Acta Univ. Sapientiae, Informatica, **1**, 2 (2009) 215–233

Towards a framework for near-duplicate detection in document collections based on closed sets of attributes

Dmitry I. Ignatov
Higher School of Economics, Department of Applied Mathematics and Information Science
Moscow, Russia
email: dignatov@hse.ru

Katalin Tünde Jánosi-Rancz
Sapientia Hungarian University of Transylvania, Cluj, Department of Mathematics and Informatics
Tg. Mureş, Romania
email: tsuto@ms.sapientia.ro

Sergei O. Kuznetzov
Higher School of Economics, Department of Applied Mathematics and Information Science
Moscow, Russia
email: skuznetsov@hse.ru

Abstract. Around 30% of documents on the web have duplicates. Near-duplicate documents bear high similarity to each other, yet they are not bitwise identical. They are identical in terms of content but differ in a small portion of the document. Thus, algorithms for detecting these pages are needed. In the course of developing a near-duplicate detection system in this article we present an approach based on frequent closed sets of attributes for constructing clusters of duplicate documents, documents being represented by both syntactic and lexical methods. We provide a prototype of software environment for those who want to utilize such methods for finding near-duplicate documents in large text collections. This software includes two syntactic methods of finding near duplicate documents, a clustering technique based on frequent closed itemsets, means of evaluation of results and a tool for generating test collections of near-duplicate documents.

AMS 2000 subject classifications: 69P99, 62H30
CR Categories and Descriptors: H.3.3 [Information Search and Retrieval]: Search Process, Clustering
Key words and phrases: near-duplicate detection, content duplication, web document

1 Introduction

The Web makes it easy for words to be copied and spread from one page to another, and the same content may be found at more than one web site, regardless of whether its author intended it to be or not. Duplicate and near-duplicate web pages are creating large problems for web search engines: they increase the space needed to store the index, either slow down or increase the cost of serving results, and annoy the users. This requires the creation of efficient algorithms for computing clusters of duplicates [4, 6, 7, 10, 11, 16, 21, 22, 30, 29].

A naive solution is to compare all pairs to documents. The first algorithms for detecting near-duplicate documents with a reduced number of comparisons were proposed by Manber [25] and Heintze [17]. Both algorithms work on sequences of adjacent characters. Brin [3] started to use word sequences to detect copyright violations. Shivakumar and Garcia-Molina [31] continued this research and focused on scaling it up to multi-gigabyte databases. Broder [5] also used word sequences to efficiently find near-duplicate web pages. Charikar [9] developed an approach based on random projections of the words in a document. Hoad and Zobel [19] developed and compared methods for identifying versioned and plagiarised documents. Henzinger [18] tests and explores how some different existing methods (Broder's [5] and Charikar's [9]) for detecting near-duplicate content could be used together to try to identify near-duplicates on the Web. A good overview of approaches to detect exact duplicates and near-duplicates of web pages can be found in [15].

We define duplicates in terms of similarity. We say that two documents are duplicates, if a numerical measure of their similarity exceeds a given threshold [7]. This can be represented by a graph, where nodes correspond to documents and the edges of the graph represent the pairs of the similarity relation. From this similarity graph we can compute the clusters of similar documents by counting the number of connected components of the graph. The main steps in finding clusters of duplicates are: representing documents by sets of attributes, making solid document images and computing clusters of similar documents. First of all, we have to remove the HTML markup and punctuation marks of the web documents. After this, as the first step, we turn these documents into strings of words, which are represented by sets of attributes. We have two options of doing this: from a syntactical approach or from a lexical approach.

In the syntactical approach we define binary attributes that correspond to each fixed length substring of words (or characters). These substrings are

called shingles. We can say that a shingle is a sequence of words. A shingle has two parameters: the length and the offset. The length of the shingle is the number of the words in a shingle and the offset is the distance between the beginnings of the shingles. We assign a hash code to each shingle, so equal shingles have the same hash code and it is improbable that different shingles would have the same hash codes (this depends on the hashing algorithm we use). After this we randomly choose a subset of shingles for a concise image of the document [4, 6, 7]. An approach like this is used in AltaVista search engine [29]. There are several methods for selecting the shingles for the image: a fixed number of shingles, a logarithmic number of shingles, a linear number of shingle (every n^{th} shingle), etc. In lexical methods, representative words are chosen according to their significance. Usually these values are based on frequencies: those words whose frequencies are in an interval (except for stop-words from a special list of about 30 stop-words with articles, prepositions and pronouns) are taken: words with high frequency can be non informative and words with low frequencies can be misprints or occasional words.

In lexical methods, like I-Match [11], a large text corpus is used for generating the lexicon. The words that appear in the lexicon represent the document. When the lexicon is generated the words with the lowest and highest frequencies are deleted. I-Match generates a signature and a hash code of the document. If two documents get the same hash code it is likely that the similarity measures of these documents are equal as well. I-Match is sometimes instable to changes in texts [22]. In lexical method [21] the focus is towards the construction of a lexicon, a set of descriptive words, which should be concise, but cover well the collection. The occurrence of a word in a document image is robust with respect to small changes in the document. When we define document images, we define a similarity relation on documents starting from a similarity measure, which takes to two documents to a number into the [0,1] interval, depending on the amount of their common description units. Then we choose a threshold. If this threshold is exceeded, it means that there is a large similarity between the documents (the two documents are very close to being duplicates). The metrics and the threshold define similarity relation on document pairs. The similarity relation on document pairs determines clusters of near-duplicates. There are several possible definitions for a cluster, but one of them often used in practice is as follows: Consider a graph, in which nodes represent the Internet documents and edges correspond to similarity relations. Then a cluster of near-duplicates is a connected component of this graph. The advantage of this definition is in the efficiency of computation: a connected component of a graph can be computed in linear time in the number of edges.

A drawback of the definition is also obvious: the relation to be near-duplicates is not transitive, so absolutely different documents can occur in a cluster. The strongest definition of a cluster is based on a graph clique, but it is much harder computationally, because generation of maximal cliques is a classical problem. We can use an intermediate formulation, which is between these two extreme definitions, and this way make a trade-off between the precision and the complexity of the cluster computation.

In this paper we consider similarity as an operation taking two documents to the set of all common elements of their concise description. Description elements can be syntactical units (shingles) or lexical units (representative words). A cluster of similar documents is defined as a set of all documents with a certain set of common description units. A cluster of duplicates is defined as a set of documents, where the number of common description units exceeds a given threshold. In this article we compare results of its application with the list of duplicates obtained by applying other methods to the same collection of documents. We examined the impact of the following parameters on the result:

- The use of the syntactical or lexical methods for representing documents
- the use of method "$\mathfrak{n}$ *minimal elements in a permutation*" or "*minimal elements in* $\mathfrak{n}$ *permutations*" [4, 6, 7] (the second method, having better probability-theoretical properties, has worse computational complexity)
- shingling parameter
- threshold value of similarity of document images.

We used a definition based on formal concepts for a cluster: clusters of documents are given by formal concepts of the context where objects correspond to description units and attributes are document names. So a cluster of very similar documents corresponds to a formal concept so that the size of the extent exceeds the threshold given by a parameter. In this approach, the problem of generating very similar documents is reduced to the problem of data mining, known as generating frequent closed item sets.

There are many web services, such as web search engines, which use near-duplicate detection techniques. These techniques are also useful for plagiarism detection in R&D reports and scientific articles [20]. To the best of our knowledge, there is no freely available framework with implementation of basic methods of near-duplicate detection. We made a first attempt to develop such a system with taking into account researcher's needs. Potthast and Stein [28]

note that there are no public collections suited for the analysis and evaluation of near-duplicate detection algorithms. Then they propose to use the Wikipedia Revision Corpus for the task. Our solution of this problem is a tool for generating near-duplicate collections based on one's own corpus of texts. We give a detailed description of our system in section 3.

2 Computational model

2.1 Document image

We used standard syntactical and lexical approaches with different parameters, for creating document images. Within syntactical approach we realized the shingling scheme and computing document image (sketch) with the method "$\mathfrak{n}$ *minimal elements in a permutation*" and the method "*minimal elements in* $\mathfrak{n}$ *permutations*", a detailed description of which can be found in [4, 6, 7]. For each text the program **shingle** with two parameters (*length* and *offset*) generates contiguous subsequences of size *length* so that the distance between the beginnings of two subsequent substrings is *offset*. The set of sequences obtained in this way is hashed so that each sequence receives its own hash code. From the set of hash codes that corresponds to the document a fixed size (given by parameter) subset is chosen by means of random permutations described in [4, 6, 7]. The probability of the fact that minimal elements in permutations on hash code sets of shingles of documents A and B (these sets are denoted by F_A and F_B, respectively) coincide, equals to the similarity measure of these documents $\mathrm{sim}(A, B)$:

$$\mathrm{sim}(A, B) = P[\mathfrak{min}\{\pi(F_A)\} = \min\{\pi(F_B)\}] = \frac{|F_A \cap F_B|}{|F_A \cup F_B|}$$

Permutations (that can be represented by renumbering of shingles) are realized by multiplying binary vectors that represent document images (each component of such a vector corresponds to the hash code of a particular shingle from the image) on random binary matrices. For each hash code from the set of hash codes of a document its number in each random permutation is computed as a product of the hash code given in the form of binary vector on the randomly generated binary matrix that corresponds to the permutation. The number of permutations is also a parameter. For each permutation (given by a binary matrix) the minimal element (i.e., hash code of a shingle that became the first after the permutation) is chosen. The image of a document in the method "*n minimal elements in a permutation*" is the set of n minimal

(first) hash codes in a permutation. The image of a document in the method "*minimal elements in n permutations*" is the set consisting of minimal (first) hash codes in n independent permutations. In both methods the images of all documents have fixed length n. The second approach has better randomization properties (see [4, 6, 7] for details), although it needs more time for computations (n times more than in the first approach).

2.2 Definition of similarity and similarity clusters by means of frequent concepts

First, we briefly recall the main definitions of Formal Concept Analysis (FCA) [12]. Let G and M be sets, called the set of objects and the set of attributes, respectively. Let I be a relation $I \subseteq G \times M$ between objects and attributes: for $g \in G$, $m \in M$, gIm holds iff the object g has the attribute m. The triple $K = (G, M, I)$ is called a *(formal) context*. Formal contexts are naturally given by cross tables, where a cross for a pair (g, m) means that this pair belongs to the relation I. If $A \subseteq G$, $B \subseteq M$ are arbitrary subsets, then *derivation operators* are given as follows:

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\},$$
$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}.$$

The pair (A, B), where $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$ is called a *(formal) concept (of the context* K*)* with *extent* A and *intent* B.

The operation $(\cdot)''$ is a closure operator, i.e., it is idempotent ($X''' = X''$), extensive ($X \subseteq X''$), and monotone ($X \subseteq Y \Rightarrow X'' \subseteq Y''$). Sets $A \subseteq G$, $B \subseteq M$ are called *closed* if $A'' = A$ and $B'' = B$. Obviously, extents and intents are closed sets. Formal concepts of context are ordered as follows: $(A_1, B_1) \leq (A_2, B_2)$ iff $A_1 \subseteq A_2 (\Leftrightarrow B_1 \supseteq B_2)$. With respect to this order the set of all formal concepts of the context K makes a lattice, called a *concept lattice* $\underline{\mathfrak{B}}(K)$ [12].

Now we recall some definitions related to association rules in data mining. For $B \subseteq M$ the value $|B'| = |\{g \in G \mid \forall m \in B(gIm)\}|$ is called *support* of B and denoted by $\sup(B)$. It is easily seen that set B is closed if and only if for any $D \supset B$ one has $\sup(D) < \sup(B)$. This property is used for the definition of a closed itemset in data mining. A set $B \in M$ is called k-*frequent* if $|B'| \leq k$ (i.e., the set of attributes B occurs in more than k objects), where k is parameter. Computing frequent closed sets of attributes (or itemsets) became important in data mining since these sets give the set of all association rules [27]. For our implementation where contexts are given by set G of description units (e.g.,

shingles), set M of documents and incidence (occurrence) relation I on them, we define a cluster of k-similar documents as intent B of a concept (A, B) where $|A| \geq k$. Although the set of all closed sets of attributes (intents) may be exponential with respect to the number of attributes, in practice contexts are *sparse* (i.e., the average number of attributes per object is fairly small). For such cases there are efficient algorithms for constructing all most frequent closed sets of attributes (see also survey [23] on algorithms for constructing all concepts). Recently, competitions in time efficiency for such algorithms were organized in a series of workshops on Frequent Itemset Mining Implementations (FIMI). By now, a leader in time efficiency is the algorithm FPmax* [14]. We used this algorithm in order to find similarities of documents and generate clusters of *very similar documents*. As mentioned before, objects are description units (shingles or words) and attributes are documents. For representations of this type *frequent closed itemsets* are closed sets of documents, for which the number of common description units in document images exceeds a given threshold. Actually, FPmax* generates *frequent itemsets* (which are not necessarily closed) and *maximal frequent itemsets*, i.e., frequent itemsets that are maximal by set inclusion. Obviously, maximal frequent sets of attributes are closed.

3 Program implementation

Software for experiments with syntactical representation comprise the units that perform the following operations:

1. XML Parser (provided by Yandex): it parses XML packed collections of web documents,
2. removing html-markup of the documents,
3. generating shingles with given parameters length-of-shingle, offset,
4. hashing shingles,
5. composition of document image by selecting subsets (of hash codes) of shingles by means of n *minimal elements in a permutation* and *minimal elements in* n *permutations* methods,
6. composition of the inverted table, the list of identifiers of documents shingle, thus preparing data to the format of programs for computing closed itemsets,

7. computation of clusters of k-*similar documents* with FPmax* algorithm: the output consists of strings, where the first elements are names (ids) of documents and the last element is the number of common shingles for these documents,

8. comparing results with the existing list of duplicates (in our experiments with the ROMIP collection of web documents, we were supplied by a precomputed list of duplicate pairs),

9. generation of test collections of near-duplicate documents.

Unit 8 (for evaluation of results) outputs five values: 1) the number of duplicate pairs in the ROMIP collection, 2) the number of duplicate pairs for our realization, 3) the number of unique duplicate pairs in the ROMIP collection, 4) the number of unique duplicate pairs in our results, 5) the number of common pairs for the ROMIP collection and our results. For the lexical method, the description units are words (not occurring in the stop list) the frequencies of which lie in a certain interval. The amount of words in the dictionary is controlled by placing closer the extreme points of the interval.

3.1 GUI

The application has a Graphical User Interface similar to a setup application.

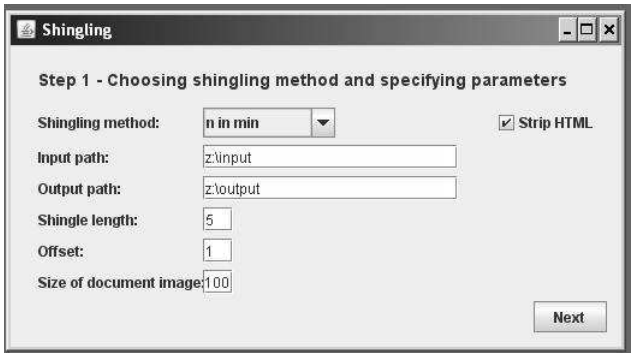

Figure 1: Input form for a duplicate search

In Fig. 1 the user specifies the shingling method and the following parameters: input path, output path, shingle length, offset, size of document image

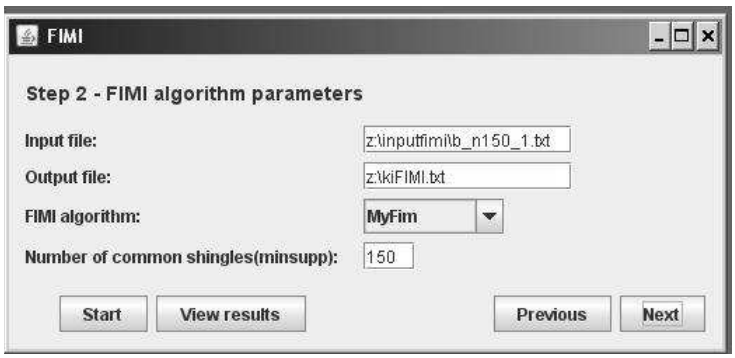

Figure 2: Input form for FIMI algorithm.

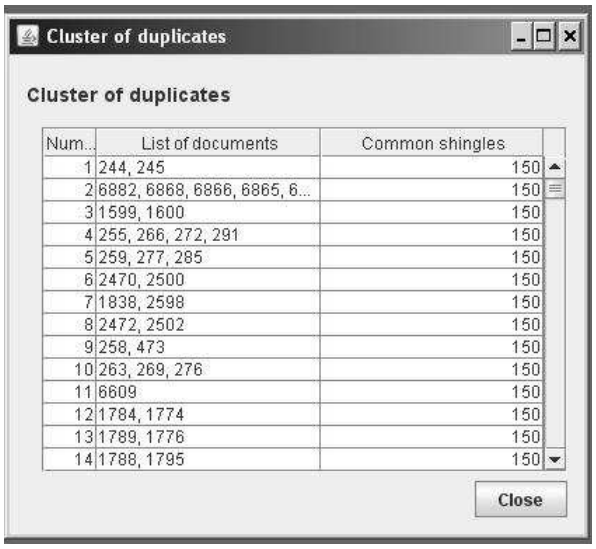

Figure 3: The output of the duplicates

and an option to strip the HTML or not. With the 'Next' button we can advance to the next form (Fig. 2).

On the next form the user has to choose the FIMI algorithm to be used and specify the parameters of the chosen algorithm: the input and output path of the algorithm and the number of common shingles. The 'Start' button starts the algorithm, the 'View results' button shows the results, see Fig. 3. With

the 'Previous' and 'Next' buttons we can either return to the previous form or advance to the next form. On the 'Clusters of duplicates' form we can see the results of the FIMI algorithm, shown in a grid view. On the last form the user can compare the results. The user has to specify the path of the list of duplicate pairs and the path of the log file. With the 'Start' button the user can start the comparison. With the 'Previous' button the user can return to the previous form. With the 'Close' button the application will exit. The screenshots of the GUI can be seen in the Fig. 1, 2, 3.

3.2 Tool for construction of near-duplicate test collection

The technique of generating test collection for near-duplicates proposed below uses the editing styles of near-duplicates: Block Edit (add or delete several paragraphs), Key Block (contains one or more well-known paragraphs), Minor Change (small editing changes), Block Reordering (reorder known paragraphs) and their combinations. Short description of these methods with indication of parameters is given in Table 1.

	Operation	Parameter(s)
1	Reordering of existing paragraphs	Percentage of reordering paragraphs
2	Deletion of existing paragraphs	Percentage of deleting paragraphs
3	Addition of existing paragraphs	Percentage of deleting paragraphs
4	Replacement of existing words	Percentage of replacing words
5	Addition of repeated paragraphs	Amount of paragraphs and number of paragraph repeats
6	replacement of characters	Set of character pairs: (initial character, new character)

Table 1: Operations for generation of test near-duplicate collection

We note that any of these operations use random number generator to construct the set of editable elements. For example, any reordering of paragraphs is random. This gives us more precise results than the use of manual edition of documents. By the way, stemming and finding synonyms of words require significant computational resources. We use simple replacement of exciting words chosen randomly from a dictionary; this makes no difference to find near-duplicate by non semantic methods. The user can apply a sequence of editing operations choosing an item from Table 1. During the loading process each document splits into paragraphs, each paragraph splits into sentences, and each sentence splits into words. As a result the user can see the statistics for each document (number of paragraphs, sentences and words). The tool

produces a log file in an output folder with names of input and generated files, number of sentences and words of input file, and parameters of the changes made. For example, for paragraph deletion the operation system saves the number of deleted words and characters. This information will be used to compare the results with those of the tested methods.

4 Experiments

As experimental data we used ROMIP collection of URLs (see www.romip.ru) consisting of 52 files of 4.04 GB general size. For experiments the collection was partitioned into several parts consisting of three to 24 files (from 5% to 50% percent of the whole collection). Shingling parameters used in experiments were as follows: the number of words in shingles was 10 and 20, the offset was always taken to be 1 (which means that the initial set of shingles contained all possible contiguous word sequences of a given length). Two methods of composing document image described in Section 2.1 were studied: $\mathfrak{n}$ *minimal elements in a permutation* and *minimal elements in* $\mathfrak{n}$ *permutations*.

The sizes of resulting document images were taken in the interval of 100 to 200 shingles. In case of the lexical representation described in Section 2.1, only words from the resulting dictionary were taken in the document image (the set of descriptive words). As thresholds defining *frequent closed sets* (i.e., the numbers of common shingles in document images from one cluster) we experimentally studied different values in intervals, where the maximal value is equal to the number of shingles in the document image, e.g., [85, 100] for document images with 100 shingles, the interval [135, 150] for document images of size 150, etc. Obviously, choosing the maximal value in the interval, we obtain clusters where document images coincide completely. For parameters taking values in these intervals we studied the relation between resulting clusters of duplicates and ROMIP collection of duplicates (computed by other methods). The ROMIP collection of duplicates consists of pairs of web documents that are considered to be duplicates. For each such pair we sought an intent, which contains both elements of the pair, and vice versa, for each cluster of *very similar documents* (i.e., for each corresponding closed set of documents with more than k common description units) we took each pair of documents in the cluster and looked for the corresponding pair in the ROMIP collection. The output of this unit is the table with the number of common number of duplicate pairs found by our method (denoted by HSE) and those in the ROMIP collection, and the number of unique pairs of HSE duplicates (document pairs

occurring in a cluster of "very similar documents" and not occurring in the ROMIP collection). The results of our experiments showed that the ROMIP collection of duplicates, considered to be a bench-mark, is far from being perfect. First, we detected that there is a large number of false duplicate pairs in this list due to similar framing of documents. For example the pages with the following information about historical personalities Garibald II, Duke of Bavaria and Giovanni, Duke of Milan were declared to be duplicates.

However these pages, as well as many other analogous false duplicate pairs in ROMIP collection do not belong to concept-based (maximal frequent) clusters generated in our approach.

In our study we also looked for *false duplicate clusters* in the ROMIP collection, caused by transitive closure of the binary relation "X is a duplicate of Y" (as in the typical definition of a document cluster in [7]). Since the similarity relation is generally not transitive, the clusters formed by transitive closure of the relation may contain absolutely nonsimilar documents. Note that if clusters are defined via maximal frequent itemsets there cannot be effects like this, because documents in these clusters share necessarily large itemsets.

4.1 Performance of algorithms and their comparison

We measured the elapsed time on the shingling stage, composing document images and generating clusters of similar documents (by algorithms for computing frequent closed itemsets). In the last stage we used and compared various algorithms: several well-known algorithms from data mining [13] and AddIntent, an algorithm which proved to be one of the most efficient algorithms for constructing the set of all formal concept and concept lattices [26]

Experiments were carried out on a PC P-IV HT with 3.0 MHz frequency, 1024 MB RAM under Windows XP Professional. Experimental results and the elapsed time are partially represented in Tables 2, 3, and 4.

In our experiments the best performance is attained by Fpmax* algorithm, followed by the AFOPT algorithm [24]. These two algorithms proved to be the fastest in FIMI competitions [13]. AddIntent* (AddIntent modified for maximal frequent itemsets) lags behind these two, although it performs much better than MAFIA [8]. Optimized implenations of APRIORI and ECLAT [2] failed to compute the output even in the case of small subcollections of documents (about 10% of the whole collection). This relative behavior of algorithms is similar to that observed in [13] in experiments with low support. In the following table we present running times in a typical experiment with different algorithms on a subcollection of about 10% of the whole collection.

FPmax		All Pairs of Duplicates		Unique pairs of duplicates		Common pairs
Input	Threshold	ROMIP	HSE	ROMIP	HSE	
b_1_20_s_100_n1-12.txt	100	105570	15072	97055	6557	8515
b_1_20_s_100_n1-12.txt	95	105570	20434	93982	8846	11588
b_1_20_s_100_n1-12.txt	90	105570	30858	87863	13151	17707
b_1_20_s_100_n1-12.txt	85	105570	41158	83150	18738	22420
b_1_20_s_100_n1-24.txt	100	191834	41938	175876	25980	15958
b_1_20_s_100_n1-24.txt	95	191834	55643	169024	32833	22810
b_1_20_s_100_n1-24.txt	90	191834	84012	155138	47316	36696
b_1_20_s_100_n1-24.txt	85	191834	113100	136534	57800	55300
b_1_10_s_150_n1-6.txt	150	33267	6905	28813	2451	4454
b_1_10_s_150_n1-6.txt	145	33267	9543	27153	3429	6114
b_1_10_s_150_n1-6.txt	140	33267	13827	24579	5139	8688
b_1_10_s_150_n1-6.txt	135	33267	17958	21744	6435	11523
b_1_10_s_150_n1-6.txt	130	33267	21384	19927	8044	13340
b_1_10_s_150_n1-6.txt	125	33267	24490	19236	10459	14031

Table 2: Results of the method $\mathfrak{n}$ *minimal elements in a permutation.*

FPmax		All Pairs of Duplicates		Unique pairs of duplicates		Common pairs
Input	Threshold	ROMIP	HSE	ROMIP	HSE	
m_1_20_s_100_n1-3.txt	100	16666	4409	14616	2359	2050
m_1_20_s_100_n1-3.txt	95	16666	5764	13887	2985	2779
m_1_20_s_100_n1-3.txt	90	16666	7601	12790	3725	3876
m_1_20_s_100_n1-3.txt	85	16666	9802	11763	4899	4903
m_1_20_s_100_n1-6.txt	100	33267	13266	28089	8088	5178
m_1_20_s_100_n1-6.txt	95	33267	15439	26802	8974	6465
m_1_20_s_100_n1-6.txt	90	33267	19393	24216	10342	9051
m_1_20_s_100_n1-12.txt	100	105570	21866	95223	11519	10347
m_1_20_s_100_n1-12.txt	95	105570	25457	93000	12887	12570

Table 3: Results for the method*minimal elements in* $\mathfrak{n}$ *permutations.*

Algorithm	Dataset	Threshold	Time elapsed sec
Fpmax*	b_1_20_s_100_n1-6.txt	95	2,0
	b_1_20_s_100_n1-6.txt	90	3,1
	b_1_20_s_100_n1-6.txt	85	5,3
	b_1_20_s_100_n1-12.txt	100	3,0
	b_1_20_s_100_n1-12.txt	95	9,0
	b_1_20_s_100_n1-12.txt	90	14,2
	b_1_20_s_100_n1-12.txt	85	25,7
	b_1_20_s_100_n1-24.txt	100	16,1
	b_1_20_s_100_n1-24.txt	95	120,0
	b_1_20_s_100_n1-24.txt	90	590,4
	b_1_20_s_100_n1-24.txt	85	1710,6
Afopt	b_1_20_s_100_n1-6.txt	100	1,39
	b_1_20_s_100_n1-6.txt	95	1,984
	b_1_20_s_100_n1-6.txt	90	2,359
	b_1_20_s_100_n1-6.txt	80	3,078
Mafia	b_1_20_s_100_n1-6.txt	100	123
	b_1_20_s_100_n1-6.txt	95	584
	b_1_20_s_100_n1-6.txt	90	1160
	b_1_20_s_100_n1-6.txt	80	2186
	b_1_20_s_100_n1-12.txt	100	1157
apriori_borgelt	b_1_20_s_100_n1-6.txt	100 - 85	failed
eclat_apriori	b_1_20_s_100_n1-6.txt	100 - 85	failed
AddIntent*	b_1_20_s_100_n1-6.txt	100	177,64
	b_1_20_s_100_n1-6.txt	95	186,765
	b_1_20_s_100_n1-6.txt	90	192,765
	b_1_20_s_100_n1-6.txt	85	204,031

Table 4: Performance of FIMI algorithms

In the contexts corresponding to these subcollections, the number of objects is relatively large compared to the threshold minsup value defined by parameters in the definition of duplicates. Thus, these are typical problems of generating frequent itemsets in low-support data and relative performance of data mining algorithms in our experiments is similar to that in survey [13].

5 Conclusions and further work

We propose a framework to detect near-duplicate documents in large text collections. Analyzing the results of our experiments with concept-based def-

inition of clusters of similar documents with ROMIP data collection we can draw the following conclusions:

- ROMIP collection of URLs is a good testbed for comparing performance of FCA and data mining algorithms in generating (maximal frequent) closed sets of attributes.
- The list of ROMIP duplicates contains many false duplicates, which are not detected as such by the methods based on closed itemsets.
- Approaches based on closed sets of attributes propose adequate and efficient techniques for both determining similarity of document images and generating clusters of very similar documents. They can be efficiently used on the stage of outputting documents relevant to a query, when the number of all found relevant documents does not exceed several thousands (around 10,000 documents). However, this algorithm may encounter major difficulties in treating larger collections of documents due to intrinsic exponential worst-case complexity of the problem of computing maximal frequent itemsets.
- For our datasets (which are very "column-sparse"), the best data mining algorithms for computing frequent closed itemsets, FPmax* and Afopt, outperform AddIntent, one of the best algorithm for constructing concept lattice, adapted for computing maximal frequent itemset.
- The results of syntactical methods essentially depend on the *shingle length* parameter. Thus, in our experiments, for the shingle length 10 the results (pairs of duplicates) were much closer to those in the ROMIP list as for the lengths of shingles equal to 20, 15, and 5.
- In our experiments the results obtained by different methods of document representation – π *minimal elements in a permutation* and *minimal elements in n permutations* – did not differ much, which testifies in favor of the first, faster method.

We would also like to create a site of our project on SourceForge.net with freely available sources of the framework. In further developments, we are going to release implementation of other methods for near-duplicate detection (NDD) like I-match, super shingling, and so on. Development of complex techniques for testing NDD methods and creation of testing collections seems to be quite of interest for computer scientists in this field.

Acknowledgements

This work was supported by the Scientific Foundation of Russian State University Higher School of Economics as a part of project 08-04-0022. The authors would like to thank Sergei A. Obiedkov, Igor A. Selitsky, and Mikhail V. Samokhin for helpful discussions and participating in the software realization of the approach.

References

[1] A. M. Hasnah, A new filtering algorithm for duplicate document based on concept analysis, *Journal of Computer Science*, **2**, 5 (2006) 434–440.

[2] C. Borgelt, Efficient implementations of Apriori and Eclat, *Proc. Workshop on Frequent Itemset Mining Implementations Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI '03)*, 2003, Melbourne, FL, USA. ⇒ 226

[3] S. Brin, J. Davis, H. Garcia-Molina, Copy detection mechanisms for digital documents, *1995 ACM SIGMOD International Conference on Management of Data* 1995, pp. 398–409. http://borgelt.net/apriori.html ⇒ 216

[4] A. Broder, On the resemblance and containment of documents, *Proc. Compression and Complexity of Sequences (SEQS: Sequences97)*. pp. 21–29. ⇒ 216, 217, 218, 219, 220

[5] A. Broder, S. Glassman, M. Manasse, G. Zweig, Syntactic clustering of the web, *6th International World Wide Web Conference*, Apr. 1997, pp. 393–404. ⇒ 216

[6] A. Broder, M. Charikar, A.M. Frieze, M. Mitzenmacher, Min-wise independent permutations, *Proc. STOC*, 1998, pp. 327–336. ⇒ 216, 217, 218, 219, 220

[7] A. Broder, Identifying and filtering near-duplicate documents, *Proc. Annual Symposium on Combinatorial Pattern Matching*, *Lecture Notes in Computer Science* (eds. R. Giancarlo and D. Sankoff) **1848**, 2000, pp. 1–10. ⇒ 216, 217, 218, 219, 220, 226

[8] D. Burdick et al., MAFIA: A performance study of mining maximal frequent itemsets, *Proc. Workshop on Frequent Itemset Mining Implementa-*

tions Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI '03), 2003, Melbourne, FL, USA. ⇒ 226

[9] M. S. Charikar, Similarity estimation techniques from rounding algorithms, *34th Annual ACM Symposium on Theory of Computing* (2002) 380–388. ⇒ 216

[10] J. Cho, N. Shivakumar, H. Garcia-Molina, Finding replicated web collections, *Proc. SIGMOD Conference*, (2000) 355–366. ⇒ 216

[11] A. Chowdhury, O. Frieder, D.A. Grossman, M.C. McCabe, Collection statistics for fast duplicate document detection, *ACM Transactions on Information Systems*, **20**, 2 (2002) 171–191. ⇒ 216, 217

[12] B. Ganter, R. Wille, *Formal concept analysis: mathematical foundations*, Springer, Berlin, 1999. ⇒ 220

[13] B. Goethals, M. Zaki, Advances in Frequent Itemset Mining Implementations: Introduction to FIMI03, *Proc. Workshop on Frequent Itemset Mining Implementations Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI '03)*, 2003. ⇒ 226, 228

[14] G. Grahne, J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, in *Proc. FIMI03 Workshop*, 2003. ⇒ 221

[15] G. S. Manku, A. Jain, A. Das Sarma, Detecting near-duplicates for web crawling, *Proceedings of the 16th international conference on World Wide Web*, May 8-12, 2007, Banff, Alberta, Canada ⇒ 216

[16] T. H. Haveliwala, A. Gionis, D. Klein, P. Indyk, Evaluating strategies for similarity search on the web, *Proc. WWW'2002*, Honolulu, 2002, pp. 432–442. ⇒ 216

[17] N. Heintze, Scalable document fingerprinting, *Proc. of the 2nd USENIX Workshop on Electronic Commerce*, 1996, pp. 191–200. ⇒ 216

[18] M. Henzinger, Finding near-duplicate web pages: a largeccale evaluation of algorithms, *Annual ACM Conference on Research and Development in Information Retrieval*, 2006, pp. 284–291. ⇒ 216

[19] T. C. Hoad, J. Zobel, Methods for identifying versioned and plagiarised documents, *Journal of the American Society for Information Science and Technology*, **54**, 3 (2003) 203–215. ⇒ 216

[20] D. I. Ignatov, S. O. Kuznetzov, V. B. Lopatmikova, I. A. Selitsky, Development and testing of system for detection of near-duplicates in collection of R&D documents, *Theoretical and Practical Journal of State University Higher School of Economics for Business Informatics* **4**, 6 (2008) 21–28 (in Russian). ⇒ 218

[21] S. Ilyinsky, M. Kuzmin, A. Melkov, I. Segalovich, An efficient method to detect duplicates of web documents with the use of inverted index, *Proc. 11th Int. World Wide Web Conference (WWW'2002)*, Honolulu, Hawaii, USA, 7-11 May 2002, *ACM*, 2002 ⇒ 216, 217

[22] A. Kolcz, A. Chowdhury, J. Alspector, Improved robustness of signature-based near-replica detection via lexicon randomization, *Proc. KDD'04* (eds. W. Kim, R. Kohavi, J. Gehrke, W. DuMouchel) Seattle, 2004, pp. 605–610. ⇒ 216, 217

[23] S.O. Kuznetzov, S.A. Obiedkov, Comparing performance of algorithms for generating concept lattices, *Journal of Experimental and Theoretical Artificial Intelligence*, **14** (2002) 189–216. ⇒ 221

[24] G. Liu, H. Lu, J. Xu Yu, W. Wei, X. Xiao, AFOPT: An efficient implementation of pattern growth approach, Proc. Workshop on Frequent Itemset Mining Implementations Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI '03), 2003. ⇒ 226

[25] U. Manber. Finding similar files in a large file system, *Proc. of the USENIX Winter 1994 Technical Conference*, 1994, pp. 1–10. ⇒ 216

[26] D. van der Merwe, S.A. Obiedkov, D. Kourie, AddIntent: a new incremental algorithm for constructing concept lattices, *Proc. International Conference on Formal Concept Analysis (ICFCA'04)*, *Lecture Notes in Artificial Intelligence*, **2961**, 2004, pp. 372–385. ⇒ 226

[27] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Efficient mining of association rules using closed itemset lattices, *Inform. Syst.*,**24**, 1 (1999) 25–46. ⇒ 220

[28] M. Potthast, B. Stein, *New issues in near-duplicate detection* in *Data Analysis, Machine Learning and Applications*, Springer, 2007, pp. 601–609. ⇒ 218

[29] W. Pugh, M. Henzinger, Detecting duplicate and near-duplicate files, *United States Patent 6658423* (December 2, 2003). ⇒ 216, 217

[30] N. Shivakumar, H. Garcia-Molina, Finding near-replicas of documents on the web, *Proc. The World Wide Web and Databases, International Workshop (WebDB'98)*, *Lecture Notes in Computer Science*, **1590**, 1999, pp. 204–212. ⇒ 216

[31] N. Shivakumar, H. Garcia-Molina. Building a scalable and accurate copy detection mechanism, *Proc. ACM Conference on Digital Libraries*, March 1996, pp. 160–168. ⇒ 216

[32] C. Xiao, W. Wang, X. Lin, J. X. Yu, Efficient similarity joins for near duplicate detection, *Proceedings of the 17th International Conference on World Wide Web*, Beijing, China, 2008, pp. 131–140.

[33] Y. Zhao, G. Karypis, Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, **55** (2004) 311–331.

Received: May 11, 2009.