UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

Textual Article Clustering in Newspaper Pages

Marco Aiello, Andrea Pegoretti

November 2004

Technical Report # DIT-04-102

Textual Article Clustering in Newspaper Pages

Marco Aiello & Andrea Pegoretti
Dep. of Information and Communication Technologies
Università di Trento
Via Sommarive, 14
38100 Trento, Italy
aiellom@dit.unitn.it andpego@supereva.it

Abstract

In the analysis of a newspaper page an important step is the clustering of various text blocks into logical units, i.e., into articles. We propose three algorithms based on text processing techniques to cluster articles in newspaper pages. Based on the complexity of the three algorithms and experimentation on actual pages from the Italian newspaper *L'Adige*, we select one of the algorithms as the preferred choice to solve the textual clustering problem.

1 Introduction

One of the first and most evident consequences of the revolution brought by the diffusion of computers and by the great success of the Internet is a gradual decrease of the quantity of paper documents in our everyday life: we can read our favorite newspaper on the Internet; we can be informed minute by minute on the last news by blogs and newsletters; we can read a book in electronic format. Nevertheless, there is a body of paper documents still circulating and being printed. These need to be converted to digital format for a number of reasons which include: digital storage, document retrieval, document transmission, document mutli-modal delivery (e.g., text-to-speach or visualization on handheld devices), document summarization, and so on.

The quantity of documents that needs conversion to digital format is thus increasing, creating the need for systems capable to extract knowledge and *'understand'* documents automatically. Such systems must be able not only to scan and store documents, but also

to process the information contained within. On a printed document, these tasks must be performed by a human operator, which involves high costs in terms of work and time, and limited coverage of the document collections.

The purpose of Document Image Analysis (DIA) is to recover information from paper documents, including not only the text but also images, layouts and all the components that characterize a printed document. There are various applications of DIA: from the digitalization of old books and newspapers to the automatic recognition of hand-compiled forms; from the automatic delivery of post parcels to the creation of text-to-speech systems capable to read any kind of paper document to visually impaired people.

DIA is the set of techniques involved in recovering syntactic and semantic information from images of documents, prominently scanned versions of paper documents. The syntactic information is represented by the layout structure of the document, where each basic document object (i.e., indissoluble unit of text, titles, captions, figures, etc.) is identified and described by its content and its topological information (position in the page, width, height, etc.). The semantic information is collected into the logical structure, comprehending both the labeling of the document object (each document object is assigned a label indicating its role: title, sub-title, plain text, page number, etc.) and the detection of the reading order, i.e., the sequence of document objects the user is supposed to read.

1.1 Related work

Various systems have been proposed for document image analysis. A great body of work is dedicated to mail automation, form processing and processing of business letters, for instance [Cesarini et al., 1998, Dengel and Dubiel, 1997]. Lee [Lee et al., 2000] describes a systems that analyzes journals from IEEE, specifically Transactions on Pattern Analysis and Machine Intelligence. Walischewski [Walischewski, 1997] presents a system to process business letters. The use of learning modules leads to more adaptable systems as those presented by Sainz and Dimitriadis [Palmero and Dimitriadis, 1999] and Li and Ng [Li and Ng, 1999]. These latter two systems ignore the important role of textual content. All the above systems are specific to some document class and all consider documents with one unique reading order. Klink [Klink et al., 2000] uses textual features and geometrical relations for the classification of document objects. The classification module is based on fuzzy-matched rules. The reading order detection problem is not addressed.

A prominent example of a system to process newspaper pages is the one developed by

Tsujimoto and Asada [Tsujimoto and Asada, 1992], which is tuned to process regular multi-column black-and-white papers. Both for layout and logical structure detection, domain knowledge is hard-coded into four transformation rules. The use of a tree based representation for the document restricts the class of documents that can be processed. In addition, the rules work only for the specific document class and cannot be adapted to other classes. No use is made of the textual content of the articles. Furthermore, the system does not work for newspapers with very complex layouts.

1.2 Three algorithms for article clustering

In [Aiello et al., 2002] we presented a system which, by using spatial reasoning [Aiello, 2002, Aiello and Smeulders, 2004] and text processing techniques [Baeza-Yates and Ribeiro-Neto, 1999], is able to extract the reading order from document images for documents having very different layouts. In particular, a spatial language for rectangles [Balbiani et al., 1998], based on the work on interval relations of Allen [Allen, 1983], is used to describe the elements in a document and general rules to analyze them. In that work, the focus is on the heterogeneity of the document collection for the extraction of one unique reading order for each document image.

In this paper instead we focus on newspaper pages, which have an additional difficulty with respect to our previous work; that is, on each page there are several independent articles and have independent reading orders. In addition, newspaper pages are an interesting application case as these are still widely used in paper format and there are vast collections of newspapers archived in paper or microfilm.

We propose three text processing based algorithms for the problem of article clustering in newspaper pages, that is, the identification of text blocks which belong to the same article. We provide a complexity analysis of the algorithms. Finally, via experimental results carried out on pages from the Italian regional newspaper *L'Adige* (`http://www.ladige.it`) we show that text processing techniques work for the problem of article clustering, and compare the performances of the various algorithms. Surprisingly, the simplest algorithm has similar performances with respect to the more sophisticated ones and significantly lower complexity, and is the one we propose as the algorithm of choice.

The reminder of the paper is organized as follows. In Section 2, we state the article clustering problem and provide three algorithms to address it. In Section 4, we provide a complexity analysis of the three algorithms. In Section 3 we provide experimental results and

a detailed discussion. Conclusions are presented in Section 5.

2 Article Clustering Algorithms

Newspaper pages are generally formed by several independent articles which are scattered through the page in columns. For simple pages a human reader can determine the subdivision into articles and the reading order from layout and typesetting information only, but in the case of complex newspaper pages the only way to understand the page is to read the text and compare the contents of the blocks. Consider the sample page from *L'Adige* in Figure 1 in which the running text blocks are highlighted by black rectangles. Some articles are immediate to separate from the rest, such as the one on the top of the page, but others are not clear from the layout. Is the text surrounding the central photograph one single article or rather are there two articles? Does the first long column on the left proceed on the small column below the photograph, or does one have to read next the long column on the right of the photograph? This can be said only after having examined the textual content of the page.

Figure 1: A page from *L'Adige*.

2.1 Problem definition

A document is formed by elementary logical units called *document objects*. Examples of document objects are the title, a figure or a caption. Important characteristics of a document object are its bounding box, its position within the page, and its content. We focus on running text document objects which form the contents of an article. For the sake of simplicity, we ignore titles, subtitles and captions; we also ignore any spatial and syntactic information: the only information we consider is the text inside blocks. Given a newspaper page, we assume all the blocks of text in which the layout divides the articles are given, together with their textual content. No other information is assumed.

Definition. (article clustering) Given a set of running text document objects from a newspaper page, we call *article clustering* the problem of partitioning the document objects into disjoint sets of articles, so that the elements of each set taken in appropriate order form a coherent piece of text.

In the following we present three algorithms to perform article clustering based on the textual content of the document objects. The three algorithms follow a general scheme and differ in the final step. First we provide the general algorithm description and then the three variants.

2.2 General algorithm

In order to group the blocks of text in a number of articles (clusters), we must solve two problems, namely:

- find the features which better describe the document objects in an article,
- find the features which better distinguish the document object in an article from the objects in the other articles.

The first set of features provides a quantification of *intra-cluster similarity*, while the second one provides a quantification of *inter-cluster dissimilarity* [Baeza-Yates and Ribeiro-Neto, 1999].

Each article (cluster) is described by the set of the words it contains (intra-cluster similarity) and it differentiates from the others by the presence of terms not present (or less present) in the other articles (inter-cluster dissimilarity).

A method to state if two blocks belong to the same article is to compare the words they contain: if they share the same words (or groups of words, or synonyms) they probably talk

about the same subject. Not all the terms are equally useful for describing the document contents. If a term appears many times in a block it can be considered more important in identifying the subject of the block than a word that appears just one time; if a term appears in the 90% of the blocks it does not provide any useful information to differentiate one block from another.

This effect is captured through the assignment of a numerical weight to each term in a block; this value comprehends a quantification of both the intra-cluster similarity and the inter-cluster dissimilarity. The first one is represented by the frequency of the term in its block and provides a measure of how well it describes the subject in the block (and consequently the subject of the entire cluster the block belongs to), while the second one is represented by a measure of the rarity of the term in the page. The two values are then combined into a single measure.

Each block is thus described by a list of terms and their corresponding weights. We can look at the block as n-dimensional vectors, one dimension for each word listed. Moreover, if all the distinct words through all the blocks are n, each block can be represented by a vector of n elements, where the elements corresponding to words not included in the block are given the value 0. In this way we represent the blocks in the page as a group of vectors in an n-dimensional space; we can consider the angle between two vectors as a measure of the similarity between them. This angle is narrow if the vectors have similar values in the common components and low values in the components they do not share, in other words, when the blocks represented by the vectors are similar. Therefore we take the cosine of the angle between two vectors as the degree of similarity between them [Baeza-Yates and Ribeiro-Neto, 1999]: the value is higher when the angle is narrower, and it takes values between 0 and 1.

When we know the degree of similarity between all the couples of blocks, we can group together all the blocks which are "sufficiently near" and state that they belong to the same article.

The general algorithm for article clustering consists of four steps:

1. *Indexing:* to obtain the list of all the words inside the blocks,
2. *Weighting:* to give a weight to each word inside each block,
3. *Computing the similarity:* to find the similarity between all the couples of vectors,

4. *Clustering:* to group together the blocks which probably belong to the same article.

Let us consider each step of the algorithm.

2.2.1 Indexing

With the process of indexing we reduce the text to a list of index terms or keywords; the set of all the index terms provides a logical view of the block. An index term is a text word whose semantics helps in identifying the document contents. The logical view can be full text if it lists all the words in the text, or it can include only some of them, through the elimination of the stopwords (functional words, such as articles, connectives, auxiliary verbs), the use of stemming (reduction of distinct words to their grammatical root), the reduction of synonyms to a single term and the grouping of sequences of words with a single lexical meaning (such as "forza multinazionale" - multinational force, or first and last name of people). We do not keep any information about the position of the terms in the text, but we keep trace of the number of times each term appears in the text, coupling each keyword in the list with its number of occurrences.

This step outputs for each page a text file containing, for each block of text, a list of terms and their frequency in the block. However, the algorithms used to perform these operations are not a subject of this paper and we refer to [Baeza-Yates and Ribeiro-Neto, 1999] for their explanation.

2.2.2 Weighting

Parsing the index files one obtains the list of words and their frequency in the blocks, in other words a frequency vector for each block. Starting from this information we build, for each block, a t-dimensional vector (where t is the total number of distinct words in the index) where each element represents the weight of a particular word in the block. Next we define weight vector.

Definition. (weight vector) Let b be the number of blocks in the page, t be the number of terms in the page, and k_i be a generic index term. Let $P = \{b_1, b_2, \ldots, b_b\}$ be the complete set of text blocks in a page, and $K = \{k_1, k_2, \ldots, k_t\}$ be the set of all index terms. We associate a value $w_{i,j}$ called *weight* with each index term k_i of a block b_j. A *weight vector* of a block b_j

is a sequence of weights $\{w_{1,j}, w_{2,j}, \ldots, w_{t,j}\}$. If a term i does not appear in the j-th block, then we set $w_{i,j}$ to 0.

Definition. (normalized term frequency) Let $freq_{i,j}$ be the raw frequency of the term k_i in the block b_j, then the *normalized term frequency* $tf_{i,j}$ is

$$tf_{i,j} = \frac{freq_{i,j}}{\max_k freq_{i,j}}$$

in words, the raw frequency divided by the maximum frequency of any term in the block.

Definition. (term rarity) Let n_i be the number of blocks in which the term k_i appears, then the *inverse document frequency* idf_i for the term k_i is

$$idf_i = \log \frac{b}{n_i}$$

the rarer a term is in the document, the more likely it is that the few blocks where it appears belong to the same article.

We notice that the weight of a term k_i of a block b_j is defined as the product of the normalized term frequency and the inverse frequency, that is,

$$w_{i,j} = tf_{i,j} \times if_i \tag{1}$$

In summary, the output of the weighting step is a weight vector for each block.

2.2.3 Computing the similarity

The next step is to fill in a matrix, named *similarity matrix*, where the element (h, k) represents the degree of similarity between the block h and the block k. The similarity is represented by the cosine of the angle between the weight vectors related to the two blocks b_h and b_k. Each element (h, k) of the similarity matrix is given by the following $sim(h, k)$ function:

$$sim(h,k) = \frac{b_h \bullet b_k}{|b_h| \times |b_k|} = \frac{\sum_{i=1}^{t} w_{i,h} \times w_{i,k}}{\sqrt{\sum_{i=1}^{t} w_{i,h}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,k}^2}} \tag{2}$$

The higher is the similarity between two blocks and the higher is the probability that they are in the same article.

2.2.4 Clustering

The clustering process consists in aggregating single blocks into sets of blocks (clusters). We represent the document objects as the nodes of a graph, called *connection graph*. An edge represents the fact that two objects belong to the same cluster. In Figure 2, a connection graph is represented where three clusters are formed, that is, the cluster formed by $\{b_1, b_3, b_4, b_7\}$, the one formed by $\{b_2, b_5\}$ and the singleton $\{b_6\}$.

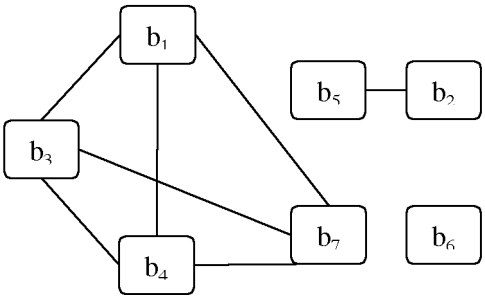

Figure 2: A connection graph.

The three algorithms to perform the article clustering differ only in this last step. All the algorithms start with a graphs with a node per document object and no edges. At each step one or more edges are added to the graph. At any iteration of the algorithm each connected sub-graph represents a portion of an article. The output of the algorithm is a graph where each fully connected component represents a cluster, that is, a complete article. In all the three variants, we fix a similarity threshold: all pairs of blocks with similarity higher than the threshold can be considered part of the same article.

2.3 Simple Clustering Algorithm

In the *simple clustering algorithm (SCA)* an edge in the connection graph is set by looking at the similarity matrix. For each element, if the value is above the threshold, then there is an edge in the graph. The algorithm is reported in Figure 3.

2.4 Comparative Clustering Algorithm

In the simple clustering algorithm all the edges are added at once. On the contrary, in the *comparative clustering algorithm (CCA)* the process of adding an edge is iterative and considers the edges present at each step in the graph. As a point of notation, we define subgraph(b) to be the set of nodes which are connected to b and we indicate its generic

```
1   compute the similarity matrix M
2   for all elements m_{h,k} =sim(h,k) in M
3     if m_{h,k} ≥ threshold then
4       there is an edge in the connection graph between b_h and b_k
```

Figure 3: The Simple Clustering Algorithm (SCA).

element with e_b. Then, the CCA algorithm is reported in Figure 5. First, the CCA algorithm

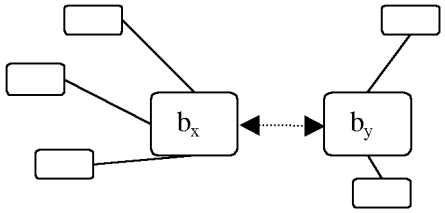

Figure 4: Comparing the subgraphs.

```
1   set the connection graph to have no edges
2   compute the similarity matrix M
3   let max=m_{x,y} be the maximum value in the similarity matrix M
4   if max ≥ threshold then
5     if for all elements e_x, e_y sim(e_x,e_y) > 0 then
6       add the edge x,y to the connection graph
7     go to step 3
8   else terminate
```

Figure 5: The Comparative Clustering Algorithm (CCA).

searches for the blocks which are more similar and then compares partially formed clusters of blocks before adding a new edge. In Figure 4, two nodes are considered to be joined by an edge only if all the nodes they are connected to have an inter-similarity greater than zero. In other words, the rationale is that all the blocks in an article must have a minimum value of similarity between them higher than 0.

2.5 Agglomerative Clustering Alogrithm

The idea of the *Agglomerative Clustering Algorithm (ACA)* is to further push the motivation behind the Comparative Clustering. When two very similar blocks are found, then the two blocks are not only linked via an edge, but they are merged as if they were a single block. This implies that after the merge, all the weights of the blocks need to be recomputed. For each term appearing in at least one of the vectors of x and y the new frequency vector of the block xy is given by the following values:

$$freq_{i,xy} = \begin{cases} \dfrac{freq_{i,x} + freq_{i,y}}{2} & \text{if } freq_{i,x} \neq 0 \text{ and } freq_{i,y} \neq 0 \\ freq_{i,x} & \text{if } freq_{i,x} \neq 0 \text{ and } freq_{i,y} = 0 \\ freq_{i,y} & \text{if } freq_{i,x} = 0 \text{ and } freq_{i,y} \neq 0 \\ 0 & otherwise \end{cases} \tag{3}$$

The idea behind the algorithm is represented in Figure 7. At each step, the two blocks which are more similar and have a similarity above the threshold are merged into a unique block for comparing it with other blocks at the following iteration. In this way, the merged blocks have more power in "attracting" the other blocks of the same article, but on the other hand when two wrong blocks are merged they lose most of the attracting power towards the groups of both, leading to unpredictable results. The reason why we merge the two frequency vectors in such an unusual way lays on the necessity of maintaining the balance of the dimension of the blocks: after few steps of the algorithm there can be a block made of two, three or more blocks, with very high frequency values; experiments not reported in this document showed that this unbalance worsen the algorithms performance.

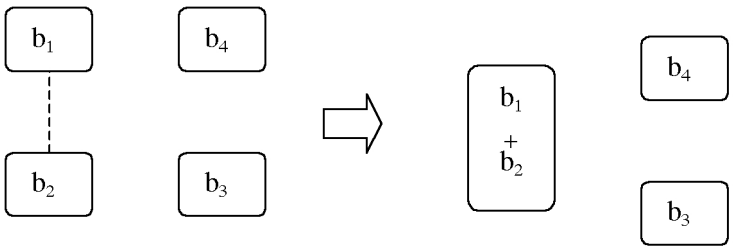

Figure 6: Merging similar blocks.

In summary, the Simple Clustering Algorithm has two main advantages: simplicity and efficiency. The Comparative Clustering adds only a little complexity with respect to the

```
1   set the connection graph to have no edges
2   compute the similarity matrix M
3   let max=m_{x,y} be the maximum value in the similarity matrix M
4   if max ≥ threshold then
5     add the edge x, y to the connection graph
6     merge the frequency vector of x and y into a single vector
7       according to Formula (3)
8     go to step 2
9   else terminate
```

Figure 7: The Agglomerative Clustering Algorithm (ACA).

Simple Clustering algorithm; the exclusion of some edges in the connection graph should entail an improvement in the correctness of the edges set with a possible loss in the fraction of edges found. Finally, the Agglomerative Clustering Algorithm may improve both correctness and completeness, with a significant handicap: it is slower than the other two, since it must re-compute several times all the weights and the similarity matrix.

3 Experimental results

To test whether the proposed algorithms are effective to solve the article clustering problem and to discover which of the strategies behind each one of them is most accurate, we have run them on a real and meaningfully heterogeneous collection of newspaper pages from *L'Adige* (`http://www.ladige.it`). The algorithms work by comparing the similarity of two blocks with a given threshold. The value of this threshold is established performing appropriate experimentation.

The first step in the experimentation is the setting of adequate accuracy measures that can give a quantitative value to the output of the algorithm. Given a newspaper page and the correct clustering of the document objects into articles as determined by a human being (*ground truth*), how close is the output of a article clustering algorithm? To answer this question we introduce the notions of precision, recall, and distribution.

Figure 8: Identification of running text blocks in a page from *L'Adige*.

3.1 Measures

Consider the page from *L'Adige* in Figure 8. The textual document objects are identified by integers ranging from 1 to 10, which form 4 articles. The first article, "La protesta dei sindacati blocca il Consiglio" is made up by blocks 1, 2, and 5; surrounded by blocks 2 and 5 we find the second article, "La Svp lavora per la riforma", which is made up by blocks 3 and 4. The third article is "Così si compromette il lavoro di mesi," composed by blocks 6, 7, 8, and 10; the last article "CGIL: Primo Schonsberg Sbaglia" is composed by block 9 alone. In this case the ground truth is (also shown in Figure 9):

Article 1: block 1, block 2, block 5

Article 2: block 3, block 4

Article 3: block 6, block 7, block 8, block 10

Article 4: block 9

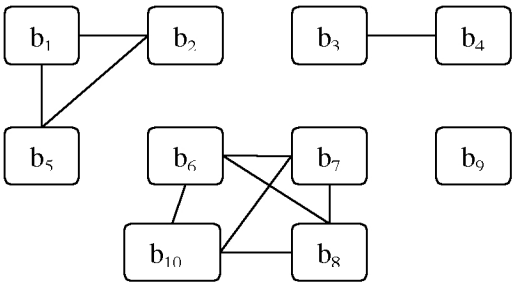

Figure 9: The ground truth for the article in Figure 8.

Assume next that the output of a clustering algorithm is the following (also shown in Figure 10):

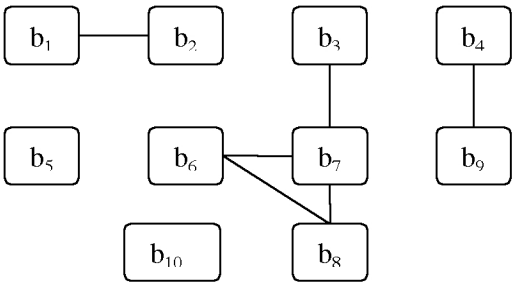

Figure 10: A clustering output for the article in Figure 8.

Article 1: block 1, block 2
Article 2: block 3, block 6, block 7, block 8
Article 3: block 5
Article 4: block 4, block 9
Article 5: block 10

To measure the difference between the output and the golden truth, we compare the respective connection graphs. In Figure 11, we merge the two graphs highlighting the correct edges present in the output graph, the missing ones and the wrongly added ones.

We use two measures taken from the information retrieval literature (see for instance [Baeza-Yates and Ribeiro-Neto, 1999]): *recall* represents the fraction of edges in the ground truth which are in the output, and *precision* represents the fraction of output edges which are in the ground truth graph. In addition, we use another measure to compare the number of articles retrieved which we name *distribution*. More formally, if

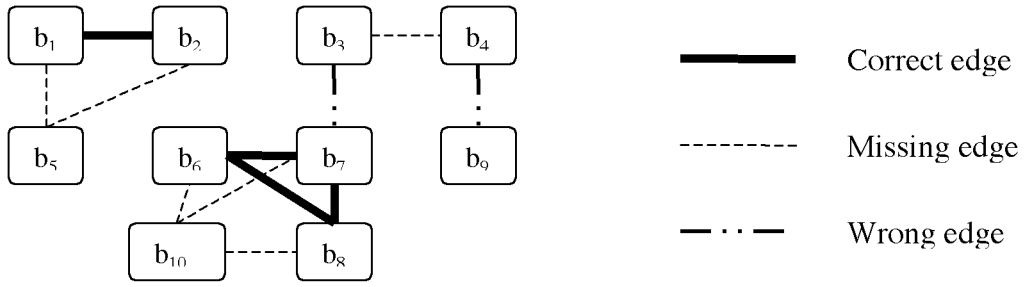

Figure 11: Comparing the connection graphs of the output and ground truth.

Truth edges = number of edges present in the truth graph
Output edges = number of edges present in the output graph
Correct edges = number of edges of the output graph that appear in the truth graph
Truth articles = number of articles in the ground truth
Output articles = number of articles found in the output

Then, we have that

$$\text{Precision} = \frac{\text{Correct edges}}{\text{Output edges}}$$

$$\text{Recall} = \frac{\text{Correct edges}}{\text{Truth edges}}$$

$$\text{Precision} = \begin{cases} \dfrac{\text{Truth articles}}{\text{Output articles}} & \text{if Truth articles} \leq \text{Output articles} \\ \dfrac{\text{Output articles}}{\text{Truth articles}} & \text{otherwise} \end{cases}$$

If the algorithms put together many wrong blocks there will be high recall values opposed to very low precision values; viceversa if the algorithms join just a few number of correct blocks, we will have good values in precision (even 100%) with bad values in recall. For instance, if the algorithm joins together all the blocks in a single article, we have a recall value of 100% (there are edges between all the couples of blocks, among which there are for sure all the correct edges) coupled with an awful precision value.

Finally, another problem needs to be addressed. Suppose to have two output obtained with different thresholds. The first one brings a precision of 60%, a recall of 40% and a

distribution of 70%. The other one has a precision of 50%, a recall of 45% and a distribution of 75%. Which one of the two output is better? In order to answer this question we add a new measure: a *Weighted Harmonic Mean* (WHM) of the three values.

$$\text{WHM} = \frac{5}{\frac{2}{\text{precision}} + \frac{2}{\text{recall}} + \frac{1}{\text{distribution}}}$$

The harmonic mean is weighted for the purpose of giving less importance to the distribution value, which is useful but very imprecise. Since distribution is merely a ratio between the numbers of articles, it can show good values even if precision and recall are very bad.

One may want to reduce all the results to single values of precision, recall and distribution, applicable to the entire group of tests. There are two ways to combine the evaluations: a macro-mean or a micro-mean. With *macro-mean* we intend the normal mean value, obtained summing the values and dividing the result by the number of addends. Let n be the number of executed test, Correct edges$_i$ be the number of correct edges found in the i-th test, Output edges$_i$ be the number of output edges found in the i-th test and so on; then the *micro-mean* is defined as follows:

$$\text{Precision micro-mean} = \frac{\sum_{i=1}^{n} \text{Correct edges}_i}{\sum_{i=1}^{n} \text{Output edges}_i}$$

$$\text{Recall micro-mean} = \frac{\sum_{i=1}^{n} \text{Correct edges}_i}{\sum_{i=1}^{n} \text{Truth edges}_i}$$

$$\text{Distribution micro-mean} = \begin{cases} \dfrac{\sum_{i=1}^{n} \text{Truth articles}_i}{\sum_{i=1}^{n} \text{Output articles}_i} & \text{if } \sum_{i=1}^{n} \text{Truth articles}_i \leq \sum_{i=1}^{n} \text{Output articles}_i \\ \dfrac{\sum_{i=1}^{n} \text{Output articles}_i}{\sum_{i=1}^{n} \text{Truth articles}_i} & \text{otherwise} \end{cases}$$

The WHM of the micro-mean is computed over the three micro-mean values.

3.2 Experimental set up

We used a document collection made of 24 pages taken from different parts of the newspaper *L'Adige* We chose this newspaper for its complex layout and the presence of many articles

in each individual page; moreover, daily on the web-site `http://www.ladige.it` pdf and html reproduction of each page are published. Half of the pages in the collection were used in the determination of the similarity threshold, the other twelve pages were used in the determination of the performances of the algorithms. The algorithms and the evaluation procedure were implemented in Perl. The experiments were run on a standard PC Pentium 4 2400 with 256 MB of RAM memory running the Windows XP with operating system. The complete experimental results are presented in [Pegoretti, 2004].

The distribution of the articles and blocks in the pages is displayed in Figure 12. The second column refers to the pages used for the first two tests, while the third column refers to the pages used in the final test.

	total	**first 12**	**last 12**
Articles	161	72	89
Blocks	407	198	209
Average articles per page	6.7	6	7.4
Average blocks per page	17	16.5	17.4
Average blocks per article	2.5	2.8	2.3

Figure 12: Article distribution in the esperimental data set.

3.2.1 Indexing

To test the three algorithms independently of the indexing technique used in the first phase, we used three different indexing strategies.

Base indexing: A way to index is simply to report all the words in the text with their respective frequency. In addition we reduce plurals and singulars to a single word (like 'governo'/'governi government/governments) and some (very few) groups of synonyms or expressions with the same meaning (e.g., 'prezzo'/'costo' 'price'/'cost' but also 'Washington'/'capitale Americana' 'Washington'/'American capital').

Stop indexing: The only advantage of the previous approach is processing speed. On the negative side, it leaves in the index all the articles, conjunctions, auxiliary verbs and so on. These words are not only so common that one can find them in all the blocks in the page, but moreover they do not add any semantic information. Therefore the second step is to eliminate all these common words, called stopwords.

Bigram indexing: The third strategy contemplates the elimination of the stopwords and also the combination of two words that make a single lexical unit (bigrams): 'dibattito parlamentare' (parliament discussion) or 'forza multinazionale' (multinational force).

In Figure 13 we summarize the distribution of terms and their occurrences through the blocks, as it is obtained from the test pages indexed in the three ways. We display information about the entire collections and then separately about the two halves of the data set.

	base	**stop**	**bigram**
Over all the pages:			
Average number of terms per block	90.0	53.9	50.8
Average number of occurrences per term	1.4	1.1	1.1
Average number of terms per page	933	747	731
First 12 pages:			
Average number of terms per block	90.2	53.3	50.2
Average number of occurrences per term	1.4	1.1	1.1
Average number of terms per page	910	720	703
Last 12 pages:			
Average number of terms per block	89.8	54.5	51.4
Average number of occurrences per term	1.4	1.1	1.1
Average number of terms per page	957	774	759

Figure 13: Data set distribution with respect to indexing strategies.

3.3 Determination of the similarity threshold

We use the Weighted Harmonic Mean to determine the optimal similarity threshold; that is, the threshold which yields the highest micro-mean WHM is the one to be chosen. Thus, the way to choose the threshold is to make tests on a set of documents representative of the collection for which the ground truth is available.

Note that if the threshold is too low, the algorithms put together many unrelated blocks, causing a high recall and low precision; on the opposite, a threshold too high means that less blocks are joined, leading to very good values in precision, but, very bad ones in recall. Moreover, the relation between the two parameters is not linear. Therefore we should find a good balance between the extremes.

The first experiment was run to determine what was the optimal value for the threshold. We applied the tests to the first half of the data set, employing all three indexing strategies. For each page, we repeated the algorithms with different thresholds, evaluated the results and kept the threshold leading to the best WHM value. In Figure 14, we summarize the results.

best value	**algorithm**	**macro-mean**	**micro-mean**
WHM	Agglomerative clustering with stop indexing	75.54%	75.74%
Precision	Comparative clustering with bigram indexing	83.26%	82.22%
Recall	Agglomerative clustering with base indexing	73.54%	71.60%

Figure 14: Results for optimal value determination.

The second group of tests was devoted to the research of the similarity threshold that should be used by a program that implements the algorithms, i.e., a single threshold that should be applied to all the pages of a given set of documents. This threshold was searched on the same data set of the previous tests. Trying the values around the mean value of the best thresholds found (in the latter test), we searched the value that, when applied to all the first half of the pages in the collection, brought the best macro and micro mean of the performance evaluations. We tested all the three algorithms on all the three groups of vectors, as done before. In Figure 15, we summarize the results.

best value	**algorithm**	**macro-mean**	**micro-mean**
WHM	Agglomerative clustering with bigram indexing	70.95%	71.50%
Precision	Comparative clustering with bigram indexing	77.36%	73.26%
Recall	Agglomerative clustering with bigram indexing	72.60%	68.80%

Figure 15: Results for the threshold value determination.

Finally, we tested the algorithms working with a threshold chosen a priori. We applied the fixed threshold found in the latter test to the second half of the document collection. We tested all the three algorithms over all the documents indexed in the three different ways. In Figure 17 we present the results with the base indexing strategy. In Figure 18 we present the results with the stop indexing strategy. In Figure 19 we present the results with the bigram indexing strategy. Finally, in Figure 16 we summarize the overall experimental results highlighting the best algorithms.

best value	algorithm	macro-mean	micro-mean
WHM	Simple clustering with stop indexing	59.77%	58.91%
	Comparative clustering with bigram indexing	59.72%	58.25%
Precision	Simple clustering with stop indexing	65.41%	58.25%
Recall	Agglomerative clustering with stop indexing	66.90%	67.02%

Figure 16: Results for the threshold value determination.

3.4 Discussion

The three algorithms gave very similar results. The best results for each test (obtained sometimes with the stop indexing and sometimes with the bigram indexing) differ one from the other for few percentage points. In the Best Result tests, where Agglomerative Clustering with bigram indexing obtained a WHM value of 75.74% (micro-mean), the other two algorithms proved to be almost equally good: Simple Clustering obtained with stop indexing a WHM of 73.34% and Comparative Clustering with stop indexing obtained a WHM of 73.23%. In the other tests the situation does not vary much: the best results of each algorithm never differ one from the other for more than 3%. Thus, there is apparently no need for the additional operations made by the two variants of the Simple Clustering Algorithm.

We can see how the Simple Clustering algorithm and the Comparative Clustering one give the same results when the base indexing is used. It is because, with all stopwords included in the vectors, each block always brings a similarity degree with all the other blocks different from 0 (they always share some terms), so the Comparing Clustering does not prevent any edge of the connection graph. The Agglomerative Clustering always shows the worst performances when it uses vectors indexed with base indexing. This way of indexing increases the probability of making errors; when the algorithm merges two wrong blocks the errors tend to propagate (errors generate more errors). Compared to the Simple Clustering, we can state that it is more robust but less stable: it commits less errors, but when it incurs in an error the results are unpredictable.

The best and the worst results always correspond to the same documents. The worst results are given by the pages with a low number of articles all discussing more of less the same subject, divided into many blocks. On the other hand, the best results are given where few articles discuss heterogeneous topics and are divided into few big blocks. Even a human reader, put in front of the raw text of the blocks, can have serious difficulties in reconstructing

Simple clustering with base indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0802	0.875	0.917	0.846	0.857
Worst	0.0802	0.183	0.333	0.100	0.750
Macro-mean	0.0802	48.44%	55.56%	53.53%	76.50%
Micro-mean		47.31%	36.57%	51.31%	83.18%
Comparative clustering with base indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0802	0.875	0.917	0.846	0.857
Worst	0.0802	0.183	0.333	0.100	0.750
Macro-mean	0.0802	48.44%	55.56%	53.53%	76.50%
Micro-mean		47.31%	36.57%	51.31%	83.18%
Agglomerative clustering with base indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0704	0.875	0.917	0.846	0.857
Worst	0.0704	0.086	0.037	0.714	0.800
Macro-mean	0.0704	43.80%	47.05%	53.86%	73.17%
Micro-mean		39.46%	25.84%	52.36%	89.90%

Figure 17: The experimental results using the base indexing strategy.

the different articles in the most complex cases.

As expected, the base indexing proved to be the worst type of indexing, leading to the worst results in every test. In particular, in the Best Result tests, it leads to a great number of wrong edges, bringing high recall value and very low precision values. Stop indexing and bigram indexing gave more or less the same results: sometimes the best result is obtained with stop indexing, sometimes with bigram indexing. Since the first one is simpler, it is preferable to the second one, but the data set is too small to exclude one of the two. Actually, bigram indexing can give good results in most cases, since the Keyword-extractor system we used to compute the indexes is still imprecise and the errors eliminate the advantages brought by the right bigrams found.

Simple clustering with stop indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0572	0.896	1.000	0.800	0.923
Worst	0.0572	0.0000	0.000	0.000	0.800
Macro-mean	0.0572	59.77%	65.41%	59.35%	82.34%
Micro-mean		58.91%	51.36%	59.16%	82.41%
Comparative clustering with stop indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0440	0.896	1.000	0.800	0.923
Worst	0.0440	0.183	0.333	0.100	0.750
Macro-mean	0.0440	58.99%	61.80%	61.08%	83.01%
Micro-mean		58.29%	50.23%	57.59%	89.00%
Agglomerative clustering with stop indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0384	0.909	0.857	0.923	1.000
Worst	0.0384	0.183	0.333	0.100	0.750
Macro-mean	0.0384	57.04%	54.54%	66.90%	87.30%
Micro-mean		55.98%	40.89%	67.02%	94.68%

Figure 18: The experimental results using the stop indexing strategy.

The best algorithm proved to be the Agglomerative Clustering applied to vectors given by stop indexing, but only for less than 2 percentage points. This is due to the fact that, even if it has the same best results as Simple Clustering, it has only 2 pages with a WHM value under 60% against 4 pages for Simple Clustering. Since Agglomerative Clustering with base indexing gave the best recall value coupled with the worst precision value, we understand that when the algorithm makes a mistake it tends to put together wrong blocks and not to separate similar blocks. The Comparative Clustering algorithm proved to be the most precise among the three, with a consequent loss in recall, due to the prevention of both wrong and right edges. Some articles include blocks that have a degree of similarity of 0 with other blocks of the same article, and the algorithm avoids these arrangements. Anyway, since they

Simple clustering with bigram indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0446	0.896	1.000	0.800	0.923
Worst	0.0446	0.207	0.096	1.000	0.800
Macro-mean	0.0446	57.03%	61.85%	63.66%	85.44%
Micro-mean		55.38%	42.25%	62.83%	89.90%
Comparative clustering with bigram indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0384	0.896	1.000	0.800	0.923
Worst	0.0384	0.245	0.115	1.000	1.000
Macro-mean	0.0384	59.72%	62.87%	60.57%	79.18%
Micro-mean		58.25%	51.43%	56.54%	86.41%
Agglomerative clustering with bigram indexing					
	threshold	**WHM**	**precision**	**recall**	**distribution**
Best	0.0291	0.811	0.833	0.769	0.857
Worst	0.0291	0.207	0.096	1.000	0.800
Macro-mean	0.0291	56.40%	55.36%	66.09%	84.39%
Micro-mean		54.47%	39.68%	64.40%	96.74%

Figure 19: The experimental results using the bigram indexing strategy.

are quite rare the worsening is roughly equivalent to the improvements (i.e., the number of cut right edges roughly equals the number of avoided wrong edges).

As for the threshold determination, one discovers the (relative) weakness of the Simple Clustering opposed to the robustness of the other two algorithms, which gain the best results in WHM, precision and recall. Our conclusion is that the Agglomerative Clustering, if applied with a good threshold, can give the best overall results. The Comparative Clustering produces the best results in precision, due to the fact that it checks all the edges before adding them to the graph.

In the overall experimentation, one can conclude that the Simple Clustering algorithm is the one with overall best performance, proving to be not only the simplest and to have less

computational complexity, but also to be more flexible than the other two.

4 Complexity analysis

It is interesting to compare the three algorithms not only from the point of view of their performance, as we did in the previous section, but also in terms of their complexity. In the following we consider worst case time complexity, best case and present a representative case. We show that the three algorithms are all polynomial, but do have an increasing time complexity.

We assume to have b blocks per page and that the total number of distinct words (per page) is n. To compute the number of operations to perform in a typical case,[1] we consider the following numbers:

different words per page	$n =$	1000
different words per block	$n_b =$	100
number of blocks	$b =$	21
number of articles	$a =$	7

4.1 Indexing, Weighting and Similarity Matrix computation complexity

In *indexing*, when we parse the lists of the indexed terms, we obtain b vectors with the frequency of the terms. Each vector has only a number of elements equal to the number of different words in the single block, that is much less than n; we have no need to allocate space for the missing terms, since we already know that their frequency is 0. In the worst case each block has n different words. In the best case each word appears just in one block and the number of different words in a block is $\frac{n}{b}$. Usually, each block has more or less $\frac{n}{10}$ different terms with base indexing and the ratio is even smaller with stopword and bigram indexing. The exact complexity of the indexing is out of the scope of the current presentation.

In *weighting*, one computes the weight of terms recurring in a given block, as the other terms have a weight of 0. The weight is computed according to Equation (1). From the indexing step we already know $freq_{i,j}$ for every i, j and the number of blocks b, but we still have to compute the maximum frequency present in each block, and the number n_i of blocks in which the term k_i appears. Both operations can be performed during the same cycle by considering all the frequency vectors, which lead to a worst case complexity of $\mathrm{O}(b \times n)$. If

[1] We talk of typical case, to avoid confusion with the well-known *average complexity* concept.

each block has $\frac{n}{b}$ elements, we have $\mathrm{O}(\frac{n}{b} \times b\ ) = \mathrm{O}\ (n)$ operations. In the typical case, the number of operations is: $100 \times 21 = 2100$ operations. As for the *idfs* values for all the terms, these are constant and can be computed in $\mathrm{o}(n)$. For the typical case, assume a logarithm and a division need 3 operations, then we have a total of $3 \times n$ operations $= 3000$ operations. Finally, the computation of the weights is simply a division and a product, that is, $\mathrm{O}(n \times b)$ in the worst case, $\mathrm{O}(n)$ in the best case and $2100 \times 2 = 4200$ operations in the typical case. In summary, for weighting we have

$\mathrm{O}(b \times n) + \mathrm{O}(n) + \mathrm{O}(b \times n) =$	$\mathrm{O}(b \times n)$	worst case
$\mathrm{O}(n) + \mathrm{O}(n) + \mathrm{O}(n) =$	$\mathrm{O}(n)$	best case
$2100 + 3000 + 4200 =$	9300 op	typical case

In *computing the similarity matrix*, one computes the similarity between all the pairs of blocks. One notices that the similarity matrix is symmetric, thus the number of computations is only half of the size of the matrix's size. The formula to compute the similarity is Equation (2). In the worst and best case, each vector contains all n words of the article yielding a complexity of $\mathrm{O}(n)$. In the typical case, some terms will be missing in both blocks. The experimentation showed that one usually compares blocks with no more than 10 common terms. The norms of the vectors are constant through all the operations, so we can calculate them just once for each vector. These can be computed once by performing n products plus a sum and a division, yielding the complexity of $\mathrm{O}(n)$ for the worst case, $\mathrm{O}(\frac{n}{b})$ for the best case, and $10+1+1 = 12$ operation for the typical case. As for the computation of the norms, in the worst case the vector has n elements, thus, we have to compute n squares, n sums, and a square root for a total of $\mathrm{O}(bn)$ operations. In the best case, each vector has $\frac{n}{b}$ elements so the we have $\mathrm{O}(\frac{n}{b} \times b) = \mathrm{O}(n)$. In the typical case, we have 100 elements, thus, 201 operations per block. In summary, for weighting we have

$\mathrm{O}(\frac{b(b-1)}{2} \times n) + \mathrm{O}(n \times b) =$	$\mathrm{O}(b^2 \times n)$	worst case
$\mathrm{O}(\frac{b(b-1)}{2} \times \frac{n}{b}) + \mathrm{O}(\frac{n}{b} \times b) = =$	$\mathrm{O}(b \times n)$	best case
$\frac{21 \times 20}{2} \times 21 + 201 \times 21 =$	6741 op	typical case

4.2 Simple clustering complexity

To compute the connection graph one needs to compare each element of the similarity matrix with the threshold value. Since the number of elements in the similarity matrix is $\frac{b(b-1)}{2}$, the cost is $O(b^2)$. In Figure 20, the total complexity of the simple clustering algorithm is reported.

Simple clustering complexity			
phase	**Worst Case**	**Best Case**	**Typical Case**
Weighting	$O(b \times n)$	$O(n)$	9300
Similarity matrix	$O(b^2 \times n)$	$O(b \times n)$	6741
Connection graph	$O(b^2)$	$O(b^2)$	289
Total	$O(b^2 \times n)$	$O(b^2 + b \times n)$	16300

Figure 20: Simple clustering complexity.

4.3 Comparative Clustering complexity

In the comparative clustering, the computation of the connection graphs is more expensive, as several cycles are necessary. In the worst case, all the blocks make up a single article and each block has n distinct words.

To connect all the b blocks we have to discover $b-1$ edges, so we have to use the first $b-1$ best values in the similarity matrix. Suppose we set in each iteration the maximum value to an infinite minimum value, so each time we have to check all the $\frac{b(b-1)}{2}$ terms to find the maximum. In total $\frac{b(b-1)}{2} \times b(b-1)$ operations are required. If we sort the values in the matrix (e.g., in time $O(\frac{b(b-1)}{2} \times \log \frac{b(b-1)}{2})$ with a quicksort) this operation has no cost. It is easy to prove by induction that at the end of the process, independently of the order in which the connections have been discovered, we have to check $\frac{b(b-1)}{2} \times \frac{b(b-2)}{2}$ similarities between blocks, due to the checks made by the Comparison Clustering. In summary, the total

cost with or without a quicksort respectively is:

$$\mathrm{O}(\frac{b(b-1)}{2} \times (b-1) + \frac{(b-1)(b-2)}{2}) = \mathrm{O}(b^3)$$
$$\mathrm{O}(\frac{b(b-1)}{2} \times \log \frac{b(b-1)}{2} + \frac{(b-1)(b-2)}{2}) = \mathrm{O}(b^2 \times \log(b^2))$$

In the best case, we have no connection between the blocks and $\frac{n}{b}$ elements in each vector. The cost is simply the one of finding the best value in the similarity matrix. If we use quicksort, the price of its cost is compensated by the fact that we know at no cost whether the maximum value is smaller than the threshold or not. In summary, the total cost with or without a quicksort respectively is:

$$\mathrm{O}(\frac{b(b-1)}{2}) = \mathrm{O}(b^2)$$
$$\mathrm{O}(\frac{b(b-1)}{2} \times \log \frac{b(b-1)}{2}) = \mathrm{O}(b^2 \times \log(b^2))$$

In the typical case, with 21 blocks in 7 articles and 3 blocks per article, referring to CCA algorithm presented in page 10, we have:

step 1 0 operations
step 2 $\frac{b(b-1)}{2} = 210$ operations
step 3 210 operations
step 4 1 operation
step 5 1 operation
step 6 1 operation
step 7 0 operations
step 8 0 operations

Step 3 to 6 are repeated for the 7 articles, plus the initialization operation. In total, we have 2961 operations. Notice that, using a sorting algorithm, one would have 3038 operations, so in the typical case there is no practical advantage to use sorting and in fact we do not use sorting.

In Figure 21, the total complexity of the comparative clustering algorithm is reported.

4.4 Agglomerative clustering complexity

In the agglomerative clustering algorithm, the weights and the similarity matrix are computed several times during one run. We do not give here all the details to compute the complexity

Comparative clustering complexity			
phase	**Worst Case**	**Best Case**	**Typical Case**
Weighting	$O(b \times n)$	$O(n)$	9300
Similarity matrix	$O(b^2 \times n)$	$O(b \times n)$	6741
Connection graph	$O(b^3)$	$O(b^2)$	2961
Total	$O(b^2 \times n + b^3)$	$O(b^2 + b \times n)$	19009

Figure 21: Comparative clustering complexity.

of each step of the algorithm, rather, we summarize the results:

Step	Total number of operations
Read frequency vector	n
Compute $idfs$	$2 \times n$
Compute weights	$2 \times n$
Compute sim. matrix	$\frac{2(b-1)^3 - 6(b(b-1)^2) - (b-1)^2 + 6b^2(b-1) - 2(b-1)}{12}$
Best value search	$\frac{2(b-1)^3 - 6(b(b-1)^2) - (b-1)^2 + 6b^2(b-1) - 2(b-1)}{12}$
Vector merge	n

Then, for the worst case, we have that computing the weights has a total cost of $O(b^2 \times n)$, the cost for the similarity matrix is $O(b^3 \times n + b^2 \times n^2)$, $O(b^3)$ is for best value search and $O(b \times n)$ is for merging vectors. For the best case, the situation is analogous, except that the vectors have size $\frac{n}{b}$, which yields values of $O(n)$, $O(b \times b)$, $O(b^2)$ and 0 for weighting, computing the similarity, searching for the best value and vector merging, respectively. Finally, for our typical case we have the following situation:

Step	Total number of operations
Read frequency vector	29.400
Compute $idfs$	30.000
Compute weights	58.800
Compute similarity matrix	47.235
Best value search	1486
Vector merge	140

Summarizing,the worst and best case complexity of the agglomerative clustering algorithm are reported in Figure 22, together with the typical case.

Agglomerative clustering complexity			
phase	**Worst Case**	**Best Case**	**Typical Case**
Weighting	$O(b^2 \times n)$	$O(n)$	118200
Similarity matrix	$O(b^3 \times n)$	$O(b \times n)$	47235
Best value search	$O(b^3)$	$O(b^2)$	1486
Merging	$O(b \times n)$	0	140
Total	$O(b^3 \times n)$	$O(b^2 + b \times n)$	167061

Figure 22: Agglomerative clustering complexity.

4.5 Discussion

The three algorithms have all polynomial complexity, though there is a strict increase in worst case complexity, going from the simple algorithm to the comparative clustering one, and finishing with the agglomerative clustering algorithm. In Figure 23, we summarize the complexity of the three algorithms. One also notices that the best case cost is the same for all the three, and in particular, it is the cost of the computation of a single similarity matrix.

Algorithms' complexity			
Algorithm	**Worst Case**	**Best Case**	**Typical Case**
Simple Clustering	$O(b^2 \times n)$	$O(b^2 + b \times n)$	16300
Comparative Clustering	$O(b^3 + b^2 \times n)$	$O(b^2 + b \times n)$	19002
Agglomerative Clustering	$O(b^3 \times n)$	$O(b^2 + b \times n)$	167061

Figure 23: Summary of the time complexity results.

The typical case gives us an indication of the fact that simple clustering and comparative have similar cost, on the other hand, agglomerative clustering is typically very expensive. The Agglomerative Clustering cost is one order of magnitude higher than the others, due to the iteration of the weighting step and of the computation of the similarity matrix.

4.6 A note on execution times

The execution times on the pages of the data-set confirm the theoretical complexity results of the previous section. In Figure 24, we show the execution times in milliseconds of the three algorithms using the three different indexing strategies implemented in Perl on a standard PC equipped with Pentium 4 2400 with 256 MB of RAM memory, running the Windows XP operating system. The times represent the average wall clock time of the execution of the algorithms on all the data-set excluding the pre-indexing step which is the same for all algorithms.

Algorithms' execution times			
indexing strategy	**base**	**stopword**	**bigram**
Clustering Algorithm			
simple (SCA)	70	48	43
comparative (CCA)	72	49	45
agglomerative (ACA)	444	291	286

Figure 24: Execution times in milliseconds for the three algorithms.

One notices that the simple and comparative clustering are very close in execution times and that the agglomerative one is significantly slower. Using the base indexing slows down computations as there are more words to compare, i.e., the vectors are bigger. The stopword and bigram strategies are similar with respect to execution time.

In summary, we may conclude that the execution times are low, all below the second, and make the approach feasible for use as part of a document understanding systems.

5 Conclusions

Newspapers are a hard test for document understanding systems as they usually have a high number of document elements, several independent articles scattered on the same page and layouts which are not standardized and not geometrically simple.

We introduced three algorithms to attack the article clustering problem within real newspaper pages. The three algorithms are based on text processing techniques and differ in their last step, that is, in the way the aggregation of blocks into articles is performed. The experimentation, though performed on a small collection of pages, has shown how the per-

formance of the algorithms does not differ much. In particular, the best results for each test (obtained sometimes with the stop indexing and sometimes with the bigram indexing) differ from another for few percentage points. For instance, in the Best Result tests Agglomerative Clustering with bigram indexing obtained a WHM value of 75.74%, while Simple Clustering obtained with stop indexing a WHM of 73.34% and Comparative Clustering with stop indexing obtained a WHM of 73.23%. If the performance of the three algorithms is not so significant, the difference in complexity is significant. All algorithms are polynomial, but the worst case complexity is one degree less for the simple clustering algorithm.

The absolute values of the performance of the algorithms are in the 70% range. These values, which may appear good, but not outstanding, should be considered in the context of a whole document analysis architecture, where our algorithms are a component of a larger system. Just as in [Aiello et al., 2002] the text processing component worked together with a spatial reasoner to identify unique reading orders from documents, a system for newspaper page understanding would have a document clustering component based both on natural language and on spatial clues.

Systems to analyze documents are commercially available. For instance, Textbridge (`http://www.scansoft.com/textbridge`) correctly handles most newspaper pages where articles are divided into blocks, but where blocks of the same article are spatially contiguous. The algorithms we propose could complete such a system to perform understanding of newspaper pages with articles whose blocks are scattered on one page.

The results presented in this article open a number of issues for future investigation. First, one may want to consider larger newspaper collections in languages different from Italian. The techniques used are independent of Italian and our implementation can be easily ported to other languages. Second, one may attempt to consider all the articles in the same newspaper, not only on the same page. It is often the case (especially on the first page) that the same article goes from one page to one or more following ones. Third, one may investigate the result of combining text processing with spatial clues to perform article clustering.

We have shown how article clustering can be effectively performed using text processing techniques and in particular we may conclude, as it has been often concluded in dealing with Information Retrieval problems, that *'simple is beautiful'*. The simple clustering algorithm performed after having simply removed stop words has a high performance rate while keeping a low computational complexity.

Acknowledgments

We thank Enrico Blanzieri, Bernardo Magnini, Christof Monz, and Emanuele Pianta for comments and discussion on the topic, in addition, Emanuele offered valuable support with the tools described in [Tonella et al., 2004]. Pegoretti thanks the Department of Information and Communication Technologies of the University of Trento for the support while completing his 'Tesi di Laurea Triennale' in Computer Science.

References

[Aiello, 2002] Aiello, M. (2002). Document image analysis via model checking. *AI*IA Notizie*, XV(1):45–48.

[Aiello et al., 2002] Aiello, M., Monz, C., Todoran, L., and Worring, M. (2002). Document Understanding for a Broad Class of Documents. *International Journal of Document Analysis and Recognition IJDAR*, 5(1):1–16.

[Aiello and Smeulders, 2004] Aiello, M. and Smeulders, A. (2004). Thick 2D Relations for Document Understanding. *Information Sciences*. To appear.

[Allen, 1983] Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley.

[Balbiani et al., 1998] Balbiani, P., Condotta, J., and Fariñas del Cerro, L. (1998). A model for reasoning about bidimensional temporal relations. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 124–130. Morgan Kaufmann.

[Cesarini et al., 1998] Cesarini, F., Gori, M., Marinai, S., and Soda, G. (1998). Informys: A flexible invoice-like form-reader system. *IEEE Transactions on PAMI*, 20(7):730–746.

[Dengel and Dubiel, 1997] Dengel, A. and Dubiel, F. (1997). Logical labeling of document images based on form layout features. In *Proceedings of the 1997 Workshop on Document Image Analysis, DIA'97*, pages 26–31, Ulm, Germany.

[Klink et al., 2000] Klink, S., Dengel, A., and Kieninger, T. (2000). Document structure analysis based on layout and textual features. In *Proceedings of Fourth IAPR International Workshop on Document Analysis Systems, DAS2000*, pages 99–111, Rio de Janeiro, Brazil.

[Lee et al., 2000] Lee, K., Choy, Y., and Cho, S. (2000). Geometric structure analsysis of document images: A knowledge approach. *IEEE Transactions on PAMI*, 22(11):1224–1240.

[Li and Ng, 1999] Li, X. and Ng, P. (1999). A document classification an extraction system with learning ability. In *ICDAR'99*, pages 197–200, Bangalore, India.

[Palmero and Dimitriadis, 1999] Palmero, G. S. and Dimitriadis, Y. (1999). Structured document labeling and rule extraction using a new recurent fuzzy-neural system. In *ICDAR'99*, pages 181–184, Bangalore, India.

[Pegoretti, 2004] Pegoretti, A. (2004). Clustering of article blocks in newspaper pages. Master's thesis, Univ. of Trento. Available from `http://dit.unitn.it/~aiellom/tesiSvolte.html`.

[Tonella et al., 2004] Tonella, P., Ricca, F., Pianta, E., and Girardi, C. (2004). Using keyword extraction for web site clustering". In *Proc. of WSE 2003, 5th International Workshop on Web Site Evolution*.

[Tsujimoto and Asada, 1992] Tsujimoto, S. and Asada, H. (1992). Major Components of a Complete Text Reading System. *Proceedings of the IEEE*, 80(7):1133–1149.

[Walischewski, 1997] Walischewski, H. (1997). Automatic knowledge acquisition for spatial document interpretation. In *Proceedings of the 4th ICDAR'97*, pages 243–247, Ulm, Germany.