NBM: An Efficient Cache Replacement Algorithm for Nonvolatile Buffer Caches

JUNSEOK PARK and KERN KOH
Seoul National University
56-1 Shillim-dong, Kwanak-gu, Seoul, 151-742
REPUBLIC OF KOREA

HYUNKYOUNG CHOI and HYOKYUNG BAHN
Ewha University
11-1 Daehyun-dong, Seodaemun-gu, Seoul, 120-750
REPUBLIC OF KOREA

Abstract: Recently, byte-accessible NVRAM (nonvolatile RAM) technologies such as PRAM and FeRAM are advancing rapidly and there are attempts to use these NVRAMs as part of buffer caches. A nonvolatile buffer cache provides improved consistency of file systems by absorbing write I/Os as well as improved performance. In this paper, we discuss the optimality of cache replacement algorithms in nonvolatile buffer caches and present a new algorithm called NBM (NVRAM-aware Buffer cache Management). NBM has three salient features. First, it separately exploits read and write histories of block references, and thus it estimates future references of each operation more precisely. Second, NBM guarantees the complete consistency of write I/Os since all dirty data are cached in NVRAM. Third, metadata lists are maintained separately from cached blocks. This allows more efficient management of volatile and nonvolatile buffer caches based on read and write histories, respectively. Trace-driven simulations show that NBM improves the I/O performance of file systems significantly compared to the NVLRU algorithm that is a modified version of LRU to hold dirty blocks in NVRAM.

Key-Words: Buffer cache, Replacement algorithm, Nonvolatile RAM, Caching, LRU.

1 Introduction

Due to recent advances in semiconductor technologies, byte-accessible NVRAMs (nonvolatile RAMs) such as MRAM (magnetic RAM), PRAM (phasechange RAM), and FeRAM (ferro electro RAM) are emerging rapidly [2, 3, 4]. There are some attempts to use NVRAMs as part of buffer caches [5, 7, 8]. By using NVRAMs together with VRAMs (volatile RAMs) as buffer cache spaces, consistency of file systems can be improved by absorbing write I/Os to the NVRAM. Since file systems generally tend to perform write I/Os to the RAM component and flush them to secondary storage periodically due to performance reasons, there is an interval of time in which consistency of the file system is compromised. By performing writes to NVRAM instead of VRAM, this period of inconsistency is removed, and consistency can be maintained completely [5].

In this paper, we discuss the optimality of cache replacement algorithms in nonvolatile buffer caches and present a new algorithm called NBM (NVRAM-aware Buffer cache Management). The storage architecture of NBM consists of the secondary storage media and two kinds of buffer caches, namely volatile buffer cache and nonvolatile buffer cache. The secondary storage is basically composed of hard disks, but NAND flash memory or other storage media can be used. The volatile buffer cache is composed of usual DRAM, and the nonvolatile buffer cache is composed of NVRAM such as PRAM, FeRAM, or MRAM. Our buffer cache replacement algorithm has three salient features. First, it separately exploits the read history and the write history of block references, and thus it estimates future references of each operation more precisely. Second, our algorithm guarantees the complete consistency of write I/Os since all dirty data are cached in NVRAM. Third, metadata lists are maintained separately from cached blocks. That is, we use two lists to maintain the recency history of references, namely the LRR (least recently read) list and the LRW (least recently written)

list. However, blocks in these lists are not necessarily identical to those blocks in the volatile and nonvolatile buffer caches, respectively. In reality, metadata of an evicted block from the buffer cache can be maintained in the list. This separation allows more efficient management of volatile and nonvolatile buffer cache spaces. Trace-driven simulations show that the proposed algorithm improves the I/O performance of files systems significantly.

The remainder of the paper is organized as follows. Before describing our algorithm, we present the formal definition of nonvolatile buffer caching problems and give an offline optimal algorithm in Section 2. Then, we explain the system architecture and present a new buffer cache replacement algorithm on this architecture in Section 3. Section 4 shows experimental results obtained through trace-driven simulations to assess the effectiveness of the proposed scheme. Finally we conclude this paper in Section 5.

Table 1. Comparison of DRAM, SRAM, NAND flash memory, and three types of byte-accessible NVRAMs.

Media characteristics	MRAM	FeRAM	PRAM	SRAM	DRAM	NAND flash
Nonvolatility	Yes	Yes	Yes	No	No	Yes
Write latency per byte	10~50ns	30~100ns	100ns~	30~70ns	50ns~	10μs~
Read latency per byte	10~50ns	30~100ns	20~80ns	30~70ns	50ns	50ns
Max. erase cycle	10^{16}	10^{12}~10^{16}	10^{12}	10^{15}	10^{15}	10^{6}
Energy consumption	~30μW	~10μW	~30μW	~300mW	~300mW	30mW

2 Formal Definition of Nonvolatile Buffer Caching

Before describing nonvolatile buffer caching, we first review the traditional buffer caching problems. Let S be the size of buffer cache, r the total number of references, and h the number of hit references. Then, the buffer cache manager needs to store the currently requested block without exceeding S. In this paper, we focus on non-lookahead, demand fetching algorithms. If the number of blocks in the buffer cache is larger than S, the replacement algorithm selects a victim and purges it from the buffer cache. The goal of the algorithm is to maximize the number of blocks referenced directly from the buffer cache after all the requests have been processed. The hit ratio, calculated by h/r, is an appropriate performance metric to measure the performance of the replacement algorithm. In the traditional volatile buffer cache environments, Belady's MIN algorithm is known to be optimal with respect to the hit ratio [1]. The MIN algorithm replaces the block that will be referenced furthest in the future. MIN is not a practical algorithm because one cannot know future references in real systems. However, MIN provides the upper bound of performance to the research community pursuing good online algorithms.

Now, let us look at the nonvolatile buffer caching problems. We assume that both volatile and nonvolatile buffer caches are used together. We also assume that all writes are performed in nonvolatile buffer cache. Hence, consistency is always guaranteed. Writes to secondary storage occurs only when a block is evicted from the nonvolatile buffer cache. Additionally, we assume that all clean blocks reside in volatile buffer cache. This assumption may be released in practical terms, but similar to previous researches we include it in theoretical analysis [5].

Let S_{NV} be the size of nonvolatile buffer cache, S_V be the size of volatile buffer cache, r the total number of read references, w the total number of write references, and h the number of hit references. Unlike traditional buffer caching problems, it is known that the hit ratio, calculated by $h/(r+w)$, is not a good performance metric for nonvolatile buffer caching environments. Instead, the performance metric should be changed to the number of disk I/Os [5]. This is because even a hit may cause I/O operations. For example, if a write reference comes and the requested block exists in the volatile buffer cache but not in the nonvolatile buffer cache, it is apparently a hit. However, this incurs a write operation to nonvolatile buffer cache. If there is no empty slot in the nonvolatile buffer cache, we need to evict a dirty block from the nonvolatile buffer cache, which essentially incurs an I/O operation.

The problem is then to reduce the total number of I/O operations. For read operations, blocks in both volatile and nonvolatile buffer caches can be referenced. However, for write operations, we should only use nonvolatile buffer cache for the consistency reason. Thus, if a write request arrives and the requested block does not exist in the nonvolatile buffer cache, the replacement algorithm should make a room in the nonvolatile buffer cache for that request. If the number of blocks in the nonvolatile buffer cache is larger than S_{NV}, the replacement algorithm selects a victim and purges it, which eventually incurs a write I/O. However, if a read request arrives, a cache hit from either volatile or nonvolatile cache does not incur an I/O operation.

With these different situations, the optimality of the replacement algorithm also changes for nonvolatile buffer caching environments. An optimal algorithm, we call NV-MIN, behaves as follows. We do not provide the formal optimality proof of this algorithm, but it can be shown intuitively. Note that this algorithm is enhanced from MIN+ presented by Lee et al [5]. When a read miss occurs, NV-MIN chooses a victim block by the following scenario. If there is an empty slot in VRAM or there is a slot whose next reference is write in VRAM, NV-MIN caches the new block here. Otherwise, NV-MIN evicts the block that will be read-referenced furthest in the future among those in VRAM.

When a write miss occurs in NVRAM, NV-MIN behaves as follows. Note that this case includes the case that a write-referenced block already exists in VRAM but not in NVRAM. If there exists an empty slot in NVRAM, NV-MIN caches the requested block in this slot. (In this case, if the block already exists in VRAM, NV-MIN removes it from VRAM.) Otherwise, NV-MIN replaces the block that will be write-referenced furthest in the future. If the evicted block will be read-referenced again in the future, it may be copied to VRAM according to the following scenario. If there exists an empty slot in VRAM or if there exists a block whose next request is write in VRAM, NV-MIN caches the evicted block to this slot. Otherwise, NV-MIN compares two blocks, namely the block that will be read-referenced furthest in the future among those in VRAM and the evicted block from NVRAM, and evicts the block that will be read-referenced further in the future. Finally, NV-MIN writes the block to the disk.

3 A New Replacement Algorithm for Nonvolatile Buffer Caches

3.1 System Architecture

The system architecture in this paper is described in Fig. 1. The storage system consists of the secondary storage media and two kinds of buffer caches, namely volatile buffer cache and nonvolatile buffer cache. The secondary storage is basically composed of hard disks, but NAND flash memory or other storage media can be used. The volatile buffer cache is composed of usual DRAM, and the nonvolatile buffer cache is composed of NVRAM such as PRAM, FeRAM, or MRAM. All writes to buffer caches are performed in nonvolatile buffer cache. Hence, all dirty blocks reside in NVRAM and thus consistency is always guaranteed. Writes to secondary storage happens only when an eviction from NVRAM occurs.

Hybrid hard disks (HHD) [9] also have a NVRAM layer between DRAM and secondary storage. However, NVRAM layer in HHD cannot be used as a system buffer cache since it is composed of flash memory and thus it is not byte-accessible. Moreover, NVRAM in this paper is equivalently used with traditional DRAM except that it is non-volatile, whereas NVRAM in HHD mainly serves as a write buffer.

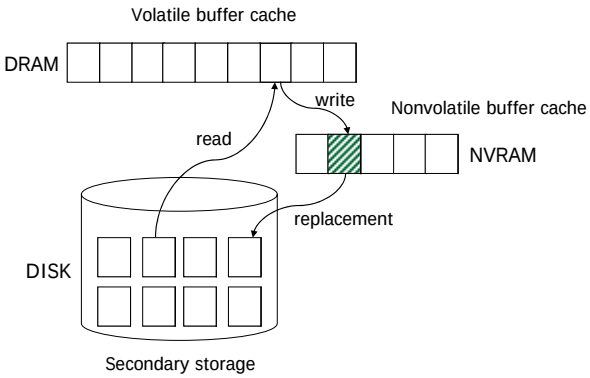


Fig. 1 System architecture of NBM.

3.2 The NBM (NVRAM-aware Buffer cache Management) Algorithm

In this subsection, we present the NVRAM-aware buffer cache management (NBM) algorithm. NBM separately exploits the read history and the write history of block references to estimates future references of each operation precisely. To do this, NBM uses two lists to maintain the recency history of references, namely the LRR (least recently read) list and the LRW (least recently written) list. Note that blocks in these two lists are not necessarily identical to those blocks in the volatile and nonvolatile buffer caches, respectively. This separated management of metadata and actual data allows more efficient management of volatile and nonvolatile buffer cache spaces. For example, if a block is recently read and written, the metadata of the block can exist in both LRR and LRW lists, but actual data is only maintained in the nonvolatile buffer cache. Thus, some blocks in the LRR list may not exist in the volatile buffer cache. This allows volatile buffer cache to hold more blocks, leading to more read-hits. Nevertheless, we maintain

the read history of the block in the LRR list because the block may return to the volatile buffer cache if it is evicted from the nonvolatile buffer cache. Fig. 3 depicts the pseudocode of the NBM algorithm.

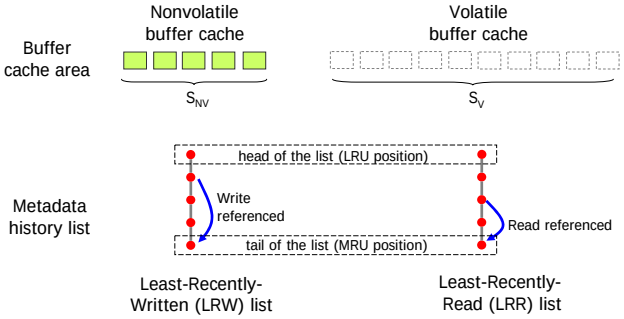


Fig. 2 Data structures used in the NBM algorithm.

```
NBM (block p, operation op)   /* p is requested block */
{   if (p is in the buffer cache) /* cache hit */
    {   if (op is read)
        {   if (p∉read_list) add p at the tail of read_list;
            else move p to the tail of read_list;
        else   /* write */
        {   if (p∉write_list) add p at the tail of write_list;
            else move p to the tail of write_list;
            if (p is not in NVRAM)
            {   if (no free block in NVRAM) replace_NVRAM ();
                move p from VRAM to NVRAM;
            }
        }
    }
    else /* cache miss */
    {   if (op is read)
        {   if (no free block in VRAM) replace_VRAM ();
            add p to VRAM;
            add p at the tail of read_list;
        }
        else   /* write */
        {   if (no free block in NVRAM) replace_NVRAM ();
            add p to NVRAM;
            add p at the tail of write_list;
        }
    }
}

replace_VRAM ()
{   evict the least recently read block p in VRAM;
    evict the read history of p from read_list;
}

replace_NVRAM ()
{   select the least recently written block p in NVRAM;
    evict the write history of p from write_list;
    if (p∈read_list)
    {   if (p is more recently read
            than the least recently read block q in VRAM)
        {   evict q from VRAM;
            evict the read history of q from read_list;
            move p from NVRAM to VRAM;
        }
    }
    else evict p from NVRAM;
}
```

Fig. 3 Pseudocode of NBM.

4 Experimental Results

We have conducted trace-driven simulations to compare the performance of buffer cache management algorithms in terms of total number of I/O operations. We used traces collected by Roselli et al. [6] from the Hewlett-Packard series 700 workstations running HP-UX. These traces are categorized into three environments, namely INS (instructional workload), RES (research workload), and WEB (Web server workload). Details of these traces are summarized in Table 2.

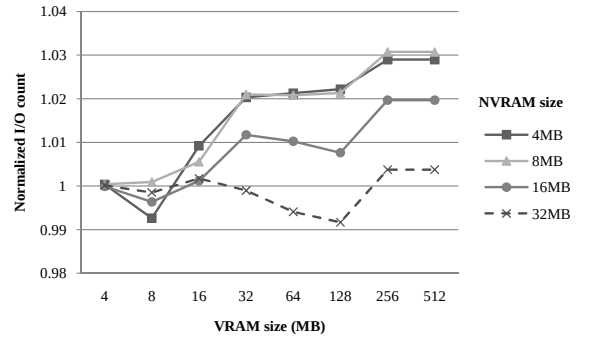
Table 2. Summary of traces used in our experiments.

	trace	INS	RES	WEB
System calls	Total	861168	393571	371019
	Reads	733549	336963	329377
	Writes	127619	56608	41642
	Read:Write	5.75	5.95	7.91
Memory usage (MB)	VRAM	106	68	316
	NVRAM	152	133	136
Read access pattern	Read clean block	1173289	39819	2035430
	Read dirty block	162212	94099	84766
	Ratio	7.23	4.22	24.01

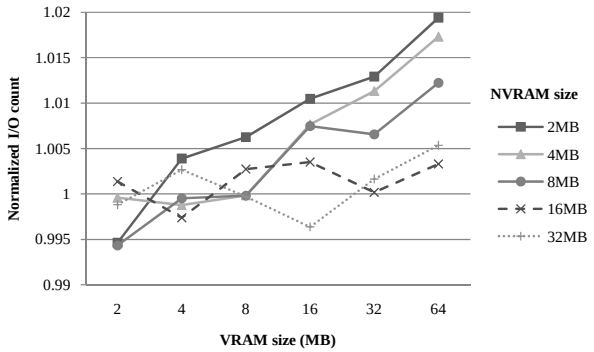
For comparison, we have designed and implemented NVLRU, which is a modified version of the traditional LRU algorithm. NVLRU holds dirty blocks in NVRAM, conserving file system consistency without periodical flush. Whenever it needs to accommodate a new dirty block but there is no free block in NVRAM, it selects and evicts the least recently used dirty block from NVRAM.

Fig. 4 shows the performance of NBM in terms of the total number of I/O operations normalized by NVLRU. NBM outperforms NVLRU especially when the system is under heavy NVRAM pressure. This is due to the major drawback of the NVLRU algorithm. NVLRU holds dirty blocks that are recently read but rarely written in NVRAM because it maintains LRU list based on not only write references but also read references for dirty blocks in NVRAM. On the other hand, NBM efficiently identifies dirty but rarely written blocks using LRW list which tracks write reference history only.

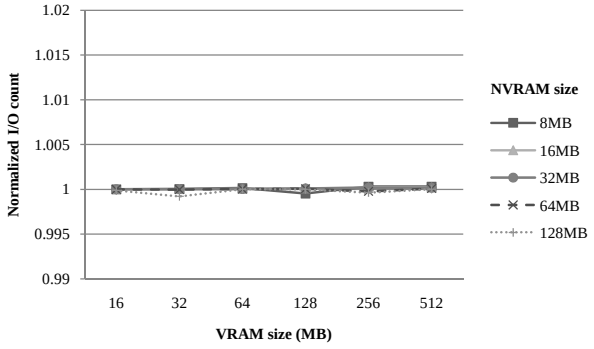
Moreover, NBM further reduces total I/O count when the VRAM space is sufficiently large. As explained earlier, for every NVRAM replacement victim, NBM decides to move the block from NVRAM to VRAM if it is recently read. This helps the system buffer cache to hold all the recently read blocks while consuming less NVRAM space. However, if VRAM is under heavier memory pressure than NVRAM is, moving a block from NVRAM to VRAM can result in worse read hit ratio.



(a) INS workload



(b) RES workload



(c) WEB workload

Fig. 4 Performance of NBM in terms of total I/O count normalized by NVLRU varying VRAM and NVRAM sizes.

NBM shows little performance improvement for workloads that rarely reads from dirty blocks. For such workloads, using LRU or LRW for NVRAM blocks makes no difference. Hence, NBM and NVLRU work similarly. This can be observed well in Fig. 4 (c).

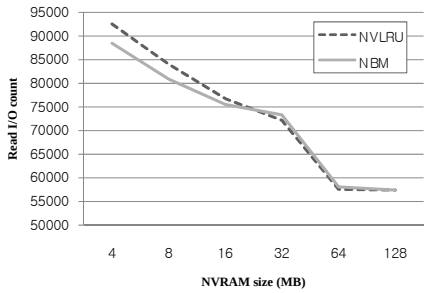
Fig. 5 shows read I/O counts and write I/O counts varying NVRAM sizes. For every workload we have considered, the two algorithms showed almost the same write I/O counts. NBM does flush before moving a dirty block from NVRAM to VRAM to conserve file system consistency. For this reason, NBM issues write I/O as much as NVLRU does. In Fig. 5 (a), we can see reduced read I/O counts by adopting NBM. This change is highlighted in memory settings with small NVRAM. As the amount of NVRAM grows, most of dirty blocks that are currently being used can be held in NVRAM even with simpler algorithms, leaving no room to improve the system performance. At some points, NBM performs even slightly worse than NVLRU in terms of read I/O count. These cases can be observed in certain memory size settings where VRAM is under relatively heavier memory pressure than NVRAM.

5 Conclusion

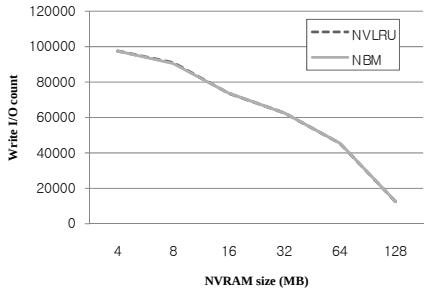
In this paper, we introduced the NBM algorithm that uses both VRAM and NVRAM as buffer caches. To reserve consistency in such systems, dirty blocks are stored in NVRAM and actual write I/O operations occur only when there is not enough free NVRAM space.

To fully utilize NVRAM space, when selecting a NVRAM block to replace, NBM efficiently identifies the least recently written block regardless of its read references by using LRW list. In addition to this, NBM can decides to move the victim block from NVRAM to VRAM depending on its read reference history which is recorded in LRR list, rather than discarding the block after flushing. Consequently, NBM fills NVRAM up only with the most recently written blocks among dirty blocks while maintaining all the recently read blocks in the buffer cache.

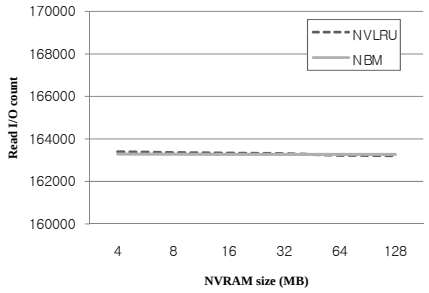
Through trace-driven simulations, we have shown that the NBM algorithm outperforms NVLRU in terms of the total number of I/O operations especially in systems with small NVRAM and large VRAM.



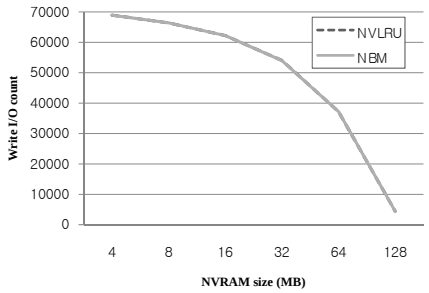
(a) Read I/O count in INS workload (VRAM=128MB)



(b) Write I/O count in INS workload (VRAM=128MB)



(c) Read I/O count in WEB workload (VRAM=256MB)



(d) Write I/O count in WEB workload (VRAM=256MB)

Fig. 5 Comparison of two algorithms in terms of read I/O count and write I/O count varying NVRAM size.

References:

[1] L. Belady, "A Study of Replacement of Algorithms for a Virtual Storage Computer," *IBM Systems Journal*, vol.5, no.2, pp.78–101, 1966.

[2] Magnetic RAM (MRAM) Product & Technology, http://www.memorystrategies.com/report/focused/mram.htm, 2008.

[3] Ferroelectric Memories, http://www.memorystrategies.com/report/focused/ Ferroelectric.htm, 2007.

[4] A Memory Strategies Focus Report: Focus on Phase Change Memory and Resistance RAMs, http://www.memorystrategies.com/report/focused/phaschange.htm, 2008.

[5] K. Lee, I. Doh, J. Choi, D. Lee, S. H. Noh, "Write-aware buffer cache management scheme for nonvolatile RAM," *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*, Phuket, Thailand, pp.29-35, 2007.

[6] D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads," *In Proceedings of the 2000 USENIX Annual Technical Conference (USENIX-00)*, Berkeley, CA, pp.41-54, 2000.

[7] T. R. Haining and D. D. E. Long, "Management policies for non-volatile write caches," *Proceedings of the IEEE International Performance, Computing and Communications Conference*, pp. 321-328, 1999.

[8] S. Akyurek and K. Salem, "Management of Partially Safe Buffers," *IEEE Transactions on Computers*, vol.44, no.3, pp. 394-407, 1995.

[9] T. Bisson, S. A. Brandt, and D. D. Long, "A hybrid diskaware spin-down algorithm with I/O subsystem support," *Proceedings of the 26th IEEE International Performance, Computing and Communications Conference*, 2007.

FCM Algorithm Based on Unsupervised Mahalanobis Distances with Better Initial Values and Separable Criterion

JENG-MING YIH
Graduate Institute of Educational Measurement and Statistics
Department of Mathematics Education, National Taichung University
140 Min-Sheng Rd., Taichung City 403, Taiwan
Taiwan
yih@mail.ntcu.edu.tw

YUAN-HORNG LIN
Department of Mathematics Education
National Taichung University
140 Min-Sheng Rd., Taichung City 403, Taiwan
Taiwan
lyh@mail.ntcu.edu.tw

HSIANG-CHUAN LIU
Department of Bioinformatics, Asia University
500 Lioufeng Rd., Wufeng, Taichung 413,Taiwan
TAIWAN

Abstract: The fuzzy partition clustering algorithms are most based on Euclidean distance function, which can only be used to detect spherical structural clusters. Gustafson-Kessel (GK) clustering algorithm and Gath-Geva (GG) clustering algorithm, were developed to detect non-spherical structural clusters, but both of them based on semi-supervised Mahalanobis distance needed additional prior information. An improved Fuzzy C-Mean algorithm based on unsupervised Mahalanobis distance, FCM-M, was proposed by our previous work, but it didn't consider the relationships between cluster centers in the objective function. In this paper, we proposed an improved Fuzzy C-Mean algorithm, FCM-MS, which is not only based on unsupervised Mahalanobis distance, but also considering the relationships between cluster centers, and the relationships between the center of all points and the cluster centers in the objective function, the singular and the initial values problems were also solved. A real data set was applied to prove that the performance of the FCM-MS algorithm gave more accurate clustering results than the FCM and FCM-M methods, and the ratio method which is proposed by us is the better of the two methods for selecting the initial values.

Key-Words: FCM-MS, FCM-M, GK algorithms, GG algorithms, Mahalanobis distance

1 Introduction

Clustering plays an important role in data analysis and interpretation. It groups the data into classes or clusters so that the data objects within a cluster have high similarity in comparison to one another, but are very dissimilar to those data objects in other clusters. Fuzzy partition clustering is a branch in cluster analysis, it is widely used in pattern recognition field. The well known ones, such as, C. Bezdek's "Fuzzy C-Mean (FCM)" [1], are all based on Euclidean distance function, which can only be used to detect the data classes with same super spherical shapes.

Extending Euclidean distance to Mahalanobis distance, the well known fuzzy partition clustering algorithms, Gustafson-Kessel (GK) clustering algorithm [3] and Gath-Geva (GG) clustering algorithm [2] were developed to detect non-spherical structural clusters, but these two algorithms fail to consider the relationships between cluster centers in the objective function, GK algorithm must have prior information of shape