Declustering Schemes with Guaranteed Performance

Chung-Min Chen
Telcordia Technologies
(formerly Bellcore)
Morristown, NJ 07960
chungmin@research.bellcore.com

Randeep Bhatia
Bell Laboratories
Murray Hill, NJ 07974

Rakesh K. Sinha
Bell Laboratories
Murray Hill, NJ 07974

Abstract

Declustering schemes reduce response time of multidimensional range queries by distributing the data blocks across multiple disks. Previous declustering methods are mostly heuristic-based and provide neither guarantee nor asymptotic analysis of the performance. In this paper we seek declustering schemes with guaranteed worse deviation from the optimal. We devise an $O(M^4)$ algorithm, where M is the number of disks, to find the optimal (in worse case sense) scheme in the class of recently proposed cyclic declustering *schemes. We also propose a new scheme based on* golden ration sequences, *which can be obtained in $O(M \log M)$. We show by analysis that, for 2-dimensional data, the new scheme has a worst case response time within a factor* 3 *of the optimal for any query, and within a factor* 1.5 *of the optimal for large queries. Furthermore, we show that the actual performance of the new scheme is even better than the theoretical bounds: it is at most 1 more than the optimal for $M \leq 22$, and is at most 4 more than the optimal for $M \leq 550$.*

Keywords: declustering, multidimensional range queries, parallel I/O.

1. Introduction

Range queries are a common class of queries in many applications, including multidimensional databases, image and GIS applications. In these applications, the underlying dataset is organized as a multidimensional grid, where each point in the grid is a data block. A range query retrieves a rectangular subset of the multidimensional data space. To speed up the response time of the query, declustering schemes are usually used. A declustering scheme distributes data blocks among multiple disks, so they can be retrieved in parallel.

Various declustering schemes have been proposed in the literature, which include those based on some "mapping" functions [6, 11, 7, 16, 4, 9] and those transform the problem into a graph theory problem with heuristic solutions [8, 14, 15, 3, 13]. While graph-based methods produce good performance, they bear much higher computing complexity than the mapping-based schemes. Yet, to the best of our knowledge, there is no evidence that the graph-based methods provide better performance than the best mapping-based schemes (for example [16]) for uniform grid files. Graph-based methods are more suited for non-uniformly distributed datasets. Among the mapping-based schemes, the recently proposed Cyclic Declustering scheme was shown to outperform previously known schemes [16, 17].

All of the previous schemes aim at minimizing "average query response time". The only exception is [5], which seeks the minimum number of disks needed if a response time of 1 is to be guaranteed for all queries of *round* shape. In this paper, we focus on declustering schemes with guaranteed worse case performance for rectangular range queries on two-dimensional (2D) data.

1.1. Problem Definition

Throughout the paper we assume a 2D grid of size $N_x \times N_y$ (i.e., N_x columns and N_y rows) and M disks. Each point in the grid is called a *cell*, which is a data block to be stored in one of the M disks. Let $\mathcal{N}_x = \{0, 1, \ldots, N_x - 1\}$ (and $\mathcal{N}_y$ be defined similarly). Then a declustering scheme is defined as a function $f : \mathcal{N}_x \times \mathcal{N}_y \rightarrow \{0, 1, \ldots M - 1\}$. That is, f assigns cell (x, y) to disk number $f(x, y)$.

A (rectangular) range query is denoted as $Q = [(x, y), l_x, l_y]$, where (x, y) is the coordinate of the southwest corner cell of Q, and l_x and l_y are the horizontal and vertical side lengths of Q, respectively. Let $f_i(Q)$ be the number of cells in Q that get assigned to disk number i by scheme f, i.e. $f_i(Q) = |\{f(x, y) = i | (x, y) \in Q\}|$. Then the *response time of Q under declustering scheme f* is defined as

$$RT(f, Q) = \max(f_0(Q), f_1(Q), \ldots, f_{M-1}(Q)).$$

We may simply use $RT(Q)$ when f is clear from the context. Tentatively, $RT(Q)$ is interpreted to be the largest number of disk seeks performed by any of the M disks. Since seek time is the dominant cost in disk I/O, $RT(Q)$ serves as a good indication of the elapsed time to answer Q.

It is not hard to see that the shortest possible response time for any query Q is given as $ORT(Q) = \lceil \frac{|Q|}{M} \rceil$, which we call the *optimal response time* of Q. A *strictly optimal* scheme is one that produces optimal response time for all queries. It is known that strictly optimal scheme does not exist in general, except under some stringent conditions (e.g. when $M = 1, 2, 3$ or 5 or $M \geq N_x N_y$) [1]. Let $DEV(f, Q) = RT(f, Q) - ORT(Q)$. Then the problem can be stated as follows:

Given N_x, N_y, and M, find the scheme f that minimizes

$$MAXD(f) = \max_{\text{all } Q} DEV(f, Q).$$

No non-exhaustive search solutions are known for the problem, which is very likely to be intractable (whether it is NP-complete, or NP-hard, remains an open problem). Thus, instead of finding the global optimal solution, we seek schemes with approximate performance.

In Section 2, we show how to find, in $O(M^4)$ time, the optimal Cyclic Declustering (CD) scheme, in the sense of worst case performance. In Section 3, we propose a new scheme based on Golden Ratio Sequences (GRS), which gives very good bound on $MAXD(f)$. The GRS scheme has the advantage that it can be constructed with much lower complexity ($O(M \log M)$) than the search overhead of the CD schemes. More interestingly, the bound on $MAXD(f)$ is independent of the grid size; it depends only on M. Therefore, the best scheme can be chosen solely based on the number of disks and applied to any size of dataset. In Section 4, we compare the worst case performance of CD and GRS schemes. We give conclusions in Section 5.

2. Optimal Cyclic Declustering Scheme

Definition:([16]) A *Cyclic Declustering* (CD) scheme is a scheme that maps cell (x, y) to disk number $(x + h \cdot y) \bmod M$, where h, called the *skip value*, is some constant between 1 and $M - 1$ that is relatively prime to M. We use CD(h) to denote a CD scheme that uses skip value h.

Cyclic schemes are actually a special case of the Generalized Disk Modulo scheme [6], which maps cell (x, y) to disk number $(ax + by) \bmod M$, where a and b are some constants. Intuitively, a CD scheme works as follows: It starts by assigning to the first row (row 0) the disk numbers $0, 1, \ldots, M-1, 0, 1, \ldots, M-1, \ldots$ Then the disk numbers assigned to row i is obtained by shifting those assigned to row $i - 1$ by h positions to the left. The property that h is relatively prime to M ensures that any M consecutive cells in any column also contain M distinct disk numbers.

Prabhakar et al. [16] proposed two methods to find the best skip value h with the objective of minimizing the average query response time. One method (called GFIB) is based on a heuristic that uses Fibonacci number. The basic idea is that if $M = F_i$ (the i'th Fibonacci number), then choose $h = F_{i-1}$. Some approximation is used if M is not an exact Fibonacci number. While this heuristic computes the skip value quickly, it does not always give close to optimal performance. The other method (called EXH) is simply an exhaustive search that evaluates all possible values of h. EXH gives close to optimal performance but the search overhead is expensive. For each value of h, there are a total of $C_2^{N_x+1} \cdot C_2^{N_y+1}$ queries to evaluate. For each query of side lengths l_x, l_y, it takes $O(l_x \cdot l_y)$ to compute the response time. Thus, the total complexity of a brute force EXH implementation is $O(M N_x^3 N_y^3)$. Usually, $N_x, N_y > M$ and thus the complexity could be far larger than $O(M^7)$.

In the following, we describe a $O(M^4)$ procedure that, given M disks, finds the optimal CD scheme f for which $MAXD(f)$ is the least among all CD candidates.

Lemma 1: Given a CD scheme $f = CD(h)$ and a query $Q = [(x, y), l_x, l_y]$. Let $q = [(0, 0), l_x \bmod M, l_y \bmod M]$. Then

$$RT(f, Q) - ORT(Q) = RT(f, q) - ORT(q).$$

Proof: See Appendix A. □

The lemma suggests that $MAXD(f)$ can be obtained by considering all possible values of q. Since there are M^2 such queries and it takes $O(M^2)$ to compute the response time of each query, one can find $MAXD(f)$ in $O(M^4)$, regardless the grid side lengths N_x and N_y. However, this amounts to an $O(M^5)$ algorithm as there could be as many as $M - 1$ different skip values to consider (when M is a prime). In the following, we describe a dynamic programming technique that finds the response time of all possible

queries q in $O(M^3)$ and thus reduces the total complexity to $O(M^4)$.

Fixing a scheme f and given a coordinate (x, y), define *disk distribution vectors* (DDV)

$$D1(x,y) = [f_0(q_1), f_1(q_1), \ldots, f_{M-1}(q_1)], \text{ and}$$

$$D2(x,y) = [f_0(q_2), f_1(q_2), \ldots, f_{M-1}(q_2)],$$

where $q_1 = [(0,0), x, y]$ and $q_2 = [(x,0), 1, y]$. It is not hard to verify the relationships:

$$\begin{aligned} D1(x,y) &= D1(x-1,y) + D2(x,y), \text{ and} \\ D2(x,y) &= D2(x,y-1) + \mathrm{I}_{f(x,y)}, \end{aligned}$$

where $\mathrm{I}_{f(x,y)}$ is a vector $[a_0, a_1, \ldots, a_{M-1}]$ such that $a_i = 1$ if $i = f(x,y)$, and $a_i = 0$ otherwise. We use $\max(D1(x,y))$ to denote the maximum component in the vector.

Based on the relationships, we may compute $D2(x,y)$, $D1(x,y)$, and subsequently $RT(x,y) = RT(q_1) = \max(D1(x,y))$, for all $0 \le x, y \le M-1$, in a column major order. An insight look at the above expressions reveals that, in practice, D_1 can be maintained in an array of M entries (instead of an $M \times M$ matrix), where each entry holds a disk distribution vector of length M. Similarly, D_2 can be maintained in a variable that holds a disk distribution vector (instead of an array of M disk distribution vectors).

The detailed algorithm is listed in Figure1. The algorithm has a time complexity of $O(M^3)$, because each step in the nested for-loop requires $O(M)$ and a total of M^2 iterations are executed. It requires $O(M^2)$ space as both variables $RT[\,]$ and $D1[\,]$ require $O(M^2)$ space. The following theorem summarizes the results.

Theorem 2: Given M disks, we may find, in $O(M^4)$ time, the optimal CD scheme f such that $MAXD(f) \le MAXD(f')$, for all CD schemes f'. The scheme f guarantees that for all query Q, $RT(f,Q) - ORT(Q) \le MAXD(f)$, regardless of the grid size.

3. Golden Ratio Declustering Schemes

Cyclic declustering schemes require that the shift distance (i.e., the skip value) between two consecutive rows be a constant. The Golden Ratio declustering scheme is a scheme that allows the shifts to be variable, as long as any M consecutive cells in any column remain a permutation of $0, 1, \ldots, M-1$. For any M, the algorithm for obtaining the golden ratio sequence (GRS) and the

```
Algorithm Compute_Response_Time
Input:
    int h;  // skip value
    int M;  // number of disks
Output:
    int RT[0..M-1,0..M-1];
    // RT[x,y]= response time of query [(0, 0), x, y]
Type:
    DDV int[0..M-1];  // disk distribution vector
Variable:
    DDV D1[0..M − 1];
    DDV D2;
Function:
    f(x, y) = (x + h · y) mod M;
    // CD scheme with skip value h
Begin
    for y = 0 to M − 1
        D1[y] = [0, 0, ..., 0];
        // initialize to zero vector
    for x = 0 to M − 1
    {
        D2 = [0, 0, ..., 0];
        for y = 0 to M − 1
        {
            D2 = D2 + I_f(x,y);
            D1[y] = D1[y] + D2;
            RT[x, y] = max(D1[y]);
        }
    }
End
```

Figure 1: Algorithm for computing table $RT[x,y]$

golden ratio declustering scheme (GRS scheme)is as follows.

Step-1: construct M pairs (i, key_i) for $0 \le i < M$, where key_i is the fractional part of $\frac{2i}{(1+\sqrt{5})}$.

Step-2: sort the first components based on key values. This will give a permutation on $0, 1, .., M-1$. This is because $\frac{2}{(1+\sqrt{5})}$ is an irrational number and hence the keys are all distinct. (In practice, we get the same effect, for all practical values of M, by using 32 bit precision arithmetic.) Call the resulting permutation GRS(M).

Step-3: compute the inverse permutation, GRS^{-1} by:

$$\text{for } i = 0 \text{ to } M-1 \ \{GRS^{-1}(GRS(i)) = i\}$$

Step-4: the GRS-declustering scheme maps point (x,y) to disk $(x - GRS^{-1}(y \bmod M)) \bmod M$.

Example 1: Consider $M = 6$, then

$$GRS(M) \ = \ 0,\ 5,\ 2,\ 4,\ 1,\ 3$$

$$GRS^{-1}(M) \;=\; 0,\ 4,\ 2,\ 5,\ 3,\ 1$$

The figure below shows the result of applying the GRS scheme to a 9 by 9 grid (the southwest corner cell is taken to be the origin (0,0)).

4	5	0	1	2	3	4	5	0
2	3	4	5	0	1	2	3	4
0	1	2	3	4	5	0	1	2
5	0	1	2	3	4	5	0	1
3	4	5	0	1	2	3	4	5
1	2	3	4	5	0	1	2	3
4	5	0	1	2	3	4	5	0
2	3	4	5	0	1	2	3	4
0	1	2	3	4	5	0	1	2

Unlike cyclic declustering schemes, GRS schemes are more amenable to analysis due to its base on golden ratio sequences, for which many properties have been known (e.g. [12, 10]). It can be proved that when M is a Fibonacci number, the GRS scheme has a very good behavior (worst case response time within a factor of 3 and average response time within 14%). Our simulation results indicate that the performance of GRS schemes varies smoothly as a function of M. So these analytical results provide strong evidence that the GRS scheme behaves very well for *all* values of M. We believe that these analytical bounds are not tight, and simulation shows the actual performance is far better than these theoretical guarantees.

The following theorem gives a worst case bound on the response time of a query.

Theorem 3: Let M be a fibonacci number. Then the response time of any query Q is at most *three* times its optimal response time. If both dimensions of Q are at least M in length, then the response time of Q is at most 1.5 times its optimal response time. In general, if Q has $r_1M + r_2$ rows and $c_1M + c_2$ columns, where $r_1, c_1 \geq 1$ and $0 \leq r_2, c_2 < M$. Then the response time of Q is at most $1 + \frac{2}{r_1 c_1 + 3}$ times its optimal response time.

Proof: See Appendix B. □

We will now state the bound on average response time. There are several (equally reasonable) ways to define average response time of a declustering scheme and how far it is from the optimal response time. We found by simulation that the relative performance of any two declustering schemes remains the same under all these metrics. For our analysis, we picked the following metric:

Definition: For any given dimension $c \times r$, we consider the highest response time of a $c \times r$ query. Then we sum these response times over all possible values of c and r. Similarly, we compute the sum of optimal response times for all these queries. The average response time of a declustering scheme is defined as the ratio of the two sums. Clearly, this metric is at least one, and a good declustering scheme should have this metric close to one.

Since we are averaging over the set of all possible queries, the average response time depends on the size and shape of the grid. We already know (from Theorem 3) that the GRS scheme is nearly optimal (even in a worst case measure) for large queries (side-lengths $\geq M$), so we are mainly interested in analyzing average performance of GRS over small and medium queries. Including larger queries will only improve the average response time. Based on above observations, we pick $N_x = N_y = M$, and analyze the average response time of the GRS scheme for all queries in this grid.

Theorem 4: Let M be a Fibonacci number. Then the average response time of GRS-clustering scheme for an $M \times M$ grid is at most 1.14.

Proof: We refer interested readers to [2] as the proof is too detailed to cover in this paper. □

Finally, we can find the metric $MAXD(f)$ for any GRS scheme f in a way similar to that for the cyclic schemes. Let f be any GRS scheme, then it is not hard to see that Lemma 1 still holds if we substitute $q = [(0,0), l_x \bmod M, l_y \bmod M]$ with $q = [(0, y \bmod M), l_x \bmod M, l_y \bmod M]$. Since there are M^3 such queries, we can find $MAXD(f)$ in $O(M^4)$ using a similar dynamic programming technique.

Theorem 5: Given M disks, the corresponding GRS scheme f can be obtained in $O(M \log M)$. We may also find $MAXD(f)$, the worse response time deviation from the optimal among all queries, in $O(M^4)$. The bound $MAXD(f)$ is independent of the grid size (it depends only on M).

4. Performance Comparison

We compare the worst case performance of the CD schemes found by the GFIB and EXH heuristics and the GRS schemes. Figure 2 shows the result. We varied the number of disks, x, from 1 to 550. However, instead of plotting $y = MAXD(f, x)$, the figure plots, for each x, $y = \max_{2 \leq m \leq x} MAXD(f, m)$. This results in

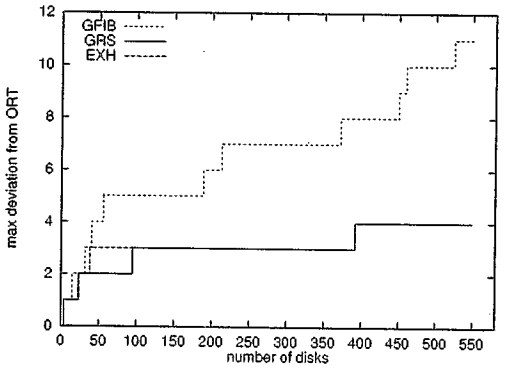


Figure 2: Worst case performance comparison of GRS and GFIB. The step functions are plotted according to the cost metric $y = max_{2 \le m \le x} max\_dev(m)$.

a step function that is more amenable to visual examination (The original figure that plots $y = MAXD(f, x)$ is difficult to read as the curves oscillates and cross-pass each other in some ranges of x).

The figure shows that, for any query Q, GRS guarantees a response time within $ORT(Q) + 1$ when $M \le 22$, $ORT(Q) + 2$ when $M \le 94$, $ORT(Q) + 3$ when $M \le 391$, and $ORT(Q) + 4$ when $M \le 550$ (we stopped the evaluation at $M = 550$ due to the long running time for large M). The results also suggest that the theoretical bound on the worst case ratio to the optimum (Theorem 3) —$3 \cdot ORT$—shall occur only for small values of ORT. Indeed, we found that, when $M \le 550$, it occurs only when $ORT = 1$.

The GFIB scheme, while giving good performance when M is a Fibonacci number, produces less efficient performance than GRS on non-Fibonacci numbers overall. As the figure shows, its deviation from the optimum grows at a rate faster than that of GRS, and reaches as high as 11 when $M \ge 525$. For EXH, it produces a response time at most $ORT(Q) + 3$ when $M \le 223$. We do not have results for $M > 223$, due to EXH's excessive running time to find the best skip value. Comparing GRS to EXH, we note that GRS schemes can be obtained efficiently for large M (finding the Golden Ratio Sequence takes only $O(M \log M)$ in time). Even for the range of number of disks ($M \le 223$) where we managed to compute the skip values for EXH scheme, GRS performs no worse than EXH in terms of maximum deviation from the optimum.

5. Conclusion

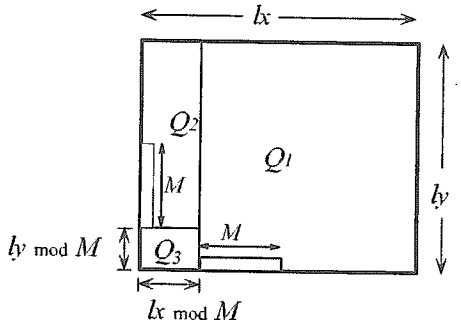
We look at the declustering problem for 2D range queries. Unlike previous work, we seek declustering schemes that minimize worse deviation from the optimal. In the class of cyclic declustering schemes, we show how the optimal schemes (in the worst case sense) can be found in $O(M^4)$, given M disks and regardless of the grid size. We also propose the golden ratio scheme, which can be easily constructed in $O(M \log M)$. We prove analytically that whenever M is a Fibonacci number, the golden ration scheme has close to optimal performance. Our simulation shows that the performance of the golden ratio scheme varies smoothly with M, which is a strong evidence that our scheme has good behavior for all values of M.

Of most importance, our results show that both schemes provide very good worse case performance guarantee. In particular, the golden ratio scheme guarantees that, for any query, the deviation from the optimal response time is at most 1 when $M \le 22$, at most 2 when $M \le 94$, at most 3 when $M \le 391$, and at most 4 when $M \le 550$.

Appendix

A. Proof of Lemma 1

We divide Q into three areas as shown in the figure below.



It is not hard to see that Q_1 and Q_2 can be partitioned into "stripes" of length M (as shown in shaded area) , where each stripe contains M different disks under the Cyclic Declustering scheme and thus contributes 1 to the response time. Thus,we have

$$\begin{aligned} RT(Q) &= RT(Q_1) + RT(Q_2) + RT(Q_3) \\ &= \lfloor \frac{l_x}{M} \rfloor l_y + \lfloor \frac{l_y}{M} \rfloor (l_x \bmod M) + RT(Q_3) \end{aligned}$$

Similarly,

$$ORT(Q) = \lceil \frac{|Q_1| + |Q_2| + |Q_3|}{M} \rceil$$

$$= \lfloor \frac{l_x}{M} \rfloor l_y + \lfloor \frac{l_y}{M} \rfloor (l_x \bmod M) + ORT(Q_3)$$

Also, it is easy to see that $RT(Q_3) = RT(q)$ (as the response time of a query under CD depends only on its side lengths) and $ORT(Q_3) = ORT(q)$ (as they have the same size). Therefore,

$$\begin{aligned} RT(Q) - ORT(Q) &= RT(Q_3) - ORT(Q_3) \\ &= RT(q) - ORT(q) \end{aligned}$$

B. Proof of Theorem 3

We will only give a sketch of the proof, due to space limitation. Interested readers may contact the first author for a full version of the paper with complete proofs.

First, we establish some properties on the permutation of golden ratio sequences. In general, given a permutation σ over $\{0, 1, 2, \ldots M-1\}$, we can define the σ-declustering as mapping point (i, j) to disk $(i - \sigma^{-1}(j \bmod M)) \bmod M$. The inverse permutation σ^{-1} is constructed from σ the same way that GRS^{-1} is constructed from GRS (see Section 3). It is not hard to see that the GRS scheme is a σ-declustering scheme. It turns out that the performance of a σ-declustering scheme is closely related to the distribution of elements in σ.

Definition: Let R be any interval (i.e., set of consecutive numbers) in $\{0, 1, 2, \ldots, M-1\}$. R can be a "wrap-around" interval, e.g. $[M-3, M-2, M-1, 0, 1]$. The *gaps* of R in σ are defined as following: mark the positions of elements of R in σ. For each of the $|R|$ marked positions, count the number of elements to the next marked position. For the last marked position, compute its distance to the first marked position assuming that the sequence σ is cyclic. These $|R|$ numbers are defined as the gaps of R in σ.

Example:
Let $M = 13$, $\sigma = 0, 5, 10, 2, 7, 12, 4, 9, 1, 6, 11, 3, 8$ and $R = \{11, 12, 0, 1\}$. Then the marked positions (of R in σ) are $\{0, 5, 8, 10\}$, and the gaps are $\{5, 3, 2, 3\}$. It is easy to see that the sum of all the gaps is always equal to M.

The following theorem states that the response time of Q under σ-declustering scheme can be characterized in terms of gaps of R in σ.

Theorem 6: Consider a range query Q with r rows and c columns, where $r, c \leq M$. Let R be the set of row-indices of Q, and let $\{g_1, g_2, \ldots, g_r\}$ be the set of gaps of R in σ. Sort the gaps in non-decreasing order so that $g_1 \leq g_2 \cdots \leq g_r$. Then the response time of Q under σ-declustering scheme is at most the maximum value of j such that $g_1 + g_2 + \cdots + g_j \leq c$.

What the theorem is saying is that in order to minimize the response time of Q, we want a permutation σ such that most of the gaps for Q are reasonably large. Finding such a permutation is a very well studied problem in mathematics and theoretical computer science [12, 10]. The GRS sequences are such permutations based on Fibonacci numbers and golden ratios.

Definition: The Fibonacci numbers are defined as follows: $F_0 = 0, F_1 = 1$, and recursively $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$. (The sequence is $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$). The *golden ratio*, $\frac{2}{\sqrt{5}+1} \approx 0.618$, is the limit of $\frac{F_{k-1}}{F_k}$.

The following theorem states that when M is a fibonacci number, the GRS sequence on $\{0, 1, \ldots, M-1\}$ has very well distributed gaps.

Theorem 7: [10] Let $M = F_k$, and GRS(M) be the Golden ratio sequence on $\{0, 1, \ldots, M-1\}$. Then for any (possibly wrap-around) interval R (in $[0, M-1]$) of length $F_i + s$, where $0 \leq s \leq F_{i-1} - 1$, the gaps of R in GRS(M) are following:

- s gaps of length F_{k-i} each
- $F_{i-2} + s$ gaps of length F_{k-i+1} each
- $F_{i-1} - s$ gaps of length F_{k-i+2} each

So, while the gaps are not of equal length, they can take at most one of three different values, which are, roughly, within a factor of 2.6 of each other.

The following lemma follows directly from Theorems 6 and 7.

Lemma 8: Fix $M = F_k$ and GRS(M)-declustering scheme. Let Q be any query with $F_i + s$ rows ($0 \leq s \leq F_{i-1} - 1$) and c columns, where both the number of rows and columns are at most M then

$$RT(Q) \leq \begin{cases} \left\lceil \frac{c}{F_{k-i}} \right\rceil \\ s + \left\lceil \frac{c - sF_{k-i}}{F_{k-i+1}} \right\rceil, \quad \text{if } c > sF_{k-i} \\ 2s + F_{i-2} + \left\lceil \frac{c - sF_{k-i} - (F_{i-2}+s)F_{k-i+1}}{F_{k-i+2}} \right\rceil, \\ \quad \text{if } c > sF_{k-i} + (F_{i-2}+s)F_{k-i+1} \end{cases}$$

(Please note that the first bound is larger than the second bound, which, in turn, is larger than the third bound. That is, we have better bounds for larger values of c.)

Finally, Theorem 3 follows from the following three lemmas, which can be proved using Theorems 6, 7 and Lemma 8.

Lemma 9: Let M be a fibonacci number, and Q be a query such that both dimensions of Q are at most M. Then the response time of Q is at most three times its optimal response time.

Lemma 10: Let M be a fibonacci number, and Q be a query such that one dimension of Q is at most M and the other dimension is at least M. Then the response time of Q is at most three times its optimal response time.

Lemma 11: Let M be a fibonacci number, and Q be a query with r_1M+r_2 rows and c_1M+c_2 columns, where $r_1, c_1 \geq 1$ and $0 \leq r_2, c_2 < M$. Then the response time of Q is at most $1+\frac{2}{r_1c_1+3}$ times its optimal response time.

References

[1] K. Abdel-Ghaffar and A. E. Abbadi. Optimal allocation of two-dimensional data. In *Proceedings of the International Conference on Database Theory*, 1997.

[2] R. Bhatia, R. K. Sinha, and C. M. Chen. *Declustering Using Golden Ratio Sequences.* Bell Laboratories, Murray Hill, NJ, Jan 1999. Technical Memorandum (a short version to appear in ICDE 2000).

[3] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: a high-performance remote-sensing database. In *Proc. of 13th Int'l Conf. on Data Engineering*, pages 375–384, Bermingham, U.K., Apr. 1997.

[4] C.M. Chen and R. Sinha. Raster-spatial data declustering revisited: an interactive navigation perspective. In *15th Int. Conf. on Data Engineering*, 1999.

[5] L.T. Chen and D. Rotem. Declustering objects for visualization. In *Proc. of 19th Int'l Conf. on Very Large Data Bases*, pages 85–96, Dublin, Ireland, Aug. 1993.

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

[7] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proc. of 2nd Int'l Conf. on Parallel and Distributed Information Systems*, pages 18–25, San Diego, CA, Jan. 1993.

[8] M.T. Fang, R.C.T. Lee, and C.C. Chang. The idea of declustering and its applications. In *Proc. of 12th Int'l Conf. on Very Large Data Bases*, pages 181–188, Kyoto, Japan, Aug. 1986.

[9] H. Ferhatosmanoglu and D. Agrawal. Concentric hyperspaces and disk allocations for fast parallel range searching. In *Proc. of 15th Int. Conf. on Data Engineering*, pages 608–615, 1999.

[10] A. Itai and Z. Rosberg. A golden ratio control policy for a multiple-access channel. *IEEE Transactions on Automatic Control*, AC-29:712–718, 1984.

[11] M.H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. of 1998 ACM SIGMOD Conference*, pages 173–182, Chicago, IL, Jun. 1988.

[12] D.E. Knuth. *The art of computer programming Vol. 3.* Addison-Wesley, Reading, MA, 1973.

[13] S. Kou, M. Winslett, Y. Cho, and J. Lee. New GDM-based declustering methods for parallel range queries. In *Int'l Database Engineering and Applications Symposium (IDEAS)*, Aug. 1999.

[14] D.R. Liu and S. Shekar. Partitioning similarity graphs: a framework for declustering problems. *Information Systems: An International Journal*, 21(6):475–496, Sep. 1996.

[15] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proc. of 10th Int'l Parallel Processing Symposium*, pages 434–440, Honolulu, Hawaii, Apr. 1996.

[16] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. E. Abbadi. Cyclic allocation of two-dimensional data. In *14th Int'l Conf. on Data Engineering*, pages 94–101, Orlando, FL, Feb 1998.

[17] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A.E. Abbadi. Efficient retrieval of multidimensional datasets through parallel I/O. In *5th Int. Conf. on High Performance Computing*, Dec. 1998.