

# MARIAN Dynamic Incremental Clustering Experiment<sup>1</sup>

FAZLI CAN & ALI ÇÖPLÜ

3 April, 1994

*Abstract.* Clustering a document database is useful for both searching and browsing. While new documents are added to the database, they should be integrated to the existing clustering structure with reasonable cost. Deleting is a similar problem. Because of these, algorithms that deal with clustering structure should handle both deletion and addition. The cover coefficient concept is introduced for handling clustering with low complexity. An extension to Cover Coefficient Based Clustering Methodology is introduced and it is called Cover Coefficient-Based Incremental Cluster Maintenance algorithm (C<sup>2</sup>ICM). In this report, the results of a previous implementation of C<sup>2</sup>ICM with samples from a large MARIAN database with 960,000 documents will be given. Also modifications on the previous implementation and execution time statistics on small MARIAN and INSPEC databases are presented. Some statistics about large MARIAN database are also provided.

---

<sup>1</sup> Based on a work done for the requirements of CS533 (Information Retrieval Systems).

# 1- Introduction

In this report, the results of experiments of previously written programs for a large document database, namely large MARIAN, with 444,651 are given. These previously written programs implemented the Cover Coefficient-Based Incremental Clustering algorithm [6]. The results of application of these programs to small databases can be found in [4,6]. Also the reader is referred to read [2,3] for further understanding of the Cover Coefficient-Based algorithms.

This document first presents an overview of the programs. The changes to these programs can be found in the appendices. The changes are necessary because the execution environment of the programs has been changed from IBM to SUN workstations. Some modifications were done on the program for executing with double precision. Although it is said in [6] that using double precision makes no difference, we found that it makes difference. The number of clusters found with double precision is not the same as the results with single precision. Disk space and memory requirements and execution times of the programs have changed. Updated results of these statistics will be given. The same results will be given for large MARIAN too. Some further comments on the  $C^2$ ICM with the results obtained from larger MARIAN will be presented.

Four appendices are supplied for additional reference. Appendix A gives an error analysis for addition of real numbers that are caused by rounding and normalization during the summation. This is needed because we obtained different results from the same program on SUN and IBM environment. Appendix B explains the interpretation of the raw files of large MARIAN database. In appendix C, the operations done on input files in order to convert to the format required by clustering programs are described. The modifications that are done on the programs are supplied in appendix D.

The rest of the report is organized as follows. A general overview of the programs with disk space and memory requirements is given in section 2. Section 3 is devoted to performance of programs on INSPEC and small MARIAN. In section 4, performance statistics on large MARIAN samples are presented. Using the results in section 4, performance estimation for larger databases are done in section 5. Section 6 describes the effects of database dynamics on the output of the algorithm. Then relationship between clustering and indexing will be given with some tables in section 7. The complexity of the  $C^2$ ICM algorithm is given in [2]. The validation of this complexity analysis is done in section 8. Section 9 summarizes results from the previous sections and provides some future research suggestions.

## 2 - General overview of the programs

The implementation of the Cover Coefficient-Based Incremental Clustering Methodology ( $C^2ICM$ ) was written in VS Pascal for use on IBM mainframe systems running VM/CMS[6]. We have converted these programs to SunOS Pascal for running on SUN workstations. In order to keep memory usage to a minimum, the algorithm was implemented in two programs which communicate via several files. The first program selects the cluster seeds, creates the cluster pointer and vector files and calculates alpha and beta values. The second program performs the clustering. The data structures used in this program can be found in [6] with details.

The following subsections are organized as the following. In section 2.1, we give the pseudo code of the programs which are actually in [6]. The precision of real numbers were changed for more accurate results. The effect of number representation with experimental results are given in section 2.2. Section 2.3 and 2.4 give the memory and disk space requirements for large MARIAN database, respectively.

### 2.1 Programs

At this stage, a pseudo code of the programs may be useful. Pseudo code is taken from [6].

#### **Cluster seed selection program:**

1. Calculate inverse row sums (alpha values), inverse column sums (beta values), and decoupling coefficients for documents and terms.
2. Calculate seed power for all documents. If a document is very distinct i.e., has a decoupling coefficient greater than a certain threshold, then assign it a seed power of zero and decrement  $n_C$  by its decoupling coefficient.
3. Sort documents by seed power.
4. Select seed documents in decreasing order of seed power. If more than one document has the same seed power, select only distinct documents.

#### **Cluster construction program:**

1. Sort seed documents by document number.
2. If this is the first clustering structure to be generated, skip to step 6.

3. For each seed document,  $x$ , in the previous clustering structure, if  $x$  has become a non-seed document in this increment, then falsify the cluster whose seed was  $x$ .
4. For each seed document,  $x$ , if  $x$  is an old document and  $x$  was a non-seed document in the previous clustering structure, then falsify the cluster which contained  $x$ .
5. For each cluster,  $y$ , falsified in the previous two steps, recluster all documents  $x$  in  $y$  if  $x$  is still in the database and is a non-seed documents. If two or more cluster seeds have the same coverage over  $x$ , assign  $x$  to the cluster initiated by the seed with highest seed power among candidates. If two or more cluster seeds have the same seed power among the candidates, assign  $x$  to the cluster initiated by the seed among the candidates with the lowest document number.
6. Cluster all new documents,  $x$ . For situations where two or more cluster seeds have the same coverage over  $x$ , follow the same policy as given in (5).

Several symbols which were introduced by [2,3] are used to characterize a document database. Table 1 gives a list of these symbols and their meanings with their values for the databases used for this document.

**Table 1: Symbols Used to Characterize a Document Database**

Symbol	Meaning	Value in database		
		INSPEC	MARIAN (Small)	MARIAN (Large)
m	Highest document number in the database	12,684	42,815	495,103*
n	Highest term number in the database	14,573	59,536	266,691
$n_c$	Number of clusters	476	5,240	17,422
$x_d$	Average number of terms used per document	32.5	11.2	14.65

\* In this database, 50,452 documents do not contain any term.

The input parameters to the programs are stored in a text file. By changing these parameters programs can run for different purposes. If the initial document is 0, it performs reclustering otherwise it performs incremental clustering algorithm. For debugging purposes, a switch parameter is also available.

## 2.2 The effect of number representation

In the first step, we tried to execute the clustering programs, cluster seed selection and cluster construction, in our SUN workstation environment. These programs were written with VS PASCAL for running on IBM environment. We converted these programs to SunOS PASCAL for running on SUN workstations.

After this, we tried to run the programs on INSPEC and small MARIAN. We observed that the number of clusters selected by our program and previous program is not same. After many debugging steps, we found that this is because of different representation of single precision real numbers in two different environments. In the cluster seed selection program, the decoupling coefficients are calculated and then the ones that are smaller than a threshold value is summed up. During this summation, after processing certain number of documents, a small number between 0 and 1 is added to a big number such as 2000. Adding a small real number to a large real number causes the elimination of some bits and leads to imprecision. For understanding the difference we have done some experiments on the data. The following table may be found useful.

**Table 2: The difference between incremental and normal sum of  $n_c$  values for small MARIAN**

	Single precision with normal sum	Single precision with incremental sum	Double precision with normal sum	Double precision with incremental sum.
On IBM	5,218	5,240	5,240	5,240
On SUN	5,239	5,239	5,240	5,240

The results are taken with decoupling values smaller than a threshold value of small MARIAN. There are 42,815 decoupling coefficients. In this table, incremental sum means that we added the decoupling coefficients with groups of 100 elements. Then we took the summation of these partial sums.

## 2.3 Memory requirements

Memory requirements of both programs for significant data structures were given in [6] but we have changed the representation of real numbers from single precision to double precision. The reason for this change is explained with details in Appendix A. The updated memory requirements of the programs are given in tables 3,4. Estimated memory requirements for large MARIAN database is also given in tables 5,6.

**Table 3: Memory Requirements of the Cluster Seed Selection Program**

Variable	Description	Bytes
address	Starting addresses of document vectors	4m
alpha	Inverse row sums of D	8m
beta	Inverse column sums of D	8n
decoupling_document	Document decoupling coefficients	8m
decoupling_term	Term decoupling coefficients	8n
seed_power	Seed power value	12m
term_presence	Counters for finding database statistics	4n
valid_seed	Validity flag to indicate equivalent seeds	m
TOTAL		33m+20n

**Table 4: Memory Requirements of the Cluster Construction Program**

Variable	Description	Bytes
address	Starting addresses of document vectors	4m
beta	Inverse column sums of D	8n
cluster_members	Cluster member listing for debugging	4m
cluster_seeds	Cluster seed listing for debugging	4n <sub>C</sub>
coverage	Coverages over document being clustered	8n <sub>C</sub>
iisd	Inverted index for seed documents	4n+12n <sub>C</sub> x <sub>d</sub>
inverted_cluster	Inverted cluster index	4m
new_cluster_index	New clustering structure	8n <sub>C</sub> +0.8(m-n <sub>C</sub> )
old_cluster_index	Previous clustering structure	9n <sub>C</sub> +8(m-n <sub>C</sub> )
seed_power	Seed power values	8n <sub>C</sub>
TOTAL		20.8m+12n+ 28.2n <sub>C</sub> +12n <sub>C</sub> x <sub>d</sub>

**Table 5: Cluster Seed Selection Program Memory Requirements for large MARIAN**

Variable	Description	Bytes
address	Starting addresses of document vectors	1,980,412
alpha	Inverse row sums of D	3,960,824
beta	Inverse column sums of D	2,133,528
decoupling_document	Document decoupling coefficients	3,960,824
decoupling_term	Term decoupling coefficients	2,133,528
seed_power	Seed power value	5,941,236
term_presence	Counters for finding database statistics	1,066,764
valid_seed	Validity flag to indicate equivalent seeds	495,103
TOTAL		21,672,219

**Table 6: Cluster Construction Program Memory Requirements for large MARIAN**

Variable	Description	Bytes
address	Starting addresses of document vectors	1,980,412
beta	Inverse column sums of D	2,133,528
cluster_members	Cluster member listing for debugging	1,980,412
cluster_seeds	Cluster seed listing for debugging	69,688
coverage	Coverages over document being clustered	139,376
iisd	Inverted index for seed documents	4,129,552
inverted_cluster	Inverted cluster index	1,980,412
new_cluster_index	New clustering structure	521,521
old_cluster_index	Previous clustering structure	3,978,246
seed_power	Seed power values	139,376
TOTAL		17,052,523

## 2.4 Disk space requirements

Updated disk space requirements of the significant files are itemized in table 7 . Table 8 shows the disk space requirements for the large MARIAN database calculated using table 7.

**Table 7: Disk space requirements**

File	Description	Bytes
dpointer	Document vector starting addresses	4m
dvector	Document vectors	$8m \times d^*$
iclsndx	Inverted cluster index	4m
icolsum	Inverse column sums of D	8n
newcpnt	New cluster index starting address	$4n_C$
newcvec	New cluster index members	$4(m' - n_C)$
oldcpnt	Old cluster index starting addresses	$4n_C$
oldcvec	Old cluster index members	$4(m' - n_C)$
seedpow	Seed power values	$8n_C$
TOTAL		$8m + 8m \times d + 8n + 8m' + 8n_C$

\*  $m'$  is the number of documents with at least 1 term (number of non-deleted documents).

**Table 8: Disk space requirements for large MARIAN**

File	Description	Bytes
dpointer	Document vector starting addresses	1,980,412
dvector	Document vectors	52,113,097
iclsndx	Inverted cluster index	1,980,412
icolsum	Inverse column sums of D	2,133,528
newcpnt	New cluster index starting address	69,688
newcvec	New cluster index members	1,708,916
oldcpnt	Old cluster index starting addresses	69,688
oldcvec	Old cluster index members	1,708,916
seedpow	Seed power values	139,376
TOTAL		61,904,033

m' is 50,452 in large MARIAN.

### 3. Performance on INSPEC and small MARIAN

After the conversion of programs is finished, its performance was tested with INSPEC, and small MARIAN as it was done in [6]. Since the representation of real numbers changed, the results in [6] are not same as our IBM results. We have done the same clustering experiments on IBM again and SUN workstations. In the following subsections, the execution times and comparison for both algorithms are given.

#### 3.1 Execution times for C<sup>3</sup>M and C<sup>2</sup>ICM

Execution times for reclustering portions of two databases on SUN workstations are given in tables 9 and 11 with number of clusters at each step ( $n_c$ ), depth of indexing ( $x_d$ ), term generality ( $t_g$ ), number of non zero entries in D matrix ( $t$ ) and sum of term weights (sum of  $w$ ). In tables 10 and 12, the observed execution times in seconds with other statistics about INSPEC and small MARIAN on IBM machine are given. These tables are useful for showing the effects of different systems on the execution of programs and the effects of changing the number representation precision on the output of the algorithm. Increments to the database were simulated by initially clustering a portion of the database then adding the remaining pieces in equal size.

The results of the following tables are updated results of tables that are given in [6]. Double precision for real numbers is used for these tables. It can be observed that the number of clusters for INSPEC with single precision and double precision are not so different. But for small MARIAN database, the difference is about 21, which is very large for such a database. This difference comes from the addition of real numbers by a computer.

The entries in the tables 9-12 can be interpreted as the following. Each row indicates the slice of database which is taken into consideration. Some parameters for this slice of the database are given with the execution times of two programs. These execution times are for reclustering of this slice of the database.

In these experiments, two different architecture are used: IBM and SUN workstations. The IBM is an ES/9000-480 with about 40 Mips, but MVS operating system which the experiments are done uses about 0.5 of the Mips. So IBM is actually 20 Mips. The experiments on SUN workstations are mainly done on 2 workstations with 22 Mips. But these machines are also file servers of the network. The capacity of these workstations for real numbers is 3.7 megaflops.

**Table 9: Execution Times for Reclustering Portions of INSPEC on SUN**

Database statistics						$n_C$	Seed Selec.	Cluster Constr.
m	n	$x_d$	$t_g$	t	sum of w		Exec. time(s.)	Exec. time(s.)
4,014	8,039	31.82	15.89	127,727	229,102	271	9.5	20.8
5,748	9,682	31.98	18.99	183,838	329,472	321	12.7	35.3
7,482	11,122	32.36	21.77	242,110	433,930	365	16.4	51.7
9,216	12,504	32.70	24.10	301,318	536,267	408	19.9	71.8
10,950	13,567	32.49	26.22	355,718	634,338	444	23.2	90.7
12,684	14,573	32.50	28.29	412,255	733,632	476	27.2	114.9

**Table 10 :Execution Times for Reclustering Portions of INSPEC on IBM**

Database Statistics						$n_C$	Seed Selec.	Cluster Constr.
m	n	$x_d$	$t_g$	t	sum of w		Exec. time(s.)	Exec. time(s.)
4,014	8,039	31.82	15.89	127,727	229,102	271	16.3	13.2
5,748	9,682	31.98	18.99	183,838	329,472	321	23.2	20.9
7,482	11,122	32.36	21.77	242,110	433,930	365	30.2	29.5
9,216	12,504	32.70	24.10	301,318	536,267	408	37.5	38.8
10,950	13,567	32.49	26.22	355,718	634,338	444	43.9	49.2
12,684	14,573	32.50	28.29	412,255	733,632	476	51.0	59.1

**Table 11: Execution Times for Reclustering Portions of small MARIAN on SUN**

Database Statistics						$n_C$	Seed Selec.	Cluster Constr.
m	n	$x_d$	$t_g$	t	sum of w		Exec. time(s.)	Exec. time(s.)
13,551	27,152	10.40	5.19	140,957	158,878	2,578	11.8	256.0
19,404	34,671	10.28	5.75	199,421	224,229	3,382	15.3	483.9
25,257	40,354	10.39	6.51	262,545	296,051	3,896	19.9	751.1
31,110	44,786	10.30	7.15	320,375	361,747	4,331	24.3	1,055.3
36,963	53,053	10.91	7.60	403,166	460,953	4,799	29.7	1,444.2
42,815	59,536	11.15	8.02	477,539	549,168	5,240	34.7	1,876.3

**Table 12: Execution Times for Reclustering Portions of small MARIAN on IBM**

Database Statistics						n <sub>c</sub>	Seed Selec.	Cluster Constr.
m	n	x <sub>d</sub>	t <sub>g</sub>	t	sum of w		Exec. time(s.)	Exec. time(s.)
13,551	27,152	10.40	5.19	140,957	158,878	2,578	19.9	89.0
19,404	34,671	10.28	5.75	199,421	224,229	3,382	27.9	163.3
25,257	40,354	10.39	6.50	262,545	296,051	3,896	36.6	246.9
31,110	44,786	10.30	7.15	320,375	361,747	4,331	44.0	342.3
36,963	53,053	10.91	7.60	403,166	460,953	4,799	54.9	477.0
42,815	59,536	11.15	8.02	477,539	549,168	5,240	64.8	615.0

One can see that the execution time of seed selection program is greater on SUN workstations. But the execution time of cluster construction program is smaller on SUN workstations. This may be due to the seeks that are done on SUN machines. On SUN workstations, SUN Pascal does not support 'seek' function. We handled this problem by calling an external C function. But when each call for the 'seek' function, a conversion from Pascal file pointer to C file pointer is needed. Because of this, the number of seeks play an important role for execution time. The seed selection program performs a constant number of disk seeks because all file processing in this program is sequential. In the cluster construction program, there are two specific instances where a disk seek is necessary

- when building the inverted index for seed documents (IISD), and
- when reclustering a document

So the execution time is related to the number of disk seeks in the cluster construction program.

When the size of the database is small, the execution time of seed selection is near to the execution time of cluster construction program. But when the database size increases, cluster construction execution time dominates. For example, the reclustering time of whole INSPEC database on SUN is 27.2+114.9. The ratio of seed selection time to cluster construction is  $27.2/114.9=0.24$ . The same ratio for small MARIAN database is  $34.7/1876.5=0.018$ . As it can be seen in the following sections, for large MARIAN this ratio is much smaller.

The execution times also depend on the load of the machines because both systems are time sharing systems. We have done the experiments more than once and tried to take reasonable values into the tables.

Tables 13 and 15 show the observed execution times of incremental cluster maintenance in seconds of CPU time on INSPEC and small MARIAN, along with the number of falsified clusters (FC) and documents (FD) on SUN workstations. We showed only the execution time of cluster construction program because the execution time for seed selection program is same in both of them. Also tables 14 and 16 show the same execution times and numbers on IBM. These are also updated versions of results given in [6].

These tables can be interpreted in the following way. The three entries in the lower left corner of table 13 indicates that the number of falsified clusters is 49 and the number of falsified documents is 1,244 when incrementing from 10,950 documents to 12,684 documents in INSPEC. The execution time of building the clustering structure during this increment in INSPEC is 26.92. The execution time of selecting the seeds for this increment is given in the last row of table 9 which is 27.2.

Another observation about these tables is the difference between the number of falsified clusters. These numbers may differ at the same database. This is due to the fact that pre-existing clustering structures may be different. For example, suppose that in one increment two documents that are in the same cluster become seeds. In this case, only one cluster is falsified. But in another increment, these two documents may be in two different clusters. This case would falsify two clusters.

The effects of representation of real numbers with single precision cannot be observed in tables 13 and 14. Because the number of decoupling values is small. It can be at most 12,684. The number of falsified clusters and documents are exactly same for all cases. The execution times are different for the reasons described above. But for small MARIAN, the differences in FC and FD are observed. In Appendix A, an analysis for the error in summation can be found. Because of this error propagation, the differences occur.

**Table 13: Execution Times for Incremental Clustering Portions of INSPEC on SUN**

		Initial size of database									
		4,014		5,748		7,482		9,216		10,950	
Final size		FC	FD	FC	FD	FC	FD	FC	FD	FC	FD
5,748		68	1,052	-	-	-	-	-	-	-	-
		17.2		-		-		-		-	
7,482		67	1,271	67	1,262	-	-	-	-	-	-
		20.7		22.8		-		-		-	
9,216		81	1,737	81	1,708	81	1,687	-	-	-	-
		26.6		29.9		29.5		-		-	
10,950		47	1,079	47	1,116	47	1,079	47	1,101	-	-
		23.2		26.3		25.7		26.0		-	
12,684		49	1,244	49	1,223	49	1,237	49	1,219	49	1,166
		26.9		30.6		36.5		30.1		29.7	

**Table 14: Execution Times for Incremental Clustering Portions of INSPEC on IBM**

		Initial size of database									
		4,014		5,748		7,482		9,216		10,950	
Final size		FC	FD	FC	FD	FC	FD	FC	FD	FC	FD
5,748		68	1,052	-	-	-	-	-	-	-	-
		12.3		-		-		-		-	
7,482		67	1,271	67	1,262	-	-	-	-	-	-
		15.1		15.3		-		-		-	
9,216		81	1,737	81	1,708	81	1,687	-	-	-	-
		20.4		19.3		19.0		-		-	
10,950		47	1,079	47	1,116	47	1,079	47	1,101	-	-
		18.0		17.5		16.9		17.0		-	
12,684		49	1,244	49	1,223	49	1,237	49	1,219	49	1,166
		21.0		19.9		19.8		19.7		19.4	

**Table 15: Execution Times for Incremental Clustering Portions of small MARIAN on SUN**

		Initial size of database									
		13,551		19,404		25,257		31,110		36,963	
Final size		FC	FD	FC	FD	FC	FD	FC	FD	FC	FD
19,404		421	3,050	-	-	-	-	-	-	-	-
		243.4		-		-		-		-	
25,257		509	3,252	510	3,231	-	-	-	-	-	-
		299.8		295.8		-		-		-	
31,110		441	4,729	440	4,757	441	4,595	-	-	-	-
		399.2		400.0		389.4		-		-	
36,963		1,105	5,516	1,105	5,533	1,105	5,566	1,105	5,728	-	-
		475.8		468.2		468.2		476.6		-	
42,815		676	4,304	676	4,298	676	4,259	676	4,190	675	3,974
		479.1		483.4		472.0		471.2		458.6	

**Table 16: Execution Times for Incremental Clustering Portions of small MARIAN on IBM**

Final size	Initial size of database									
	13,551		19,404		25,257		31,110		36,963	
	FC	FD	FC	FD	FC	FD	FC	FD	FC	FD
19,404	419	3,027	-	-	-	-	-	-	-	-
	83.4		-		-		-		-	
25,257	509	3,273	510	3,252	-	-	-	-	-	-
	105.2		104.0		-		-		-	
31,110	447	4,813	446	4,836	447	4,642	-	-	-	-
	90.3		136.6		136.5		-		-	
36,963	1,108	5,542	1,108	5,564	1,108	5,601	1,108	5,762	-	-
	163.8		164.8		169.3		166.3		-	
42,815	680	4,332	680	4,328	680	4,285	680	4,218	679	3,999
	168.4		170.4		168.2		167.0		163.3	

### 3.2 Execution Time Comparison of C<sup>3</sup>M and C<sup>2</sup>ICM

Tables 17-20 summarize the information presented in tables 9-16. They show that incremental clustering (C<sup>2</sup>ICM) costs less than simple reclustering (C<sup>3</sup>M) in terms of execution time. The savings at each increment are shown in these tables. The results are given again both on IBM and SUN.

The tables were prepared in the following way. For table 17, execution times for C<sup>3</sup>M is the summation of entries columns 8 and 9 from table 9. For example, first entry in table 7 column 3 row 6 is 142.1. In table 9, row 6 column 8 and 9 gives 27.2+114.9=142.1. Execution times of C<sup>2</sup>ICM was calculated using table 9 and 13. For example, first entry in table 17 column 4 row 6 is 54.1. In table 9, row 6 column 8 is 27.2 and in table 13 column 1 row 5 is 26.9. So row 6 column 6 in table 17 is 26.9 + 27.2 = 54.1. Note that cluster construction time for C<sup>2</sup>ICM is taken from the first column of tables 13-16. Tables 18-20 were prepared in the same way.

As the database size increases, the savings in execution time grow. Also the execution time per document while using C<sup>3</sup>M increases as the database size increases. The execution time per document while using C<sup>2</sup>ICM decreases as the database size increases. An estimation based on the numerical values for small MARIAN is given in [6]. We will give the same kind of estimations for large MARIAN and compare the results with actual results in the following sections.

**Table 17: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM for INSPEC on SUN**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	4,014	30.3	30.3	0.0
4,014	5,748	48.0	29.9	18.1
5,748	7,482	68.1	37.1	31.0
7,482	9,216	91.7	46.5	45.2
9,216	10,950	113.9	46.4	67.5
10,950	12,684	142.1	54.1	88.0
Total Savings				249.8

**Table 18: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM for small MARIAN on SUN**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	13,551	267.8	267.8	0.0
13,551	19,404	499.2	258.7	240.5
19,404	25,257	771.0	319.7	451.3
25,257	31,110	1,079.6	423.5	656.1
31,110	36,963	1,473.9	505.5	968.4
36,963	42,815	1,911.0	513.8	1,397.2
Total Savings				3,713.5

**Table 19: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM for INSPEC on IBM**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	4,014	29.6	29.6	0.0
4,014	5,748	44.1	35.5	8.6
5,748	7,482	59.8	45.3	14.4
7,482	9,216	76.4	57.9	18.4
9,216	10,950	93.1	61.9	31.2
10,950	12,684	110.1	71.9	38.2
Total Savings				110.9

**Table 20: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM for small MARIAN on IBM**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	13,551	108.8	108.8	0.0
13,551	19,404	191.2	111.3	79.9
19,404	25,257	283.4	141.9	141.6
25,257	31,110	386.2	134.3	252.0
31,110	36,963	531.9	218.7	313.2
36,963	42,815	679.8	233.3	446.5
Total Savings				1,233.1

## 4. Performance on large MARIAN

First part of this report is dealt with the results on INSPEC and small MARIAN databases. After the programs are proven to be valid in our SUN environment, we started to work with large MARIAN. First step is to convert the raw MARIAN files into the format required by programs. This took too much time than expected. Since files are very large, operations on them took very long time. The last step in this process is using a four-way merge program. But this program is designed for smaller number of documents. The execution time of the merge program increases exponentially as the new documents and terms come. The maximum document number in the raw files is about 956,000. Although the program ran more than 5,000 minutes, the maximum document number processed is 495,103. We tried this twice. Since the system is a time-sharing system, we could not be able to finish it. Because of these restrictions, we decided to use this available part. At the end, we converted this part into the required format. We decided to do our experiments with two different portions of the database. First part is the slice between document number 230,000 to 270,000. Second part is the slice between document number 450,000 to 495,103. We add the slices of 10,000 documents at each step although this amount is very large for a database. In order to see the effects of large increments, we did the third experiment with increment slices of 50,000. The results of these three portions are given in tables 22-24. The results of incremental clustering are given in tables 25-27. The comparison of these two results are given in tables 28-30.

An important point is that we could only do our experiments on a SUN workstation environment. We could not do it on IBM because we did not have enough disk space and memory to run the programs.

During our experiments, we found that some documents in the database do not contain any terms. The programs handle this case and consider them as deleted documents. In table 21, we give the document numbers used in experiments ( $m$ ), the number of documents with 0 terms, and the actual number of documents( $m'$ ). After this point in this report, we will use the actual number of documents instead of document numbers. Thus when we use  $m$ , it means  $m'$  unless it is stated explicitly.

**Table 21: The number of documents and actual document numbers**

m	m with 0 terms	m'
230,000	242	229,758
240,000	258	239,742
250,000	265	249,735
260,000	278	259,722
270,000	283	269,717
300,000	29,978	270,022
350,000	50,358	299,642
400,000	50,402	349,598
450,000	50,448	399,552
460,000	50,448	409,552
470,000	50,452	419,548
480,000	50,452	429,548
490,000	50,452	439,548
495,103	50,452	444,651

In figures 1 and 2, the percentage of the added documents are given with the percentages of the number of falsified clusters and documents. In these figures, Db. Inc. means the number of new coming documents as a percentage of the last database. Fls. Doc. is the percentage of falsified documents. Fls. Cl is the falsified clusters percentage over previous size of the database. The data are taken both from first and second parts of the database. The last entry in figure 2 looks smaller because the number of documents is the half of the amount in the other increments. A general observation in these figures is that the percentage of added documents decrease at each step. But when the database size increases, this decrease also slows down.

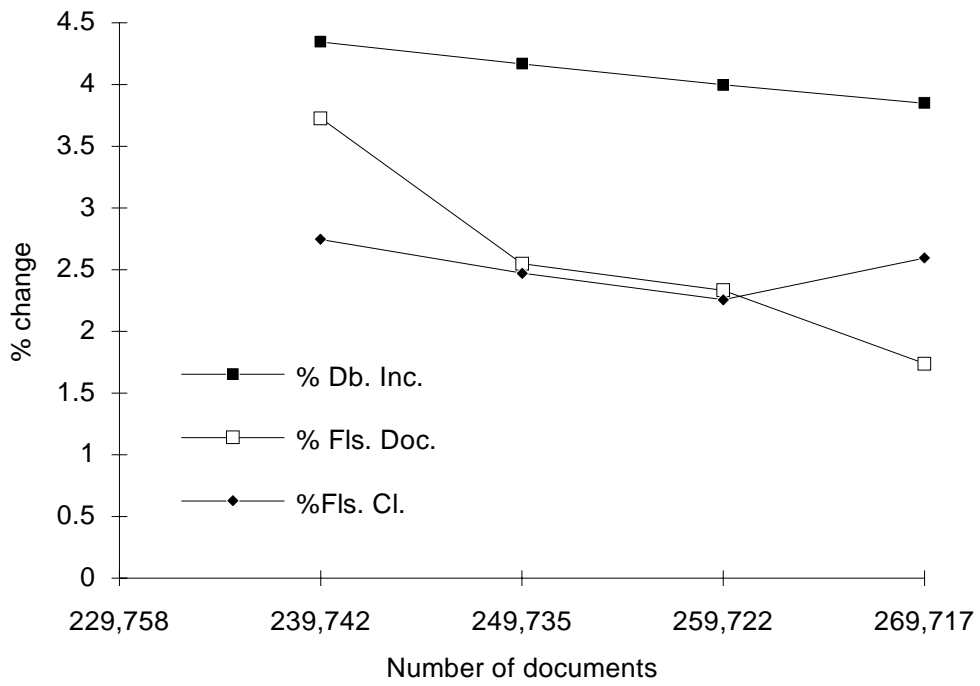


Fig. 1 The percentage change of database increments in large MARIAN part 1

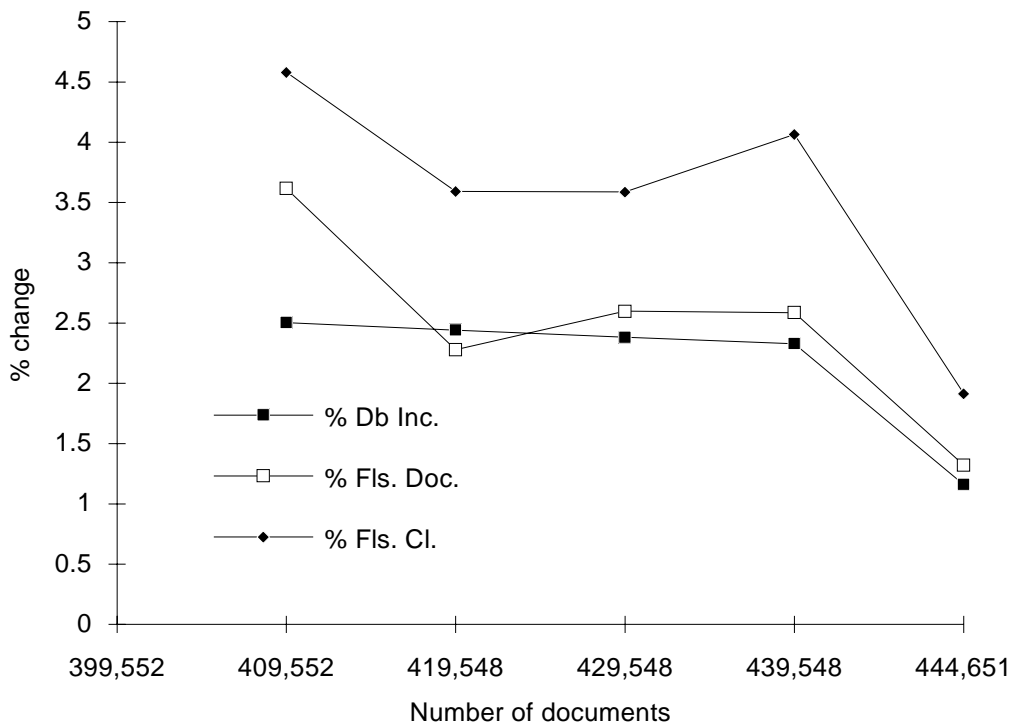


Fig. 2 The percentage change of database increments in large MARIAN part 2

## 4.1 Execution times on large MARIAN

In this section, the results on large MARIAN that are similar to INSPEC and small MARIAN databases are given. We did incremental and reclustering in three parts. Part 1 is document numbers from 230,000 to 270,000 with increments of 10,000 documents. Part 2 is the portion with document numbers from 450,000 to 495,103 with the same amount of increments. Part 3 consists of document numbers from 300,000 to 495,103 with 50,000 increments. Since there are deleted documents, the number of documents that are given in tables are not same as the document numbers. The same statistics with INSPEC and small MARIAN for large MARIAN parts are given in tables 22-24. The ratio of execution time of seed selection program to execution time of cluster construction program decreases. For large MARIAN part 1, this ratio is  $233.3/24,893.4=0.009$ . For large MARIAN part 2, this ratio is  $463.2/66,102.5=0.007$ .

**Table 22: Execution Times for Reclustering Portions of large MARIAN Part 1**

Database Statistics						$n_c$	Seed Selec. Exec. time(s.)	Cluster Constr. Exec. time(s.)
m	n	$x_d$	$t_g$	t	sum of w			
229,758	151,380	12.18	18.48	2,797,907	3,257,666	12,678	200.2	19,419.3
239,742	155,338	12.16	18.76	2,914,649	3,394,772	13,024	208.9	20,823.3
249,735	158,971	12.14	19.08	3,032,765	3,537,310	13,341	216.5	22,164.5
259,722	162,397	12.15	19.44	3,156,628	3,680,943	13,609	229.7	23,605.6
269,717	165,454	12.08	19.70	3,259,035	3,813,548	13,924	233.3	24,893.4

**Table 23: Execution Times for Reclustering Portions of large MARIAN Part 2**

Database Statistics						$n_c$	Seed Selec. Exec. time(s.)	Cluster Constr. Exec. time(s.)
m	n	$x_d$	$t_g$	t	sum of w			
399,552	240,941	14.08	23.35	5,626,610	6,829,368	16,591	401.3	54,644.7
409,552	246,977	14.22	23.58	5,823,995	7,083,105	16,783	413.5	57,085.7
419,548	252,369	14.34	23.84	6,017,675	7,331,963	16,973	424.7	59,673.5
429,548	257,510	14.46	24.12	6,212,520	7,583,552	17,144	439.8	62,299.4
439,548	263,600	14.59	24.32	6,411,324	7,838,864	17,327	451.8	64,827.4
444,651	266,691	14.65	24.42	6,513,436	7,970,079	17,422	463.2	66,102.5

**Table 24: Execution Times for Reclustering Portions of large MARIAN Part 3**

Database Statistics						$n_c$	Seed Selec. Exec. time(s.)	Cluster Constr. Exec. time(s.)
m	n	$x_d$	$t_g$	t	sum of w			
270,022	165,657	12.09	19.71	3,264,511	3,820,371	13,931	233.5	26,532.2
299,642	182,918	12.56	20.57	3,763,379	4,459,694	14,627	269.5	33,451.9
349,598	211,471	13.38	22.11	4,676,366	5,616,331	15,617	331.2	43,315.6
399,552	240,941	14.08	23.35	5,626,610	6,829,368	16,591	401.3	64,827.4
444,651	266,691	14.65	24.42	6,513,436	7,970,079	17,422	463.2	66,102.5

## 4.2 Performance with incremental clustering

In this subsection, the results of incremental clustering are given. The execution times of seed selection program are same with the reclustering algorithm. Because of this, the execution times of cluster construction program are given in tables 25-27. The number of falsified documents and clusters are given in the same tables. As it is said in the previous subsection, in the first increment the number of falsified documents is very high with respect to other steps.

In tables 28-30, the savings of C<sup>2</sup>ICM algorithm over C<sup>3</sup>M algorithm can be found. An important result is that as the database size increases, the savings amount increase very much. This saving is  $(64,351.9/66,565.7)*100=96.7\%$  for the whole database (last row in table 29).

**Table 25: Execution Times for Incremental Clustering Portions of Large MARIAN Part 1**

	Initial size of database							
	229,758		239,742		249,735		259,722	
Final size	FC	FD	FC	FD	FC	FD	FC	FD
239,742	348	8,565	-	-	-	-	-	-
	1,668.2		-		-		-	
249,735	322	6,109	322	6,127	-	-	-	-
	1,502.6		1,493.8		-		-	
259,722	301	5,825	301	5,849	301	5,943	-	-
	1,536.7		1,548.4		1,543.7		-	
269,717	353	4,519	353	4,629	353	4,702	353	4,774
	1,327.1		1,327.5		1,339.4		1,335.6	

**Table 26: Execution Times for Incremental Clustering Portions of Large MARIAN Part 2**

	Initial size of database									
	399,552		409,552		419,548		429,548		439,548	
Final size	FC	FD	FC	FD	FC	FD	FC	FD	FC	FD
409,552	760	14,450	-	-	-	-	-	-	-	-
	3,544.5		-		-		-		-	
419,548	603	9,333	603	9,440	-	-	-	-	-	-
	2,926.6		2,919.8		-		-		-	
429,548	609	10,901	609	10,880	609	10,679	-	-	-	-
	3,173.8		3,192.8		3,144.8		-		-	
439,548	697	11,107	697	11,177	697	11,199	697	10,918	-	-
	3,276.2		3,254.2		3,279.3		3,222.8		-	
444,651	332	5,815	332	5,821	332	5,742	332	5,903	332	5,943
	1,750.6		1,773.5		1,741.3		1,790.6		1,793.4	

**Table 27: Execution Times for Incremental Clustering Portions of Large MARIAN Part 3**

Initial size		Final Size							
		299,642		349,598		399,552		444,651	
270,022	FC	2,737	7,894.3	3,702	12,578.1	3,497	14,473.2	2,716	14,319.0
	FD	34,861		72,215		55,136		49,626	

**Table 28: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM on Large MARIAN Part 1**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	229,758	19,619.5	19,619.5	0.0
229,758	239,742	21,032.2	1,877.1	19,155.1
239,742	249,735	22,381.0	1,719.1	20,661.9
249,735	259,722	23,835.3	1,766.4	22,068.9
259,722	269,717	25,126.7	1,560.4	23,566.3
Total Savings				85,452.2

**Table 29: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM on Large MARIAN Part 2**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	399,552	55,046.0	55,046.0	0.0
399,552	409,552	57,499.2	3,958.0	53,541.2
409,552	419,548	60,098.2	3,351.3	56,746.9
419,548	429,548	62,739.2	3,613.6	59,125.6
429,548	439,548	65,279.2	3,728.0	61,551.2
439,548	444,651	66,565.7	2,213.8	64,351.9
Total Savings				295,316.8

**Table 30: Comparison of Execution Times for C<sup>3</sup>M and C<sup>2</sup>ICM on Large MARIAN Part 3**

Increment		Execution Times(sec)		
From	To	C <sup>3</sup> M	C <sup>2</sup> ICM	Savings(sec)
0	270,022	26,765.7	26,765.7	0
270,022	299,642	33,721.4	8,163.8	25,557.6
299,642	349,598	43,646.8	12,909.3	30,737.5
349,598	399,552	65,228.9	14,874.5	50,354.4
399,552	444,651	66,565.7	14,782.2	51,783.5
Total Savings				158,433.0

The following is two figures that are showing the change in the number of terms used while the database size is changing. It can be observed that the number of terms increase while the number of documents increase. But the change in the number of terms decrease as the database is growing. The number of terms saturates after some point. In figure 4, it can be observed more precisely.

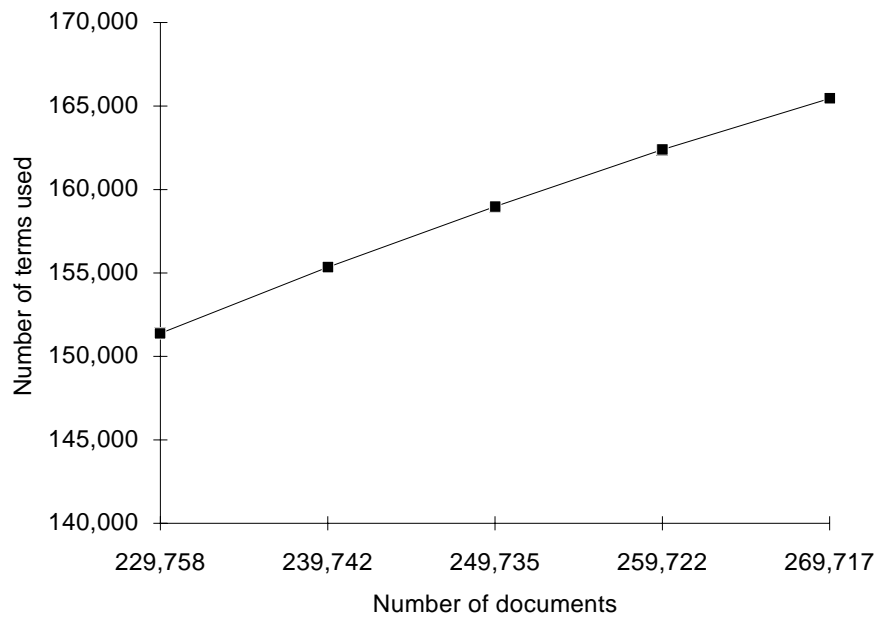


Fig. 3 The change of number of terms for large MARIAN part 1

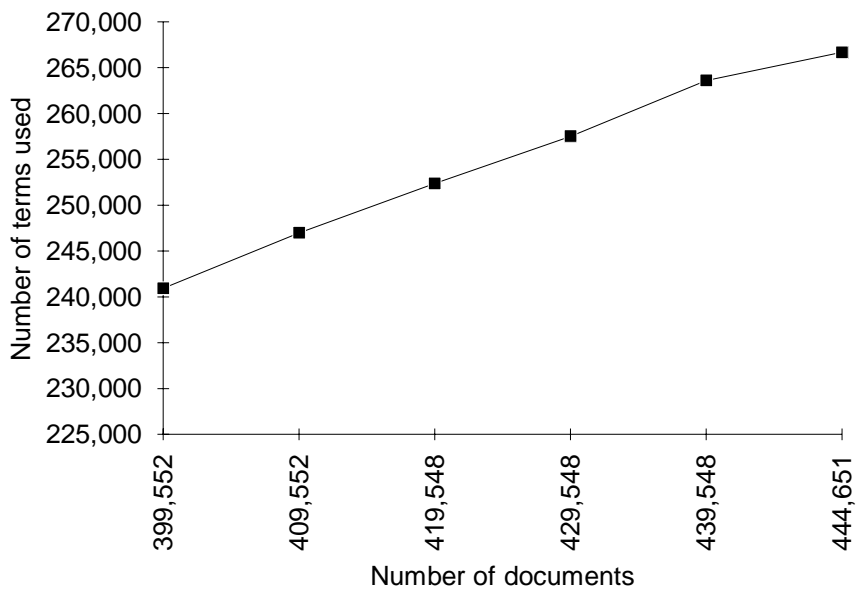


Fig. 4 The change of number of terms for large MARIAN part 2

## 5 - Performance estimation for larger databases

One of the purposes of this project is to estimate the execution time of  $C^2ICM$  algorithm for larger databases. In the previous sections, we gave the results of  $C^2ICM$  and  $C^3M$  algorithms. In this section, we try to estimate the execution time of larger databases.

In tables 31 and 32, average time per document for incremental and reclustering algorithms for large MARIAN are given. In these tables,  $t_{rec}$  means the execution time for  $C^3M$  algorithm. It is the reclustering time of the whole database.  $t_{inc}$  is the execution time for  $C^2ICM$  algorithm. These data are taken from previous tables.  $t_{rec}/m$  is the average time per document when using  $C^3M$  and  $t_{inc}/m$  is the average time per document when using  $C^2ICM$  algorithm. In both tables,  $t_{inc}$  is 0 for first row. Because this is the same as reclustering. Because of this reason, we did not take them into account.

**Table 31 : Average time per document for incremental and reclustering algorithms for large MARIAN Part 1**

m	$t_{rec}$	$t_{inc}$	$t_{rec}/m$	$t_{inc}/m$
229,758	19,619.5	0.0	0.085392	-
239,742	21,032.2	1,877.1	0.087728	0.00783
249,735	22,381.0	1,719.1	0.089619	0.006883
259,722	23,835.3	1,766.4	0.091772	0.006801
269,717	25,126.6	1,560.4	0.093159	0.005785

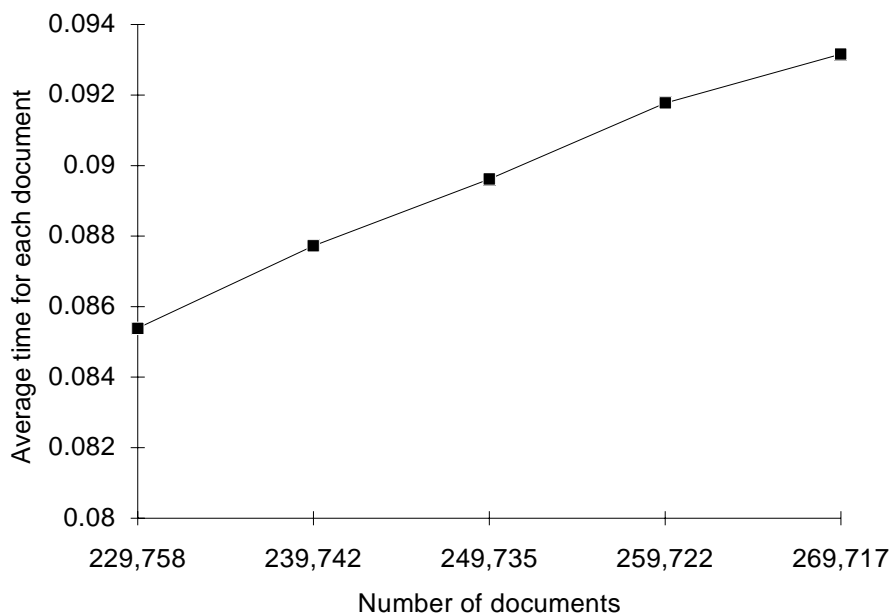


Fig. 5 The change in average time per document for  $C^3M$  algorithm on large MARIAN part 1

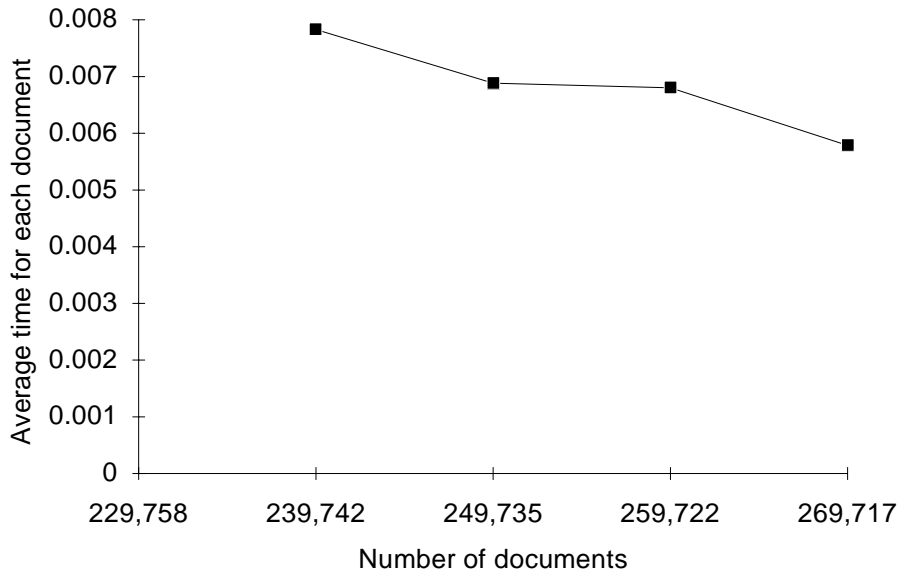


Fig. 6 The change in the average time per document for  $C^2ICM$  algorithm on large MARIAN part 2

In figure 5 and 6, the data in table 28 is summarized. Average time per document while using  $C^3M$  increases as the number of documents increases. When the database size is 444,651 it should be at least  $0.093159 \times 444651 = 41,423.24$ . Because the average time per document increases as the database grows. Experimentally we found this number as 66,565.7. This amount is also greater than the amount calculated if we consider the increase ratio as linear.

**Table 32: Average time per document for incremental and reclustering algorithms for large MARIAN Part 2**

m	$t_{rec}$	$t_{inc}$	$t_{rec}/m$	$t_{inc}/m$
399,552	55,045.9	0.0	0.137769	
409,552	57,499.2	3,958.0	0.140395	0.009664
419,548	60,098.2	3,351.3	0.143245	0.007988
429,548	62,739.1	3,613.6	0.146058	0.008412
439,548	65,279.2	3,728.0	0.148514	0.008481
444,651	66,565.7	2,213.8	0.149703	0.004979

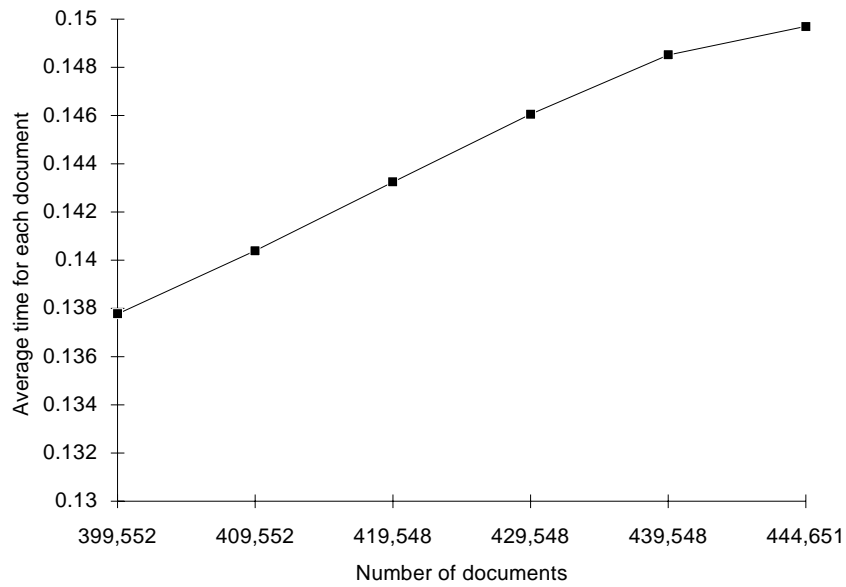


Fig. 7 The change in average time per document for  $C^3M$  algorithm on large MARIAN part 2

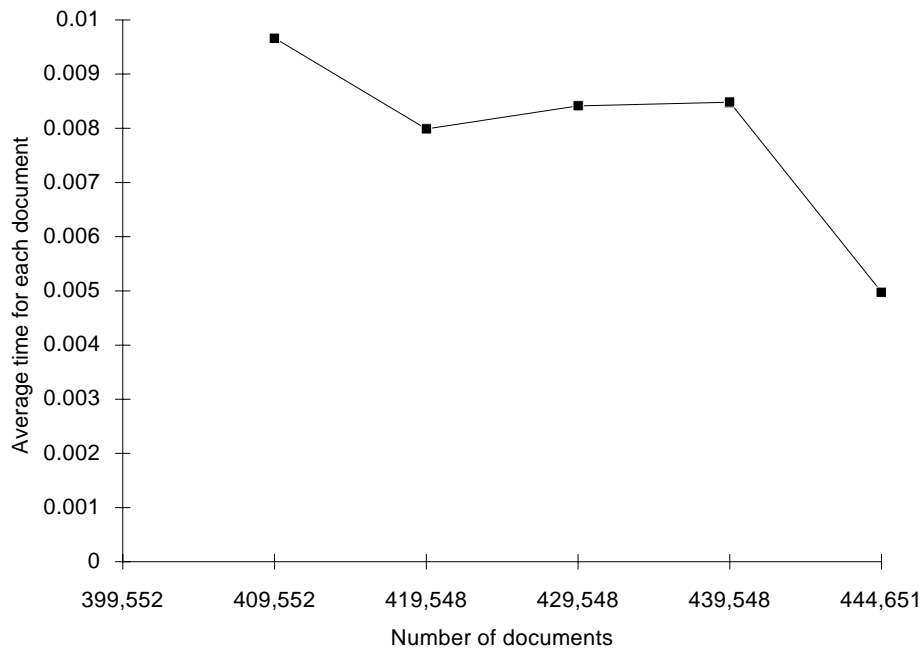


Fig. 8 The change in average time per document for  $C^2ICM$  algorithm on large MARIAN part 2

For  $C^2ICM$ , the average time per document may be at most 0.005785. So time estimation for a database with 444,651 documents is 2,572.3. Experimentally we found this as 2,213.8. So we observed that  $C^2ICM$  performs better than  $C^3M$  and these savings increase as the database grows.

Using the data in table 32, we can estimate the execution time of both algorithms for a database of 1,000,000 documents. For C<sup>3</sup>M, the average time per document always increase so the average time per document may be at least 0.149703. The execution time may be at least 0.149703x1,000,000=149,703 secs = 41,5 hr's. For C<sup>2</sup>ICM, the average time per document decreases, so the average time per document may be at most ( in the worst case ) 0.004979x1,000,000 = 4,979 secs = 1,38 hr's. When C<sup>2</sup>ICM is used, the saving is about 40 hours.

## 6-Effects of database dynamics on the output of the algorithm

In this section, we try to investigate the effects of database dynamics on the output of the algorithm. Database dynamics means the change in m and n, the number of documents and terms used respectively. In [2], new variables for estimating the effects of database are defined. They are  $n_{c,k}$  and  $n_k$  indicate the number of clusters and number of terms at kth database increment (  $k=0$  indicates the initial database ). The following formula is derived in the same paper.

$$n_{c,k} = \left( \frac{n_k}{n_{k-1}} \right) \cdot n_{c,k-1} \quad \text{for } k > 0$$

That is the number of clusters at the kth step can be estimated using the number of terms used in kth step, (k-1)th step and number of clusters at (k-1)th step. The formula shows an important point. As we increase the number of terms used,  $n_k$ , the number of clusters,  $n_{c,k}$ , increases. The number of clusters grow as new terms are added to the database, but its rate of growth decreases as the database size increases.

In tables 33 and 34, we tried to find experimental values of the above formulae.  $n_k$  is the number of terms used.  $n_{c,k}(\text{actual})$  is the experimental number of clusters at each step.  $n_{c,k}(\text{formula})$  is the number obtained from the above formula. For the first part of large MARIAN, theoretical results are near to the experimental results. There is at most %0.4 error. That is theoretical results differ from experimental ones at most %0.4. But for the second part, they differ very much such as %2.5. The estimation depends on the increase in the number of terms used. If the new coming documents use the existing terms mostly, the estimation becomes more accurate.

**Table 33: Number of cluster estimation for large MARIAN part 1**

m	n <sub>k</sub>	n <sub>c,k</sub> (actual)	n <sub>c,k</sub> (formula)
229,758	151,380	12,678	-
239,742	155,338	13,024	13,009
249,735	158,971	13,341	13,329
259,722	162,397	13,609	13,629
269,717	165,454	13,924	13,865

**Table 34: Number of cluster estimation for large MARIAN part 2**

m	n <sub>k</sub>	n <sub>c,k</sub> (actual)	n <sub>c,k</sub> (formula)
399,552	240,941	16,591	-
409,552	249,977	16,783	17,213
419,548	252,369	16,973	16,944
429,548	257,510	17,144	17,319
439,548	263,600	17,327	17,549
444,651	266,691	17,422	17,530

## 7-Relationship between clustering and indexing

The Cover-Coefficient based indexing clustering relationships are formulated in [2] as follows.

$$n_c = \frac{t}{x_d \cdot t_g} = \frac{m \cdot n}{t} = \frac{m}{t_g} = \frac{n}{x_d} \quad \text{and} \quad d_c = \frac{m}{n_c} = t_g$$

It means that number of clusters can be estimated ( or calculated) using the depth of indexing ( $x_d$ ), term generality ( $t_g$ ) and the number of non-zero entries in D-matrix ( $t$ ). When we are doing experiments with large MARIAN, we tried to look for the relationships between these entities. We summarize our experimental results in tables 35-36. In these tables, we showed term generality, standard deviation for term generality, coefficient of variation which is defined as (standard deviation / average). We gave the values of same parameters for  $d_c$ , which is the average cluster size. There are several formulae for standard deviation. We used the following formula:

$$\sqrt{\frac{\sum (x_i - \mu)^2}{n}}$$

where  $x_i$  is the numbers and  $\mu$  is the average of these numbers.

We prepared these tables in the hope that coefficient of variations for both  $t_g$  and  $d_c$  are similar to each other. But the standard deviation in  $t_g$  became very large. This is because of some terms that are used approximately by half of the documents in database. That means

some terms are used in 150,000 documents. But the averages for both variables are very near to each other. These results validate the theoretical results.

We observed that the average cluster sizes are same for both incremental and reclustering algorithms. But standard deviation indicates that cluster sizes are more different in the results of incremental clustering algorithm.

**Table 35: Averages for term generality and cluster size with reclustering algorithm**

m	avg. $t_g$	std. $t_g$	coeff. of var. for $t_g$	avg. $d_c$	std. $d_c$	coeff. of var. for $d_c$
399,552	23.35	713.71	30.57	24.08	46.42	1.93
409,552	23.58	727.30	30.84	24.40	45.43	1.86
419,548	23.84	741.67	31.11	24.72	45.69	1.85
429,548	24.12	756.36	31.36	25.05	45.84	1.83
439,548	24.32	768.98	31.62	25.37	47.17	1.86
444,651	24.42	775.45	31.76	25.52	47.24	1.85

**Table 36: Averages for term generality and cluster size with incremental clustering algorithm**

m	avg. $t_g$	std. $t_g$	coeff. of var. for $t_g$	avg. $d_c$	std. $d_c$	coeff. of var. for $d_c$
399,552	23.35	713.71	30.57	-	-	-
409,552	23.58	727.30	30.84	24.40	46.85	1.92
419,548	23.84	741.67	31.11	24.72	48.14	1.95
429,548	24.12	756.36	31.36	25.05	49.43	1.97
439,548	24.32	768.98	31.62	25.37	50.14	1.98
444,651	24.42	775.48	31.76	25.52	50.68	1.99

## 8- Validation of the Complexity of the Algorithm

In [2] and [3], the complexity of  $C^2ICM$  and  $C^3M$  algorithms are given. The complexity of  $C^3M$  algorithm is given as  $O(m.x_d.t_{gs})$  where  $m$  is the number of documents used,  $x_d$  is the depth of indexing, and  $t_{gs}$  is average number of seed documents per term. Also  $t_{gs}$  accounts only for terms having a generality of two or more. We calculated  $t_{gs}$  as the terms having more than one occurrence in the database ( which is denoted as  $n_2$  in table 37) divides the number of entries in the inverted index for seed documents, which is denoted as (# of e iisd) in table 37. These parameters are summarized in tables 37 and 38 for large MARIAN part 1 and 2, respectively.

Since the execution time is proportional to  $m.x_d.t_{gs}$ , we tried to find a constant, which is called  $r$ , where  $r=(m.x_d.t_{gs})/Execution\ time$ . The numbers for large MARIAN part 1 is given in table 39. Here  $r$  is changing for each database size. But the difference is not so big. We can

consider it as a constant. The same discussion can be made for large MARIAN part 2 (table 41).

The complexity of one incremental step of  $C^2$ ICM algorithm is given as  $O(x_d.(m_1+m'))$  in [2]. Here  $m_1$  is the previous size of the database,  $m'$  is the size of added database,  $x_d$  is depth of indexing. But a constant cannot be obtained. In our environment, experimental analysis for  $C^2$ ICM is a difficult task to deal with. Because first step takes more time than the following steps. So first step in tables 40 and 42 cannot be used as criterion. In these tables, a constant seems after the first increment.

**Table 37: Parameters of large MARIAN Part 1 needed for analyzing execution times**

m	229,758	239,742	249,735	259,722	269,717
$n_2$	65,970	67,767	69,508	71,262	72,572
# of e iisd	322,009	331,571	340,073	348,070	355,634
$t_{gs}$	4.88	4.89	4.89	4.88	4.90

**Table 38: Parameters of large MARIAN Part 2 needed for analyzing execution times**

m	399,552	409,552	419,548	429,548	439,548	444,651
$n_2$	104,764	107,265	109,628	112,026	114,614	115,888
# of e iisd	545,732	560,263	571,964	583,973	600,149	608,102
$t_{gs}$	5.21	5.22	5.22	5.21	5.24	5.25

**Table 39: Analysis of the execution times of large MARIAN part 1 for reclustering**

m	229,758	239,742	249,735	259,722	269,717
exec time	19,619.5	21,032.2	22,381.0	23,835.3	25,126.6
$t_{gs}$	4.88	4.89	4.89	4.88	4.9
$x_d$	12.18	12.16	12.14	12.15	12.08
r	696.07	677.80	662.41	646.08	635.39

**Table 40: Analysis of the execution times of large MARIAN part 1 for incremental clustering**

$m^*$	229,758	239,742	249,735	259,722	269,717
exec time	0	1,877.1	1,719.1	1,766.4	1,560.4
$x_d$	12.18	12.16	12.14	12.15	12.08
r	0	1,553.07	1,763.59	1,786.47	2,088.04

\*For a given step, m in table= previous size of database + size of added database

**Table 41: Analysis of the execution times of large MARIAN part 2 for reclustering**

m	399,552	409,552	419,548	429,548	439,548	444,651
exec. time	55,045.9	57,499.2	60,098.2	62,739.1	65,279.2	66,565.7
t <sub>gs</sub>	5.21	5.22	5.22	5.21	5.24	5.25
x <sub>d</sub>	14.08	14.22	14.34	14.46	14.59	14.65
r	532.46	528.71	522.56	515.80	514.78	513.77

**Table 42: Analysis of the execution times of large MARIAN part 2 for incremental clustering**

m*	399,552	409,552	419,548	429,548	439,548	444,651
exec time	0	3,958.0	3,351.3	3,613.6	3,728.0	2,213.8
x <sub>d</sub>	14.08	14.22	14.34	14.46	14.59	14.65
r	0	1,471.41	1,795.22	1,718.86	1,720.23	2,942.51

\*For a given step, m in table = previous size of database + size of added database

## 9- Conclusion and Future Research

Clustering is an important tool increasing the efficiency of retrieval from larger databases. For clustering, Cover-Coefficient based Clustering Methodology (C<sup>3</sup>M) is introduced in [3] for increasing efficiency. Then due to the dynamic nature of databases, deletion and addition of documents, Cover-Coefficient Incremental Clustering Methodology (C<sup>2</sup>ICM) was proposed in [2]. An implementation of both algorithms has been done and results on INSPEC and small MARIAN are given in [6]. In this project, we extended this implementation for running on SUN environment. The experiments were done on a database of 444,651 documents called large MARIAN. Initial conversion of this database into the format required by the previous implementation has been done. Due to the restrictions explained in the report, we could not try the intended database size. If the conversion program can be written again using another data structures, which is explained in appendix C, the whole database can be obtained and experimented with.

The execution time of C<sup>3</sup>M and C<sup>2</sup>ICM algorithms are compared. It is proven that C<sup>2</sup>ICM is more faster than C<sup>3</sup>M for larger databases. Also, some theoretical results that are given in [2] and [3] are tried to be proved experimentally. Most of the experimental results are consistent with the theoretical results.

The next step in this direction may be comparing the validity of clustering structure that are generated by C<sup>2</sup>ICM and C<sup>3</sup>M algorithms. This can be done by designing queries and evaluating the effects of clustering structure on the results of these queries.

# ACKNOWLEDGMENTS

To Dr. Edward Fox of Computer Science Department and the Computing Center at Virginia Tech, for his work with the MARIAN database.

To Robert France of the Computing Center at Virginia Tech, for work with the MARIAN database and his assistance with my experiments which used it.

To Cory Snively of Beacon Graphics corporation for his assistance with my experiments.

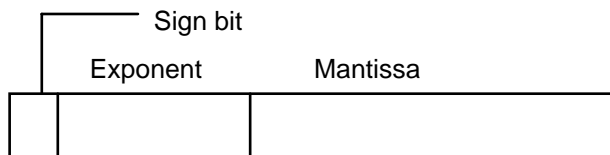
## REFERENCES

- [1] Atkinson, K.E. "An introduction to Numerical Analysis." John Wiley&Sons Inc., New York, 1978.
- [2] Can, F. "Incremental Clustering for Dynamic Information Processing." *ACM Transactions on Information Systems (TOIS)*. Vol 11, No 2 (April 1993), pp 143–164
- [3] Can, F. and Ozkarahan, E.A. "Concepts and Effectiveness of the Cover Coefficient-based Clustering Methodology for Text Databases." *ACM Transactions on Database Systems (TODS)*. Vol 15, No 4 (December, 1990), pp 483–517.
- [4] Can, F., Snaveley, C.D., Fox, E.A., France, R.K., "Incremental Clustering for Very Large Document Databases Initial MARIAN experience." In *Proceedings of the 8th International Symposium on Computer and Information Sciences ISCIS VIII* (Istanbul, Turkey, November). Istanbul 1993.
- [5] Fox, E.A., France, R.K., Sahne, E., Daoud, A., &Cline, B.E. "Development of a Modern OPAC:From REVTOLC to MARIAN." In *Proceedings of the 16th Annual International ACM SIGIR Conference* (Pittsburgh, PA, June). ACM, New York 1993, pp 248–259.
- [6] Snaveley, C. "Implementation of a Cover Coefficient-Based Incremental Clustering Algorithm for Very Large Document Databases." Technical Report, SAN Departmental Honors Program, Miami Univ. of Ohio, Dec. 16, 1992.

# Appendix A

## Effects of Number Representation

When we are trying to calculate the number of clusters, we should add all decoupling values smaller than a threshold value. But during the summation of these values, we obtained different results from the previous results. After some experiments, we found that this was because of real number representation of decoupling values. During the summation, we add numbers between 0 and 1 to an increasing sum. After some time, we found ourselves adding small numbers such as 0.5 to 2000. While summing up this kind of numbers, rounding is a problem. Also for adding to numbers, the exponents should be equaled. So for equaling the exponents, the small numbers should be shifted each time. When we normalize the number, we lose many significant digits. If the precision is small, this difference become apparent very quickly. The representation of real numbers can be given in the following way:



Here mantissa is the normalized representation of the real number. Normalized means there is no 0 after the decimal point. For example,  $0.00025 \times 10^{-7}$  is represented as

sign bit = 1    exponent = -10    mantissa = 25.

If a number  $z$  is represented in floating-point form, then it is stated essentially in the form  $z = (\text{sign } z) \cdot (a_1 a_2 a_3 \dots a_n)_b \cdot b^e$  the characters  $a_i$ 's are all digits in the base  $b$  system,  $0 \leq a_i \leq b-1$  with  $a_1 \neq 0$ . The number  $e$  is called the *exponent* or *characteristic*; and the number  $.a_1 a_2 a_3 \dots a_n$  is called *mantissa*. The mantissa contains  $n$  digits in the base  $b$  system. All numbers that are longer must be shortened to this length in some way, thus limiting the possible precision of any calculation.

When working with only a limited number of digits in a number, as on a computer, rounding errors are inevitable with most arithmetic calculations. Assume a real number is given in the following way

$$z = s \cdot (a_1 a_2 a_3 \dots a_n)_b \cdot b^e, \quad s = \pm 1, \quad a_1 \neq 0$$

That is  $s$  is the sign of  $z$ . Then rounded machine version of  $z$  is

$$fl(z) = \begin{cases} \sigma(a_1 a_2 \dots a_n)_\beta \cdot \beta^e & 0 \leq a_{n+1} < \frac{\beta}{2} \\ \sigma[(a_1 a_2 \dots a_n)_\beta + (.00\dots 01)_\beta] \beta^e & \frac{\beta}{2} \leq a_{n+1} \leq \beta \end{cases}$$

in which  $(.00\dots 01)$  means  $\beta^{-n}$ . We round up if the  $(n+1)$ st digit is greater than or equal to  $\frac{\beta}{2}$  and we round down if it is less than  $\frac{\beta}{2}$ .

For the error in  $fl(z)$ , first assume  $0 \leq a_{n+1} < \frac{\beta}{2}$

Error in  $z = z - fl(z) = s(.00\dots 0a_{n+1}a_{n+2}\dots)_\beta \times b^e = s(a_{n+1}a_{n+2}\dots)_\beta \times b^{e-n}$  so

$$|z - fl(z)| \leq \frac{1}{2} \beta^{e-n}$$

By a similar argument, the same result is true for  $\frac{\beta}{2} \leq a_{n+1} \leq \beta$ . To obtain a convenient form, let  $\epsilon$  be defined by  $\frac{z - fl(z)}{z} = -\epsilon$  where  $|\epsilon|$  is the relative error. Then we can write as  $fl(z) = (1 + \epsilon)z$   $|\epsilon| \leq \frac{1}{2} \beta^{-n+1}$ . This form is much used in the analysis of the propagation of rounding errors.

In the light of the above definitions, we analyze the computation of the sum

$$S = \sum_{i=1}^m x_i$$

In a computer with  $x_1, \dots, x_m$  in floating-form that is  $x_i = fl(x_i)$   $i = 1..m$ . Define  $S_i$  as

$S_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \epsilon_2)$  with  $|\epsilon_2| \leq \frac{\beta}{2} \beta^{-n}$  if numbers are rounded. Define recursively

$$S_{r+1} = fl(S_r + x_{r+1}) \quad r = 1, 2, \dots, m-1$$

$$\text{Then } S_{r+1} = fl(S_r + x_{r+1})(1 + \epsilon_{r+1}) \quad |\epsilon_{r+1}| \leq \frac{\beta}{2} \beta^{-n}$$

Expanding the first few sums, we obtain the following

$$\begin{aligned}
S_2 - (x_1 + x_2) &= \varepsilon_2(x_1 + x_2) \\
S_3 - (x_1 + x_2 + x_3) &= (x_1 + x_2)\varepsilon_2 + (x_1 + x_2)(1 + \varepsilon_2)\varepsilon_3 + x_3\varepsilon_3 \\
&\approx (x_1 + x_2)\varepsilon_2 + (x_1 + x_2 + x_3)\varepsilon_3 \\
S_3 - (x_1 + x_2 + x_3 + x_4) &\approx (x_1 + x_2)\varepsilon_2 + (x_1 + x_2 + x_3)\varepsilon_3 \\
&\quad + (x_1 + x_2 + x_3 + x_4)\varepsilon_4
\end{aligned}$$

We have neglected cross-product terms  $\varepsilon_i\varepsilon_j$  since they will be of much smaller magnitude.

By induction, we obtain

$$\begin{aligned}
S_m - \sum_{i=1}^m x_i &\approx (x_1 + x_2)\varepsilon_2 + \dots + (x_1 + x_2 + \dots + x_m)\varepsilon_m \\
&= x_1(\varepsilon_2 + \varepsilon_3 + \dots + \varepsilon_m) + x_2(\varepsilon_2 + \varepsilon_3 + \dots + \varepsilon_m) \\
&\quad + x_3(\varepsilon_3 + \varepsilon_4 + \dots + \varepsilon_m) + \dots + x_m\varepsilon_m
\end{aligned}$$

So the error in summation can be given in this way. In our experiment, all  $x_i$ 's are near to each other that is  $x_1 \approx x_2 \approx \dots \approx x_m \approx x$  and all error values are also near to each other, i.e.  $\varepsilon_1 \approx \varepsilon_2 \approx \dots \approx \varepsilon_m \approx \varepsilon$ . So if we substitute these values, we get

$$\begin{aligned}
S_m - \sum_{i=1}^m x_i &\approx x(\varepsilon + \varepsilon + \dots + \varepsilon) + x(\varepsilon + \varepsilon + \dots + \varepsilon) \\
&\quad + x(\varepsilon + \varepsilon + \dots + \varepsilon) + \dots + x\varepsilon \\
&= x\varepsilon((m-1) + (m-1) + (m-2) + \dots + 1) \\
&= x\varepsilon(m-1 + \sum_{i=1}^{m-1} i) \\
&= x\varepsilon(m-1 + \frac{(m-1)m}{2})
\end{aligned}$$

In our experiments, we know  $0 \leq x \leq 1$ ,  $|\varepsilon| \leq \frac{2}{2} 2^{-22} = 2^{-22}$  and  $m=42,815$ . Here a typical value for  $x$  is the average of decoupling values, that is  $5,240/42,815=0.12$ .  $\varepsilon = 2^{-22}$  in the worst case, we get  $0.12 * 2^{-22} * (42,814 + (42814 * 42815)/2) = 26.75$ . Experimentally we found this error as 21. The above value is an upper bound for the error. Most of the above analysis are taken from [1].

# Appendix B

## Interpretation of Marian files in MARIAN database

MARIAN is a library database. It consists of documents which can be a book, periodical, manual, government documents etc. Each document contains many fields, there are indexes on 4 of them. The data we used are taken using these indexes. These fields are

Title : Title of the document ex. 'Information Retrieval'.

Note : Some note field about the document ex. 'Electronic devices meeting'.

Subject : Subjects of the documents which are given for each document

document 1 : Computer Science, Information Retrieval

document 2 : Computer Science, Computer languages, Prolog

Author : Author(s) of the file

document 1 : Peter Ustinov, Peter Selimov

document 2 : Ali Yilmaz, Ahmet Yilmaz

The database is partitioned according to these above fields. There are 10 files

occursInNote.sort.full  
occursInTitle.sort.full  
occursInSubject.sort.full  
occursInPersAuthor.sort.full  
occursInConfAuthor.sort.full  
occursInCorpAuthor.sort.full  
Subject.link.full  
PersAuthor.link.full  
ConfAuthor.link.full  
CorpAuthor.link.full

The format of first 4 files are as the following:

TermClass<tab>TermId<tab>Recordnumber<tab>Weight<tab>Evidence

Evidence is the field which we can ignore for our case.

In occursInNote and occursInTitle files record number corresponds to actual document number. During the creation of files, each word in titles and notes is assigned a term class and term id and the appearance of each word is written into the corresponding file (\*Title.sort.full or \*Note.sort.full) with its associated number of occurrences.

In occursInSubject file, record number corresponds to distinct subject name. Here each subject name takes a record number. For example 'computer science' is given record number 204. The terms that are appearing in this record are given with their associated weights. We mean there may be more than one line in occursInSubject file whose record number is 204. Record number 204 contains 'computer' once and 'science' once. There may be records which contain each word more than once. For example record 'computer science and computer languages' (record number 205). According to this explanation we can wait the following lines appear in occursInSubject file.

```
101:233445 204 1 ( for word 'computer')  
101:233446 204 1 ( for word 'science' )  
101:233445 205 2 ( for word 'computer')  
101:233446 205 1 ( for word 'science' )  
101:233447 205 1 ( for word 'languages')
```

For occursInSubject file we can find the links to actual document numbers using the file Subject.link.full. In \*.link.full files, the tuple format is the following,

```
documentnumber<tab>recordnumber
```

The record numbers in occursInSubject file corresponds to record numbers in Subject.link.full. So this means that one document can have more than one subject records and one record can be contained by more than one documents. We should find the documents that contain any given record and insert the document numbers into occursInSubject file and obtain a file that are compatible with other files.

In occursIn\*Author file, record number corresponds to distinct corporate, conference or personal author. Again each full author name is given a record number. In the following example, 'Peter Ustinov' is record 123. 'Peter Selimov' is record number 124. So the following lines should appear in the occursInPersAuthor file

```
2:40530 123 1 ( for 'Peter')  
2:40531 123 1 ( for 'Ustinov')  
2:40530 124 1 ( for 'Peter')  
2:40532 124 1 ( for 'Selimov')
```

Also a group of authors may be in the same record. A book may be written more than one author. For example 'Ali Coplu, Ali Karci, Ali Sirin, Ahmet Yilmaz, Ahmet Cetin'. So weights in these author files may be more than 1. Maximum weight observed in these files is 6. This

amount can be allowable in these files. Another example for authors from actual database is "*Augustine of Selima, the Younger. Also called 'the Arbiter' and 'Augustinius Doradus'*".

The records for author names were partitioned into 3 parts in order to classify the authors. Corporate authors are names of corporations such as 'IBM, Thinking Machines Corporation'. Conference authors are the names of conferences. Personal author is the author of any kind of document. Also here these three files cannot share common records. But they can share terms. Here term means author name. For example 'Peter'. It is possible that a corporate author record contains 'Peter' 3 times and personal author record contains 'Peter' 4 times. So after inserting document numbers instead of record numbers occursInPersAuthor and occursInCorpAuthor can contain the lines

2:40530 456 3 in occursInCorpAuthor

2:40530 456 4 in occursInPersAuthor

The same argument about the \*.link.full files is valid for author files.

Each field is actually a collection of possible fields. For instance, author includes performers, editors, and authors of individual parts of collections as well as the main author of the work. Notes include summaries, contents listings, and manuscript descriptions, as well as simple notes like "includes bibliography". An example record from the database may clarify the points;

FIELD NAME	VALUE
=====	=====
CALL NUMBER	: TK7800 .1585
AUTHOR	: International Electron Devices Meeting.
TITLE	: Technical digest/ International Devices Meeting
OTHER TITLE	: I.E.D.M. technical digest
OTHER TITLE	: IEDM technical digest
IMPRINT	: New York:Institute of Electrical and Electronics Engineering
DESCRIPTION	: v.:ill.;28 cm.
FREQUENCY	: Annual
PUBLISHED	: 1973-
SUPPLEMENTS	: Supplements accompany some years.
NOTE	: Sponsored by: IEEE Group on Electron Devices, 1973-1975
SUBJECT	: Electronic apparatus and appliances--Congresses.
ADDED ENTRY:	Taffanel, Claude Paul, 1844--1908.
ADDED ENTRY:	IEEE Group on Electron Devices.
ADDED ENTRY:	IEEE Electrical Devices Society.
NOTE	: International Electronic Devices Meeting.

The added entries are all authors. This record also has three titles, two notes and a compound subject.

In this database, there exists some documents that do not contain any terms. These documents are government documents and they are not fully analyzed for some reasons. The clustering programs handle these documents successfully. They are considered as deleted documents.

# Appendix C

## Conversion of Marian files in MARIAN database

At the beginning, there are 10 files in native format. The names of the files are the following:

occursInNote.sort.full  
occursInTitle.sort.full  
occursInSubject.sort.full  
occursInPersAuthor.sort.full  
occursInConfAuthor.sort.full  
occursInCorpAuthor.sort.full  
Subject.link.full  
PersAuthor.link.full  
ConfAuthor.link.full  
CorpAuthor.link.full

The native format of the MARIAN database is a tuple format. Each tuple is a term with the record number that contains this term. For fields NOTE, TITLE and SUBJECT there is a corresponding file (occursInNote.sort.full, occursInTitle.sort.full, and occursInSubject.sort.full). For the AUTHOR field, there are three files in this version (occursInPersAuthor.sort.full, occursInConfAuthor.sort.full and occursInCorpAuthor.sort.full). Here extension *sort.full* means that these files are sorted according to term class and term id as major and minor respectively. These 6 files are with the form of following tuple:

TermClass<tab>TermId<tab>Recordnumber<tab>Weight<tab>Evidence

Evidence is the field which we can ignore for our purposes. The term class and term ID together uniquely identifies a term.

In the files for TITLE and NOTE fields, record number corresponds to actual document number that contain this term. In other files, the record number corresponds to a distinct author name or subject. To convert these files to same format with other files, we should use other four files (Subject.link.full, PersAuthor.link.full, ConfAuthor.link.full, CorpAuthor.link.full). They are in the following format:

documentnumber<tab>recordnumber

The document number corresponds to actual document number, and the record number corresponds to the record number in the corresponding occursIn\* file. In order to have four files in the same format with actual documents numbers, we used the many-to-many mappings supplied by \*.link.full files. To obtain a document vector information for a document

x, we should find all tuples of the form x<TAB>r in \*.link.full file. Then for each r found, find all tuples of the form TermClass<tab>TermId<tab>r<tab>Weight in \*.sort.full files.

After making these operations, the files should be merged somehow in order to convert data in the format required by the programs. To do this, a four-way merge is used. Also the input files should be sorted according to document numbers.

Instead of maintaining the supplied term numbering structure, the terms are renumbered as single integers starting from 1 as they encountered. The four-way merge program builds a linked list for each term class that contains old term ID numbers and their assigned integers. This caused many problems during the conversion. We used the previous four-way merge program. For a database with this much documents, this implementation executes very slowly. This is because of searching time. When a new term with its class and ID comes from the input files, the program should search the linked list of this term class whether it is given a number or not. If not found give it a number and insert it into the linked list. This search and addition occurs in  $O(n)$  time. When the number of entries increase, the search time increases. If another data structure such as heap with  $O(\log n)$  search and addition time is used, the merge operation can be done in shorter time.

The entire conversion can be enumerated with the following steps:

1 - Convert the tab-delimited tuples to space-delimited tuples in all files. This is accomplished via CONVERT.C program in SUN environment.

2 - Change the format of the occursInSubject.sort.full and occursIn\*Author.sort.full with the corresponding \*.link.full files. This is accomplished via the EXPAND program.

3 - Sort the all resulting 6 files using the external sorting routine of the operating system. But the files are very large. The largest file is with 7,500,000 tuples. So OS sorting routine could not sort these files. In order to sort these files, divide the files into 100,000 tuples files. Then sort these files and then merge the files again. These operations are accomplished via the programs DIVIDE.P and MERGE.P. Also eliminate the tuples that contain the same term class, term ID and document number with MERGE.P. This situation is possible. The reason is explained in appendix B.

4 - Merge the author files. There is no other conversion needed because they have the same meaning. Also during the merge operation, eliminate the tuples having the same term class,

term ID and document number. In this case, we only add the weight of these tuples and write them to output only once. This is accomplished by MERGEAUTHOR.P program.

5 - Merge the four resulting files corresponding to each field and renumber the terms as they are encountered. This is accomplished via MARIAN.P program.

The output from this procedure will be the document vector and pointer files for the MARIAN database and a term mapping file which contains tuples of the following format:

```
termnumber<SPACE>termclass<SPACE>termID
```

This mapping can be used for some purpose, if desired.

# Appendix D

## Modifications on the programs

1- First problem encountered during the conversion is the 'seek' function SunOS Pascal does not support the 'seek' function which provides random accessing to a binary file. We handled this problem by calling an external function from C language library. But at each call to this function, pointer conversion from Pascal file pointer to C file pointer is needed. This slows down the execution time.

2- The second change is the representation of real numbers, we changed the single precision real variable definitions to double precision real variables. These are SHORTREAL and REAL in SunOS Pascal, respectively.

3- Other problem is with 'get' procedure in SunOS Pascal. After some debugging, we found that when we make a seek to a binary file, one record is retrieved into the file buffer area. This is used in the program when terms of a document is transferred into memory. In IBM Pascal, the program makes a 'seek' function call then calls 'get' function in order to get the first record that is pointed by file pointer. But in SunOS Pascal, after calling the 'seek' function, first record that is pointed by file pointer is already in memory. Because of this, we changed the place of 'get' function in the 'while' loops. After making a seek, the program starts a 'while' loop, if first record is available, there is no need to call the 'get' function. So we placed the 'get' function calls at the end of 'while' loops.

4- When the programs are used for reclustering, that is *old\_m*, which is a parameter of the programs, is given as 0, the deleted documents are directly placed in the ragbag cluster. The reason is that if *old\_m* is 0, then the procedure *recluster\_falsified* is not called and new documents, in this case the whole database, are clustered if they are not seed documents. The coverages of seed documents are 0 and this kind of documents are placed in the ragbag cluster. But when *old\_m* is greater than 0, means incremental clustering, the documents are checked whether they are empty or not. If they are empty, they are discarded directly, and do not appear anywhere in the clustering structure. The previous programs count these empty documents as falsified documents. Also when calculating the average depth of indexing ( $x_d$ ), these documents were considered. We changed this. Only non-empty documents are considered when calculating the  $x_d$ .