, since the matched string is "uma rosa é uma rosa é uma ". The search for the remaining of the SiString also does not return $N_{ci} > 15$, so, the final value of Sci is 26. Thus, $S_{ti} = 26/31 = 83.9\%$.

Patricia trees are very efficient because only O(log$n$) bit inspections are necessary to obtain the whole set of SiStrings answering a query [3], where n is the number of indexing points.

3. Use of Shingles in the Similarity Measure

According to [4], two documents A and B can present the relations of "resemblance" and "containment". The w-shingling $S(D, w)$ of a document D is the set of whole shingles with size w contained in D. This set represents the information used to calculate the similarity between documents. For example, the 4-shingling of the text "uma rosa é uma rosa é uma rosa" is:

$S(D, 4) = \{$(uma, rosa, é, uma), (rosa, é, uma, rosa), (é, uma, rosa, é)$\}$,

resulting in three different shingles with $w = 4$. In this work, the shingles that occur more than once in the text will appear only once in answer set, as with the two first shingles from the example. Experiments demonstrate that a better performance is obtained for this situation.

From the distinct set of whole shingles of two documents, the absolute similarity between them is calculated using the concept of intersection and union of sets, as shown:

$$r(P, C) = \frac{|S(P) \cap S(C)|}{|S(P) \cup S(C)|} \quad (1)$$

where $S(P)$ represents the set of whole shingles of the plagiarized document and $S(C)$ the set of shingles of the candidate document.

In practice, we have $S(P) \cap S(C)$ representing the total number of shingles occurring in the plagiarized document and in the candidate document and $S(P) \cup S(C)$ representing the sum of the number of shingles occurring simultaneously in two documents plus the number of shingles that occurs in each of the documents that do not occur in the other one.

In the same way, it is possible to verify how much of a candidate document C is contained in another plagiarized document P, as in the following equation:

$$c(P, C) = \frac{|S(P) \cap S(C)|}{|S(P)|} \quad (2)$$

Figure 6 presents the pseudo-code of the algorithm. It considers shi_p as the set of shingles of the plagiarized document and shi_c as the set of shingles of the candidate document.

```
1. read D_p and D_ci;
2. shi_p = GenerateShingles(D_p);
    for every shi_pi:
        res = SearchHash(shi_pi);
        if (result == 0): /*not found*/
            InsertHash(shi_pi);
            S(P)++;
3. shi_c = GenerateShingles(D_c);
    for every shi_ci:
        result = PesquisaHash(shi_ci);
        if (result! = 0):
            (S(P) ∩ S(C)) + +;
            DeleteHash(shi_ci);
5. return c(P, C);
```

Figure 6. Shingles algorithm

As an example of the execution of the algorithm consider D_p = "uma rosa é uma rosa é uma rosa", D_{ci} = "uma rosa é uma rosa vermelha ou branca.", and $w = 4$. By step 2, the w-shingles of D_p are taken and inserted in a hash table. Each different shingle is inserted only once, even if it occurs more than once. The total number of inserted shingles stored in $S(P)$ is 3, for this example: (uma, rosa, é, uma), (rosa, é, uma, rosa) and (é, uma, rosa, é). In step 3, shingles from D_{ci} are taken: (uma, rosa, é, uma), (rosa, é, uma, rosa), (é, uma, rosa, vermelha), (uma, rosa, vermelha, ou) and (rosa, vermelha, ou, branca); following, they are searched in the hash table. The total number of successful searches is stored in $(S(P) \cap S(C))$. In this case, two shingles are found. In step 4, the obtained similarity S_{ti} is calculated by $c(P, C) = 2/3 = 66.7\%$.

Some other works use hash to verify the similarity between documents. In the tool COPS [6], the sentences of documents are inserted in the hash table, while SCAM [13] inserts each different term in the table. Other works as [9, 12, 5, 7] use hashing with different heuristics for deciding the information to be stored, as well as different hash functions.

4. Experimental Results

4.1. Automatic Generation of Plagiarized Documents

We created a synthetic set of documents as follows. We composed a set of documents from passages of documents available in the Web, whose themes are given by the word from the query. The aim of the system is to simulate a composition of a document made by an user using pieces of documents from the web. The number of documents that might be used in the composition of a plagiarized document is given, as well as the number of terms that the plagiarized document might have related to the size of the documents returned from the search. It is also possible to insert the paragraphs in the plagiarized document. This functionality is added since the user that makes a plagiarism normally completes the document with its own text. This situation was not treated in the experiments.

In the initial step the system collects the first ten documents returned from a query performed by the search engine TodoBR [14]. Following, the HTML document is parsed to obtain the text in ASCII format, which is separated in paragraphs. Random paragraphs from each document are used to compose the plagiarized document, always maintaining the percentage of common terms that the plagiarized document has related to each document used in the composition. This information is the expected similarity, S_e, of the plagiarized document related to the candidate document.

Figures 7, 8, 9 and 10 present different documents [1] that are used to generate the plagiarized document presented in Figure 11. Next we will explain how the plagiarized document of Figure 11 was obtained.

"O amor quer abraçar e não pode.
A multidão em volta,
com seus olhos cediços,
põe caco de vidro no muro
para o amor desistir.

Figure 7. Document A

Document A from Figure 7, with 25 terms, will be considered as containing passages written by the user. The document is created with 60% of its text composed by the documents from the query (documents B, C and D, from Figures 8, 9 and 10, with 39, 16 and 22 terms, respectively) and 40% of its text composed from the user document (in this case document A). Thus, we hope that the plagiarized docu-

1 Pieces extracted from the poem "Corridinho", written by Adélia Prado.

O amor usa o correio,
o correio trapaceia,
a carta não chega,
o amor fica sem saber se é ou não é.
O amor pega o cavalo,
desembarca do trem,
chega na porta cansado
de tanto caminhar a pé.

Figure 8. Document B

Fala a palavra açucena,
pede água, bebe café,
dorme na sua presença,
chupa bala de hortelã.

Figure 9. Document C

Tudo manha, tudo truque, engenho:
é descuidar, o amor te pega,
te come, te molha todo.
Mas água o amor não é"

Figure 10. Document D

ment would be composed by at least $(60/3) = 20.0\%$ of the terms from each one of the three documents used in this example.

The system reads randomly paragraphs from each document and save, also randomly, paragraphs in plagiarized document. For document C, which contains 16 terms, 20% means approximately three terms. When taking each paragraph, considering that they have 4 terms each, no another paragraph from this document would be taken, since the similarity value is already above 20%. The expected similarity will be given by $S_e = (4 * 100)/12 = 33.3\%$. The paragraph used would be inserted in the plagiarized document randomly.

é descuidar, o amor te pega,
A multidão em volta,
dorme na sua presença,
"O amor quer abraçar e não pode.
a carta não chega,
para o amor desistir.
O amor pega o cavalo,

Figure 11. Example of plagiarized document

As the new document of Figure 11 has 15 terms from document A, 9 terms from document B, 4 terms from document C and 6 terms from document D, the expected similarity between the plagiarized document and each of its generators is 60.00%, 23.07%, 25.00% and 27.27%, respectively, for documents A, B, C e D, according to the total number of terms of each document used to compose the plagiarized document.

As the algorithm works by taking whole paragraphs from documents until the minimal similarity calculated is obtained (for our example, 20% for each document from the search and 40% for the user document), some values of expected similarity can be significantly greater than this limits, as happens with document A.

4.2. The Experiments

The collection used in the experiments was composed by the 10 first HTML pages returned from 900 different queries submitted to the search engine TodoBR [14]. The steps for the performance evaluation of the two implemented methods where:

1. Generation of plagiarized documents containing passages from the first 10 documents returned from 900 queries;
2. Use of shingles to determinate the similarity between the documents;
3. Use of the Patricia tree to determinate the similarity between the documents;
4. Calculation of the difference between the expected and obtained similarity for both Patricia and shingles algorithms;
5. Calculation of the average differences for each algorithm.

Step 1 was described in section 4.1. Step 2 was performed for w varying from 2 to 10. At this point, each document used to compose the plagiarized document D_p is matched against D_p using the shingles algorithm presented in section 6. In step 3, a procedure similar to the one presented in step 2 is performed using the Patricia tree. Step 4 obtains the absolute difference between each value of the expected similarity (obtained in step 1) and the obtained similarity, for each running of the algorithms. Step 5 returns a list of average differences obtained in step 4.

4.3. Results

Experiments were performed to evaluate the differences between the expected similarity (obtained by the generator) and the similarity obtained by the Patricia and shingles algorithms. For the shingles algorithm, different values were evaluated for w, from 2 to 10. The value of $w = 1$ was not considered.

The arithmetic average of differences between the expected and obtained similarity was obtained for each method, between each candidate document D_c and the plagiarized document D_p, for each query. We considered 900 queries for each method,

Table 1 presents the results for the two methods. The numbers represent the total arithmetic average obtained for each method, including the variations for the shingle algorithm, with different values of w. A value close to zero represents the best possible result, that is, the smallest difference between the expected and the obtained similarity by each method.

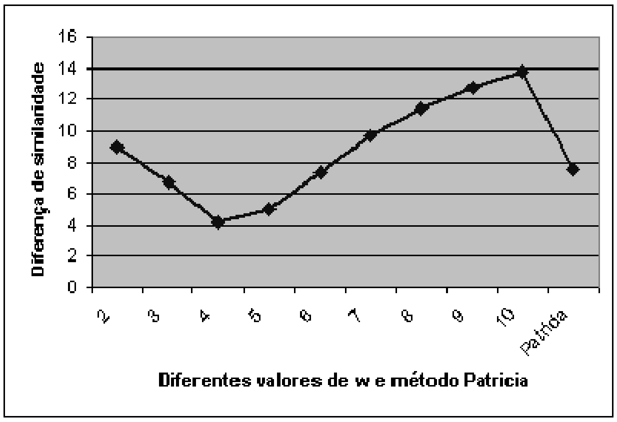

Figure 12. Graphic of the average of differences between expected and obtained results

The algorithm implementing shingles presents better results than the algorithm implementing the Patricia tree. The best value verified for w was 4, whose difference between expected and obtained results was 4.13%. The same measure for the Patricia tree was 7.50%.

5. Conclusions

This paper have presented and compared two effective methods to evaluate the syntactic similarity between documents. The methods are able to detect plagiarism in a given document, and documents collected from the Web.

The first method implemented the Patricia tree, which has time complexity O($n$log$n$) for the tree construction, where n is the size of the document, and constant complexity for searching similar passages.

The second method used shingles, which also has time complexity O($n$log$n$).

The best result obtained was with shingles algorithm, for the value of $w = 4$, whose average difference between the expected result and the obtained result was 4.13%. The same measure for the Patricia tree was 7.50%.

As a future work, we will study ways of determining the best fingerprinting to a query document. This fingerprinting might be used in the search in the Web for similarity candidate documents.

Acknowledgements

This work was supported in part by the SIAM project–grant MCT/FINEP/CNPq/PRONEX 76.97.1016.00, the GERINDO project–grant MCT/CNPq/CT-INFO 552.087/02-5, and by CNPq grant 520.916/94-8 (Nivio Ziviani).

References

[1] L. C. A. Albuquerque. Recuperação eficiente de informação em bancos de dados não estruturados. Master's thesis, DCC/UFMG, Belo Horizonte, Brasil, Março 1987.

[2] G. H. Gonnet; R. A. Baeza-Yates and T. Snider. *Information Retrieval: Data Structures and Algorithms*, chapter New Indices for Text: Pat Trees and Pat Arrays, pages 66–82. Prentice-Hall, 1992.

[3] R. Baeza-Yates, E. F. Barbosa, and N. Ziviani. Hierarchies of indices for text searching. *Information Systems*, 21(6):497–514, 1996.

[4] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society, 1998.

[5] D. M. Campbell; W. R. Chen and R. D. Smith. Copy detection systems for digital documents. 78-88. In *IEEE Advances in Digital Libraries (ADL 2000)*, pages 78–88, Washington, May 22-24 2000.

[6] S. Brin; J. Davis and H. Garcia-Molina. Copy detection mechanisms for digital documents. In *ACM SIGMOD Annual Conference*, pages 398–409, San Francisco, May 1995.

[7] R. A. Finkel, A. Zaslavsky, K. Monostori, and H. Schmidt. Signature extraction for overlap detection in documents. In Michael J. Oudshoorn, editor, *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002. ACS.

[8] W. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, North Virginia, 1992.

[9] N. Heintze. Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, November 1996.

[10] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searchs. In *1st ACM-SIAM Symposium Discrete Algorithms*, pages 319–327, San Francisco, 1990.

Methods	w = 2	w = 3	w = 4	w = 5	w = 6	w = 7	w = 8	w = 9	w = 10	Patricia
Differences	8.97%	6.72%	4.13%	5.04%	7.34%	9.67%	11.42%	12.71%	13.78%	7.50%

Table 1. Average of differences between expected and obtained results

[11] D. R. Morrison. Practical algorithm to retrieve information coded in alphanumeric. *ACM*, 15(4):514–534, Oct 1968.

[12] R. L. Ribler and M. Abrams. Using visualization to detect plagiarism in computer science classes. In *IEEE Symposium on Information Vizualization 2000*, pages 173–178, Salt Lake City, Utah, October 09 - 10 2000.

[13] N. Shivakumar and H. Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, Austin, Texas, June 1995.

[14] TodoBR. http://www.todobr.com.br, 2003.

[15] E. Ukkonen. On-line construction of suffix trees. In *Algorithmica*, pages 14:249–260, 1995.

[16] N. Ziviani. *Projeto de Algoritmos com Implementações em Pascal e C*. São Paulo, Brazil, 1993.