CoDet: Sentence-based Containment Detection in News Corpora

Emre Varol, Fazli Can, Cevdet Aykanat, Oguz Kaya
Computer Engineering Department, Bilkent University
Ankara, Turkey
{ evarol, canf, aykanat } @cs.bilkent.edu.tr, oguzkaya87@gmail.com

ABSTRACT
We study a generalized version of the near-duplicate detection problem which concerns whether a document is a subset of another document. In text-based applications, document containment can be observed in exact-duplicates, near-duplicates, or containments, where the first two are special cases of the third. We introduce a novel method, called CoDet, which focuses particularly on this problem, and compare its performance with four well-known near-duplicate detection methods (DSC, full fingerprinting, I-Match, and SimHash) that are adapted to containment detection. Our method is expandable to different domains, and especially suitable for streaming news. Experimental results show that CoDet effectively and efficiently produces remarkable results in detecting containments.

Categories and Subject Descriptors
H.3.7 [**Digital Libraries**]: Collection, System Issues.

General Terms: Algorithms, Experimentation, Performance, Reliability

Keywords: Corpus Tree, Document Containment, Duplicate Detection, Similarity, Test Collection Preparation.

1. INTRODUCTION
Near-duplicate detection is an important task in various web applications. Due to reasons such as mirroring, plagiarism, and versioning such documents are common in many web applications [17]. For example, Internet news sources generally disseminate slightly different versions of news stories coming from syndicated agencies by making small changes in the news articles. Identifying such documents increases the efficiency and effectiveness of search engines.

We consider a generalized version of the near-duplicate detection problem and investigate whether a document is a subset of another document [2]. In text-based applications, document containment can be observed in near-duplicates and containments. We refer to identifying such document pairs as the document containment detection problem. We study this problem within the context of news corpora that involve streaming news articles.

If a document d_C possesses all the information that document d_A has, then d_C is said to contain d_A, which is denoted as $d_C \supseteq d_A$, and this relation is called *containment*. Moreover, if two documents contain roughly the same content, they are *near-duplicates* [6]. Although near-duplicate condition is a special case of containment, these two cases are not usually distinguished from each other [15]. Similar to the "conditional equivalence" concept defined by Zobel and Bernstein [17], if $d_C \supseteq d_A$, then a news-consumer who have already read d_C would have no need to read d_A. Of course, $d_C \supseteq d_A$ does not necessarily imply $d_A \supseteq d_C$, i.e. containment relation is asymmetric. By detecting $d_C \supseteq d_A$, news consumers that have already seen d_C can be informed to skip reading d_A.

In related studies, containment problem is addressed by near-duplicate detection algorithms. Therefore, we compare performance of CoDet with well-known near-duplicate detection approaches.

Contributions of this study are the following. We introduce a sentence-based containment detection method adaptable to different text-based problem domains, and especially suitable for streaming news; show that our approach outperforms commonly known near-duplicate detection methods; and construct a test collection using a novel pooling technique, which enables us to make reliable judgments for the relative effectiveness of algorithms using limited human assessments.

2. RELATED WORK
In near-duplicate detection similarity measurement plays an important role [11]. By using similarity, two documents are defined as duplicates if their similarity or resemblance [3] exceeds a certain threshold value. Such approaches are applied to identify roughly the same documents, which have the same content except for slight modifications [1]. In comparisons, factors other than similarity may also play a role. Conrad and Schriber [7] after consulting librarians deem that two documents are duplicates if they have 80% overlap and 20 variations in length.

Similarity measures may use all words in documents in calculation. Instead of using each word, a sequence of them, shingles, may be used. In shingling approaches, if two documents have significant number of shingles in common, then they are considered as similar (near-duplicate). Well-known shingling techniques include for example COPS [1] and DSC (Digital Syntactic Clustering) [3]. COPS uses the sentences (or small units) to generate hash codes and stores these in a table to see if a document contains a sentence. Wang and Chang propose using the sequence of sentence lengths for near-duplicate detection and they evaluated different configurations of sentence-level and word-level algorithms [14].

Shingling and similarity approaches suffer from efficiency issues. As a result a new strategy emerged which is based on hashing of the whole document. I-Match [6] is a commonly known approach that uses this strategy. It filters terms based on collection statistics (*idf* values). Charikar's [5] Simhash method is based on the idea of creating a hash by using document features (words, bigram, trigrams, etc.). It compares bit differences of these signatures to decide if two documents are near-duplicate or not. Yang and Callan [15] use clustering concepts for efficiency. While clustering documents they use additional information

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10...$10.00.

extracted from documents and structural relationships among document pairs.

Hajishirzi et al. [9] propose an adaptable method for near duplicate detection by representing documents as real valued sparse k-gram vectors, where weights are learnt to optimize a similarity function. Zhang et al. [18] address the partial-duplicate detection problem by doing sentence level near-duplicate detection and sequence matching. Their algorithm generates a signature for each sentence and sentences that have the same signature are considered as near-duplicates. Theobald et al. [13] propose SpotSigs algorithm that combines stopword antecedents with short chains of adjacent content terms to create signatures.

3. CODET ALGORITHM

3.1 Containment Similarity Concept

CoDet is a novel sentence-based containment detection algorithm. It employs a new similarity measure called containment similarity (CS). It measures to what extent a document d_A is contained by another document d_C, which is defined as

$$CS(d_C, d_A) = \sum_{s_i \in S_C} \sum_{s_j \in S_A} cs(s_i, s_j) \quad (1)$$

where S_A and S_C denote the set of sentences in d_A and d_C, respectively. The function $cs(s_i, s_j)$ indicates containment similarity between sentences s_i and s_j, which is calculated as

$$cs(s_i, s_j) = \sum_{k=1}^{len_{f(s_i, s_j)}} k \times idf\left(w_{f(s_i,s_j),k}\right) \quad (2)$$

where $f(s_i, s_j)$ denotes the word sequence representing the longest word prefix match of the sentences s_i and s_j, len_t is the length of the word sequence t, and $w_{t,k}$ stands for the k^{th} word in the word sequence t. For example, let s_1 be "John is happy." and s_2 be "John is sad." then $f(s_1, s_2)$ is a word sequence (John, is), $len_{f(s_1,s_2)}$ is 2 and $w_{f(s1, s2),1}$ is *John*.

As can be seen in Formula 2, containment similarity between two sentences grows significantly as their word prefix match gets longer. The containment similarity of a document to itself is referred to as self-containment similarity (SCS).

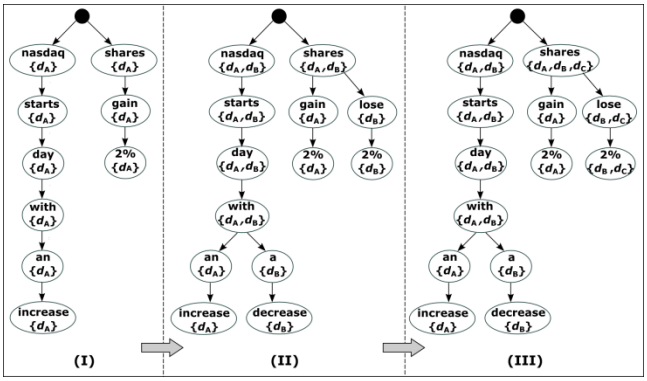

Fig. 1. Insertion of three documents d_A: "NASDAQ starts day with an increase. Shares gain 2%.", d_B: "NASDAQ starts the day with a decrease. Shares lose 2%.", and d_C: "Shares lose 2%.". For the sake of clarity, words of sentences are not sorted according to their *idf* values.

3.2 Containment Similarity Calculation

For efficient calculation of containment similarities, we utilize a data structure called *corpus tree*. The corpus tree begins with a virtual root node which contains a pointer list storing the locations of the children nodes in the next level. In addition to pointer list, nodes other than the root contain a label and a document list. The label represents the node's term and the document list contains visiting document ids.

Let d_A denote a document with a set of sentences $S_A = \{s_1, s_2 \dots s_n\}$. Processing of d_A involves processing all of its sentences. Insertion of s_i ($1 \le i \le n$) to the corpus tree is performed as follows: First, words of s_i are sorted according to their *idf* values in descending order. Let $<w_1, w_2 \dots w_m>$ denote the sequence of words in s_i after sorting. These words are inserted into the corpus tree starting from the virtual root node. If the root has a child ch_{w_1} with label w_1, then similarity values of d_A with all documents in ch_{w_1}'s document list are increased according to Formula 2. Otherwise, a new child node ch_{w_1} with label w_1 is created and added to the root's pointer list. In the next step, we treat ch_{w_1} as we did the root, and insert the following word w_2 of s_i similarly. The insertion of s_i finishes after all of its words are processed. The remaining sentences of d_A are handled in the same manner. The same is done for the remaining sentences of d_A.

Fig. 1. shows how the corpus tree grows with sentence insertions. In Fig. 1-I, d_A's sentences "NASDAQ starts day with an increase." and "Shares gain 2%." are inserted to the corpus tree starting from the virtual root, which is shown by a dark circle. Since the tree is initially empty, while inserting the first sentence all the nodes with labels *<nasdaq*, *starts*, *day*, *with*, *an*, *increase>* are created. Similarly, insertion of the second sentence creates nodes with labels *<shares*, *gain*, *2%>*. In Fig. 1-II, during the insertion of the sentence "NASDAQ starts day with a decrease." previously created nodes with labels *<nasdaq*, *starts*, *day*, *with>* are visited and updated. Also, two nodes with labels *<a, decrease>* are created. Insertion of the sentence "Shares lose 2%." visits the node with label *shares* and creates two nodes with labels *<lose, 2%>*. Thus, similarity value of d_A and d_B is increased by summation of each revisited node's impact values, which is calculated by multiplication of node's depth and *idf* value of its label. For example, contribution of the node with label starts is $(2 \times (\log(3/2) + 1))$ because its depth is 2 and word starts appears in 2 of 3 documents (in the experiments, the *idf* values are obtained from a large reference collection). The final structure of the corpus tree after the insertion of d_C is shown in Fig. 1-III.

To decide whether a document d_A is contained by another document d_C, CoDet uses $CS(d_A, d_C)$ as well as $SCS(d_A)$ values. If $(CS(d_A, d_C) / SCS(d_A))$ exceeds the equivalency threshold level (ETL), d_C is said to contain d_A. In the experiments, different ETL values are tested.

3.3 Complexity Analysis

For each scenario, let n denote the number of documents and let c denote the average number of words per document, which is treated as constant.

First Scenario (One Content, n Documents): In this case, each document has the same content; therefore, *corpus tree* contains c nodes. Each node contains n integers in its document list. As a result, the memory requirement of the corpus tree is O(n) but due to pairwise containment similarity increase operations the algorithm takes $O(n^2)$ time.

Second Scenario (n Different Contents, n Documents): In this case, each document has totally different content. Thus, corpus tree contains nc nodes (one node for each word). Each node contains only one document id in its document list. Therefore, asymptotically the memory requirement of the corpus tree is O(n) and the algorithm takes O(n) time.

The first scenario is the worst case for CoDet, where the algorithm performs nonlinearly. The second one is the best case for CoDet and the algorithm runs in linear time. In practice the algorithm behaves as if it is linear because average number of near-duplicate per

document is significantly smaller than n. Also CoDet is especially suitable for streaming news since with a time window concept, which makes older documents to be removed from the corpus tree, the corpus tree does not grow too much.

4. EXPERIMENTAL SETUP

We used four algorithms to compare their effectiveness and efficiency with CoDet. These algorithms are:

DSC: Every *three* overlapping substrings of size *four* in the documents are hashed. If a document d_C contains *60%* of d_A's hash values, we say $d_C \supseteq d_A$ [3].

Full Fingerprinting (FFP): For each document, all substrings of size *four* are hashed. If document d_C contains *60%* of d_A's hash values, then, $d_C \supseteq d_A$.

I-Match: First *two* words with the highest *idf* values are ignored. After that, *ten* words with the highest *idf* values are used to create a fingerprint for each document. When a pair of documents has the same fingerprint, the pair is marked as containment [6].

SimHash: First *two* words with the highest *idf* values are ignored. Then, each unique term of a document is hashed. We use a vector v, whose size is equal to the hash value bit size, to determine the final SimHash [5] value. For each term t, i^{th} element of the vector v is updated as follows: If i^{th} bit of the hash value of t is *zero*, then it is decreased by *idf* of w. It is increased by the *idf* otherwise. Finally, if i^{th} element of v is positive, i^{th} bit of the SimHash value is set to *one*; otherwise it is set to *zero*. When a pair of documents' SimHash values has a Hamming distance less than *three*, the pair is considered as containment.

For hashing, SHA1 [6] algorithm is used in all methods. Stopword elimination and a word truncation-based stemming (first-5) are performed before the detection process. I-Match, SimHash and CoDet requires *idf* values. These values are obtained from a large reference collection (defined in the next section). In order to do a fair evaluation, each algorithm's parameters are optimized to give the best results for efficiency.

We performed the experimentation on a machine with quad 2.1Ghz six-core AMD Opteron processors with six 128 KB L1, 512 KB L2, and one 6MB L3 cache. It has 128 GB memory and operating system Debian Linux v5.0.5.

4.1 Test Collection Preparation

There is no gold-standard test collection for containment detection in news corpora; therefore, we prepared a test dataset from the Turkish TDT (Topic Detection and Tracking) news collection *(BilCol-2005)* [4] which contains 209,305 streaming (time-ordered) news articles obtained from five different Turkish web news sources.

For efficiency measurement, we used all documents of *BilCol-2005*. For effectiveness measurement, we used the first 5,000 documents of *BilCol-2005*. It is practically impossible to provide human assessment for each document pair in this sub-collection. Our approach to human assessments is similar to the pooling method used in TREC for the evaluation of IR systems [16]. For the creation of the dataset, we obtained a number of possible containments by running all five methods (including CoDet) with permissive parameters. In this way, methods nominate all pairs that would normally be chosen with their selective parameters, together with several additional pairs as containment candidates. Since the methods are executed with permissive parameters, we expect that most of the real containments will be added to the test collection. All pairs of documents, which are marked as containments by any of the methods, are brought to the attention of human assessors to determine whether they actually are containments. Note that in order to measure the effectiveness of a new algorithm with this test dataset, adding human assessments only for containment candidates that are nominated solely by this new algorithm to our dataset is sufficient.

By this approach, our dataset includes only true positive (TP) and false positive (FP) document pairs returned by any of our permissive algorithms. excluding true negative and false negative pairs do not change the relative effectiveness rankings of selective algorithms during the test phase; because, if a permissive algorithm marks a pair as negative (non-containment), then its selective counterpart should also marks that pair as negative. Therefore, including TN and FN pairs of permissive algorithms in our dataset would not contribute to the number of positive pairs (TP's and FP's) returned by any selective algorithm during the test phase. Hence, using our pruned dataset, precision[1] values of the selective algorithms remain unchanged with respect to precision values they would obtain in a full dataset having annotations for all possible document pairs. Similarly, recall values of the selective algorithms decrease proportionally (with the same ratio of total number of containments in the pruned dataset to the total number of containments in the full dataset, for all algorithms) with respect to recall values they would obtain in the full dataset.

Our pooling process generated 4,727 document pairs nominations. We performed a human-based annotation to obtain a ground truth. The pooled document pairs are divided into 20 groups containing about the same number of nominations. Each document pair is annotated by two assessors. The assessors are asked if the nominated document pairs are actually containments. The assessors identified 2,875 containment cases. The size of our dataset is comparable with the annotated test collections reported in related studies [13].

In information retrieval, human assessors may have different opinions about the relevance of a document to a query. A similar situation arises in our assessments. For example, for the document pair d_C = "XYZ shares increase 10% from 100 to 110." and d_A = "XYZ shares increase from 100 to 110.", some assessors may say that d_C and d_A are near-duplicates, while some others may claim d_C contains d_A, but the d_A does not contain d_C. In such cases we expect disagreements among human assessors. In order to validate the reliability of the assessments, we measured the agreements of the judgments by using the Cohen's Kappa measure, and obtained an average agreement rate of 0.73. This indicates almost a substantial agreement [10], which is an important evidence for the reliability of our test dataset. Furthermore, such conflicts are resolved by an additional assessor.

5. EXPERIMENTAL RESULTS

In this section, we first investigate the impacts of the following parameters on the performance of CoDet: Processed Suffix Count (PSC), Depth Threshold (DT), and Word Sorting (WS). This discussion is followed by efficiency and effectiveness performance of CoDet with those of four well-known near-duplicate detection algorithms. Effectiveness measurement is done by precision, recall and F_1 values. Impacts of parameters and effectiveness experiments are done on prepared test collection. Efficiency experiment is performed with the whole *BilCol-2005*.

5.1 Impacts of Parameters

Processed Suffix Count (PSC): It determines how many suffixes of each sentence are inserted to the corpus tree. If the PSC is 3, the processed suffixes for "NASDAQ starts day with an increase." are the sentence itself, <starts, day, with, an, increase> and <day, with, an, increase> Increasing PSC increases space requirement

[1] Precision (P) = |TP| / (|TP| + |FP|). where |S| is the cardinality of the set S. Recall (R) = |TP| / (|TP| + |FN|). F_1 = 2PR / (P + R).

but do not change effectiveness considerably as shown in Fig. 2: Different PSC values result in close F_1 scores.

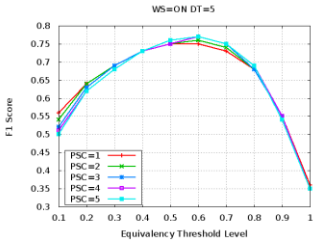

Fig. 2. Effect of Processed Suffix Count (PSC).

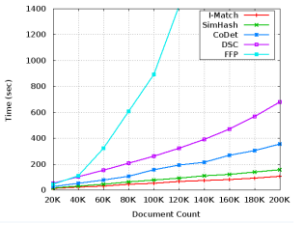

Fig. 3. Efficiency Comparison: Execution Time vs. Document Count.

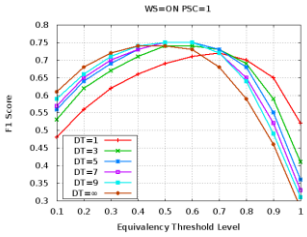

Fig. 4. Effect of Depth Threshold (DT): Word Sorting (WS) is on.

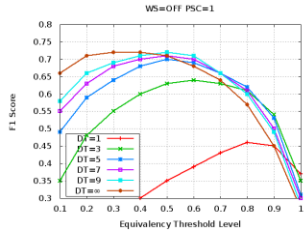

Fig. 5. Effect of Depth Threshold (DT): Word Sorting (WS) is off.

Depth Threshold (DT): It determines how many words of a sentence are processed. If the DT is 3, the processed words "NASDAQ starts day with an increase." are *<nasdaq, starts, day>*. Fig. 4 and 5 show the effect of DT on F_1 score. Sorting words of a sentence by *idf* values places representative words close to the virtual root. Thus, results are better for small DT values when word sorting is enabled. It avoids the noise effect of insignificant words in similarity calculations. In the experiments, DT value of 5 gives the best result; also smaller DT values yield a similar performance. Thus, instead of having the corpus tree structure, an algorithm that considers only a few most significant words from each sentence can improve efficiency without sacrificing effectiveness significantly.

Word Sorting (WS): Sorting words in sentences by *idf* values causes important words to be located close to the virtual root. Since most sentences start with common words, by using word sorting, we avoid many redundant similarity calculations. In the experiments, enabling word sorting decreases average number of calculated similarity values per document from 341 to 3.53.

5.2 Comparing with Other Algorithms

The efficiency results are given in Fig. 3. As the number of documents increase, execution time of full fingerprinting increases non-linearly. It calculates similarity values for each document pair that has at least one substring in common. Hence, it is not feasible for large collections. CoDet performs as the third best algorithm in time efficiency; the corpus tree accesses impose many random memory accesses, which disturb cache coherency. Our results show that I-Match, SimHash and CoDet are scalable to large collections.

Table 1 shows the effectiveness results. The best performance with a value of 0.85 F_1 score is observed with FFP since it calculates text overlaps between document pairs having a common substring. Therefore, without making any semantic analysis, it is difficult to outperform FFP in terms of effectiveness with a time-linear algorithm. CoDet finds text overlaps by only using important words of sentences and is the second best in terms of effectiveness with an F_1 score of 0.76. I-Match, SimHash, and DSC perform poorly with respective F_1 scores of 0.45, 0.39, and 0.30. FFP is not feasible for large collections; thus, CoDet is the most suitable algorithm for containment detection in news corpora.

Table 1. Effectiveness Comparison

Algorithm	Precision	Recall	F_1 Measure
FFP	0.82	0.88	0.85
CoDet	0.75	0.76	0.76
I-Match	0.72	0.33	0.45
SimHash	0.53	0.30	0.39
DSC	0.22	0.45	0.30

6. CONCLUSIONS

In this work we investigate containment detection problem, which is a more generalized version of the near-duplicate detection problem. We introduce a new approach, and compare its performance with four other well-known methods. As the experimental results demonstrate CoDet is preferable to all these methods; since it produces considerably better results in a feasible time. It also has desirable features such as time-linear efficiency and scalability, which enriches its practical value. Our method is versatile, can be improved, and can be extended to different problem domains.

7. REFERENCES

[1] Brin, S., Davis, J., Garcia-Molina, H.: Copy detection mechanisms for digital documents. ACM SIGMOD Conf. (1995) 398-409.

[2] Broder, A.: On the resemblance and containment of documents. Proc. of Compression and Complexity of Sequences (1997) p.21.

[3] Broder, A. Z., Glassman, S. C., Manasse, M. S., Zweig, G.: Syntactic clustering of the web. Computer Networks and ISDN Systems, 29(8-13) (1997) 1157-1166.

[4] Can, F., Kocberber, S., Baglioglu, O., Kardas, S., Ocalan, H. C., Uyar, E.: New event detection and topic tracking in Turkish. *JASIST* 61(4) (2010) 802-819.

[5] Charikar, M.: Similarity estimation techniques from rounding algorithms. ACM STOC (2002) 380-388.

[6] Chowdury, A., Frieder, O., Grossman, D., McCabe, M. C.: Collection statistics for fast duplicate document detection. ACM TOIS, 20(2) (2002) 171-191.

[7] Conrad, J. G., Schriber, C. P.: Managing déjà vu: Collection building for the identification of duplicate documents. *JASIST*, 57(7) (2006) 921-923.

[8] Deng, F., Rafiei, D.: Approximately detecting duplicates for streaming data using stable Bloom filters. ACM SIGMOD Conf. (2006) 25-36.

[9] Hajishirzi H., Yih W., Kolcz A.: Adaptive near-duplicate detection via similarity learning. ACM SIGIR Conf. (2010): 419-426.

[10] Landis, J. R., Koch, G. G.: The measurement of observer agreement for categorical data. *Biometrics*, 33(1) (1977) 159–174.

[11] Manku, G. S., Jain, A., Sarma, A. D.: Detecting near-duplicates for web crawling. ACM WWW Conf. (2007) 141-150.

[12] Monostori, K., Zaslavsky, A. B., Schmidt, H. W.: Efficiency of data structures for detecting overlaps in digital documents. ACSC (2001) 140-147.

[13] Theobald, M., Siddharth, J., Paepcke, A.: SpotSigs: robust and efficient near duplicate detection in large web collections. ACM SIGIR '08 Conf. 563-570.

[14] Wang, J. H., Chang, H. C.: Exploiting sentence-level features for near-duplicate document detection. AIRS Conf. (2009) 205–217

[15] Yang, H., Callan, J. P.: Near-duplicate detection by instance-level constrained clustering. ACM SIGIR Conf. (2006) 421-428.

[16] Zobel, J.: How reliable are the results of large-scale information retrieval experiments? ACM SIGIR Conf. (1998) 307-314.

[17] Zobel, J., Bernstein, Y.:. The case of the duplicate documents measurement, search, and science. *LNCS*, Vol. 3841. (2006) 26-39.

[18] Zhang, Q., Zhang, Y., Yu, H., Huang, X.: Efficient partial-duplicate detection based on sequence matching. ACM SIGIR Conf. (2010) 675-682.