

Bit Sequences and Biclustering of Text Documents

Selim Mimaroglu and Dan A. Simovici
University of Massachusetts at Boston,
Department of Computer Science,
Boston, Massachusetts 02125, USA,
{smimarog, dsim}@cs.umb.edu

Abstract

We propose a new technique for clustering of text documents that relies on a biclustering structure constructed on terms and documents. Our approach makes use of a greedy algorithm applied to bit sequences associated with each group of synonym terms. The use of bit sequences allows us to achieve superior time performance. Additionally, our algorithm provides meaningful cluster descriptions.

1 Introduction

Immense amount of data can be found in digital format due to the cheap storage devices and advances in data compression. In the World Wide Web (WWW), the biggest data collection ever created, each web page can be treated as a text document with some additional features such as hyperlinks. Thus, finding documents that are relevant for a query or searching a collection of documents related to a subject is increasingly more expensive.

To contain the cost of searches various techniques were developed in Information Retrieval (IR), Natural Language Processing (NLP), and in Text Data Mining (TDM). Although each of these fields borrow from each other, there are some distinctions.

The IR approach may be the most common one because it is preferred by the current Internet search engines. The input of a typical IR algorithm consists of a collection of keywords treated as a small document, and the algorithm returns those documents on the WWW that are close to the collection of keywords by a specific dissimilarity criterion; the process is using ranking functions [3].

NLP, which started as a subfield of Artificial Intelligence, is concerned mainly with communication in natural languages, that is, with the understanding and generating natural language [14]. While searching the text documents, NLP uses the semantics of words as extracted from the ambient document to produce more refine search results. This

deeper analysis is, generally, more expensive compared to the IR approach.

TDM uses standard data mining techniques on text documents; for example k -means, and hierarchical clustering algorithms. These algorithms generally require a certain amount of pre-processing that creates suitable structures. Since our aim is to present a new text clustering algorithm, we will discuss briefly several other text clustering techniques.

In [19] the authors investigate the outcome of the k -means and hierarchical clustering algorithms and conclude that an improved version of k -means (which modifies the centroids incrementally, and runs multiple times because of the local minimum problem) yields better results. In the same source [19] the bisecting k -means algorithm is introduced and mentioned as yielding even better results than k -means.

A hybrid approach that combines the hierarchical clustering algorithm with the k -means algorithm for browsing documents is presented in [4]. Initial clusters are found by hierarchical clustering, then the centroids of the initial clusters and their members are refined using k -means. This hybrid approach is proposed in other sources such as [19].

In addition to the difficulties in determining the number of clusters the k -means generally requires multiple runs to achieve better clusters, because randomly selected initial centroids produce a different clustering at each run. Hierarchical clustering algorithms such as UPGMA [11, 13] are slow by operating in quadratic time complexity.

In TDM it is very common to represent documents as vectors, where each unique term's frequency is specified. Let $tf(d, t)$ be the number of occurrences of a term t in a document d . A term is considered unimportant if it appears in many documents; therefore a scaling factor $idf(t)$ (inverse document frequency) is needed to assign an importance value to each term. If t occurs in n_t documents among a collection of n documents, then $idf(t) = \log \frac{n}{n_t}$. The numbers $tf(d, t)$ and $idf(t)$ are combined together

(see [20]) to yield the *tfidf* measure as

$$tfidf(d, t) = tf(d, t) \times idf(t).$$

If (t_1, \dots, t_n) is the sequence of terms under consideration the *tfidf*-vector of a document d is the sequence $(tfidf(d, t_1), \dots, tfidf(d, t_n))$. The cosine similarity measure is the cosine of the angle between two normalized document vectors. In most cases, *tfidf*-document vectors do not contain any components for stop words, and the terms are stemmed. A popular stemming algorithm can be found in [18]. Stemming algorithms are heuristic approaches for stripping suffixes. For example using [18] the words “stemming”, “stems”, and “stemmed” will be stemmed to “stem” which is a valid word in the dictionary. On the other hand “January” will be stemmed to “januari” which is an invalid entry in an English dictionary.

An entirely different approach is adopted in [2] where overlapping (by HFTC Algorithm) and non overlapping (by FTC Algorithm) clusters are formed by using the frequent terms. Using the frequent term sets reduces the dimensionality and provides description of clusters. Authors of [2] mention that overlapping clusterings generated by their algorithm have superior clustering quality compared to the non overlapping clusterings. We observe that generating frequent terms may be an overwhelming task for especially large vocabulary. Total runtime around three minutes is reported in [2] for just 30 terms and 6,000 documents.

Each unique stemmed term creates a dimension in the vector space of the documents. If we think the corpus of documents as a matrix where documents are the rows and terms are the columns, singular value decomposition used in principal component analysis may be used to reduce the dimensionality of the matrix [17]. Latent Semantic Analysis (LSA) or Latent Semantic Indexing [5] is a method built on to the singular value decomposition and it relates semantically similar terms. As a result, semantically similar terms are mapped into a single principal component term, and this new term can be considered to indicate that the document is related to a topic which is represented by the semantically related terms [17]. In other words, LSA overcomes relating two terms literally, two different terms having similar meanings will be matched using LSA.

Stemming algorithms such as [18] reduce dimensionality very little. It has been a known fact that people use different words to mention the same thing. In [10, 9, 8] there has been an interest for using related *concepts*. A *concept* is set of synonyms in their terminology. In the same source it is also mentioned that this solves the synonym problem. Our approach is different than [10, 9, 8], because we do not use *concepts* we use the closest synonyms as explained in Section 3. As different documents use different terminologies, it is wise in our opinion, to look for synonyms and relate the documents even though the documents do not have identi-

cal terms. We use WordNet [16] which is a popular lexical database for English to find out synonyms. Our strategy for finding the synonyms is explained in detail in Section 3.

By using bit sequences for representing the associations between the terms and documents instead of *tfidf* vectors, we achieve faster computational results. In our approach each unique term is denoted by a bit sequence, where these bit sequences form a binary data matrix. Binary data matrix representation is used in the literature; for example, [20] refers to this representation as a *binary spreadsheet of words in documents*. We reduce the dimensionality by coalescing synonyms into a single bit sequence using WordNet [16]. Instead of using a stemming algorithm, we use *base* of a term provided by WordNet. In WordNet terminology the base of a term is also known as *lemma*. Better clusterings are reported in [10] when *lemmas* are used instead of stemmed terms.

Our algorithm finds out the core vocabularies for each topic and clusters the documents using same vocabulary. If a document contains more than one core vocabulary, for example *trade* and *mortgage* vocabularies, it is assigned to both *trade* and *mortgage* clusters by our algorithm. Thus, the clusterings that we compute are, in general, overlapping.

The use of bit sequences allows us to develop a fast algorithm and achieve low memory requirements. As a side-product of the algorithm, we obtain meaningful cluster descriptions as the term sets that characterize these clusters.

The paper is structured as follows. In Section 2, we define bit sequences and examine a few properties that are relevant to our algorithm. The algorithm itself is presented in Section 3, followed by a section dedicated to experimental results. A final section contains our conclusions and plans for future work.

2 Bit Sequences and Document Systems

Starting from the algorithm presented in [1] we develop a biclustering algorithm that works on collections of documents and uses bit sequences. In following, we introduce several basic concepts.

Definition 2.1 A *document system* is a triplet $\mathcal{S} = (\mathcal{D}, \mathcal{T}, \text{to})$, where \mathcal{D} and \mathcal{T} are two non-empty finite sets referred to as the *set of documents* and the *set of terms*, respectively, and $\text{to} : \mathcal{D} \times \mathcal{T} \rightarrow \{0, 1\}$ is a function known as the *term occurrence*.

We have $\text{to}(d, t) = 1$ if t occurs in d and $\text{to}(d, t) = 0$, otherwise.

Let λ be a number such that $1 \leq \lambda \leq |\mathcal{D}|$. A *k-term λ -bicluster* of \mathcal{S} is a pair of non-empty sets (T, D) such that $T \subseteq \mathcal{T}$, $D \subseteq \mathcal{D}$, $\text{to}(d, t) = 1$ for every $d \in D$ and $t \in T$, $|D| \geq \lambda$, and $|T| = k$.

□

Example 2.2 Consider the document system $\mathcal{S} = (\mathcal{D}, \mathcal{T}, \mathbf{to})$, where $\mathcal{D} = \{d_0, d_1, d_2, d_3, d_4, d_5\}$ and $\mathcal{T} = \{t_0, t_1, t_2, t_3, t_4\}$. Suppose that the matrix that represents \mathcal{S} is:

$$\mathbf{B}_{\mathcal{S}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

There is only one 2-term 4-bicluster, namely $B_1 = \{\{t_2, t_3\}, \{d_0, d_1, d_2, d_3\}\}$. B_2 and B_3 are 3-term 3-biclusters:

$$\begin{aligned} B_2 &= \{\{t_0, t_2, t_3\}, \{d_0, d_2, d_3\}\}, \\ B_3 &= \{\{t_1, t_2, t_3\}, \{d_0, d_1, d_3\}\}. \end{aligned}$$

□

Document systems can be represented by binary matrices starting from the operations \wedge , \vee and $'$ which are defined on the set $B = \{0, 1\}$ by:

$$a \wedge b = \min\{a, b\}, a \vee b = \max\{a, b\}, a' = 1 - a,$$

for $a, b \in \{0, 1\}$. These operations are extended componentwise to bit sequences and the set of bit sequences of length n , $B^n = \{0, 1\}^n$ can be equipped with the operations \vee , \wedge and $'$. Thus, we obtain a Boolean algebra having $\mathbf{0}^{tr} = (0, \dots, 0)$ as its least element, and $\mathbf{1}^{tr} = (1, \dots, 1)$ as its greatest element.

If $\mathcal{S} = (\mathcal{D}, \mathcal{T}, \mathbf{to})$, the binary matrix that represents \mathcal{S} is the matrix $\mathbf{B}_{\mathcal{S}}$ defined by:

$$(\mathbf{B}_{\mathcal{S}})_{ij} = \begin{cases} 1 & \text{if } t_j \text{ occurs in } d_i \\ 0 & \text{otherwise.} \end{cases}$$

The j -th column \mathbf{b}^{t_j} of the matrix $\mathbf{B}_{\mathcal{S}}$ corresponds to the term t_j .

The notion of bit sequence of a term can be extended to bit sequences for term sets. Namely, if $L = \{t_{\ell_1}, \dots, t_{\ell_r}\}$ is a term set, then $\mathbf{b}^{\wedge L}$ and $\mathbf{b}^{\vee L}$ are given by:

$$\mathbf{b}^{\wedge L} = \mathbf{b}^{t_{\ell_1}} \wedge \dots \wedge \mathbf{b}^{t_{\ell_r}}. \quad (1)$$

and

$$\mathbf{b}^{\vee L} = \mathbf{b}^{t_{\ell_1}} \vee \dots \vee \mathbf{b}^{t_{\ell_r}}. \quad (2)$$

We have $\mathbf{b}_p^{\wedge L} = 1$ if and only if the document d_p has all the terms in L , and $\mathbf{b}_p^{\vee L} = 1$ if and only if the document d_p has at least one term in L .

For a bit sequence $\mathbf{b} \in \{0, 1\}^n$ denote by $\|\mathbf{b}\|$ the number $\sqrt{\mathbf{b} \cdot \mathbf{b}^{tr}} = \sqrt{\sum_{p=1}^n b_p}$. The row that is obtained by transposing the column \mathbf{b} is denoted by \mathbf{b}^{tr} .

The Euclidean norm of \mathbf{b} considered as a sequence in \mathbb{R}^n is denoted by $\|\mathbf{b}\|$. Since $b^2 = b$ for every $b \in \{0, 1\}$ we have $\|\mathbf{b}\| = \sqrt{\sum_{i=1}^n b_i}$.

Note that (T, D) is a $|T|$ -term λ -bicluster if $\|\mathbf{b}^{\wedge T}\|^2 \geq \lambda$.

3 The Biclustering Algorithm

We make basic use of semantics for clustering text documents by using synonyms, as mentioned earlier. We search the synonyms using the publicly available dictionary WordNet [16]. Our implementation uses a Java API designed to work with WordNet, called JWNL [7].

Before applying the algorithm we execute a pre-processing that consists of two parts which are carried out in parallel. For each term we find its base term, and for each term we determine its most related synonym. As we mentioned earlier, we do not stem the terms. Instead, we use WordNet for finding the base form of each word. While looking up the base terms we find out the most related synonyms by using the WordNet.

Synonyms of a term are referred as *synset*. A term may have several synsets, which are provided in the order of relevance by WordNet

In following we discuss in detail the implementation of pre-processing phase. For each term t_i , we use WordNet to carry out the algorithm described in Figure 1. Stop words are removed, therefore pre-processing is applied to the terms that are not stop words.

Using the pre-processing algorithm from Figure 1 on each term in the test file, we create a new numerical file. Note that every term t is treated as identical with its most related synonym. Starting from this file we create the bit sequences of terms such that a unique bit sequence $\mathbf{b}^{\vee L_t}$ exists for the set of synonyms L_t of each term t .

Our algorithm, *Text Biclustering Using Bit Sequences* (TEBUBS) is presented in Figure 2. The variable S (line 9) denotes the current set of biclusters. Step 8 of the algorithm discovers all the 2-term λ -biclusters, which is the computationally most demanding step of the entire algorithm. For this task, we reduce the number of computations by considering *potentially useful terms* only, instead of using all the terms in the corpus. A term, t_i is said to be a *potentially useful term*, if number of 1s in its bit sequence is at least λ , that is $\|\mathbf{b}^{t_i}\|^2 \geq \lambda$. Note that if a term is not a potentially useful term, it cannot be a useful term, hence it cannot take part in forming a bicluster.

Expansion of an n -term λ -biclusters (T, D) to $(n+1)$ -term λ -biclusters $(T \cup \{t_i\}, D)$ is done as follows. The algorithm selects a term t_i which is a useful term, but is not one of the terms in T as stated in line 15. Observe that for each $t \in T$ all the needed 2-term biclusters $(\{t, t_i\}, D')$ are computed (and stored in a hash table) in line 8. Therefore,

Procedure Pre-Processing:
Input:
a term t_i
Output
an identification number
Method:
if t_i does not have a base (it does not appear in WordNet) **then**
check the hash table to see if t_i appears in the hash table;
if t_i does not appear in the hash table **then**
assign it a new id, store (t_i, id) in the hash table, output id,
halt.
else
use the already assigned id obtained from the hash table,
output id, halt.
else it has base(s)
check the hash table to see if any of its bases already
exist;
if any of its bases appear in the hash table **then**
obtain already assigned id from the hash table, output id,
halt;
else no bases appear in the hash table;
while (t_i has a synset) **do**
obtain the next synset of t_i from WordNet
for (syn:=each member of the synset)
check the hash table to see if syn appears in the
hash table
if syn does not appear in the hash table **then** continue;
else syn appears in the hashtable
obtain the already assigned id, output id, halt;
create a new unique id, store (base of t_i , id) in the hash table,
output id, halt;
end Pre-Processing

Figure 1. Pre-Processing Algorithm

checking the conditions at line 16 requires no new computation. If conditions at line 16 are satisfied, (T, D) will be expanded to $(T \cup \{t_i\}, D)$ as shown in line 17, assuming that it is not a duplicate. Duplications can be avoided easily by using a hash table or by checking only $n-1$ term biclusters when creating n term biclusters. We choose the second approach that is more memory efficient. Expansion of terms are carried out until no further expansion is possible.

4 Experiments

We conducted experiments on Reuters-21578 [15] collection. We included both multi-label and single-label documents; however, documents without any predefined labelling were not included in the testing.

Our document clusters are non-exclusive collections of documents. Formally, a *non-exclusive document clustering* is a family \mathcal{K} of subsets of a document set \mathcal{D} such that $\bigcup \mathcal{K} = \mathcal{D}$, that is, a collection of sets of documents such that each document $d \in \mathcal{D}$ belongs to a set in \mathcal{K} . To com-

1 **Procedure TEBUBS:**
2 **Input:**
3 M (data matrix of documents and terms
 $M = (D, T)$)
4 λ (minimum number of documents for every
bicluster)
5 **Output**
6 B (final set of biclusters)
7 **Method:**
8 Initialize B to contain all the λ -biclusters (T, D)
with $|T| = 2$, where $D = \{d | \mathbf{b}_d^T = 1, \|\mathbf{b}^T\|^2 \geq$
 $\lambda\}$
9 $S = \emptyset$
10 **repeat**
11 $B' = B - S$
12 $Useful_Terms = \bigcup \{T | (T, D) \in B'\}$
13 $S = B$
14 **for each** bicluster $(T, D) \in B'$
15 **for each** $t' \in (Useful_Terms - T)$
16 **if for each** $t'' \in T$ there is a bicluster
 $(\{t', t''\}, D')$ with $D \subseteq D'$
17 Add $(T \cup \{t'\}, D)$ to B if not already
added
18 **until** $|B| = |S|$
19 **end** TEBUBS

Figure 2. The TEBUBS Algorithm

pare non-exclusive document clusterings we need to define an adequate metric on the family $NEC_{\mathcal{D}}$ of all such clusterings.

Let $\mathcal{K}, \mathcal{K}' \in NEC_{\mathcal{D}}$ be two collections of documents, where $\mathcal{K} = \{K_1, \dots, K_m\}$, $\mathcal{K}' = \{K'_1, \dots, K'_n\}$, and $\mathcal{G} = \{d_1, \dots, d_\ell\}$.

Consider the tri-partite graph \mathcal{G} that has as vertices the disjoint union $\mathcal{K} \uplus \mathcal{D} \uplus \mathcal{K}'$. An edge exists between a document d_h and a cluster K_i (or a cluster K'_j) if $d_h \in K_i$ (or $d_h \in K'_j$, respectively). Denote by u_i and v_i the number of edges that join d_i to sets of documents from \mathcal{K} and \mathcal{K}' , respectively. Then, the distance d between the collections \mathcal{K} and \mathcal{K}' is defined as the Canberra distance d_∞ defined on the set \mathbb{R}^ℓ between the vectors $\mathbf{u} = (u_1, \dots, u_\ell)$ and $\mathbf{v} = (v_1, \dots, v_\ell)$, given by:

$$d(\mathcal{K}, \mathcal{K}') = d_\infty(\mathbf{u}, \mathbf{v}) = \max_h |u_h - v_h|.$$

Example 4.1 If $\mathcal{D} = \{d_1, d_2, d_3, d_4, d_5\}$ and

$$\begin{aligned} \mathcal{K} &= \{\{d_1, d_2, d_4\}, \{d_2, d_3, d_5\}, \{d_1, d_5\}\} \\ \mathcal{K}' &= \{\{d_1, d_2, d_3\}, \{d_2, d_3, d_4, d_5\}, \{d_2, d_3, d_5\}, \\ &\quad \{d_1, d_3, d_5\}\}, \end{aligned}$$

the tri-partite graph of this collection is shown in Figure 3. \square

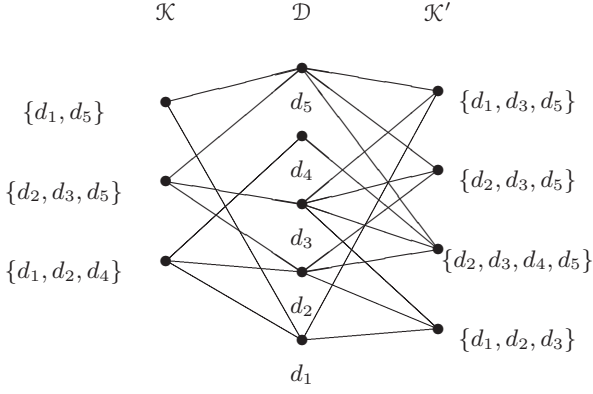


Figure 3. Tripartite Graph of the Collections $\mathcal{K}, \mathcal{K}'$

This distance can be used to evaluate the difference in the distribution of the documents in the collections \mathcal{K} and \mathcal{K}' , respectively. Actually, we shall use a modification of this distance that can be obtained using *Steinhaus transform* (see, for instance [6], p. 118), which we describe next.

Let \mathcal{H} be an arbitrary collection of sets of documents. The Steinhaus transform of the distance d is the distance $d_{\mathcal{H}}$ defined by

$$d_{\mathcal{H}}(\mathcal{K}, \mathcal{K}') = \frac{d(\mathcal{K}, \mathcal{K}')}{d(\mathcal{K}, \mathcal{H}) + d(\mathcal{H}, \mathcal{K}') + d(\mathcal{K}, \mathcal{K}')}.$$

The new metric $d_{\mathcal{H}}$ has the advantage of being bounded (it varies in the interval $[0, 1]$). Distance between covers $\mathcal{K}, \mathcal{K}'$ that are close to \mathcal{H} appear relatively larger than distances between pairs that are situated at larger distances from \mathcal{H} . If we select \mathcal{H} to be any partition of the set of documents, then

$$d(\mathcal{H}, \mathcal{K}) = \max_h |u_h - 1| = \max_h u_h - 1,$$

where (u_1, \dots, u_ℓ) is the vector of the collection \mathcal{K} . Thus, we obtain

$$d_{\mathcal{H}}(\mathcal{K}, \mathcal{K}') = \frac{\max_h |u_h - v_h|}{\max_h |u_h - v_h| + \max_h u_h + \max_h v_h - 2}.$$

For example, the vectors corresponding to the non-exclusive clusterings of Example 4.1 are $\mathbf{u} = (2, 2, 1, 1, 2)$ and $\mathbf{v} = (2, 3, 4, 1, 3)$. Thus, the distance $d_{\mathcal{H}}(\mathcal{K}, \mathcal{K}')$ is 0.428.

We compared our clustering results with the predefined classes of Reuters-21578. Fig. 4 shows the variation of the distance between the clustering we generate and the Reuters predefined classes when λ , the minimal number of documents per cluster, ranges between 5 and 40.

The smallest distance, 0.35, is obtained for $\lambda = 30$, when TEBUBS generates 12 clusters. Hierarchical agglomerative clustering using group average linkage method

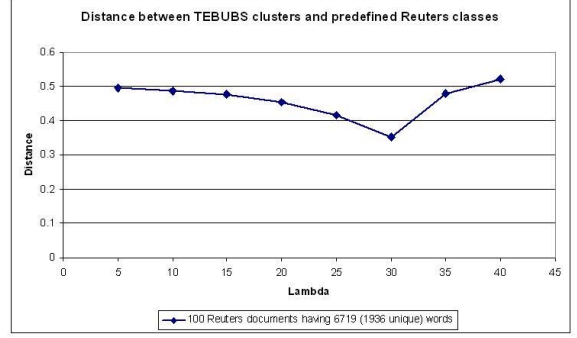


Figure 4. Distance between our generated clusters and predefined Reuters classes

produces a distance of 0.48 (not shown), which is considerably higher than our result. Fig 5 shows the execution time of TEBUBS for the corresponding values of λ . We also

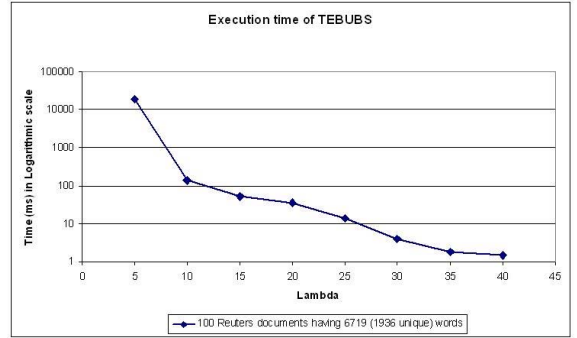


Figure 5. Execution time of TEBUBS

compared our results to those obtained by using CLUTO (see [12]), a software package for clustering high dimensional data, that has been used for clustering documents (see [19]) and is implemented in C. Specifically, we used CLUTO's bisecting k -means implementation. It is worth mentioning that TEBUBS is fully implemented in Java.

For experiments we used a 32-bit, Pentium 4 3.0 GHz processor with 2GB of physical memory; only 1.5GB of that memory was assigned TEBUBS (CLUTO did not have such a restriction). Each test is conducted 10 times and average of 10 runs is reported as the outcome. TEBUBS and CLUTO take different input parameters therefore we fixed a number of generated clusters, which is an input to CLUTO, and for each generated number of cluster we used the corresponding λ as an input to TEBUBS. In the table below λ s and corresponding number of clusters are reported.

λ	# generated clusters
35	3
30	12
25	31

Fig. 6 compares the execution time of TEBUBS and CLUTO. Under the conditions outlined above TEBUBS clearly outperforms CLUTO.

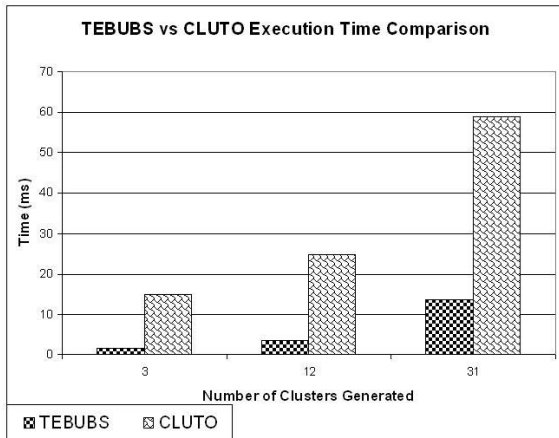


Figure 6. Execution time of TEBUBS and CLUTO

5 Conclusion and Future Work

We presented a novel technique which uses biclusters, term synonyms, and bit sequences to generate overlapping clusterings. By using synonyms we reduced the dimension of the document space considerably, and discovered otherwise unnoticed similarities between documents. The usage of bit sequences allows us to obtain considerable speed gain compared with the other state-of-the-art clustering software packages. Furthermore, our technique provides meaningful cluster descriptions a feature not frequently present in clustering algorithms.

Our future experimental work will involve dealing with document sets that cannot fit into the memory. We expect that TEBUBS will perform very well by utilizing the speed gains and low memory requirements of bit sequences.

References

- [1] J. S. Aguilar-Ruiz, D. S. Rodriguez, and D. A. Simovici. Bi-clustering of gene expression data based on local nearness. In *Proceedings of EGC 2006*, pages 681–692, 2006.
- [2] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery*

and data mining, pages 436–442, New York, NY, USA, 2002. ACM Press.

- [3] S. Brin and L. Page. The anatomy of a large-scale hypertextual (web) search engine. In *Proc. 7th International World Wide Web Conference (WWW7)*, 1998.
- [4] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'92*, pages 318–329, 1992.
- [5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [6] M. M. Deza and M. Laurent. *Geometry of Cuts and Metrics*. Springer-Verlag, Berlin, 1997.
- [7] J. Didion. JWNL 1.3 java wordnet library, 2001. <http://sourceforge.net/projects/jwordnet>.
- [8] A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Proceedings of the 2003 IEEE International Conference on Data Mining*, pages 541–544 (Poster, Melbourne, Florida, November 19-22, 2003. IEEE Computer Society.
- [9] A. Hotho, S. Staab, and G. Stumme. Text clustering based on background knowledge. Technical report, University of Karlsruhe, Institute AIFB, 2003.
- [10] A. Hotho, S. Staab, and G. Stumme. Wordnet improves text document clustering. In *Proc. SIGIR Semantic Web Workshop*, Toronto, 2003.
- [11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
- [12] G. Karypis. CLUTO - a clustering toolkit. <http://glaros.dtc.umn.edu/gkhome/views/cluto>.
- [13] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.
- [14] M. Konchady. *Text Mining Application Programming*. Charles River Media, Boston, 1 edition, 2006.
- [15] D. Lewis. Reuters-21578 text categorization test collection. <http://www.daviddlewis.com/resources/testcollections>.
- [16] G. A. Miller, C. Fellbaum, R. Tengi, P. Wakefield, R. Poddar, H. Langone, and B. Haskell. WordNet a lexical database for the english language, 1998. <http://wordnet.princeton.edu/>.
- [17] S. Mitra and T. Acharya. *Data Mining: Multimedia, Soft Computing, and Bioinformatics*. John Wiley and Sons, New Jersey, 1 edition, 2003.
- [18] M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.
- [19] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Proceedings of Text Mining Workshop, KDD 2000*, 2000.
- [20] S. Weiss, N. Indurkha, T. Zhang, and F. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, New York, 1 edition, 2004.