多磁碟系統中動態檔案之配置方法
A Dynamic Allocating Approach for Dynamic Files on Multi-disk Systems*

陳良安　　錢炳全　　陳建源
Liang-An Chen, Been-Chian Chien and Chien-Yuan Chen

義守大學資訊工程系
Department of Information Engineering
I Shou University,
1,Section 1, Hsueh-Cheng RD. Ta-Hsu Hsiang,
Kaohsiung, Taiwan, R.O.C.
Email:cbc@csa500.isu.edu.tw

摘要

本篇論文中將針對多磁碟系統中之動態的多重屬性檔案配置提出解決策略。我們的方法是以部份重新配置的方式來達到減少磁碟存取次數的目標。根據實驗顯示動態的部份重新配置策略在多磁碟的動態檔案上確實可以比靜態演算法減少不必要的磁碟存取次數，達到增進效益的目的。

關鍵字：資料庫，多磁碟，平行存取，部份符合查詢。

Abstract

In this paper, we are concerned with the problem of allocating records in a dynamic file among multiple disks. We attempt to construct a dynamic approach to avoid reallocating all existed records in a dynamic file and obtain less number of concurrent disk accesses than that of traditional static allocation methods. We find that our approach improves the performance of concurrent disk accesses indeed by our experiments. The experimental results show that our partial reallocation method can obtain a good improvement under reasonable extra reallocation cost.

Keywords: database, multi-disk, strictly optimal, parallel access, partial matching queries.

1. Introduction

Owing to the fast evolution of hardware and the upgrading of parallel techniques, the cost of hardware has been greatly decreased. The applications of multi-disk systems and disk array have greatly grown. The related researches become more important. In this paper, we consider the problem of allocating multi-attribute records in a dynamic file among multiple disks. About the problem of multi-attribute allocation, approximately, there are two main themes to be discussed. The first theme is that:

Given a set of data records in a database system, the data records can be arranged into buckets in such a way that the average number of buckets to be accessed is reduced. The second one is that:

Given a set of buckets, the buckets are allocated to the different disks appropriately, so that the buckets can be accessed simultaneously for reducing the maximum number of accesses among the disks.

The emphasis of traditional researches focuses on the first theme. The second theme was proposed by Du and Sobolewski in 1982[11]. According to Du and Sobolewski's work, there were many allocation methods proposed for solving the multi-disk allocation problem[1][4][5][9]. Most of the presented methods focus on the Cartesian product files. The Cartesian product file for partial matching queries on the multiple-disk system was first shown in [11] .

It is effective for partial matching queries if a Cartesian product file can be stored properly in advance. There are many multiple-disk allocation methods for allocating Cartesian product files. Some of these allocation methods are called **strictly optimal** when the conditions for strict optimality are satisfied. An allocation method is said to be strictly optimal if the maximum of $\lceil n/m \rceil$ buckets need to be accessed on any one of m independently accessible disks to examine the n buckets in response to a query. For instance, there are 20 buckets totally in a three-disk system, the system is called strictly optimal if the maximum disk accesses is $\lceil 20/3 \rceil = 7$ for any query. The Disk Modulo (DM) method proposed by Du and Sobolewski [11] is strictly optimal for two-disk systems and three-disk systems, the Gray Code (GC) method and Symbolic Gray Code (SGC) method proposed by Chang and Chen [4] [5] are also strictly optimal for two-disk systems.

* This research was partially supported by National Science Council, Taiwan, Republic of China, under contract NSC 86-2621-E-214-005-T(1996).

Furthermore, Chang, Lee and Du [6] showed that it is very effective if a Cartesian product file can be allocated properly into multi-attribute files in advance. By Cartesian product files, we can arrange the records into buckets in such a way that the maximum number of buckets to be examined, over all possible partial matching queries, is minimized.

However, the traditional allocation methods proposed before are strictly optimal to assign records in a static file but not in a dynamic file. In this paper, we will try to construct a new partial reallocation method to avoid reallocating all existed records in a dynamic file. The experimental results show that the performance of our method is better than the DM method even though extra reallocation cost in considered.

This paper is organized as follows. Section 2 presents the comparisons between static files and dynamic files according the features of dynamic files. In Section 3, we introduce new dynamic allocation method. Section 4 shows the experimental results of our method. Finally, some future works for dynamic allocation approaches are outlined.

2. Static Files versus Dynamic Files

A **static file** is that the domains of every attribute and the records in the file are known in advance. However, the applications usually do not have fixed domains. The records can not be inserted at a time. The domains of attributes will increase by inserting new records.

So, a **dynamic file** is defined to be a file whose records can be inserted or deleted dynamically.

Since the records will not be known in dynamic files in advance, the features of dynamic files are different from static files. The features of a static file are listed as follows:

(1) A file contains multi-attribute records which have fixed number of attributes.
(2) The domains of attributes are predefined.
(3) The total records in a static file are known in advance.

Compared with the features of a static file, the features of a dynamic file are as follows:

(1) The domain of each attribute may increase when a record is inserted.
(2) The total number of records is unknown.
(3) The inserting sequences of records are random.

Therefore, a traditional allocation method is strictly optimal in static files, but it is not in dynamic files. Now we take the following example to show that.

Example 1: Let F_1 (A_1, A_2, A_3) be a static file. The domain size of each attribute is $A_1=4$, $A_2=3$, and $A_3=2$. There are 24 records in the file. The records are arranged into four independently accessible disks by a traditional allocation method, as Figure 1(a).

Another dynamic file F_2 (A'_1, A'_2, A'_3) with three attributes. The size of the domain is not specified in advance. Of course, the maximal numbers of records are unknown. Now the following six records are inserted by the sequence of number order ascendantly.

(1) (0, 0, 0),
(2) (1, 1, 1),
(3) (2, 2, 0),
(4) (1, 2, 0),
(5) (2, 1, 0),
(6) (3, 0, 0).

The records are arranged into four independently accessible disks by the same traditional allocation method as Figure 1(b).

The traditional allocation method is strictly optimal in the static file F_1, but it is not in the dynamic file F_2. It is obvious that disk 1 and disk 2 are empty even though the most of records is allocated to disk 3. In such situation, the accesses of the records will not behave as good as the case in Figure 1(a).

According the behavior of a dynamic file, we predict the relationship between the response time and the record number as Figure 2. In Figure 2, the performance of static approaches is worse than the strictly optimal values. If there is an appropriate dynamic approach, the performance should behave between the static approach and strictly optimal values.

3. Our Approach

Since the records in a dynamic file are inserted randomly, the allocated disk of records cannot be the same as a static file. To minimize the number of disk accesses, the traditional allocation method should reallocate all of the records in a file after a new record is inserted each time. However, such complete reallocation is time consuming. Like the dynamic hashing schemes [9][12] partial reallocation must be needed when the dynamic multi-attribute file on multi-disk cannot obtain a good allocation result for some inserted records. There are three considerations while applying the partial reallocating strategy to the multi-disk allocation problem.

(1) When will the record(s) be moved ?
(2) Which record(s) should be moved ?
(3) Where will the record(s) be moved to ?

In this section, we propose an algorithm which will satisfy the three considerations. Thus, the algorithm can handle the dynamic allocation of multi-attribute files. We propose the algorithm as follows:

Algorithm:

Let F be a k-attribute file and let m be the number of independently accessible disk units (labeled $0, 1, \ldots, m-1$).

Input: A bucket B and the upper bound C. Here , B is a bucket $[i_1, i_2, \ldots, i_k]$, in F and C is the upper bound of the maximum difference of the number of disk

accesses for one partial matching query among m disks.

Output: The disk label n.

Step 1: Allocate a new bucket B to the disk by DM method.

Step 2: For one of partial matching queries, RT_1 and RT_2 are the maximum number of disk accesses and the minimum number of disk accesses among the m disks respectively.

If $(RT_1 - RT_2) \geq C$ then goto step 3,

else goto step 1 to allocate another new bucket.

Step 3: Find a bucket B' on the disk with the maximum number of disk accesses.

If $(i_1 + i_2 + \ldots + i_k) \geq (i_1 + i_2 + \ldots + i_k)$

then $B' = [i_1, i_2, \ldots, i_k]$

else $B' = [i_1, i_2, \ldots, i_k]$

Step 4: New disk label n,

$n = \left(\sum_{i=0}^{m-1} D_i \times m_i\right) \bmod m$, where D_i is disk label.

$m_i = 1$, if D_i is the ith disk label with the minimal response time of one partial matching query.

$m_i = m$, otherwise.

Step 5: Move the bucket B' to the new disk n.

Step 6: Repeat step 1 to step 5 till all records in F are already inserted.

Example 2: Here is an example of our algorithm. There is a three-attribute dynamic file and a four-disk system. Assume that the total inserted records are 11 so far. The condition of moving a bucket is that $(RT_1 - RT_2) \geq C = 3$.

The following records are inserted so far:
(0, 0, 0), (0, 1, 1), (1, 0, 1), (2, 1, 1), (1, 0, 2), (2, 1, 2), (0, 1, 0), The part of partial matching queries are shown in Figure 3(c). We can find that the partial matching query, (2, *, *), needs 3 disk accesses at Disk *0*, 1 disk accesses (2, 0, 2), (1, 2, 3), (2, 1, 3).Now a new record, (2, 3, 3), is inserted as shown in Figure 3(a).

at Disk *1*, 1 disk accesses at Disk *2* and 0 disk accesses at Disk *3*. So, RT_1=3 and RT_2=0. Because RT_1-$RT_2 \geq 3$, a bucket on Disk *0* will be moved to Disk *3*. Then the number of disk accesses of partial matching query, (2, *, *), is reduced to 2 as shown in Figure 3(d).

From Example 2, our algorithm is not a strictly optimal allocation method. But, our algorithm can keep up the number of disk accesses "near balanced" among disks. The results are our expectancy.

4. Experiments and Comparisons

The records in dynamic files are randomly generated and assigned to the corresponding disk units by the allocation method. If the unbalanced condition of our algorithm occurs, a bucket is reallocated to the destination disk.

In this section, we will make a comparison among the DM method, our algorithm and the strictly optimal values. Our experiments consider the following cases:

(1) The numbers of independently accessible disks are 3, 4 and 5.

(2) The dynamic file is a multi-attribute file.

(3) The sample numbers of records are 100, 200, 500, 1000, 2000 and 4000.

We consider that the experimental results of 3-disk systems are fundamental results since the DM method are strictly optimal in 3-disk systems. The experimental results of 4-disk and 5-disk systems are the comparisons of 3-disk systems. The experiments will show the variation between the different sample numbers of records

The experimental results will be shown in the following tables and figures.

(1) The strictly optimal values are the best number of disk accesses on each disk in theory. In Table 1, it shows the average values of the maximum access number in the DM method, our algorithm and strictly optimal values.

(2) It shows the cost of reallocation in Table 2. In Table 3, it shows the difference of average maximum access number among the DM method, strictly optimal values and the sum of Algorithm and the reallocation cost.

(3) In Figure 4, show the comparisons of maximum number of disk accesses among the DM allocation method, our algorithm and strictly optimal values.

(4) The average cost of reallocation is in Table 2(b). In Figure 5, they show the average cost of reallocation per 50 new records inserted when C=2 and C=3.

In Table 1 shows the maximum number of disk accesses among disks in the 3-disk, 4-disk and 5-disk systems when the upper bound C=3. It is obvious that the DM method behaves worse than our algorithm. The more are the total numbers of records, the greater are the difference of maximum number of disk accesses between the DM method and our algorithm.

From Table 1, we can deduce Figure 4 which show the comparison of maximum number of disk accesses among the DM method, our algorithm and the strictly optimal values. The results of experiments show that the performance of our algorithm behaves between the DM method and strictly optimal values. The forecast in Figure 2 is similar to the experimental results.

The result are concluded from Table 1 and Figure 4 as follows:

(1) Strictly optimal values are theoretical results of maximum number of disk accesses.

(2) The results of the DM method applied to dynamic files are similar to the forecast of static approaches in Figure 2.

(3) Our algorithm behaves better than the DM method.

In our algorithm, partial reallocation is necessary.

The cost of reallocation is shown in Table 2(a). The average cost per record is shown in Table 2(b).

In Table 1, our algorithm always behaves better than the DM method in different cases. But, the cost of partial reallocation should be considered in our algorithm. From Table 2, we predict that the average partial relocation cost per record N_{cost}/N_{rec} is constant when the number of records is large enough. To find the approximation of N_{cost}/N_{rec}, we construct another table, Table 3.

Table 3 shows the difference among the DM method, our algorithm and strictly optimal values. From Table 3, and Figure 5 are deduced. Figure 5, for a 5-disk system, respectively, show the cost of reallocation per 50 new records inserted. From these figures, we find that N_{cost}/N_{rec} will be a constant if the total number of records is great enough.

In Table 3, some items are defined as follows:

(1) $D_{DO} = N_{DM} - N_{SO}$, where D_{DO} is the difference between the DM method and strictly optimal values.

(2) $D_{AO} = (N_{Algorithm2} + N_{COST}/N_{rec}) - N_{SO}$, where D_{AO} is the difference between Algorithm including the cost of partial reallocation and strictly optimal values.

(3) $D_{DA} = N_{DM} - (N_{Algorithm2} + N_{COST}/N_{rec})$, where D_{DA} is the difference between our algorithm including the cost of partial reallocation and the DM method.

5. Conclusions

In the applications of information systems, the contents of database systems are not defined in advance. Records are inserted to or deleted from files dynamically. The traditional allocation method cannot behave a good performance in the dynamic allocation problem. An appropriate dynamic allocation method can handle dynamic allocation problems well.

The traditional allocation methods have to reallocate all records in a file when some of the records are inserted to or deleted from the files. Our approach is proposed to avoid reallocating all existed records in a file. However, partial reallocation is needed.

In the future, there are some points for improving the performance of dynamic allocation approaches. We conclud the future works as follows:

(1) the total cost of reallocation,
(2) which bucket to be moved,
(3) how to find RT_1 and RT_2,
(4) the dynamic allocation algorithm,
(5) theoretical analysis.

References

[1] K. A. S. Abdel-Ghaffar and El Abbadi, " Optimal Disk Allocation for Partial Matching Queries, " *ACM Trans. on Database Systems*, Vol. 18, No. 1, pp. 132-156, March 1993.

[2] L. Bentley, " Multidimensional Binary Search Trees Used for Associative Searching, " *CACM*, Vol. 18, pp. 509-516, Sept., 1975.

[3] A. Bolour, " Optimality Properties of Multiple Key Hashing Function, " *Journal of ACM*, Vol. 26, No. 2, pp. 196-210,.1979.

[4] C. C. Chang and C. Y. Chen, " Gray Code As a De-clustering Scheme for Concurrent Disk Retrieval, " *Information Science and Engineering (3)*, pp. 177-188, 1987.

[5] C. C. Chang, H. Y. Chen and C. Y. Chen, " Symbolic Gray Code As a Data Allocation Scheme for Two-disc Systems, " *The Computer Journal*, Vol. 35, No. 3, pp. 299-305, 1992

[6] C. C. Chang, M. W. Du and R. C. T. Lee, " Performance Analysis of Cartesian Product Files and Random Files, " *IEEE Trans. on Software Engineering*, Vol. SE-10, No. 1, pp. 88-99, January 1984.

[7] C. C. Chang, R. C. T. Lee and H. C. Du, " Some Properties of Cartesian Product Files, " *in Proc. ACM-SIG MOD 1980 Conf., Santa Monica, CA*, pp. 157-166, May 1980.

[8] C. C. Chang and D. H. Su. " Some Properties of Multiattribute File System Based upon Multiple Key Hashing Functions, " *proc. 21th Annu. Allerton Conference Commun. Control Comput.*, pp. 675-682, 1983.

[9] H. C. Du, " Disk Allocation Methods for Binary Cartesian Product Files, " *BIT 26*, pp. 138-147, 1986.

[10] H. C. Du and R. C. T. Lee, " Symbolic Gray Code As Multi-Key Hashing Function, " *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 83-90 (1982).

[11] H. C. Du and J. S. Sobolewski, " Disk Allocation for Cartesian Product Files on Multiple-Disk Systems, " *ACM Trans. on Database Systems*, Vol. 7, No. 1, March 1982, pp. 82-101.

[12] R. Fagin, J. Nieverglet, N. Pippenger and H. R. Strong, " Extendible Hashing - A Fast Access Method for Dynamic Files, " *ACM Trans. on Database Systems*, Vol. 4, No. 3, Sept. 1979, pp. 315-344.

[13] M. T. Fang, R. C. T. Lee and C. C. Chang, " The Idea of De-Clustering and its Application, " *the 12th International Conference on Very Large Data Base (VLDB), Kyoto, Japan*, pp. 181-188, August 1986.

[14] J. H. Friedman, F. Baskett and L. J. Shustek, " An Algorithm for Finding Nearest Neighbors, " *IEEE Trans. on Computers*, Vol. 24, No. 10, pp. 1000-1006, 1975.

Disk 0	Disk 1	Disk 2	Disk 3
(0, 0, 0)	(0, 0, 1)	(0, 1, 1)	(0, 2, 1)
(1, 2, 1)	(0, 1, 0)	(0, 2, 0)	(1, 1, 1)
(2, 1, 1)	(1, 0, 0)	(1, 0, 1)	(1, 2, 0)
(2, 2, 0)	(2, 2, 1)	(1, 1, 0)	(2, 0, 1)
(3, 0, 1)	(3, 1, 1)	(2, 0, 0)	(2, 1, 0)
(3, 1, 0)	(3, 2, 0)	(3, 2, 1)	(3, 0, 0)

(a)

Disk 0	Disk 1	Disk 2	Disk 3
(0, 0, 0)			(1, 1, 1)
(2, 2, 0)			(1, 2, 0)
			(2, 1, 0)
			(3, 0, 0)

(b)

(*, *, 0)	Disk 0	Disk 1	Disk 2	Disk 3
F_1	3	3	3	3
F_2	2	0	0	3

(c)

Figure 1: (a) The allocation of all records among 4 disks in a static file F_1
(b) The allocation of records among 4 disks in a dynamic file F_2
(c) The number of disk accesses of F_1 and F_2 according to the partial matching query (*, *, 0)

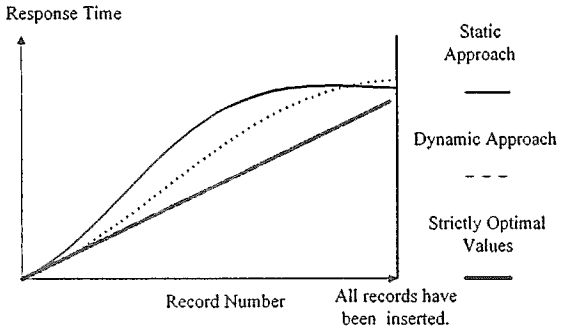

Figure 2: The behavior of a dynamic file

Disk 0	Disk 1	Disk 2	Disk 3
(0, 0, 0)	(2, 1, 2)	(0, 1, 1)	(1, 0, 2)
(2, 1, 1)	(0, 1, 0)	(1, 0, 1)	
(2, 0, 2)		(1, 2, 3)	
(2, 3, 3)		(2, 1, 3)	

(a)

Disk 0	Disk 1	Disk 2	Disk 3
(0, 0, 0)	(2, 1, 2)	(0, 1, 1)	(1, 0, 2)
(2, 1, 1)	(0, 1, 0)	(1, 0, 1)	
(2, 0, 2)		(1, 2, 3)	
		(2, 1, 3)	(2, 3, 3)

(b)

PMQ	Disk 0	Disk 1	Disk 2	Disk 3
...	...	...	...	...
(0, *, *)	1	1	1	0
(1, *, *)	0	0	2	1
(2, *, *)	3	1	1	0
(*, 0, *)	2	0	1	1
(*, 1, *)	1	2	2	0
(*, 2, *)	0	0	1	0
(*, 3, *)	1	0	0	0
...	...	...	...	...

(c)

PMQ	Disk 0	Disk 1	Disk 2	Disk 3
...	...	...	...	...
(0, *, *)	1	1	1	0
(1, *, *)	0	0	2	1
(2, *, *)	2	1	1	1
(*, 0, *)	2	0	1	1

(d)

Figure 3:
(a) The records on each disk are inserted by our algorithm
(b) The record, (2, 3, 3) on Disk 0, are reallocated to Disk 3
(c) The part of partial matching queries after the bucket (2, 3, 3) is allocated to disk 0.
(d) The part of partial matching queries after the bucket (2, 3, 3) is reallocated to Disk 3

N_{rec}	N_{DM}			$N_{Algorithm}$			N_{SO}		
	Disk=3	Disk=4	Disk=5	Disk=3	Disk=4	Disk=5	Disk=3	Disk=4	Disk=5
100	48.23	40.37	36.43	41.87	33.7	28.93	34	25	20
200	87.37	71.20	63.13	75.20	60.73	50.17	67	50	40
500	198.13	157.73	136.23	180.10	137.47	113.77	167	125	100
1000	379.67	294.77	246.73	349.03	263.90	215.67	334	250	200
2000	727.73	566.37	464.10	685.43	519.17	418.30	667	500	400
4000	1419.63	1096.27	892.10	1360.93	1026.37	832.40	1334	1000	800

(a)

N_{rec} : the total number of inserted records. N_{DM} : the number of disk accesses of the DM method,
$N_{Algorithm}$: the number of disk accesses of Algorithm, N_{cost} : the cost of reallocation,
N_{cost} / N_{rec} : the average cost per record,

Table 1: (a) The comparisons of the maximum number of disk accesses between, N_{DM} , $N_{Algorithm}$ and N_{SO} , where C=3.

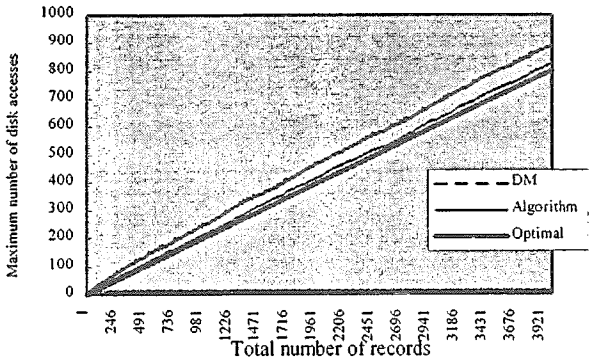

Figure 4: The results show DM method, Algorithm and strictly optimal values in a 5-disk system, where the total number of records are 4000.

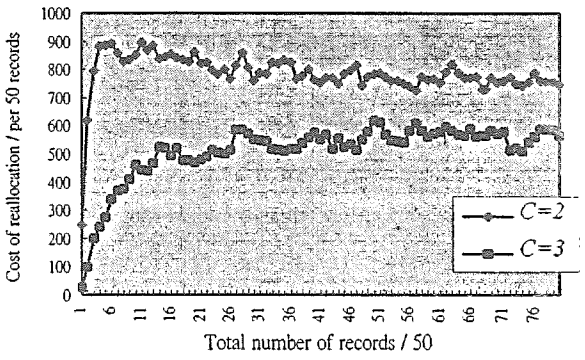

Figure 5: The cost of reallocation per 50 records inserted for a 5-disk system , where the total number of records is 4000.

N_{rec}	Disk Number=3		Disk Number=4		Disk Number=5	
	C=2	C=3	C=2	C=3	C=2	C=3
100	763.9	144.8	771.2	125.9	864.0	126.9
200	2148.7	608.7	2260.5	556.2	2540.5	563.4
500	6530.8	3196.4	7272.2	2674.3	7689.8	2787.3
1000	14272.9	8463.8	15465.1	7847.0	16240.8	7614.8
2000	28782.0	19440.4	30433.2	18760.5	32256.1	18238.1
4000	55956.1	40350	59753.2	39741.7	62903.9	40744.5

(a)

N_{rec}	Disk Number=3		Disk Number=4		Disk Number=5	
	C=2	C=3	C=2	C=3	C=2	C=3
100	7.6	1.4	7.7	1.3	8.6	1.3
200	10.7	3.0	11.3	2.8	12.7	2.8
500	13.1	6.4	14.5	5.3	15.4	5.6
1000	14.3	8.5	15.5	7.8	16.2	7.6
2000	14.4	9.7	15.2	9.4	16.1	9.1
4000	14.0	10.1	14.9	9.9	15.7	10.2

(b)

N_{cost} : the cost of reallocation, N_{cost}/N_{rec} : the average cost per record.

Table 2: (a)The cost of reallocation, N_{cost}.
(b)The average cost per record, N_{cost}/N_{rec} .

N_{rec}	Disk Number=3			Disk Number=4			Disk Number=5		
	D_{DO}	D_{AO}	D_{DA}	D_{DO}	D_{AO}	D_{DA}	D_{DO}	D_{AO}	D_{DA}
100	14.2	9.1	5.1	15.4	10.3	5.1	16.4	10.5	5.9
200	20.4	11.7	8.7	21.2	12.6	8.6	23.1	12.6	10.5
500	31.1	17.1	14.0	32.7	19.5	13.2	36.2	19.2	17.0
1000	45.7	24.2	21.5	44.8	22.9	21.9	46.7	22.2	24.5
2000	60.7	25.0	35.7	66.4	30.3	36.1	64.1	27.0	37.1
4000	85.6	32.7	52.9	96.3	41.3	55.0	92.1	33.0	59.1

Table 3: The results of D_{DO}, D_{AO} and D_{DA} , where C=3