Efficient Disk Allocation for Fast Similarity Searching *

Sunil Prabhakar Divyakant Agrawal Amr El Abbadi
Department of Computer Science
University of California
Santa Barbara
CA 93106, U.S.A.

Abstract

As databases increasingly integrate non-textual information it is becoming necessary to support efficient similarity searching in addition to range searching. Recently, declustering techniques have been proposed for improving the performance of similarity searches through parallel I/O. In this paper, we propose a new scheme which provides good declustering for similarity searching. In particular, it does global declustering as opposed to local declustering, exploits the availability of extra disks and does not limit the partitioning of the data space. Our technique is based upon the cyclic declustering schemes which were developed for range and partial match queries. We establish, in general, that cyclic declustering techniques outperform previously proposed techniques.

1 Introduction

Identifying objects that are similar to each other is a challenging problem for many applications, such as multimedia repositories or digital libraries. Examples of such applications are the QBIC project [NBE+93] and the Alexandria Digital Library [CDN+97]. In order to automate similarity searching, it is first necessary to define a similarity measure. The most common approach is to map each data object into a high-dimensional space such that the distance between two such points is a measure of the similarity between the corresponding data objects – the closer the points in the high-dimensional space, the more similar the data objects [MM96, Jag91]. The number of dimensions necessary for satisfactory mapping can be very large, and dimensionality reduction techniques are first employed to reduce the number of dimensions to smaller values [AFS93]. Searching for similar objects therefore transforms into a problem of locating the nearest points. A nearest-neighbor query is evaluated as follows. Given a query data object for which the most similar object(s) are to be

*Work supported by a research grant from NSF/ARPA/NASA IRI9411330 and NSF instrumentation grant CDA-9421978.

found from a collection of objects, the object is first mapped into the high-dimensional space. Next, the nearest point(s) from among all the points in the database is(are) determined. The object(s) corresponding to these points are the object(s) most similar to the query object.

Typical applications contain large numbers of data objects. Searching for the nearest points in such high-dimensional space is a very computation and I/O intensive operation. Several index structures have been proposed in the literature that significantly improve the search operation in multidimensional spaces. Some examples of these include the R-tree [Gut84], R^*-tree [BKSS90], the X-tree [BKK96], HB-tree [LS90], GiST [HNP95] and others [WJ96, Rob81, KS91, Gun89]. The performance of these structures is good for few dimensions, but as the number of dimensions increases beyond 10, the performance degrades significantly [AFS93]. An effective technique for improving the performance of the index structures is to employ parallelism. The major bottleneck is the large number of disk I/O operations that need to be performed. This number typically grows rapidly as the number of dimensions increases. Therefore the key to improving the performance is distributing the data among several I/O devices such that the data retrieved by any query is evenly spread across all the I/O devices (typically disks). Ghandeharizadeh, DeWitt and Qureshi [GD90, GD92] demonstrated, using simulation, that declustering data on multiple disks results in significant improvements in response time for range queries.

The problem of declustering a high-dimensional space across multiple disks has been well studied in the context of relational databases, where the goal has been to improve the performance of range and partial match queries. Prominent examples of such declustering schemes are HCAM [FB93], Fieldwise Exclusive or FX [KP88] and CMD [LSR92] or Disk Modulo [DS82] and error-correcting based techniques [FM89, AE93]. In [BBB+97], a new declustering technique is developed that optimizes data allocation on multiple disks for nearest-neighbor queries. The authors argue that the existing declustering techniques do not generate adequate declustering for nearest-neighbor queries. Through experimentation, it is shown that their approach gives much better performance for nearest-neighbor and 10 nearest-neighbors queries as compared to the HCAM approach, which has been shown to give superior performance for square range queries in two dimensions [FB93].

In a recent study [PAAE98], we proposed a new class of declustering algorithms called *Cyclic* allocation schemes for two-dimensional data. The schemes were shown to give the best declustering performance for range and partial match queries. It is also shown that the performance of the HCAM approach degrades when non-square queries are considered. The performance of HCAM for the nearest-neighbor queries is also found to be poor (achieving a speed-up of only 3 with 16 disks) [BBB+97]. The new approach developed in [BBB+97] limits the number of divisions along each dimension to 1. Also, the availability of additional disks does not improve the performance of the declustering. The other approaches do not suffer from these limitations. In this paper, we propose a new scheme which provides good declustering for similarity searching. In particular, it does global declustering as opposed to local declustering, exploits the availability of extra disks and does not restrict the partitioning of the data space. We establish, in general, that Cyclic declustering techniques outperform previously proposed techniques.

The rest of the paper is organized as follows. A brief description of the problem of declustering for nearest-neighbor searches is given in Section 2. In Section 3, the previous approaches for declustering data are presented. In Section 4 we present our new Cyclic allocation scheme. In Section 5, the performance of our scheme in comparison to previously proposed schemes is analyzed, and Section 6 concludes the paper.

2 Nearest-Neighbor Searching

In this section we discuss the problem of nearest-neighbor searching. The problem can be formally defined as:

Definition 1 *Given a collection of d-dimensional points, $\mathcal{C}$, and a query point, p, find the point in $\mathcal{C}$ that is closer to p than any other point in $\mathcal{C}$. In other words,*

$$Nearest\_Neighbor(p) = \{q | q \in \mathcal{C} \text{ and } \forall r \in \mathcal{C}, \|r - p\| \geq \|q - p\|\}$$

where $\|v\|$ is the norm or length of vector v.

In order to distribute the data points across the disks, the domain is first divided into smaller parts, called *buckets*. This is typically achieved by dividing each dimension into several parts. Each bucket is then identified by a set of numbers or coordinates corresponding to each dimension.

We borrow and extend the definitions of direct and indirect neighbors from [BBB$^+$97] (the model in their paper does not allow splitting any dimension into more than two parts).

Definition 2 *Two buckets are direct neighbors if their coordinates differ in only one dimension. Moreover, the magnitude of the difference in the coordinates should be 1.*

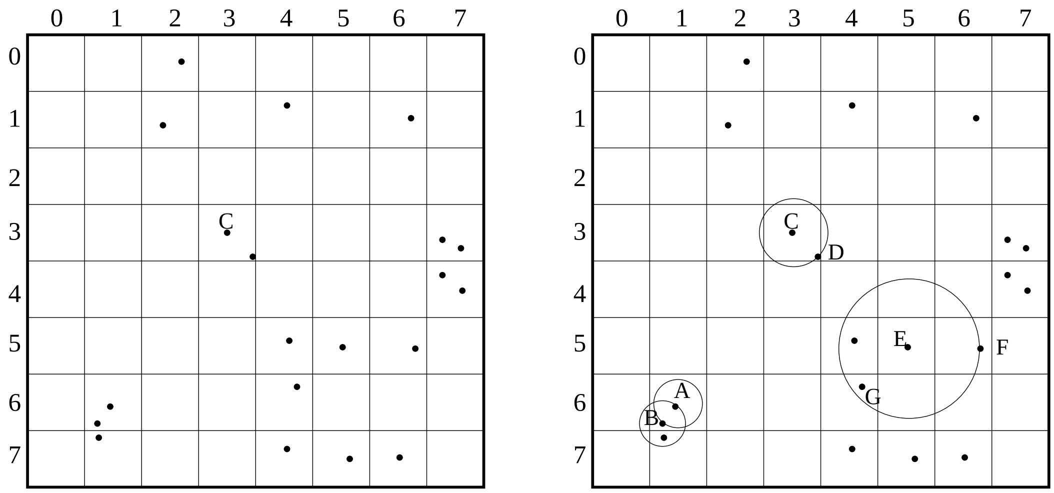
Thus, in two dimensions, bucket (i, j) is a direct neighbor of buckets $(i+1, j), (i-1, j), (i, j+1)$ and $(i, j-1)$, if they exist.

Definition 3 *Two buckets are indirect neighbors if their coordinates differ in only two dimensions. Moreover, the magnitude of the difference in both dimensions should be 1.*

Thus, in two dimensions, bucket (i, j) is an indirect neighbor of buckets $(i-1, j-1), (i-1, j+1), (i+1, j-1)$ and $(i+1, j+1)$, if they exist. Note that these definitions hold for any number of dimensions. Similarly, buckets that differ by 1 in exactly three dimensions can be said to be doubly indirect neighbors and so on.

As was established in [BBB$^+$97], for d dimensions, the likelihood that data will be clustered around the $(d-1)$-dimensional surface increases as d becomes larger. Consequently, the need to access direct and indirect neighbors when searching for nearest-neighbors becomes greater. Therefore it is important to effectively decluster direct and indirect neighbors. The scheme proposed in this paper not only optimizes for direct and indirect neighbors, but aims at more global declustering, wherein doubly indirect neighbors and so on are also declustered.

For ease of exposition, we consider the case of only two dimensions. An example of a set of points in two-dimensional space is given in Figure 1(a). These are the points to which the objects in the database are mapped. In order to decluster the data over multiple disks, it is first necessary to divide it into buckets. The region is partitioned into 64 buckets by dividing each dimension into 8 parts. Each bucket can then be identified by the two coordinates in each dimension. Let the horizontal coordinate be the first coordinate and the vertical coordinate be the second. The coordinate values are assumed to go from 0 through 7 as shown in Figure 1(a). Thus the top-left corner bucket is identified as $(0, 0)$, and the bottom-right corner bucket as $(7, 7)$.



(b) (b)

Figure 1: Data points in 2-dimensional space

A nearest-neighbor search would proceed as follows. Given the query point, the bucket containing the query point is first retrieved from the disk. The data points contained in that bucket are analyzed to determine the point that is closest to the query point. Then, depending upon the distance to the closest point and the location of the query point in the bucket, different steps need to be taken. Consider a query asking for the nearest-neighbor of point A in bucket $(1, 6)$, as shown in Figure 1(b). The nearest point in that bucket is point B. As can be seen from the circle centered at A and passing through B, no other bucket can have any point closer to A than B. Therefore, the search for the nearest-neighbor of A concludes with the retrieval of one bucket. On the other hand, if the query were to find the nearest-neighbor of point B, then bucket $(1, 6)$ would again be retrieved first. Point A is found to be the closest neighbor of point B, however, there are other buckets which can have points closer to B than A, as can be seen from the circle centered at B and passing through A. This circle passes through buckets $(0, 6), (0, 7)$ and $(1, 7)$. Thus it is necessary to retrieve these buckets and look for the nearest-neighbor of B, which in this case is a point in bucket $(1,7)$. Therefore, if the three buckets were assigned to three different disks, they could be retrieved in one parallel disk I/O, as opposed to making three sequential disk I/Os from a single disk.

Two other examples of queries are also shown in Figure 1(b). In searching for the nearest-neighbor of point C, the nearest point in its bucket is point D. The circle to point D passes through four other buckets - $(3,2),(2,3),(3,4)$ and $(4,3)$, which are the direct neighbors of bucket $(3,3)$. Thus the direct neighbors could be retrieved in parallel to speed up the search. Note that it is not necessary to retrieve the indirect neighbors (buckets $(2,2),(4,2),(2,4)$ and $(4,4)$) of the bucket containing point C because the distance to D is smaller than the shortest distance to any of these buckets). In the case of point E, no other point is found in bucket $(5,5)$. Thus, all direct neighbors are first searched – i.e. buckets $(4,5),(5,4),(5,6)$ and $(6,5)$. These can be retrieved in parallel if they have been declustered. Since the nearest-neighbor found from these buckets is point F, it is necessary to retrieve all indirect neighbors too, which could also be fetched in parallel. (Note that $(0,6)$ and $(1,7)$ are direct neighbors and $(0,7)$ is an indirect neighbor.) Alternatively, all direct and indirect neighbors could be retrieved at the same time.

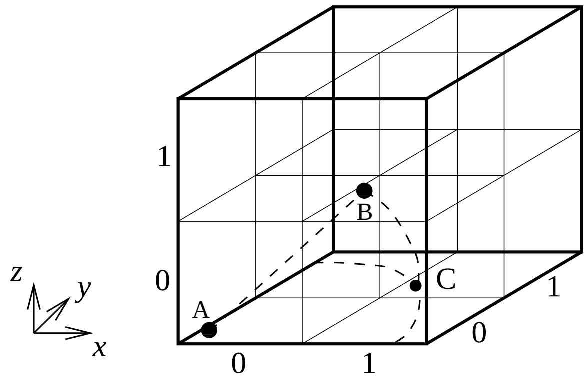


Figure 2: Example for 3 dimensional data

From the example of point B, it can be seen that it may be necessary to retrieve the direct and indirect neighbors in one step. In higher dimensions, the likelihood of such access increases. Consider the example shown in Figure 2, where A is the query point, and B is a point in the same bucket as A and is the closest such point. In three dimensions, we need to search all buckets that intersect the sphere centered at A and passing through B. The radius of this sphere can be large enough to makes it necessary to search the indirect neighbors as well as the direct neighbors. This can be seen from the arc shown in the figure corresponding to the intersection of the $x-y$ plane and the sphere. Thus it would be beneficial to retrieve all the direct and indirect neighbors of $(0,0,0)$ in parallel. Note that it may be necessary to examine buckets which are not just direct or indirect neighbors, depending upon the points already retrieved. If the query is to find the

Symbol	*Meaning*
M	Number of Disks
d	Number of Dimensions
N_i	Number of Buckets in Dimension i
x_i	Coordinate of Bucket in Dimension i

Table 1: Meaning of symbols used

k nearest-neighbors, then the circle (or hyper sphere) that determines which buckets should be examined is the one that passes through the current kth nearest-neighbor.

3 Related Work

In this section we review the declustering schemes that have been proposed in the literature. The first four schemes discussed have been developed for optimizing range and partial match queries. The last scheme has been designed specifically for nearest-neighbor queries. In the following discussion, it is assumed that the domain in each dimension is divided into several parts, thus yielding a collection of buckets, e.g., if each dimension of a 5-dimensional space is divided into four parts, we get $4^5 = 1024$ 5-dimensional buckets. In the following, we will use N_i to denote the number of buckets in dimension i. If the number of buckets in each dimension is the same, then N will be used to represent this number. Each bucket is identified by a set of coordinates: $(x_0, x_1, \cdots, x_{d-1})$ for a d-dimensional space, where each coordinate, x_j is in the range $[0, N_j - 1]$ since dimension j is divided into N_j parts. Also, M represents the number of disks over which the buckets are to be declustered, and d is the number of dimensions. The meaning of each symbol is summarized in Table 1.

The Disk Modulo or DM approach [DS82] allocates bucket $(x_0, x_1, \cdots, x_{d-1})$ to disk

$$(\sum_{j=0}^{d-1} x_j) \bmod M.$$

The Fieldwise eXclusive or FX approach [KP88] allocates the same bucket to disk

$$(b_0 \oplus b_1 \oplus \cdots \oplus b_{d-1}) \bmod M$$

where b_j is the binary representation of x_j and $\oplus$ represents the bitwise exclusive-OR operator. The HCAM approach [FB93], allocates the bucket to disk

$$\mathit{hilbert\_order}(x_0, x_1, \cdots, x_{d-1}) \bmod M$$

where the function $\mathit{hilbert\_order}()$ returns the Hilbert sequence of the input coordinates. The Hilbert sequence maps a multidimensional space into a linear order. Figure 3 shows the allocation generated for a 2-dimensional space with $N = 8$ and $M = 5$, i.e., eight buckets in each dimension and 5 disks. Figure 3(a) shows the allocation generated by the DM scheme, Figure 3(b) shows the allocation generated by the FX scheme and Figure 3(c) shows the allocation for HCAM. The order generated by the hilbert curve is also shown by the directed lines beginning at the top left corner.

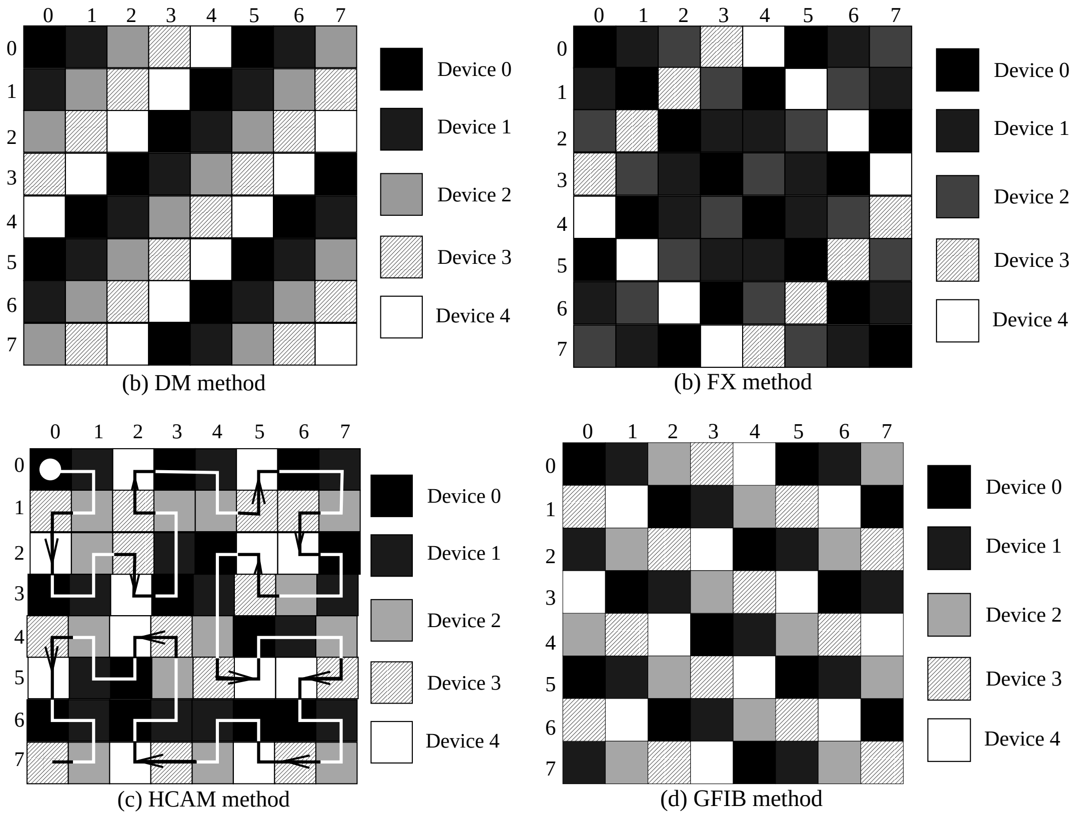


Figure 3: Allocation generated by existing methods

The Cyclic allocation schemes [PAAE98], defined only for two dimensions, allocate bucket (x_0, x_1) to disk

$$(x_0 + x_1 * H) \bmod M$$

where the value of H is appropriately chosen to ensure good declustering. Different values of H, ranging from 1 through $M-1$, produce different allocation schemes. Each of these is called a Cyclic scheme. The DM allocation method is also a Cyclic scheme, where $H = 1$. The value of H, also called the skip value, is the key factor determining the performance of the Cyclic scheme. A good choice of H depends entirely upon the value of M. Three different techniques for determining an appropriate choice of H are described in [PAAE98]. The key idea is that H should be relatively prime with respect to M and $H \neq 1$. One of the techniques for determining the value of H is called Generalized Fibonacci or GFIB. It is based upon Fibonacci numbers and is an extension of a declustering scheme proposed earlier [CLRS86] for improving the performance of raster graphics displays. The original scheme is restrictive because it limits the number of disks to odd order Fibonacci numbers. The GFIB scheme works as follows. Let us first assume that M is a Fibonacci number. The best H value is found to be the previous Fibonacci number. Generalizing this idea to other values of M, we determine the "index", r, of M in the Fibonacci sequence and choose H to be a relatively prime value close to the Fibonacci number corresponding to the index $r-1$. Note that in order to allow arbitrary values of M, we need to move from integral indices in the Fibonacci sequence to non-integral indices. The explicit closed form formula for Fibonacci numbers and integral indices provides a convenient vehicle for this:

$$F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}}$$

where ϕ is the golden ratio, $\frac{1+\sqrt{5}}{2}$ and $\hat{\phi}$ is its complement, $\frac{1-\sqrt{5}}{2}$. The allocation generated by GFIB is shown in Figure 3(d).

The relative performance of the above schemes for range queries in two dimensions is studied in [PAAE98]. The effectiveness of declustering achieved by each scheme is measured as follows. For a query which retrieves A buckets, an optimal declustering would result in no more than $\lceil \frac{A}{M} \rceil$ buckets being allocated to any one of the M disks. This therefore represents the optimal or minimum cost for the query. The cost for evaluating the query with a particular declustering scheme is given by the largest number of buckets retrieved from a single disk. The ratio of this cost to the optimal cost is taken to be a measure of the quality of the declustering for that query. This ratio is averaged over all possible queries to yield a single ratio for each scheme and value of M. An overall ratio of 1 indicates that the declustering is optimal. The higher the ratio, the poorer the declustering. Figure 4 shows the results for the above schemes. The x-axis gives the number of disks and the y-axis gives the ratio that reflects the performance of the query. It is clear to see that the GFIB Cyclic schemes outperforms all the other schemes for all values of M for range queries.

In [BBB+97], an allocation scheme for declustering buckets to optimize nearest-neighbor queries is presented. This scheme requires that each dimension be divided into exactly two parts, thus the value of each coordinate is either 0 or 1. Each bucket can therefore be identified by a binary number, where each bit identifies the bucket in each dimension. The authors argue that a nearest-neighbor query in high-dimensions needs to access direct and indirect neighbors of the bucket containing the query point. Therefore, the goal of their allocation scheme is to ensure that any two buckets that are direct or indirect neighbors of each other, are allocated to different disks. They define such an allocation to be near-optimal. It is shown that DM, FX and HCAM do not achieve near-optimal declustering. Given a bucket $(x_0, x_1, \cdots, x_{d-1})$, the [BBB+97] scheme assigns it to disk

$$\left(\bigoplus_{i=0}^{d-1} \begin{cases} i+1 & \text{if } x_i = 1 \\ 0 & \text{otherwise} \end{cases} \right)_{10}$$

where $\oplus$ is the bitwise exclusive-OR operator. This allocation guarantees near-optimal declustering, assuming that there are enough disks available. The number of disks required is given by $2^{\lceil \log_2(d+1) \rceil}$.

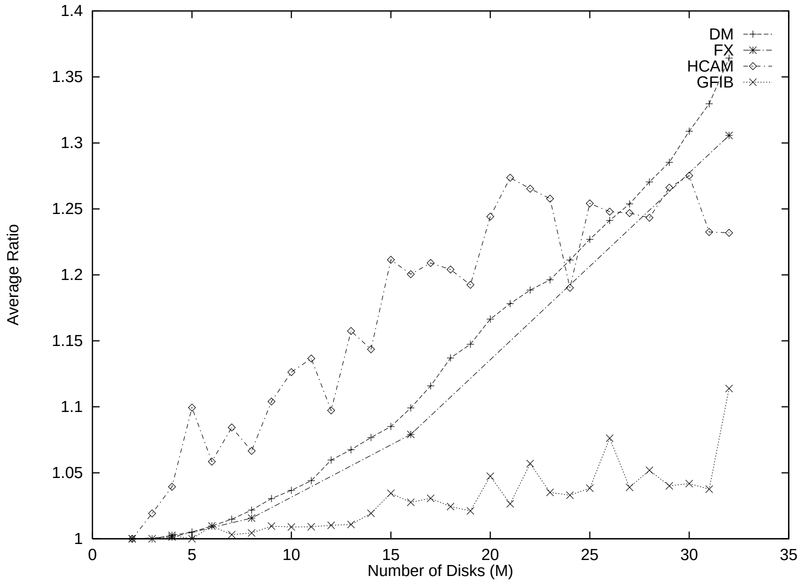


Figure 4: Relative performance for range queries

In the rest of the paper, we will therefore call their scheme *Near-optimal Declustering, or NoD*. If there are less disks than the required number, then higher disk numbers are mapped to lower numbers. However, if there are more disks available then the required number, the NoD approach does not use the extra disks – thus the extra disks are wasted.

Through experiments, it is shown in [BBB+97] that the performance of 10 nearest-neighbors queries is improved by almost a linear factor in the number of disks through declustering using NoD. The HCAM approach is able to achieve only a two fold improvement using 16 disks. Therefore, the NoD approach gives almost a factor of 5 improvement over HCAM for 16 disks. The experiments reported are for 15 dimensions, and up to 16 disks. Note that 16 disks are sufficient to ensure a near-optimal declustering for a 15-dimensional space using NoD declustering. The success of NoD is attributed to the near-optimal declustering achieved. In Figure 5, the ability of the two approaches to achieve near-optimal declustering is shown. The x-axis gives the number of disks and the y-axis gives the total number of pairs of buckets that are direct or indirect neighbors and are allocated to the same disk. Therefore, a count of 0 implies that near-optimal declustering has been achieved. The number of dimensions for the experiment is 15. It can be seen that NoD achieves near-optimal declustering as soon as 16 disks are available. HCAM, on the other hand, does not achieve near-optimal declustering even for 32 disks. In fact, we expect that HCAM will not achieve near-optimal declustering for even much larger numbers of disks.

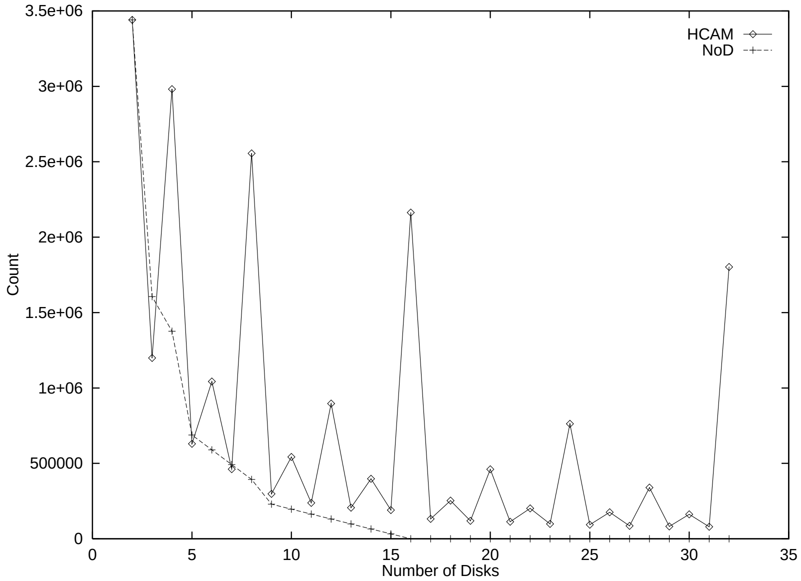


Figure 5: Comparison of HCAM and NoD for 15 Dimensions

4 Cyclic declustering for similarity search

In this section we develop a new disk allocation method for optimizing the performance of nearest-neighbor queries based upon Cyclic allocation. The Cyclic schemes developed in [PAAE98] were optimized for declustering two-dimensional data for range or partial match queries. The first step is extends the schemes to make them applicable to more than two dimensions. The 2-dimensional allocation scheme can be viewed as allocating bucket (x_0, x_1) to disk $(x_0 * H_0 + x_1 * H_1) \bmod M$, where $H_1 = 1$. Now the extension to multiple dimensions is simply to use H values for each dimension. Thus bucket $(x_0, x_1, \cdots, x_{d-1})$, is allocated to disk

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M$$

where the values of $H_0 \cdots H_{d-1}$ range from $1 \cdots M-1$. The key to the performance of the scheme, as in the case of range queries, is in the judicious choice of values for the skips, $H_0 \cdots H_{d-1}$.

As was observed in Section 2, a nearest-neighbor search often involves the retrieval of direct neighbors, indirect neighbors or both direct and indirect neighbors. Therefore, it is important that direct and indirect neighbors be declustered across the disks. Intuitively, direct neighbors differ by ± 1 in one dimension, therefore they will be allocated to disks that are $\pm H_j$ apart, if they differ in dimension j. Thus if all the H_j values are non-zero, no two direct neighbors will be allocated to the same disk. Similarly, indirect neighbors differ by ± 1 in two dimensions. Thus they would be allocated to disks that are $(\pm H_j \pm H_k) \bmod M$ apart, if they differ in dimensions j and k. Therefore, if no two H_i values add up to or differ by M or a multiple of M, then no two indirect neighbors will be allocated to the same disk. Lemmas 1 and 2 establish these statements more formally.

Lemma 1 *If all the skip values in a Cyclic scheme are non-zero (modulo M), then no two direct neighbors are allocated to the same disk.*

Proof: Let any bucket $(x_0, x_1, \cdots, x_{d-1})$, be allocated to disk D. Therefore:

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M = D$$

A direct neighbor of this bucket must have exactly the same coordinates in $d-1$ dimensions, and must differ by ± 1 in one dimension. Let the differing dimension be j. Therefore, the direct neighbor will be allocated to disk:

$$D' = (x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1} \pm H_j) \bmod M$$

Using the following well known properties of the modulo operator:

$$(a+b) \bmod c = ((a \bmod c) + (b \bmod c)) \bmod c$$

$$(a-b) \bmod c = ((a \bmod c) - (b \bmod c)) \bmod c$$

$$(a+b) \bmod c = a \Rightarrow b \equiv 0 (\bmod c)$$

we see that $D' = (D \pm H_j) \bmod M$. Therefore, D' can be equal to D if and only if $H_j \equiv 0(\bmod M)$. Since we ensure that no value of H is equal to 0 (modulo M), we are assured that no two direct neighbors will be allocated to the same disk. □

Lemma 2 *If neither the sum nor the difference of any two skip values in a Cyclic scheme equals 0 (modulo M), then no two indirect neighbors are allocated to the same disk.*

Proof: Let any bucket $(x_0, x_1, \cdots, x_{d-1})$, be allocated to disk D. Therefore:

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M = D$$

An indirect neighbor of this bucket must differ in only two coordinates by ± 1. Let the differing dimensions be j and k. The indirect neighbor is therefore allocated to disk:

$$D' = (x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1} \pm H_j \pm H_k) \bmod M$$

i.e. $D' = (D \pm H_j \pm H_k) \bmod M$. Using the above properties of the modulo operator, we once again see that $D' = D$ if and only if either $H_j + H_k \equiv 0(\bmod M)$ or $H_j - H_k \equiv 0(\bmod M)$. (Note that the other two cases are also covered because $a \not\equiv 0(\bmod M) \Rightarrow (-a) \not\equiv 0(\bmod M)$. Since such a choice of skip values is not allowed by the lemma, we see that no two indirect neighbors will be allocated to the same disk. □

These two lemmas provide the sufficient and necessary conditions for a Cyclic allocation to provide near-optimal declustering. Note however, that in order to obtain d skip values, a minimum number of disks is required. We now show that for d-dimensions, $2d$ disks are sufficient.

Theorem 1 *For a d-dimensional space, it is possible to find d values that satisfy Lemmas 1 and 2 if* $2d$ *disks are available.*

Proof: Pick the skip values to be $1, 2, \cdots, M/2$. Since none of these values are equal to 0, or larger than M, Lemma 1 is satisfied. Since all H values (with the exception of $M/2$) are smaller than $M/2$, no two can add up to M. Also, since there is only one value of $H = M/2$, and all other values are smaller than $M/2$, no two H values can add up to M or more. Since all values are different no two values can have a difference of 0. Since all values are positive and less than M, no two can have a difference greater than or equal to M. Therefore Lemma 2 is also satisfied. Thus with this choice of H values, no two direct or indirect neighbors are allocated to the same disk. □

The Cyclic approach requires a choice of skip values. The specific values chosen result in different performance. We now develop a Cyclic based allocation method that achieves near-optimal declustering. Based upon Lemmas 1 and 2 and Theorem 1, the Cyclic approach for near-optimal declustering is as follows. For d-dimensional data, allocate bucket $(x_0, x_1, \cdots, x_{d-1})$ to disk

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M$$

The choice of the skip values is made as follows:

- If $d \leq M/2$, choose the values to be $1, 2, \cdots, d$.
- If $M/2 < d < M$, choose the first $M/2$ skips to be $1, 2, \cdots, M/2$, then choose the next $d - M/2$ skips to be $M/2 + 1, \cdots, M-1$.
- Otherwise, pick the first $M-1$ skips to be $1, 2, \cdots, M-1$, and then cycle through the same values again.

From Lemmas 1 and 2, it is clear that this scheme guarantees near-optimal allocation. We could design other Cyclic schemes by choosing different skip values so as to achieve other goals. For example, if the goal is to ensure maximum parallelism for direct and indirect neighbors (i.e. *all* direct and indirect neighbors of any bucket are on different disks), in addition to the earlier conditions, we choose skip values such that no two skip values add up to or differ by a third skip value and that the sum or difference of no two different pairs of skip values is the same. Naturally, this may require more than $2d$ disks.

Figure 6 shows the disk requirements of the Cyclic and NoD approaches. As can be seen, the NoD approach has a lower requirement than the Cyclic approach. A lower bound on the number of disks required by any approach that achieves near-optimal declustering is $d+1$. This follows from the following argument. Consider a bucket B. In order to allocate direct neighbors of B to different disks, it is necessary to have at least 2 disks - one disk to which B is allocated and at least one other to which B's direct neighbors are allocated. Every pair of direct neighbors of B, are also indirect neighbors of each other (this is because any two direct neighbors differ from B in one dimension,

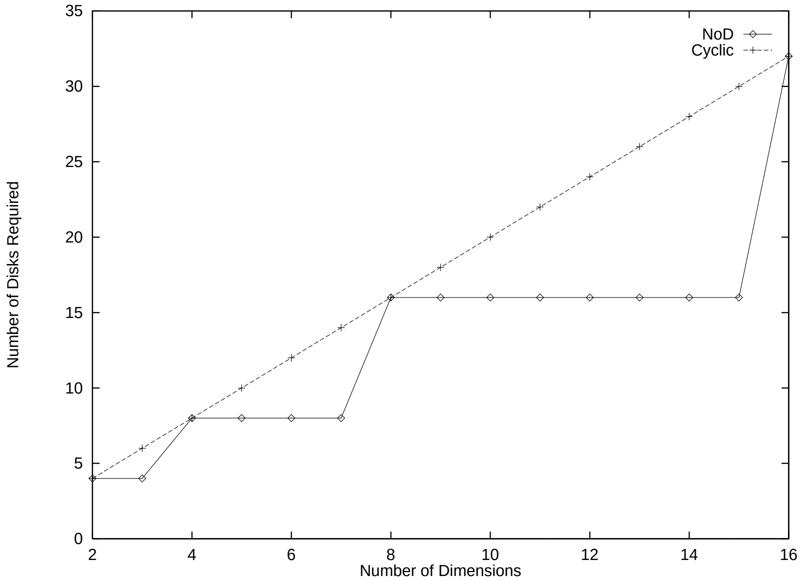


Figure 6: Disk requirements for achieving near-optimal declustering

therefore they differ from each other in two dimensions). Note that we are limiting this discussion to the case $N = 2$ because NoD is not defined otherwise. Therefore no two direct neighbors of B can be allocated to the same disk. The lower bound on the number of disks required to achieve near-optimal declustering is therefore $d+1$, since there are d direct neighbors of each bucket. Thus both approaches are within a factor of 2 of the lower bound. Moreover, the bound for Cyclic is a tight upper bound for NoD.

It should be noted that although NoD requires fewer disks for achieving near-optimal allocations, it does not make use of more disks even if they are available. Thus any extra disks are wasted instead of being used to improve the performance of the nearest-neighbor query through greater parallel I/O. The Cyclic scheme, on the other hand, makes use of all available disks, even when there are more disks than are needed to achieve a near-optimal declustering ($2d$).

5 Experimental Evaluation

In this section we analyze the performance of the new Cyclic allocation scheme for nearest-neighbor queries relative to the other schemes. We start by using a metric that measures the degree of near-optimal declustering since in [BBB+97] achieving near-optimal declustering was the main criterion for predicting the performance of an allocation scheme. However, achieving near-optimal declustering in itself does not accurately reflect the degree of parallelism achieved. Gains from parallelism are maximized when buckets that are retrieved together are spread among all available disks as uniformly as possible. We therefore define a new criterion for comparing the quality of declustering for direct and indirect neighbors. Using any declustering scheme, the cost of accessing all the direct and indirect neighbors together is given by the maximum number of buckets that are retrieved from the same disk. The average of this cost over all buckets is a measure of the quality of the declustering. The lower this average, the better the performance. A lower bound on the cost of any declustering scheme is given by $\lceil \frac{A}{M} \rceil$, where A is the total number of direct and indirect neighbors of the bucket. This measure is a modification of the measure used for comparing declustering schemes for range queries. A similar metric can be used for any set of buckets that are accessed together, such as direct neighbors only or doubly indirect neighbors and indirect neighbors, etc. The evaluations are based upon both criteria:

1. *Counts* - this is a measure of how far the allocation is from being a near-optimal allocation, as defined in [BBB+97]. The count for an allocation scheme is the sum of the count for each bucket. The count for a bucket is the number of direct and indirect neighbors of the bucket that are allocated to the same disk as the bucket itself. Thus a lower count indicates better declustering of direct and indirect neighbors. A count of 0 indicates near-optimal declustering.

2. *Cost* - as explained above, this measure is similar to the standard measure used for evaluating the performance of declustering schemes for range queries. Given a set of A buckets to be retrieved in parallel, the cost of the retrieval is given by the maximum number of buckets retrieved from a single disk. The optimal cost with M disks is given by $\lceil \frac{A}{M} \rceil$. The cost over all buckets is averaged to yield an overall cost. We define five different cost measures for each scheme.

   - Direct cost - this represents the average cost for retrieving all the direct neighbors of a bucket in parallel.
   - Indirect cost - this represents the average cost for retrieving all the indirect neighbors of a bucket in parallel.
   - Doubly Indirect cost - this represents the average cost for retrieving all the doubly indirect neighbors of a bucket in parallel.
   - Direct + Indirect cost - this represents the average cost for retrieving all the direct and indirect neighbors of a bucket in parallel.
   - Direct + Indirect + Doubly Indirect cost - this represents the average cost for retrieving all the direct, indirect and doubly indirect neighbors of a bucket in parallel.

   Note that other cost measures could be used (such as those involving triply indirect neighbors) but, as seen earlier, the above five represent the most common sets of buckets that are retrieved together for nearest-neighbor queries.

In the remainder of this section, the performance of the various schemes based upon the above metrics is presented. In Subsection 5.1 the counts measure is used to compare the ability of all the schemes to produce near-optimal declustering. In Subsection 5.2, the direct, indirect and direct + indirect cost metrics are the basis of the comparison of all schemes. The purpose is to determine the improvement in parallel I/O achieved by each scheme using multiple disks. In Subsection 5.3, the doubly indirect and the direct + indirect + doubly indirect cost are used to study how the two schemes, NoD and Cyclic, decluster sets of neighbors for which they are not explicitly optimized. In the first two subsections, the performance for 2-way partitioning is presented first, followed by 4-way partitioning. For 2-way partitioning, results for 8 dimensions as well as 15 dimensions are shown. These two values are chosen because they represent the case where NoD and Cyclic have very different disk requirements (15 dimensions) and the case where they have the same disk requirements (8 dimensions).

5.1 Achieving near-optimal declustering

We begin by studying how effective the various schemes are in achieving near-optimal allocations. For this purpose, we compare the performance of the various schemes under the counts metric. First, the case of 2-way partitioning is discussed. The performance of all the schemes for 15 dimensions is presented in Figure 7. FX has extremely poor performance, in fact it performs as though there were only two disks, irrespective of how many disks are available. The reason for this is that each coordinate can be either 0 or 1 since this is 2-way partitioning. Therefore, the value of the exclusive-OR can be only 0 or 1, hence only these two disks are ever used. DM also has poor performance, which doesn't change beyond 3 disks. The constant behavior of DM is due to the fact that any two buckets which have the same number of coordinates equal to 1, will be assigned to the same disk. Therefore, all direct or indirect neighbors whose coordinates are permutations of the same numbers will always be allocated to the same disk. This count remains constant as long as M is larger than 3. For lower values, even more direct or indirect neighbors have to be allocated to the same disk. HCAM has very variable performance, depending largely upon the value of M. It is good for odd values of M but poor for even values, particularly, for powers of 2. NoD is able to achieve near-optimal declustering with as few as 16 disks. The performance of the Cyclic approach is very close to that of NoD, except that it takes 30 disks for Cyclic to achieve near-optimal declustering. This difference depends upon the number of dimensions. In Figure 8, the performance of all schemes is shown for 8 dimensions. Here we see that NoD and Cyclic both generate near-optimal allocations with 16 disks. In general, for 2-way partitioning, NoD is able to achieve near-optimal declustering with fewer disks than Cyclic. Other than this difference, Cyclic and NoD have performance very similar to each other.

Now we consider the case where the partitioning is not 2-way. In Figure 9 the performance of all schemes is shown for 4-way partitioning with 8 dimensions. We note that the performance of FX has improved since the coordinates now have two bits, thus up to four disks are useful. The performance of HCAM follows the same pattern as for 2-way partitioning. The Cyclic scheme gives

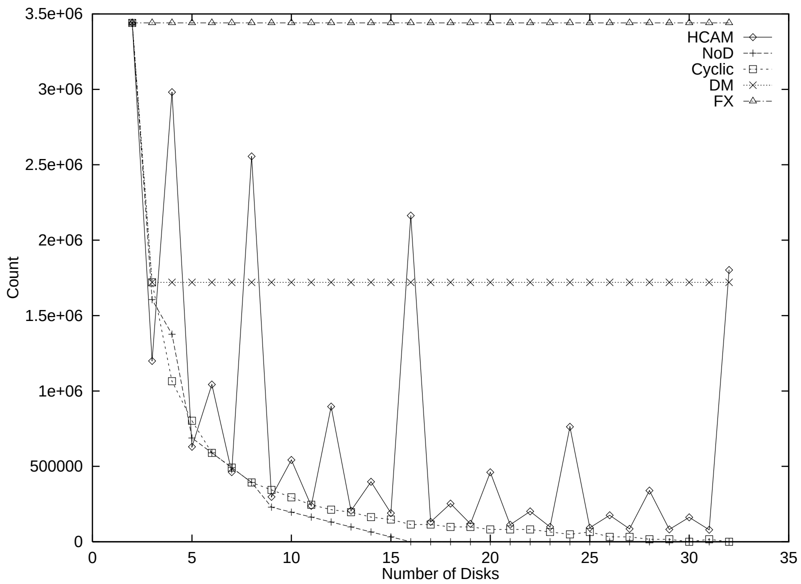


Figure 7: Counts for 15 dimensions, 2-way partitioning

the best performance achieving near-optimal declustering with 16 disks. The NoD scheme is not defined beyond 2-way partitioning.

5.2 Performance improvement through parallel I/O

In this subsection the performance of the various schemes in terms of achieving parallel I/O for direct and indirect neighbors is presented. For this purpose, the *Cost* metrics are used to compare the schemes. In each graph the lower bound that can be achieved by any allocation scheme is shown. For two-dimensional range queries, it has been shown that strictly optimal declustering can be achieved only in very rare cases [AE97]. We expect that for nearest-neighbor searching too, in general, optimal declustering will be achievable only in very few cases. The case of 2-way partitioning is discussed first. In Figure 10, the performance of the various schemes for 2-way partitioning with 15 dimensions is shown. The graph on the left shows all schemes and the graph on the right shows only NoD and Cyclic. The FX and DM schemes, as expected, do not give good performance. The performance of HCAM is again highly variable. The performance of NoD and Cyclic is interesting. Both schemes obtain significantly higher levels of declustering of direct and indirect neighbors consistently. Cyclic achieves better declustering than NoD for most values of M. It is interesting to note that even for small values of M, Cyclic outperforms NoD. This was not the case with the counts metric. Therefore, we see that although NoD achieves a near-optimal allocation with fewer disks, Cyclic results in a lower parallel access cost. The performance of Cyclic keeps improving as the number of disks increases. NoD, on the other hand, shows no improvement in parallelism beyond 16 disks. This is because it does not utilize more than 16 disks irrespective

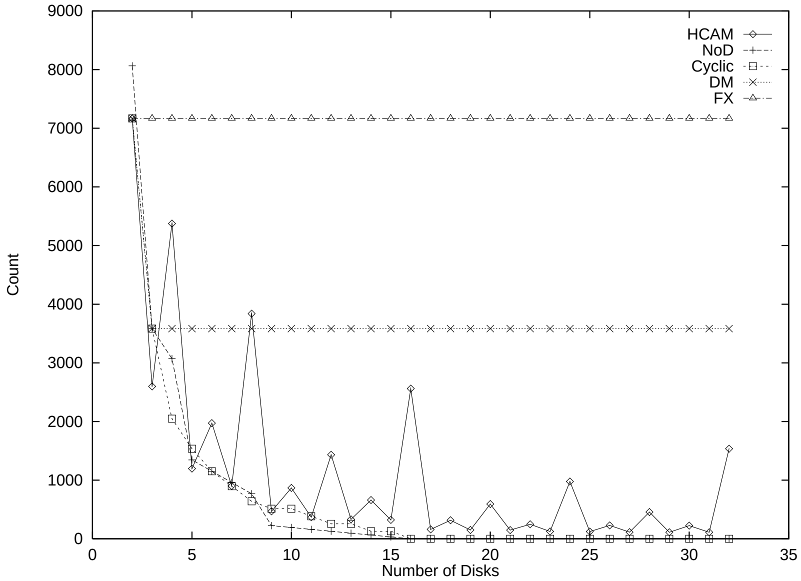


Figure 8: Counts for 8 dimensions, 2-way partitioning

of how many disks are available. It should be pointed out that for 16 disks, the performance of NoD becomes optimal at which point it saturates, i.e. the performance doesn't improve. It continues to achieve better performance than Cyclic for 17, 18, 19 and 20 disks. However, after that point, Cyclic's performance continues to improve and outperforms NoD. The behavior of NoD and Cyclic for 8 dimensions and 2-way partitioning is shown in Figure 11(a). Their relative behavior is unchanged, except that NoD does not achieve optimal declustering and Cyclic outperforms NoD more often than for 15 dimensions. From these graphs we conclude that the Cyclic approach is most effective in increasing parallel I/O when both direct and indirect neighbors are retrieved concurrently.

Let us now consider the declustering when only direct or only indirect neighbors are retrieved. Figure 11(b) gives the performance of NoD and Cyclic (the other schemes have performance very similar to that in the earlier tests) for 2-way partitioning and 8 dimensions for retrieving direct neighbors only. Figure 12 shows the performance of all the schemes for 2-way partitioning and 15 dimensions based only on direct neighbors. Similarly, Figure 13 shows the results when only indirect neighbors are retrieved for 15 dimensions. The performance of FX, DM and HCAM is almost the same as that for both direct and indirect neighbor retrieval. For direct neighbors only, we see that NoD is able to achieve optimal declustering as soon as there are 16 disks. Similarly, Cyclic is able to achieve optimal declustering as soon as 16 disks (for 8 dimensions) or 30 disks (for 15 dimensions) are available. This goes to show that if there are less than $2d$ disks, NoD gives better declustering for direct neighbors than Cyclic. For indirect neighbors however, the two schemes give performance similar to their performance for both direct and indirect neighbors, i.e.

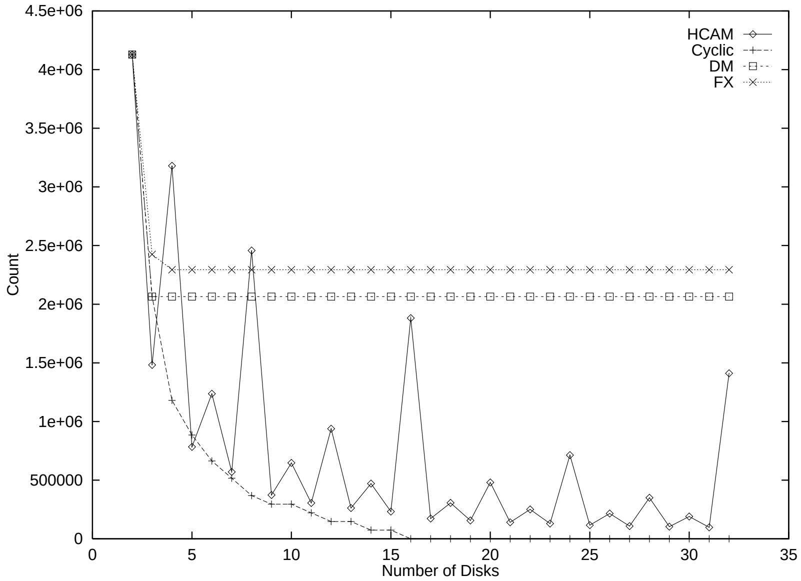


Figure 9: Counts for 8 dimensions, 4-way partitioning

Cyclic has better performance for most values of M. Note that optimal performance for direct and indirect neighbors can be achieved by using a different cyclic scheme which ensures that all direct and indirect neighbors of the same bucket are allocated to different disks, as was mentioned in Section 4.

Now we examine the behavior of the schemes for 4-way partitioning based upon the cost for direct and indirect neighbors. The performance of the schemes for 8 dimensions is given in Figure 14. The relative performance of all schemes is very much the same as that for a 2-way partitioning. Note however that the NoD scheme is not applicable because there are more than two buckets in each dimension. Cyclic is clearly the approach that gives the best performance consistently.

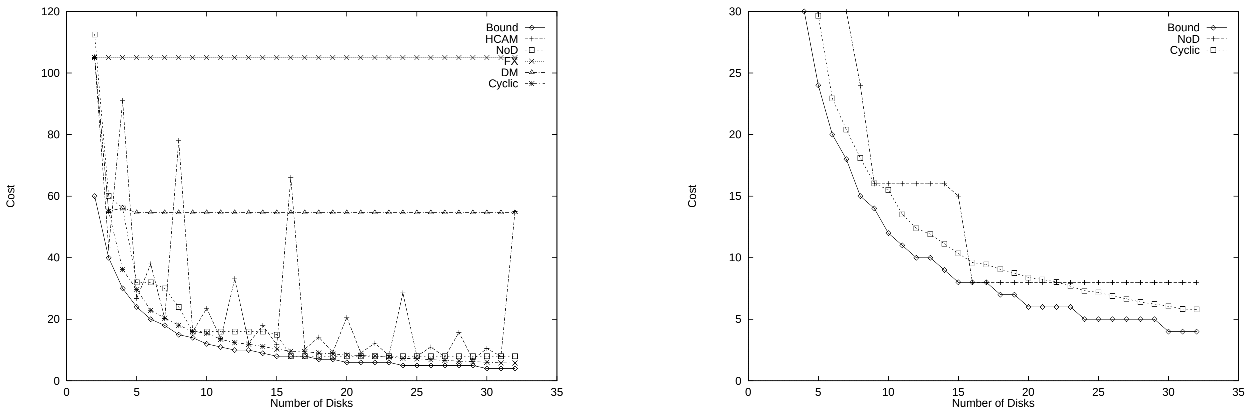


Figure 10: Cost for direct and indirect with 15 dimensions, 2-way partitioning

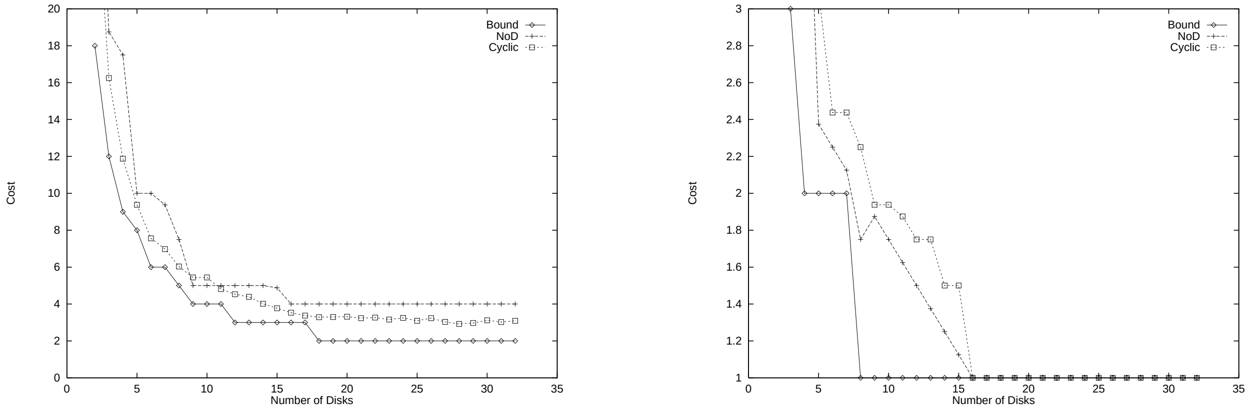


Figure 11: Cost for (a) direct and indirect and (b) direct with 8 dimensions, 2-way partitioning

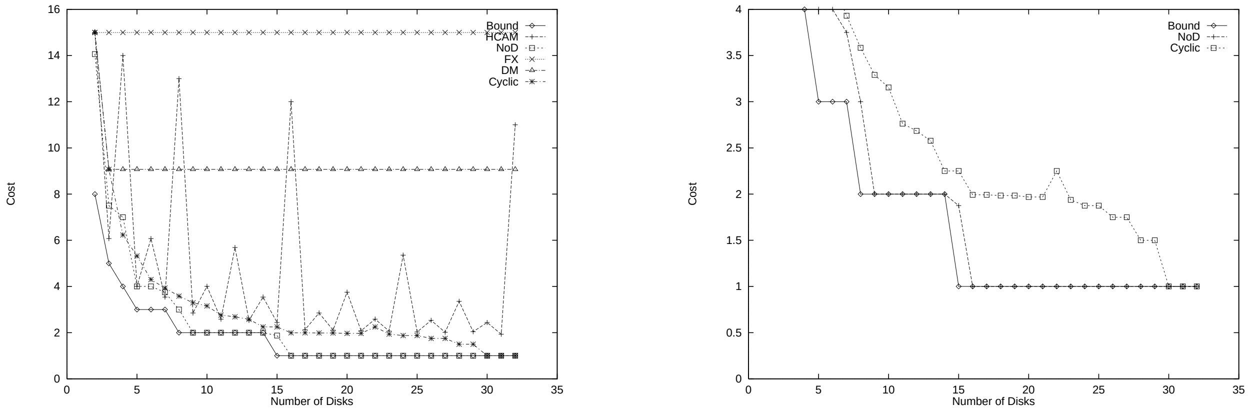


Figure 12: Cost for direct with 15 dimensions, 2-way partitioning

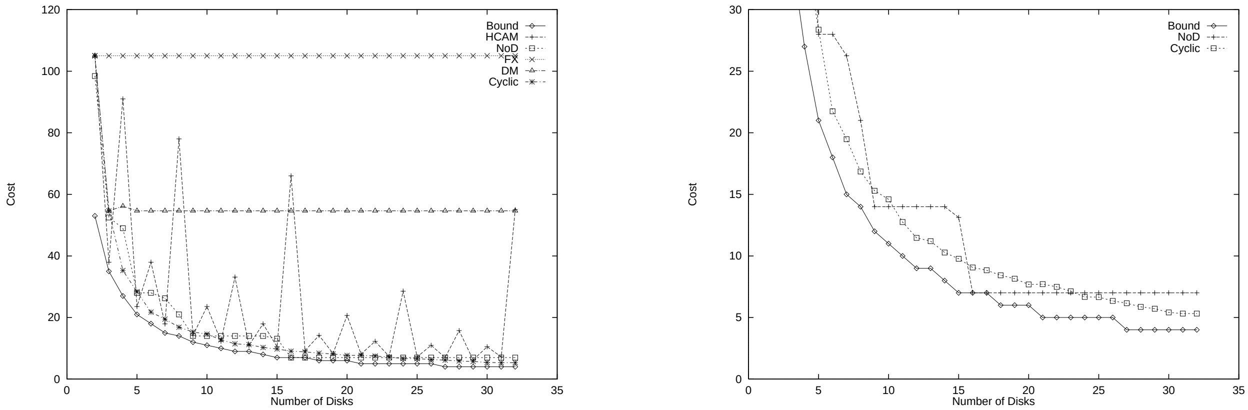


Figure 13: Cost for indirect with 15 dimensions, 2-way partitioning

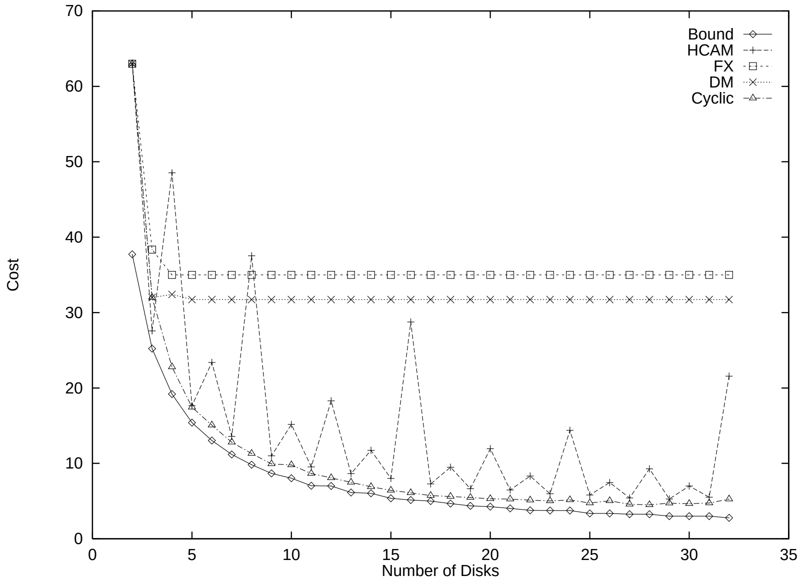


Figure 14: Cost for direct and indirect with 8 dimensions, 4-way partitioning

Although for odd values of M, HCAM has much better performance than for even values, it does not perform better than Cyclic. The superior performance of Cyclic is seen also when only direct or only indirect neighbors are considered, as seen in Figures 15(a) and 15(b).

From these results, it is clear that the new Cyclic approach achieves greatest parallelism when direct and indirect neighbors are retrieved together. Also, when only indirect neighbors are retrieved, Cyclic gives the greatest parallelism. For the case of retrieving only direct neighbors, NoD gives better declustering when there are few disks, and both schemes achieve optimal declustering with sufficient disks.

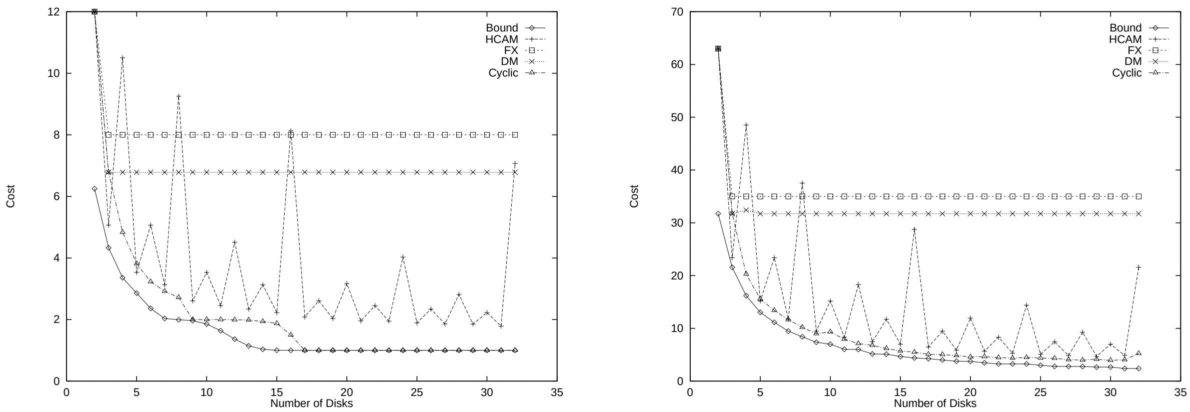


Figure 15: Cost for (a) direct and (b) indirect with 8 dimensions, 4-way partitioning

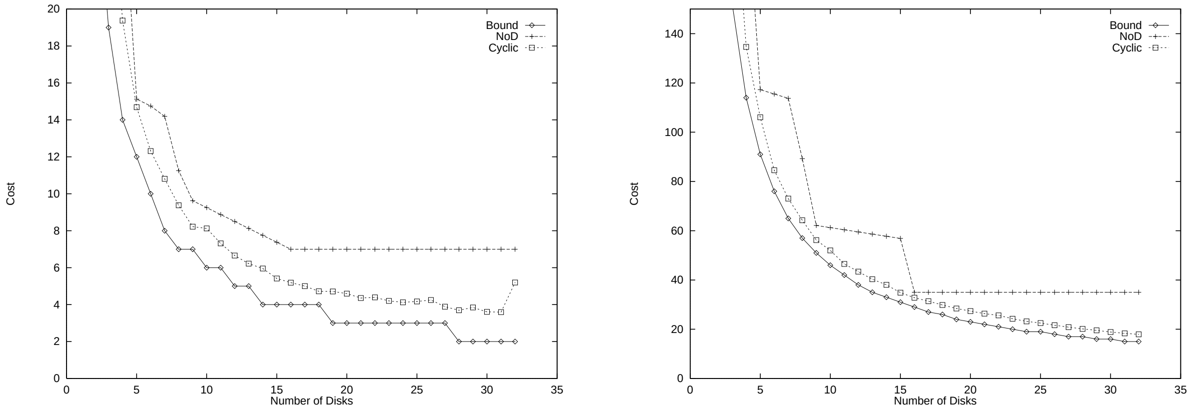


Figure 16: Cost for doubly indirect with (a) 8 dimensions and (b) 15 dimensions

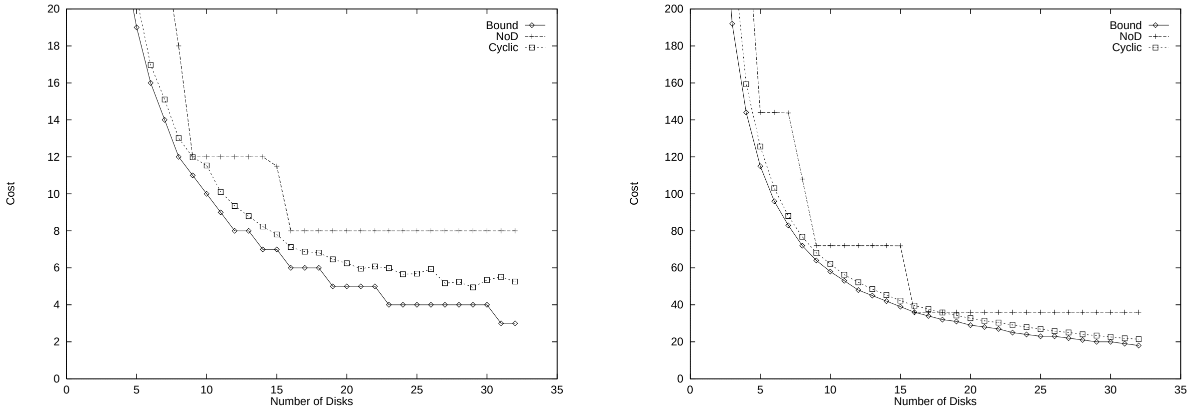


Figure 17: Cost for direct, indirect and doubly indirect with (a) 8 dimensions and (b) 15 dimensions

5.3 Declustering doubly indirect neighbors

In this subsection, the ability of the NoD and Cyclic schemes to decluster doubly indirect neighbors is studied. The goal is to show that the Cyclic approach achieves better declustering even for doubly indirect neighbors, for which neither scheme attempts to optimize explicitly. Both schemes attempt to ensure that no two direct or indirect neighbors are allocated to the same bucket. We compare the schemes using two cost measures. The first is the cost for retrieving only doubly indirect neighbors together and the second is the cost for retrieving direct, indirect and doubly indirect neighbors together. In Figure 16, the costs for retrieving only doubly indirect neighbors is shown for both NoD and Cyclic. The graph on the left is for 8 dimensions and the graph on the right is for 15 dimensions. The tests are for 2-way partitioning. In Figure 17, graphs for accessing direct, indirect and doubly indirect neighbors are shown. Note that the performance of the other schemes is similar to that seen earlier and is not shown to improve readability of the graphs. The Cyclic scheme achieves much better declustering than NoD except for a couple of points in Figure 17.

Once again, the performance of NoD remains constant beyond 16 disks. The Cyclic scheme shows consistent and regular improvement tracking the lower bound as the number of disks increase. Based upon these results we expect that for even higher levels of indirect neighbors, the Cyclic approach will give even better relative performance.

6 Concluding Remarks

The problem of efficient similarity searching is becoming important for databases as non-textual information is stored. The problem reduces to one of finding nearest-neighbors in high-dimensional spaces. In this paper, a new disk allocation method for declustering high-dimensional data to optimize nearest-neighbor queries is developed. The new scheme, called Cyclic allocation, is simple to implement and is a general allocation method in that it imposes no restrictions on the partitioning of the data space. Furthermore, it exploits the availability of any number of disks to improve performance. Finally, by varying the skip values the method can be adapted to yield allocations that are optimized for various criteria.

We demonstrate the superior performance of the Cyclic approach compared to existing schemes both those that were originally designed for range queries (FX, DM and HCAM) as well as those designed specifically for nearest-nieghbors (NoD). The FX and DM schemes are found to be inappropriate for nearest-neighbor queries. HCAM performs reasonably well for odd numbers of disks, but extremely poorly for even numbers. NoD was found not to achieve as much parallelism as Cyclic for most cases, except when retrieving only direct neighbors with a small number of disks. NoD also has the potential to give better performance for some dimensions when the number of disks is close to that required to achieve near-optimality. On the other hand, NoD is restricted to 2-way partitioning of each dimension, and its cost remains the same even when more disks beyond those required for near-optimal declustering are available. This results in a saturation of the gains produced by NoD beyond this point. In contract, the Cyclic approach is not restricted to 2-way partitioning and makes use of all available disks. In fact, its cost tracks the lower bound and reduces as the number of disks increases. Overall we observe that the Cyclic scheme gives the best performance for nearest-neighbor queries more consistently than any other scheme. Given the success of the cyclic schemes for two-dimensional range queries [PAAE98], and the flexibility for nearest-neighbor queries, we expect that it will give good performance for systems that require both types of queries.

References

[AE93] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal disk allocation for partial match queries. *Proc. ACM Symp. on Transactions of Database Systems*, 18(1):132–156, March 1993.

[AE97] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimensional data. In *International Conference on Database Theory*, pages 409–418, Delphi, Greece, January 1997.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th Int. Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[BBB+97] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Arizona, U.S.A., 1997.

[BKK96] S. Berchtold, D. A. Keim, and H. P. Kreigel. The X-tree: An index structure for high-dimensional data. In *22nd. Conference on Very Large Databases*, pages 28–39, Bombay, India, 1996.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangl es. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[CDN+97] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, A. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *IEEE International Conference on Advances in Digital Libraries, ADL*, pages 70–81, Washington, D.C., 1997.

[CLRS86] B. Chor, C. E. Leiserson, R. L. Rivest, and J. B. Shearer. An application of number theory to the organization of raster-graphics memory. *Journal of the Association for Computing Machinery*, 33(1):86–104, January 1986.

[DS82] H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Transactions of Database Systems*, 7(1):82–101, March 1982.

[FB93] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18 – 25, San Diego, CA, Jan 1993. IEEE Computer Society.

[FM89] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM Symp. on Principles of Database Systems*, pages 253–258, 1989.

[GD90] S. Ghandeharizadeh and D. J. DeWitt. A multiuser performance analysis of alternative declustering strategies. In *Proc. Int. Conf. Data Engineering*, pages 466–475, Los Angeles, California., February 1990.

[GD92] S. Ghandeharizadeh and D. J. DeWitt. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 29–38, San Diego, 1992.

[Gun89] O. Gunther. The design of the cell tree: An object-oriented index structure for geometric databases. In *Proc. Int. Conf. Data Engineering*, pages 598–605, 1989.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[HNP95] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 562–573, September 1995.

[Jag91] H. V. Jagdish. A retrieval technique for similar shapes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 208–217, 1991.

[KP88] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 173–182, Chicago, 1988.

[KS91] C. Kolovson and M. Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 138–147, 1991.

[LS90] D. B. Lomet and B. Salzberg. The hb-tree: A multi-attribute indexing method with good guaranteed performance. *Proc. ACM Symp. on Transactions of Database Systems*, 15(4):625–658, December 1990.

[LSR92] J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 3–14, Vancouver, Canada, August 1992.

[MM96] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–42, August 1996.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, February 1993.

[PAAE98] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *International Conference on Data Engineering (To Appear)*, Orlando, Florida, Feb 1998. IEEE Computer Society. Copy available at http://www.cs.ucsb.edu/TRs/TRCS97-08.ps.

[Rob81] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[WJ96] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. Int. Conf. Data Engineering*, pages 516–523, 1996.