Pergamon

PII: S0306-4379(96)00024-5

Information Systems Vol. 21, No. 6, pp. 475–496, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/96 $15.00 + 0.00

PARTITIONING SIMILARITY GRAPHS: A FRAMEWORK FOR DECLUSTERING PROBLEMS†

DUEN-REN LIU[1] and SHASHI SHEKHAR[2]

[1]Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
[2]Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA

(Received 27 July 1994; in final revised form 12 August 1996)

Abstract — Declustering problems are well-known in the databases for parallel computing environments. In this paper, we propose a new similarity-based technique for declustering data. The proposed method can adapt to the available information about query distribution (e.g. size, shape and frequency) and can work with alternative atomic data-types. Furthermore, the proposed method is flexible and can work with alternative data distributions, data sizes and partition-size constraints. The method is based on max-cut partitioning of a similarity graph defined over the given set of data, under constraints on the partition sizes. It maximizes the chances that a pair of atomic data-items that are frequently accessed together by queries are allocated to distinct disks. We describe the application of the proposed method to parallelizing Grid Files at the data page level. Detailed experiments in this context show that the proposed method adapts to query distribution and data distribution, and that it outperforms traditional mapping-function-based methods for many interesting query distributions as well for several non-uniform data distributions. Copyright ©1996 Elsevier Science Ltd

Key words: Similarity Graph, Geographic Databases, Declustering, Grid File, Parallel Databases

1. INTRODUCTION

With an increasing performance gap between processors and I/O systems, parallelizing I/O operations by declustering [12, 11, 30] data is becoming essential for high performance applications. Database machines, multi-processors and parallel computers can all benefit from effective declustering.

The declustering problem can be stated as follows: Given a set of atomic data-items, N disks, and a set of queries, divide the set of data items among the N disks, respecting the disk capacity constraints, to minimize response time for the given set of queries. Unfortunately, this problem is NP-complete in several contexts, which include partial match queries on cartesian product files [11] and join queries on a set of relations [30]. Thus any method to solve this problem in polynomial time will be heuristic.

We address the declustering problem in a single processor with a multi-disk environment. We abstract the properties of multi-disk secondary storage systems in terms of their capability of carrying out N-independent disk operations in parallel. The storage system is viewed as a collection of logical disks, each with an independent read/write head and an independent channel to transfer data to/from the processor's memory. Disk block accesses over different logical disks are independent and can be carried out in parallel. Thus the storage system can reduce the response time for large I/O volumes by a factor of N, where N is the number of disks in the system. We focus on I/O cost only. Readers are referred to MAGIC [17] for a more general cost model that includes communication costs. Furthermore, the data items are assumed to be atomic, i.e., a data item will not be split across disks. Data items like records, objects, pages and page-clusters are likely to satisfy this assumption. This assumption excludes strategies such as splitting a data item (e.g. files) across disks.

Several heuristic methods have been proposed that are based on the ideas of mapping functions, similarity and load-balancing. The mapping-function-based techniques have been proposed for k-dimensional and spatial data with partial match queries and range queries. These methods provide

†Recommended by Patrick O'Neil

a mapping function from the domain of data-items to the set of disk-ids, assuming that all data-items and queries are equiprobable. Several single-attribute functions including round robin, hash-partitioning, key-range partitioning [6, 15], and a hybrid of these [16], as well as multi-attribute functions including grid-based multi-dimensional key-range-partitioning [17], disk modulo [8, 27], generalized disk modulo (linear) [8, 35], field-wise-XOR [24], Hilbert [9], error-correcting code [11], latin-square [23, 35], random [3], vector-based declustering [4] and lattice [35] have been proposed and evaluated. A survey of multi-attribute functions can be found in [11, 9]. These methods are limited inxdvi the case of managing updates, non-uniform data-distributions and non-uniform data-sizes. Furthermore, they are limited in their ability to adapt to available information about query distribution and size constraints. Lastly, these techniques are not designed for schema partitioning.

Index-specific load-balancing based declustering methods have been proposed for B-tree [31], R-tree [20] and the temporal index [25], etc. Dynamic file allocation methods are proposed in the FIVE system [33]. These methods are incremental in nature to balance the load (e.g. storage, I/O time) in various partitions for a local window (i.e. a subset of existing data-items in partitions) around the new data-item. The incremental nature of the load-balancing methods allows them to work well with indexing methods (e.g. B-tree, R-tree) in the face of updates, non-uniform data distributions and non-uniform access frequencies to data-items. However, they do not take advantage of query distribution information, beyond looking at the access frequencies of the individual data-items.

Fang et al. [12] introduced the idea of similarity (e.g. the nearest neighbor) for declustering. Two groups G_1 and G_2 are considered similar if, for every point p in G_1, there exists at least one point q in G_2 such that either p is a nearest neighbor of q, or q is a nearest neighbor of p. Minimal spanning trees and shortest spanning paths were proposed in [12] to divide a set of given data-items into two "similar" groups. Similarity for more than two groups is not addressed. Furthermore, it is not obvious that the nearest-neighbor based notion of similarity is appropriate for all kinds of query and data types.

Our Approach and Contributions: We propose a new similarity-based technique which can take advantage of the available information about query distribution, data distribution, data-item size, and constraints on partition size. The declustering scheme is based on max-cut partitioning of a similarity graph that has data-items as nodes. The edges have weights that represent similarity between the end nodes. The similarity between two data-items measures the likelihood that the pair will be accessed together by queries in the query set of interest. Max-cut partitioning maximizes the chances that a pair of data-items that are frequently accessed together by queries are allocated to distinct disks. We show that the proposed technique adapts to query distribution, data distribution and data-size information. We also propose an incremental max-cut method, based on the ideas of max-cut similarity and load-balancing, which is a simple greedy technique best suited for dynamic allocations as well as for static declustering for large data sets.

Theoretical analysis and experiments show the superiority of the proposed method in several contexts. We also describe how the proposed technique can be used to parallelize grid files. Detailed experiments in this context show that the proposed technique can adapt to alternative data distributions and query distributions. It outperforms mapping-function-based techniques for several query distributions on uniformly distributed data sets, and almost always does so for non-uniform data distributions. Experiments also show that the proposed incremental max-cut method outperforms the load-balancing based declustering method and provides the best trade-off between parallel response time for queries and the cost of declustering.

Outline: The remainder of this paper is organized as follows. Section 2 illustrates the basic concepts and presents our scheme. Section 3 describes the heuristic techniques of the proposed scheme. Theoretical analysis of the proposed scheme is presented in Section 4. Section 5 discusses the extension of our method to handle data-items of different sizes. Section 6 presents the application of the proposed method to parallelizing Grid Files. Experiments in this context are also presented. Finally, Section 7 presents the conclusion and suggests future work.

2. SIMILARITY-BASED APPROACH TO DECLUSTERING

Fang et al. [12] introduced the similarity idea for declustering as follows: Given a set of data, partition it into two groups such that these two groups are similar to each other. Two groups G_1 and G_2 are considered similar if, for every point p in G_1, there exists at least one point q in G_2 such that either p is a nearest neighbor of q, or q is a nearest neighbor of p. The similarity between G_1 and G_2 is measured by the number of data-items which have a nearest neighbor in the same group. A small value of this measure is desired for effective declustering.

We propose a new similarity-based approach to the case of multiple disks with possibly different capacities and to the case of non-uniform data sizes. We also generalize the notion of "similarity" to capture the available information about query distribution. This section defines the basic concepts and formulates our similarity-based approach to declustering.

2.1. Basic Concepts and Definitions

In this section, we describe some basic concepts and give the definitions of the parameters of the problem. The symbols and their definitions are listed in table 1.

Symbol	Meaning
N	The number of disks
$w(u, v)$	The relative frequency that u and v are likely to be accessed together
$R(q)$	The set of data-items that qualify for the given query
$\pi(V)$	A partitioning of data set V
E_c	The set of edges $e(u, v)$ whose end points u and v fall in different groups of a partitioning $\pi(V)$
$S(\pi(V))$	The degree of similarity among groups of a partitioning $\pi(V)$ i.e., the sum of the weights on all the edges in E_c
$t(v)$	The number of time units required to retrieve the given data item v
$f(q)$	The relative frequency of the occurrence of the query q
$rt(q)$	The response time of the given query q
T^{π}	The expected query response time under a partitioning $\pi(V)$

Table 1: Symbols and definitions

Definition 1 A *data set* V is a collection of data-items. Each data-item v in V is associated with $size(v)$, representing the storage required to contain it. A *group of data-items from* V is a subset of V. A *partitioning of data set* V, $\pi(V)$, is a collection of mutually disjoint groups, $G_1^{\pi}, G_2^{\pi}, \ldots, G_N^{\pi}$, such that their union is equal to V, i.e., $\cup_{i=1}^{N} G_i^{\pi} = V$, and $G_i^{\pi} \cap G_j^{\pi} = \emptyset$.

Definition 2 A *query-set* $Q_s = \{q_1, q_2, \ldots, q_k\}$ is a set of queries over data-set V. The *query response set* $R(q)$ to a query q is the set of data items that qualify for the given query q. *Query distribution* f is a function that maps query set Q_s to a relative frequency, i.e., $f(q)$ provides the relative frequency of the occurrence of query q.

If all queries are equiprobable, then $f(q) = \frac{1}{|Q_s|}$ for all $q \in Q_s$. We note that query distribution function $'f'$ can be generalized to model the situation where the relative frequencies of query classes (e.g. range query, square-range query or partial-match query) are known, even though the relative frequencies of the individual queries are not known.

Definition 3 The *retrieval time* $t(v)$ of a given data item v is the number of time units required to retrieve v. The *response time* $rt(q)$ on a query q is defined as $max\{T_1, T_2, \ldots, T_N\}$, where T_i $(1 \leq i \leq N)$ is the total number of time units required to retrieve qualifying data items on disk i, i.e., $T_i = \sum t(v)$ over all $v \in R(q)$ and v on disk i. If all data items are of equal size, and the retrieval time of a data item is assumed to be the unit of time, then $t(v) = 1$ for all $v \in V$ and $rt(q) = max\{|R_1(q)|, |R_2(q)|, \ldots, |R_N(q)|\}$, where $|R_i(q)|$ $(1 \leq i \leq N)$ is the number of qualifying data items on disk i.

In general, the retrieval time of an atomic data item is linear in terms of the size of the data item. The definitions of $t(v)$ and $rt(q)$ take into account data items of different sizes, i.e., different retrieval times. We note that most existing methods assume that data items are of equal size and that the retrieval time of a data item (e.g. a page or bucket) is assumed to be the unit of time. To make our discussion simple to understand, we first present the definitions, proposed scheme and theorem based on the same assumption. In Section 5, we discuss the generalization of our proposed scheme to the case where atomic data items might have different sizes.

Definition 4 An allocation method is *strictly optimal* for a query q if and only if a maximum of $\lceil\frac{|R(q)|}{N}\rceil$ data items need to be accessed on any of the N disks. An allocation method is *strictly optimal* with respect to a query set $Q_s = \{q_1, q_2, \ldots, q_k\}$ if and only if the allocation method is strictly optimal for all q_i, with i = (1,2,...,k).

Definition 4 is similar to that used by Du et. al [7, 18, 11, 35]. We note that no methods can achieve strict optimality for all query-sets [7, 18, 35]. For example, no method can be strictly optimal for all range queries if the number of disks is greater than 5 [18, 35]. However, some of the existing declustering methods have been proved to be strictly optimal for simpler query sets [8, 11, 35].

We use a weighted similarity graph to capture the similarity relationship between data items. The nodes in the graph represent the data items. The similarity between two data items is quantified as the weight on the edge connecting them. There exists an edge connecting two nodes, if the weight between them is not zero. The weighted similarity graph is formally defined as the following.

Definition 5 Let $WSG = (V, E)$ be a weighted similarity graph, where V is a data set and $E = \{\ e(u, v) \mid u \in \text{V},\ v \in V$, and u and v are qualified to be accessed together in a query $\}$. Each edge $e(u, v)$ in E is associated with a weight $w(u, v)$. The weight $w(u, v)$ represents the relative frequency with which data items u and v are likely to be accessed together by a query of interest. The weight on an edge $e(u, v)$, contributed from a query set Q_s, is equal to $\sum f(q)$, over all $q \in Q_s$, where $u \in R(q)$ and $v \in R(q)$.

The similarity between data-items u and v is measured by $w(u, v)$, the weight on edge $e(u, v)$. Thus data-items with a high degree of similarity are likely to be accessed together. Similarity between two groups, G_i and G_j, of data-items, is measured by $s(G_i, G_j)$, representing the aggregation of similarity of the data-items across the two groups. Formally, $s(G_i, G_j) = \sum_{u \in G_i} \sum_{v \in G_j} w(u, v)$. Two groups G_i and G_j with high $s(G_i, G_j)$ are likely to be accessed together by queries in the query set Q_s. Thus, higher values of $s(G_i, G_j)$ are desired for effective declustering.

The degree of similarity among the groups in a partition $\pi(V)$ is measured by the aggregation of the pairwise similarities among its groups. To formalize this concept, we introduce the notion of the cut-set E_c to represent the set of edges e(u, v) whose end points u and v fall in different groups of $\pi(V)$. Now $S(\pi(V))$, the degree of similarity among groups $G_1^\pi, G_2^\pi, \ldots, G_N^\pi$ of a partitioning $\pi(V)$, can be formalized as the sum of the weights on all the edges in the cut-set, as follows:

$$S(\pi(V)) = \sum_{(G_i, G_j)\ \textit{is a doubleton subset of}\ \pi(V)} s(G_i, G_j) = \sum_{e(u,v) \in E_c} w(u, v)$$

A doubleton set is a set with two elements. The set of doubleton subsets is used to avoid redundancy during computing the weights on the edges in the cut-set.

2.2. Our Approach

The basic idea behind our approach is as follows. Since maximizing parallelism in retrieval is desirable, the end-nodes u and v of an edge $e(u, v)$ with a high weight (similarity) should be allocated into different disks. Thus, maximizing the degree of similarity among N groups in a partition should generally provide good concurrency in retrieval. Our approach to declustering strives to discover a partitioning of data, $\pi(V)$, which maximizes $S(\pi(V))$ under disk number and disk capacity constraints. Formally, the proposed scheme, namely max-cut declustering, can be stated as follows:

Declustering Scheme : Max-Cut Partitioning of the WSG

Given a weighted similarity graph $WSG = (V, E)$, the number of disks N and the size-constraint predicates L_i, where $1 \leq i \leq N$.

Find a partition $\pi(V) = (G_1, G_2, \ldots, G_N)$ among N disks,

To maximize $S(\pi(V))$, i.e., the total weight on all the edges in the cut set, denoted as

$$\sum_{e(u,v)\in E_c} w(e(u,v)), \quad where \quad E_c = \{e(u,v) | e \in E, \ u \in G_i \ and \ v \notin G_i\}.$$

Such that $\quad L_i(G_i) \ = \ True, \ \ where \ 1 \leq i \leq N$

We refer to the above max-cut declustering scheme as **max-cut partitioning of the** *WSG*, as it maximizes the total weight on the edges in the cut-set of WSG. The expected query response time can be reduced, since data-items with higher similarity are more likely to be distributed into different groups. The objective of maximizing $S(\pi(V))$ is referred to as the **max-cut similarity criterion**.

We note that existing declustering methods are all heuristic methods. The max-cut declustering scheme is also a heuristic approach for declustering problems. However, it exhibits strict optimality under special conditions, as shown in Section 4.

The size-constraint predicate L_i for each disk may be determined based on the disk load-balance criterion, available disk capacity, N and sizeof(V). It could represent either disk-capacity constraints, $\sum_{v \in G_i} size(v) \leq Capacity(disk_i)$, or storage-load balance constraints. It could balance the "heat" [16] or frequency of access to data-items in each group. We note that the max-cut partitioning scheme explicitly accounts for available information about query distribution via the weights on the edges. It also accounts for data sizes, data distribution, and partition size constraints.

The query profiles can be collected on the basis of the available database statistics for access frequencies. One way to gather such statistics would be to record the frequency of query occurrence and the access frequencies of data items. Another way could be based on the use of available information in the application domain. For example, in decision support systems, several reports are generated periodically (daily, weekly or monthly). In a terrain-visualization system, the simulation of a ground vehicle usually issues a fixed size (e.g. 8 km * 8 km) range queries [32]. In spatial databases, the proximity index [20] can be used to estimate the probability of a random query retrieving a given pair of rectangles.

We view the declustering not as a one-time operation, but as an operation that may need to be executed periodically to reorganize the allocation of data items, if query profiles change substantially. However, a complete study of the effect of change query profiles is outside the scope of this paper and needs further study. We plan to explore it in our future work.

If no information about query distribution is available, then each query is assumed to be equally likely, and the scheme still takes advantage of data-distribution information in partitioning. The scheme can also take advantage of partial information about query distribution. Weights in WSG can be defined on the basis of the frequency of query classes, even though the frequency of individual queries are not known. For example, the assumption that smaller range queries on spatial data are more likely than larger range queries can be used to define a query-distribution function.

Unlike most previous methods, the proposed scheme makes no assumption about the semantics of the data items in terms of dimensionality or domain. Thus the method can be applied to a diverse range of declustering problems. Furthermore, it can be adapted to declustering at various data granularity levels, and thus can achieve good declustering that is based on a real data set (e.g. non-uniformly distributed data pages, rather than the grid cells).

In general, the I/O time of a single disk is linear in terms of the size of the data item to be retrieved. The above scheme assumes that the size of each data-item is the same. Thus, it takes the same amount of I/O time to retrieve each data item (e.g. a bucket or a page). A generalized scheme for modeling different sizes of data-items is discussed in Section 5.

2.3. Max-cut Graph Partitioning Techniques

Our proposed max-cut declustering scheme aims at partitioning the WSG so as to maximize the total weight on the edges in the cut-set of WSG. Unfortunately, the max-cut graph-partitioning problem, as stated, remains NP-complete [14], which can be shown by reducing it to the complementary min-cut graph partitioning problem [22]. The min-cut graph partitioning problem is to partition the nodes of a graph with weights on its edges into subsets of given sizes, so as to **minimize** the sum of the weights on all of the edges in the cut-set. Kernighan and Lin [22] have shown that by changing the signs of all the edge weights, the max-cut graph partitioning problem can be transformed into the min-cut graph partitioning problem.

Many good heuristic algorithms which have been developed for the min-cut graph partitioning problem can also be applied to efficiently solve the max-cut graph-partitioning problem. We note that the minimum-spanning-trees algorithm can also be used as a simple heuristic for the graph-partitioning approach. However, the ratio-cut [5], KL [22] and other methods [21] are superior [5, 21, 22]. These heuristics are based on spectral partitioning and iterative approaches. In spectral techniques [1], eigenvalues and eigenvectors of the matrix representation of the graph are computed. Partition results are derived by mapping the information provided by eigenvectors onto actual partitions. Iterative approaches [5, 13, 22, 34] start from an initial partition, then iteratively apply pairwise swapping or moving of nodes across partitions to minimize edge cuts.

In general, graph partitioning algorithms are expensive and their CPU and memory costs often grow non-linearly with the size of the graph. There are two ways to manage the cost: (i) use an efficient and effective graph partitioning algorithm; (ii) reduce the size of the individual graph to be partitioned by using an incremental declustering method. A survey of graph-partitioning algorithms, along with their costs and quality of partition is available in [21]. The latest developments have yielded algorithms that can partition graphs with a hundred thousand nodes in a couple of minutes [21] on workstations. Similar run-times (e.g. within 50 seconds on a Sparc 2 workstation for partitioning a graph with 170,000 nodes and 230 K edges into 64 partitions) are also reported in [2]. An incremental declustering algorithm is presented in Section 3.1 and is experimentally evaluated for a large data set in Section 6.3.4. In future work, we would like to evaluate the suitability of the latest graph-partitioning algorithm for declustering.

3. HEURISTIC TECHNIQUES FOR MAX-CUT DECLUSTERING

In this section, we present two heuristic techniques, Incremental max-cut and Global max-cut graph partitioning, for the max-cut declustering scheme. Incremental max-cut is based on the max-cut similarity criterion and local load-balancing. The global max-cut graph partitioning technique is based on a global min-cut graph partitioning algorithm with adaptation of the max-cut similarity criterion.

The Incremental max-cut is a simple greedy technique best suited for dynamic allocations, as well as for static declustering of large data sets. It achieves competitive quality of declustering at lower cost. The global max-cut graph partitioning technique is more expensive and might not be suitable for frequent reallocations. It is likely to achieve better quality of declustering; however, it will have a higher cost.

3.1. Incremental Max-cut Declustering

The incremental max-cut allocates data items to the disks in a greedy manner, using the max-cut similarity criterion and local load-balancing. Different from the classical local load-balancing strategy [20, 31], which allocates a data item to the disk with the lowest load (e.g. storage) over a local window, Incremental max-cut declustering aims at allocating a data item to a disk, such that the max-cut similarity criterion within a local window is best fulfilled.

A local window around a data item u defines a subset of data items which are likely to be accessed together with u via many queries. For example, a local window in a grid file may be 8x8 grid cells centered around a newly created page. We note that the definition of a local window is determined based on the application domain and the overall cost of incremental declustering strategy depends on the size of the window. In presenting the incremental max-cut technique, without loss of generality, we assume that the definition of local window has been provided. We will discuss the local window for the grid file in Section 6. A key issue in the incremental declustering technique is the degree of reorganization over a given window. In general, many data items within the window may be reallocated; however, for simplicity, we will focus on the case where existing data items are not reallocated across disks.

```
Procedure: IncrMaxCutAllocOne($u$: data item; $N$: number of disks)
    $P$: set of data items;
    similarity: set of weights $w(a,b)$;
    Alloc-diskid[]: array of disk-id of disk in which data item is allocated;
begin
    $P$ = local-window($u$); // comments : $P$ includes $u$
    // comments: $P$ is a set of data items within the local window defined around $u$
    similarity = create-sub-$WSG$($u$, $P$);
    Alloc-diskid[$u$] = GreedyAlloc($u$, $P$, similarity, $N$);
end;
```

Fig. 1: Incremental Max-Cut Algorithm for one Data Item

Figure 1 shows the incremental max-cut declustering algorithm for allocating one data item. Let u be the data item (e.g. the newly inserted data item) to be allocated. Let P be the set of data items within the local window, defined around u. The similarity (weight) between u and other data items of P is derived first. A local weighted similarity graph $WSG^\delta = (V^\delta, E^\delta)$ is constructed next, where $V^\delta = \{ a \mid a \text{ is } u, \text{ or } a \in P \text{ such that } w(u,a) > 0 \}$ and $E^\delta = \{e(u,b) \mid b \in V^\delta \text{ and } w(u,b) > 0\}$. We note that only the edges between u and other members of P are considered, and the edges among other members of P are not considered. Incremental max-cut declustering aims at allocating u to a disk, such that the max-cut similarity criterion within the local window is best satisfied. The IncrMaxCutAllocOne() algorithm calls the GreedyAlloc() algorithm to allocate the data item.

The GreedyAlloc() algorithm is described in Figure 2. Given the data item u to be allocated, the local window P and the *similarity* (weight) among u and data items in P, the algorithm allocates u to a disk such that the max-cut similarity criterion within the local window P is best satisfied. For every disk D, we compute the **similarity measure** which is defined as the summation of *similarity(u, v)*, where $v \in P$, and v is stored in D. The data item u is allocated to the disk that has the lowest similarity measure. We break the tie by choosing the disk with the lowest storage load (e.g. the fewest data items). It can be shown that the above strategy best meets the max-cut similarity criterion within the local window, among all possible allocations of u.

The algorithm presented in Figure 1 can be generalized for declustering a set of data items, as shown in Figure 3. A simple way to generalize is to invoke the IncrMaxCutAllocOne() algorithm for each data item in the data set. The cost of this iterative procedure can be reduced by grouping the unallocated data items by the local windows. The IncrMaxCutAllocSet() algorithm in Figure 3 starts by selecting an unallocated data item p as a seed node to form a subset P which contains all data items (both allocated and unallocated) within the local window defined around the data item p. Let U^P be the set of all unallocated data items in P. For every data item u in U^P, the similarity between u and other data items of P is derived first. A local weighted similarity graph $WSG^\delta = (V^\delta, E^\delta)$ is constructed next, where $V^\delta = \{ a \mid a \in U^P$, or $a \in P$ such that there exists

```
Procedure: GreedyAlloc($u$: data item; $P$: set of data items;
        $similarity$: set of weights; $N$: number of disks) : return disk-id;
    $Disk_i$: set of data item; $v$ : data item;
    Alloc-diskid[]: array of disk-id of disk in which data item is allocated;
begin
    for disk-id $i = 1$ to $N$ do
        Let $Disk_i = \{$data item $v$ | data item $v \in P$ and Alloc-diskid[$v$] is equal to $i\}$
        compute similarity measure $M_i = \sum_{v \in Disk_i} similarity(u, v)$;
    endfor
    return disk $i$, where $M_i$ is the lowest
                among disks which satisfy disk-load capacity constraints;
end;
```

Fig. 2: Greedy Allocation Algorithm

$u \in U^P$ and $w(u,a) > 0$ } and $E^\delta = \{e(a,b) \mid a \in U^P,\ b \in V^\delta$ and $w(a,b) > 0\}$. Finally, all the unallocated data items in the local window, i.e., members of U^P, are allocated to disks one by one in a greedy manner by using the GreedyAlloc() algorithm. The process repeats until all the data items have been allocated to disks. The quality of declustering can also be improved by simultaneously allocating all unallocated data items in P to disks.

Incremental max-cut declustering reduces the size of the data set to be allocated by incrementally declustering subsets of data items within a local window. Thus the method can handle larger data sets. The ordering of unallocated data items may affect the quality of declustering. We plan to study the effect of ordering in future work.

```
Procedure: IncrMaxCutAllocSet($V$: set of data items; $N$: number of disks)
    $P$, $Disk_i$: set of data items; $u$, $v$ : data item;
    $similarity$: set of weights $w(a,b)$;
begin
    Repeat
        Select an unallocated data item $p$ in $V$
        $P$ = local-window($p$);
        // comments: $P$ is a set of data items within the local window defined around p
        // comments: $p \in P$
        Let $U^P$ be the set of all unallocated data items in $P$
        $similarity$ = create-sub-$WSG(U^P, P)$;
        for each unallocated data item $u \in U^P$
            Alloc-diskid[$u$] = GreedyAlloc($u$, $P$, $similarity$, $N$);
        endfor
    Until all data items have been allocated;
end;
```

Fig. 3: Incremental Max-Cut Algorithm for a set of data items

3.2. Global Max-Cut Graph Partitioning

The global max-cut graph partitioning technique is based on partitioning a single similarity graph over all data items. This technique declusters all data items simultaneously, in contrast to the incremental technique discussed in Section 3.1. The declustering of a set of data items into multi-disks is based on a *heuristic N-way max-cut graph-partitioning algorithm* to partition the nodes of WSG into N groups that satisfy the disk-load size constraints, such that the total weight on all the edges in the cut-set is maximized. Those data items which belong to the same group are allocated to the same disk.

In general, N-way partitioning can be implemented via generalizing the two-way partitioning or via repeated applications of the two-way partitioning [22]. We implement the N-way max-cut graph partitioning by repeated applications of two-way graph partitioning, using the modified ratio-cut heuristic [5]. Chen and Wei [5] define the ratio cost metric for a two-way min-cut partition to be $W_c/|A| * |B|$ where W_c is the sum of the weights of the edges in E_c, and that $|A|$ and $|B|$ are the sizes of the two partitions. The cost metric for the two-way max-cut algorithm is modified to be $W_c * |A| * |B|$, and the sign of the weight on the edge is changed to be negative, to maximize the weight on the edges in the cut-set. We will abstract two-way max-cut graph partitioning in this paper as follows:

2-way-maxcut-partition(V: set of nodes; E: set of edges; $A1_{limit}$; $A2_{limit}$) $\rightarrow < A1, A2 >$
where $A1$ and $A2$ are the set of nodes, sizeof($A1$) $\leq A1_{limit}$and sizeof($A2$) $\leq A2_{limit}$.

The 2-way-maxcut-partition algorithm adapts the iterative approach [5], which starts from an initial partition (i.e. two subsets), and then iteratively moves nodes across subsets in an attempt to achieve a global minimum weight on the edges in the cut set. We note that the sign of the weight on the edge is changed to negative, such that the effect of maximizing the total weight on the edges in the cut-set can be achieved. The implementation of 2-way-maxcut-partition algorithm is based on the bucket-list data structure and requires a time complexity of $O(|E|)$ [13] with respect to the number of edges $|E|$.

We have chosen a competent algorithm, ratio-cut heuristic algorithm [5]. Our max-cut partitioning scheme can adapt any existing graph partitioning algorithms as the basis for declustering. Readers are referred to [21] for a survey and comparison of alternate graph partitioning algorithms.

Figure 4 shows the global max-cut declustering algorithm based on the max-cut graph partitioning technique. The procedure starts by finding an initial partition into N subsets, via using the procedure find-initial-partition(). Then, it repeatedly applies the 2-way-maxcut-partition procedure to pairs of subsets and makes the partition as close as possible to being pairwise optimal (pairwise max-cut). Pairwise optimization is performed by choosing pairs of subsets and applying the 2-way-max-cut-partition to these pairs. Notice that the disk-load parameter is used for the size constraint in the 2-way-max-cut-partition. The disk-load for each disk is assumed to model available disk capacity, storage-load balance criterion, sizeof(V) and N, etc.

```
Procedure: GlobalMaxCutDeclustering($V$: set of nodes; $E$: set of edges; $N$: number of disks,
                disk-load: array[1..$N$] of disk load-size constraint): partition
    $P$ : partition // comment: set of groups;
    $A_i$, $A_j$ : group // comment: set of nodes;
begin
    // comment: find initial partition of N subsets
    $P$ = find-initial-partition($V, E, N$);
    // comment: pairwise optimal
    Repeat
        Choose a candidate pair of $A_i$ and $A_j$ from $P$
        E'={(u,v) | (u,v) ∈ E, u ∈ $A_i \cup A_j$ and v ∈ $A_i \cup A_j$}
        < $A_i$, $A_j$ > = 2-way-max-cut-partition($A_i \cup A_j$, E', disk-load[i], disk-load[j])
        // comment: $sizeof(A_i) \leq$ disk-load[i] and $sizeof(A_j) \leq$ disk-load[j]
    until no candidate pair can yield improvement on $S(P)$ or the number of trials > $T$
        //comment: $S(P)$ is the total weight on all edges in the cut-set of $P$.
    return $P$;
end;
```

Fig. 4: The Global Max-Cut Declustering Algorithm

Pairwise optimization is performed until no more improvement can be found to the cut set, or until the number of passes is greater than a predefined number, T. The pairwise optimization process may converge quickly, if the number of disks (groups) is not very large. The CPU cost for partitioning can also be controlled by T, which limits the number of passes for pairwise optimization. As mentioned in [22], pairwise optimality is only a necessary condition for global optimality. There may be situations when some complex interchange of three or more elements from three or more groups is required to achieve global optimum. Other multi-way graph partitioning methods [34] can also be used as the basis of our scheme to further improve the result of partitioning, if computation complexity and CPU cost is not a concern.

To find the initial partition, we use the following two approaches suggested in [22] †. In the case that N is a power of 2, we apply the 2-way-maxcut-partition() procedure to partition the initial data set into two equally balanced subsets, then we repeatedly partition each of the subsets into two equally balanced subsets until N subsets are found. Otherwise, if N is not a power of 2, the initial data set of $|V|$ elements is partitioned into a subset of $\frac{|V|}{N}$ elements and a remaining subset of $(N-1)\frac{|V|}{N}$ elements. The 2-way-maxcut-partition() procedure is repeatedly applied to the remaining subset to split out a subset of $\frac{|V|}{N}$ elements, until N subsets have been found. Alternatively, the incremental max-cut declustering technique presented in Section 3.1 can also be used to create the initial partition.

4. ANALYSIS OF MAX-CUT DECLUSTERING SCHEME

In this section, we present the analysis of the max-cut declustering scheme. The analysis will demonstrate in the Theorem 1 that the max-cut declustering scheme is capable of achieving strictly optimal declustering, if strictly optimal declustering exists.

We characterize the situation where the proposed max-cut declustering scheme has strict optimality in Theorem 1, and illustrate two interesting cases in corollary 1 and 2. We recapitulate the relevant notations and properties here first. The weighted similarity graph, with respect to a query q and a query set Q_s, has several properties. The weighted similarity graph with respect to a query q, $WSG(q)$, is a *complete graph* $G^q = (V^q, E^q)$, where $V^q = \{\, v \mid v \in R(q)\,\}$, $E^q = \{\,(u,v) \mid u \in R(q)$ and $v \in R(q)\,\}$, and the weight with respect to a query q, $w^q(u,v)$ on each edge $e(u,v) \in E^q$ is identical. The cut-set under the partition of $\pi(V^q)$ is denoted by E^q_c, where $E^q_c = \{e(u,v) \mid e \in E^q$, and u and v are in different groups$\}$. The in-set under the partition of $\pi(V^q)$ is denoted by E^q_{in}, where $E^q_{in} = E^q - E^q_c$. Let $W(E^q)$, $W(E^q_c)$ and $W(E^q_{in})$ be the total weight on the E^q, E^q_c and E^q_{in}, respectively.

The weighted similarity graph with respect to a query set Q_s, $WSG(Q_s)$, is $G = (V, E)$, where $V = \{\, v \mid v \in R(q)$, for $q \in Q_s\,\}$, $E = \{e(u,v) \mid u \in R(q)$ and $v \in R(q)$, for $q \in Q_s\,\}$. The weight $w(u,v)$ on edge $e(u,v) \in E$ is equal to $\sum f(q)$, where $q \in Q_s$, $u \in R(q)$ and $v \in R(q)$. The cut-set under the partition of $\pi(V)$ is denoted by E_c.

Lemma 1 *An allocation/partition of $WSG(q)$, for a query q into N groups, has the maximal weight on the cut-set, if and only if the allocation/partition is strictly optimal with respect to q.*

Proof. For simplicity, let us assume that $R(q)$ is divisible by N. The proof for the case, where $R(q)$ is not divisible by N, can be found in [28]. Let the partition of $R(q)$ be $R_1(q), R_2(q), \ldots, R_N(q)$, where $R(q) = \cup R_i(q)$, for i = 1 to N. $WSG(q)$ is a complete graph with weight on the edges all equal to a constant k. Without loss of generality, let us assume that $k = 1$, i.e., $w^q(u,v) = 1$ for all $e(u,v) \in E^q$. Then $W(E^q)$ is equal to $|R(q)|(|R(q)|-1)/2$ and $W(E^q_c)$ is equal to $W(E^q)$ - $W(E^q_{in})$, where

$$\begin{aligned} W(E^q_{in}) &= \sum_{i=1\ to\ N} |R_i(q)|(|R_i(q)|-1)/2 \\ &= \frac{|R_1(q)|^2 + |R_2(q)|^2 + \ldots + |R_N(q)|^2}{2} - \frac{|R_1(q)| + |R_2(q)| + \ldots + |R_N(q)|}{2} \end{aligned}$$

†One may also use simple mapping-function-based methods to create the initial partition, if the data semantics allow that.

From the *Schwarz inequality,*

$$(|R_1(q)|^2 + |R_2(q)|^2 + \ldots + |R_N(q)|^2)(1^2 + 1^2 + \ldots + 1^2) \geq (|R_1(q)| + |R_2(q)| + \ldots + |R_N(q)|)^2 \quad (1)$$

From the above equation, we can derive the following.

$$W(E^q_{in}) \geq \frac{(|R_1(q)| + |R_2(q)| + \ldots + |R_N(q)|)^2}{2N} - \frac{|R(q)|}{2} \geq \frac{|R(q)|^2}{2N} - \frac{|R(q)|}{2}$$

$W(E^q_{in})$ is the minimum when the equality in equation 1 holds. The equality holds if only if $|R_1(q)| = |R_2(q)| = \ldots = |R_N(q)|$. $W(E^q_c)$ is the maximum when $W(E^q_{in})$ is the minimum, since $W(E^q_c)$ is equal to $W(E^q) - W(E^q_{in})$. This shows that the partition achieves maximal weight on the cut-set if and only if the partition is strictly optimal with respect to q. □

Lemma 2 *If an allocation/partition method is strictly optimal with respect to a query set Q_s, then it has the maximal weight on the cut-set of $WSG(Q_s)$.*

Proof. From definition 4, the allocation/partition method is strictly optimal for all q_i in the query set Q_s. From lemma 1, it has the maximal weight on the cut-set of $WSG(q_i)$, i.e., $W(E^{q_i}_c)$ is the maximum with respect to the partition of $WSG(q_i)$, for every $q_i \in Q_s$. The total weight on the cut-set of $WSG(Q_s)$ is equal to the summation of all the $W(E^{q_i}_c)$. It is the maximum, since every $W(E^{q_i}_c)$ is the maximum for the corresponding q_i. □

Theorem 1 *If there exists a strictly optimal allocation/partition method for a query set Q_s, then max-cut declustering is also strictly optimal with respect to the query set Q_s.*

Proof. From lemma 2, a strictly optimal method for Q_s leads to a partitioning π_p with maximal weight $S(\pi_p)$ on any cut-set of $WSG(Q_s)$. Let $\delta(q, \pi_p)$ be the sum of the weights on the cut-set imposed by π_p on $WSG(q)$. Note that $S(\pi_p) = \sum_{q \in Q_s} \delta(q, \pi_p)$, and $\delta(q, \pi_p)$ is the maximal weight on any cut-set of $WSG(q)$ due to lemma 1. Now we show that if max-cut partitioning discovers a partition π_m for $WSG(Q_s)$, then $\delta(q, \pi_m) = \delta(q, \pi_p)$, for any $q \in Q_s$. This can be shown from the following two observations: (a) $S(\pi_m) = S(\pi_p)$, from lemma 2 and the definition of max-cut partitioning; (b) $\delta(q, \pi_m) \leq \delta(q, \pi_p)$, from lemma 1. The constraints, (a) and (b), can be satisfied only if $\delta(q, \pi_m) = \delta(q, \pi_p)$, for all $q \in Q_s$. Thus, max-cut partitioning π_m is strictly optimal for each $q \in Q_s$ according to lemma 1, and it is strictly optimal with respect to Q_s. □

Corollary 1 *The max-cut declustering method is a strictly optimal allocation method with respect to the Row/Col queries over a two-dimensional grid.*

Proof. A linear allocation method, $Disk(x, y) = (px + qy + r) mod N$, where $GCD(p, N) = GCD(q, N) = 1$, is strictly optimal with respect to Row/Col queries over a two-dimensional grid [35]. Therefore, the corollary follows from Theorem 1. □

The above result can be extended to Row/Col type queries over a k-dimensional grid. The partial match queries with only one unspecified attribute over a cartesian product file are instances of Row/Col type queries over a k-dimensional grid.

Corollary 2 *The max-cut declustering method is a strictly optimal allocation method with respect to the Row/Col queries over one dimension of a k-dimensional grid.*

Proof. The Disk Modulo method is strictly optimal for all partial match queries, with only one unspecified attribute [8]. Therefore, the corollary is derived from Theorem 1. □

Theorem 1 demonstrates that the max-cut declustering (partition) is capable of achieving strictly optimal declustering if a strictly optimal declustering exists. However, strictly optimal declustering does not exist for all query-sets, for example range queries [35]. In this situation, no declustering method can achieve theoretically optimal response time. In addition, heuristic implementation of max-cut declustering may not always provide strictly optimal declustering.

5. HANDLING DATA ITEMS WITH DIFFERENT SIZES

In previous sections, we assume that the size of each data item was identical. However, the sizes of data-items can vary a great deal in databases containing data items like composite application objects, spatial objects (e.g. polygons), maps (vector or raster), images, etc. Such databases include geographical information systems (GIS), statistical and scientific databases, object-oriented databases, etc. Declustering techniques therefore need to be generalized to handle data items of different sizes.

In this section, we demonstrate the capability of the proposed method to work with complex data-types (e.g. polygons) that have varying sizes of data-items. We assume that data items are atomic, i.e., that a data item will not be split across multiple disks. In future work, we plan to relax this assumption to model the situation of data items split across disks.

Let $t(v)$ be the number of time units required to retrieve the given data item v. In general, $t(v)$ is linear in terms of the size of the data item to be retrieved, assuming that the entire data item is not allowed to be split across disks. The $size(v)$ can be used to denote $t(v)$, if the information for $t(v)$ is not available. Considering two data items u and v that are accessed by a query, if both data items are allocated into the same disk, then the response time required to retrieve both v and u is equal to $t(u) + t(v)$. However, if they are allocated into different disks, the response time for accessing both of them is the maximum of $t(u)$ and $t(v)$. The possible savings in response time achieved by putting them into different disks vs. putting them into the same disk is thus equal to the minimum of $t(u)$ and $t(v)$. With the objective of minimizing the expected response time over the query set of interest, it is highly desirable to maximize possible savings in response time. This gives us the following generalized max-cut declustering scheme, which includes the possible response time savings in the cost metric.

Generalized Max-Cut Partitioning of the WSG

Given a weighted similarity graph $WSG = (V, E)$, the number of disks N and the size-constraint predicate L_i, where $1 \leq i \leq N$,

Find a partition $\pi(V) = (G_1, G_2, \ldots, G_N)$ among N disks,

To maximize possible response time savings, ϕ^π, denoted as

$$\sum_{e(u,v) \in E_c} w(e(u,v)) * min\{t(u), t(v)\}, \quad where \ E_c = \{e(u,v) | e \in E, \ u \in G_i \ and \ v \notin G_i\}.$$

Such that $L_i(G_i) = True, \ where\ 1 \leq i \leq N$

Methods used for the max-cut declustering scheme can also be applied to the generalized max-cut declustering scheme. This can easily be done by changing the weight on any edge $e(u,v)$ in WSG, to be $w'(e(u,v))$, where $w'(e(u,v)) = w(e(u,v)) * min\{t(u), t(v)\}$. Obviously, generalized max-cut partitioning reduces to max-cut partitioning as described in Section 2.2, if the data items are identical in size.

We have constructed the following theorem regarding the set of **binary queries**, where the number of atomic data items that qualify for each given query is two.

Theorem 2 *Generalized max-cut declustering is an optimal declustering method for the set of binary queries.*

Proof. The proof can be found in [28]. □

Theorem 2 should be interpreted in the proper context. It suggests that the max-cut declustering scheme provides the best declustering for a set of binary queries of atomic data items. Theorem 2 assumes that each data item is allocated to a single disk, ruling out the splitting of data items.

Example 1 We use a simple example to illustrate the application of max-cut partitioning to handling data-items of different sizes. The proposed scheme has been applied to decluster polygonal

allocation			Query	(1)	(2)
disk	(1)	(2)	q	rt(q) * f(q)	rt(q) * f(q)
	p1	p1	p1, p2	4 * 2	4 * 2
1			p1, p3	5 * 2	9 * 2
	p4	p3	p1, p4	5 * 2	4 * 2
			p2, p3	7 * 1	5 * 1
	p2	p2	p2, p4	2 * 2	3 * 2
2			p3, p4	5 * 1	5 * 1
	p3	p4	T^π	44	50
			ϕ^π	15	9

Table 2: Two allocations of example database

maps which contain polygons of different sizes [32]. In Figure 5 (a), we show a database for four polygons. For simplicity, we assume that all the queries in this particular database are polygons-intersection queries involving two polygons (spatial join queries).

The polygons are shown as vertices of the graph in Figures 5 (a) and (b). The number on the vertex denotes the size of the data item, as well as the number of time units required to retrieve the data item, assuming that the retrieval time is proportional to the size of the data item. We note that the numbers on the edges in Figure 5 (a) represent their weights, i.e., query likelihood. In Figure 5 (b), the numbers on the edges denote the response-time savings which are equal to $w(u,v) * min\{t(u),t(v)\}$, where $w(u,v)$ is the number (i.e. weight) on the edges that connect vertices u and v as shown in Figure 5 (a). For example, the sizes of $p1$ and $p3$ are equal to 4 and 5, respectively. Also, $t(p1) = 4$ and $t(p3) = 5$. In Figure 5 (a), edge($p1$,$p3$) has the weight $w(p1,p3)$ equal to 2. In Figure 5 (b), edge($p1$,$p3$) has weight of $w(p1,p3) * min(t(p1),t(p3))$, i.e., 8.

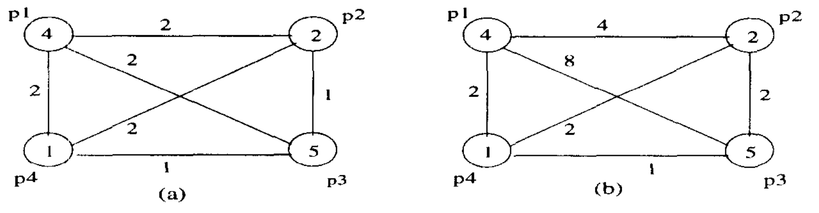


Fig. 5: An example database; (a): weighted similarity graph; (b): graph to be partitioned

Assume that the number of disks is two and that the capacity of each disk is ten units. Figure 5 (b) shows the possible response-time savings that corresponds to Figure 5 (a). According to Figure 5 (b), we have the following allocation. It is beneficial to place $p1$ and $p3$ on different disks, since the join query of $p1$ and $p3$ gives the greatest possible response-time savings. Meanwhile, it is better to put $p2$ and $p1$ on separate disks to maximize possible response-time savings. By following this plan, max-cut partitioning maximizes the possible savings of response time via putting $p1, p4$ on the same disk and $p2, p3$ on the same disk.

Table 2 lists the two-way polygon intersection queries and their expected response-time under two allocations, respectively. Allocation 1 is obtained via max-cut partitioning. In order to show the effect of max-cut partitioning, we also list another possible declustering, allocation 2. We can see that allocation 1 is more effective than allocation 2, since the expected response time (T^π) for allocation 1 is less than that for allocation 2. Allocation 1 achieves a higher savings of response time, i.e., ϕ^π (the total weight on the cut-set of the graph in Figure 5 (b)) than allocation 2 does.

6. EXPERIMENTS WITH GRID FILES

Previous sections have theoretically illustrated the ability of the proposed declustering method to adapt to query distributions, data distributions, data sizes and data semantics. In this section, we focus on a detailed experimental evaluation in the context of a real-world indexing method, namely the Grid Files. The experiments allow a comparative evaluation of the method with mapping-function and load-balancing based declustering methods. These experiments also facilitate the characterization of the dominance region of the proposed method in the context of multi-dimensional data. In the future, we intend to extend the comparisons to other domains.

6.1. *Parallelizing Grid Files*

We apply the max-cut declustering scheme to parallelizing Grid files [29], a well known access method for multi-dimensional and spatial data. Multi-dimensional data refers to a collection of data values embedded in a coordinate space which has dimension ≥ 2. The Grid file partitions the coordinate space into rectangular grids called cells. We will work with a two-dimensional coordinate space for simplicity. The result can be generalized to higher dimensions. Queries on multi-dimensional data represent various subsets of the data. To process a query, the database manager has to retrieve the data points contained inside each cell which intersects with the query.

Most existing declustering methods for grid files [18, 35] are based on coordinate mapping of the data pages in the grid-directory to the disk-id, assuming that the data are uniformly distributed. However, for many nonuniform distributions, multiple grid cells may need to share a disk block [29], and the mapping-function based methods will then need to resolve conflicts. In addition, the mapping-based methods need to be extended to deal with the splitting and merging that result from updates, since these events may change the grid-coordinates of existing data-pages.

Parallel implementations of the Grid-File need to address two distinct situations: (i) declustering a given Grid File, and (ii) determining disk allocation for the new page generated during updates (incremental declustering). For the former problem, we use the max-cut declustering technique at the data page level. To apply the max-cut declustering technique, a weighted similarity graph is created for the data-pages, using the grid directory and the query distributions that define the query class frequency. The nodes in the WSG are the data pages of the grid files. The similarity (weight) between two data pages D_1 and D_2 can be derived by estimating the frequency with which given query types will access both D_1 and D_2. This computation needs to trace the data-page pointer of the grid-directory cells. Note that only the query-type information is used to determine the weights. The details of individual queries and their expected frequency are not used.

For incremental declustering during updates, we use the proposed incremental max-cut technique. Incremental max-cut techniques need to define the notion of a local window for the grid file. We use the following approach to define a local window. A local window of upper-bound order RxW can be defined as being centered around the bounding rectangle of a selected data page p. The bounding rectangle for a data page p is the union of those grid cells which share the same data page p. The data page p could be the newly created page or the merged page. Let the grid-directory coordinates of the center of the bounding rectangle be (x_c, y_c), and the size of the allocation window AW be RxW. Then the allocation window, AW, is bounded by $(x_c - R/2, y_c - W/2)$ and $(x_c + R/2, y_c + W/2)$. The allocation data set P is defined as follows: $P = \{$data page D | the bounding rectangle of D is within AW or intersects with AW $\}$. To apply incremental max-cut declustering, the similarity is estimated in terms of the local window, i.e., the sub-grid directory. The order of the windows controls the extent and cost of incremental declustering and also depends upon the number of disks and the ratio of read/update queries since the last allocation.

6.2. *Experiment Design*

We examine range queries over a two-dimensional data set that is partitioned by the grid file. The response time averaged over the queries of interest is used as the measure of performance. Uniform and non-uniform data distributions are generated and partitioned by the grid file. The non-uniform data set includes the hot-spot data distributions. Various declustering methods are applied to decluster the data pages indexed by the grid-cells directory. Query execution is simulated to measure the response time, i.e. the number of parallel I/O required to fetch the data pages qualifying in the queries.

Query Sets: We introduce simple notations to describe the grid file and query sets. A $R \times M$ grid file has a grid directory with R rows and M columns. For example, a grid file storing a uniform data distribution with no sharing of data pages will have $R * M$ pages. A $p \times q$ query accesses

$p * q$ grid cells in the grid directory, spanning p rows and q columns. In a dense grid directory, it accesses $p * q$ data pages as well.

We consider several types of queries, including square-shaped, Row/Column and diagonal range queries. The square-shaped queries represent range queries with equal lengths on both sides and have been used to evaluate declustering methods in recent studies [9, 18]. In our experiments, we examine all possible square-range queries. Suppose that the grid file is an $M \times M$ grid. The row-query (Row) set is a set of $1 \times M$ rectangle-shaped queries, and the column-query (Col) set is a set of $M \times 1$ rectangle-shaped queries. The diagonal queries include the Principal Diagonal (PD) queries and the Anti-Diagonal (PA) queries [35]. PD is represented by a set of queries $\{PD_0, PD_1, PD_{-1}, \ldots, PD_n, PD_{-n}\}$, where query $PD_j = \{(x_i, y_i) \mid \text{cell } (x_i, y_i) \text{ and } x_i - y_i = j\}$, for all j=$\{0, 1, -1, 2, -2, \ldots, M, -M\}$. Similarly, query set PA is represented by a set of queries $\{PA_0, PA_1, PA_{-1}, \ldots, PA_n, PA_{-n}\}$, where query $PA_j = \{(x_i, y_i) \mid \text{cell } (x_i, y_i) \text{ and } x_i + y_i = j\}$, for all j={0..2M-2}.

Data Sets: The uniform data (UU) set, being factorizable, is a collection of points (x, y) where x and y are independent, uniformly distributed random variables. The hot spot (HS) data-set of K points is generated from an initial uniform distribution ($K/4$) over the unit square. We then generate and insert $3K/4$ other points from the normal distribution, with a small standard deviation. We note that the hot-spot data-set is not factorizable, and that it has been used in the literature [26] to simulate skewed distributions.

It is possible to use the "fractal dimension" [10] to generate experimental data. However, it is not likely to change the major trends (e.g. the proposed method outperforms others for non-uniform data distributions). In fact, the proposed methods are already being used for real data sets (e.g. the polygonal map of Killeen, Texas) [32]. The relative performance and rankings of declustering methods have been found to be the same as those predicted in this paper. Interested readers please refer to [32].

Grid File Creation: These data sets are stored into the grid file via a sequence of insert operations with the grid access method. Each data page has 16 records at most. A given data set can result in many different grid-directories based on the insertion order, split policies and the initial grid. The data was inserted into the grid file in the order of a space-filling curve function, Z-curve [19]. If one of the data pages overflows, the row or column of the grid that contains that data page is split. The split policy alternates between splitting rows and columns. The data pages may be shared by multiple cells after the split, as is natural in grid files. Sharing of the data page by cells introduces a distortion. For example, a grid-directory for uniformly distributed data may not have uniform distribution over data pages of cells in the grid-directory. To isolate the effect of page-sharing by cells, we use two kinds of initial grids. For the experiments on the effects of query sets and data distributions, the grid file is provided with an initial grid based on its distribution so as to reduce page splits and page sharing among grid-cell entries. For the experiments on the effects of page-sharing, data was inserted into the grid file with an initial 1x1 grid directory structure.

Declustering Methods: We compare max-cut declustering with two well-known mapping-function-based methods, the Hilbert (MF-Hilb) allocation method [9] and the Linear method [27, 8, 35]. The Hilbert method is chosen for comparison, since it achieves good performance for square-shaped range queries [9]. Two types of linear methods are used in our experiments. The first one, denoted as MF-CMD, uses the modulo function $(x + y) \bmod N$. The second one, represented as MF-GDM, uses the modulo function $(x + 5y) \bmod N$. The MF-CMD method is also known as the CMD method [27] or the Disk Modulo method (DM) [8] in a two-dimensional space. The specific mapping function method, MF-GDM, that we choose in the experiment, belongs to the class of Generalized Disk Modulo methods (GDM) [8], vector based declustering methods [4] and lattice allocation methods [35].

Two global max-cut declustering strategies are derived which use different approaches to estimate the weights. The first approach derives the weight of the graph in terms of these query types : row/column, $N \times 1/1 \times N$, and small square-range queries (e.g. $\sqrt{N} \times \sqrt{N}$ queries). The second approach derives the weight based on specific queries of interest in the experiment. We will consider the first approach as a general max-cut declustering technique (SM-GP-G), since it will not be adapted to a query set of interest in an experiment. The second one will be called a specific max-cut declustering technique (SM-GP-S), since it is adapted to the query sets. Instances of SM-GP-S use weights derived from the proper subset of square-range queries, row/column and diagonal queries in relevant experiments. Both SM-GP-G and SM-GP-S are based on heuristic global max-cut graph partitioning, as described in Section 3.2.

We evaluate the incremental max-cut technique (SM-INCR) along with the load-balancing method, LoadBal. The SM-INCR derives the weight between data pages in terms of these query types: row/column, $N \times 1/1 \times N$ and small square-range queries (e.g. $\sqrt{N} \times \sqrt{N}$ queries), over the local window (i.e., sub grid-directory). We used a row-major ordering of the cells of the grid-directory to determine the order in which data items are allocated to disks. Load-balancing based declustering techniques allocate page p to the disk with the lowest load over the local window. The load on individual disks represents the objective function of declustering. The LoadBal method uses the storage load, i.e., the number of pages within the local window residing on the disk, as the measure [31].

Choice of Parameters: Our experiments with the grid file focus on the effect of query distribution and data distribution. We leave the exploration of the effect of partition constraints out of this study, as it has not been explored in related work. We do not explore the effect of the number of disks, since it is not likely to impact the ranking of alternative declustering methods. MF-CMD has been used for performance comparison in previous research. However, we do not report the performance of MF-CMD in our comparison, since MF-CMD is a special case of MF-GDM. In our experiments, MF-CMD has not performed better than MF-GDM.

6.3. Experimental Observations

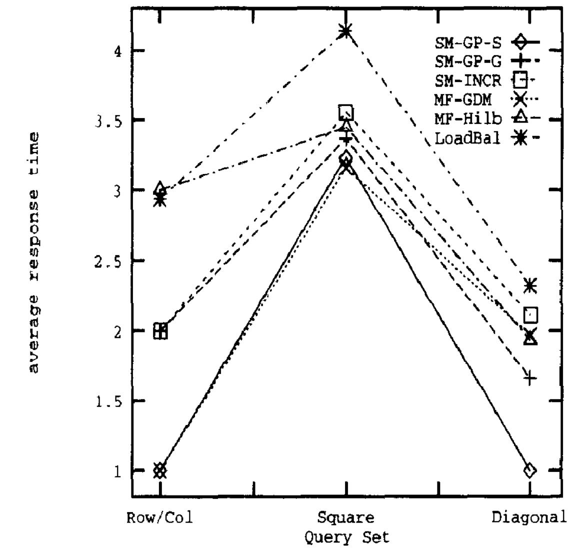
In this subsection, we compare alternative declustering techniques under the effect of query sets, data distributions and page sharing. Mapping-function based methods MF-GDM, MF-CMD and MF-Hilb need to resolve conflicts in the disk-allocation of pages that are shared by multiple cells of the grid file. We broke the tie in a fair manner by choosing a disk at random. This is consistent with the tie-breaking approaches used previously in mapping-function based methods to decluster grid files [35]. Each query is executed by searching qualified grid cells to retrieve data pages. The response time for each query is computed by counting the maximum number of qualified data pages on the disk.

6.3.1. Effect of Query Sets

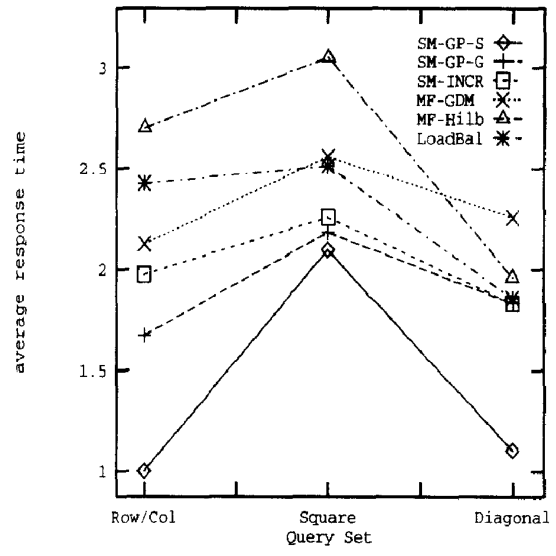
In this experiment set, data was inserted using an initial 16x16 grid partitioning of the domain of the data distribution provided to the grid file so as to reduce page sharing. The declustering was done statically after the final grid was formed. The final grid file for the UU data set has a 16x16 grid directory with no page sharing, where the final grid file for the HS data set has a 20x20 grid directory with data page sharing among grid cells. Figure 6 shows the average response time for three query sets on grid files for the UU data set and for the HS data set, which are declustered by alternate methods for 16 disks. The average response time measures the average number of parallel I/O needed over all queries in each query set.

SM-GP-S (query-set specific max-cut declustering) outperforms other methods. It illustrates the ability of the proposed method to adapt to query distribution. For the UU data set, MF-GDM (mapping function based general disk modulo) performs well on row/column and square-shaped query sets. However, it does not perform well on diagonal queries. For the HS data set, SM-GP-G (query-set independent max-cut declustering) outperforms all declustering methods except SM-GP-S. We note that SM-GP-G does not take advantage of query-set information and thus represents a weak case of similarity max-cut graph partitioning. Yet this method can outperform mapping function based methods for a non-uniform HS data set.

The detailed results for the square query set are shown in Figure 7, using the HS data set. The performance of SM-GP-S is the best, followed very closely by that of SM-GP-G and SM-INCR. The performance of LoadBal, MF-GDM and MF-Hilb follow behind that of the similarity max-cut techniques.



(a) UU Data Set



(b) HS Data Set

Fig. 6: Average response time v.s. query set

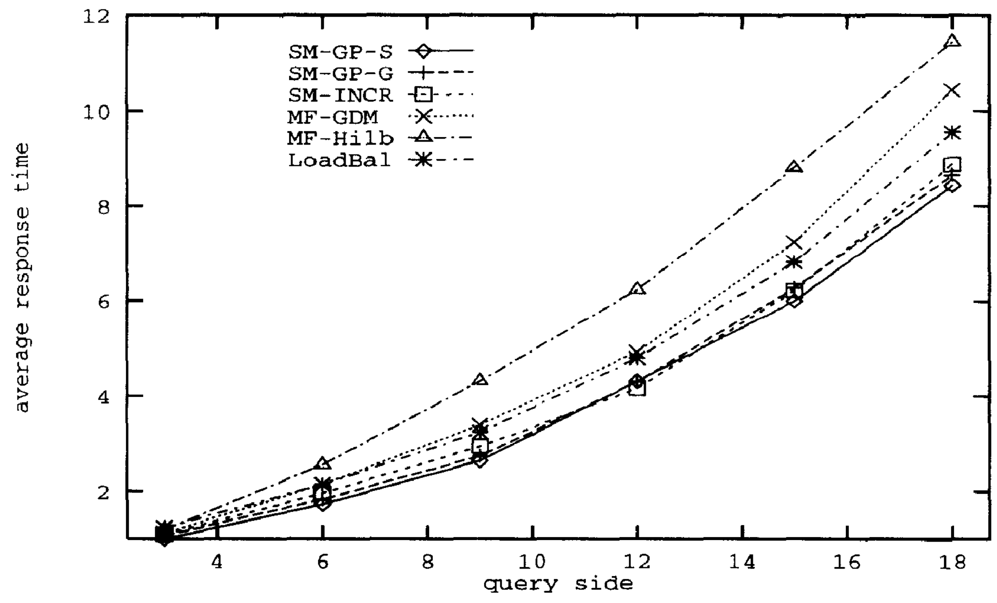


Fig. 7: Average response time v.s. query side r (Query = rxr, Grid size = 20x20, HS data set)

6.3.2. *Effect of Data Distributions*

We examine the results shown in Figure 6. On the average, the performance of SM-GP-S is the best under both data distributions, followed by that of SM-GP-G and SM-INCR. MF-GDM is very competitive with SM-GP-S with the UU data set. However, the performance of MF-GDM with the non-uniform (HS) data set drops behind that of SM-GP-G and SM-INCR. The result shows that the proposed similarity max-cut techniques adapt to different data distributions and also outperform the mapping-function based methods.

6.3.3. *Effect of Page Sharing*

We examine the case of a high page-sharing grid file in this section. The experiment is conducted on the uniformly distributed data set with 16 disks. Data is inserted into a Grid-file with an initial 1x1 grid directory. The resulting grid file has a 55x56 grid directory with a very high degree of page sharing (a data page shared by 1 to 30 cells). The incremental max-cut (SM-INCR) and load-balancing (LoadBal) methods assign pages to disks incrementally, as splits occur. The global max-cut graph partitioning (SM-GP-G and SM-GP-S) and mapping function (MF-GDM and MF-Hilb) methods perform declustering at the end, after all the data is inserted and the final grid has been created.

The result is shown in Figure 8 (b). To illustrate the effect of page sharing, we also list the results from Figure 6 (a) in Figure 8 (a), which show the results for a dense (i.e., without page sharing) grid file.

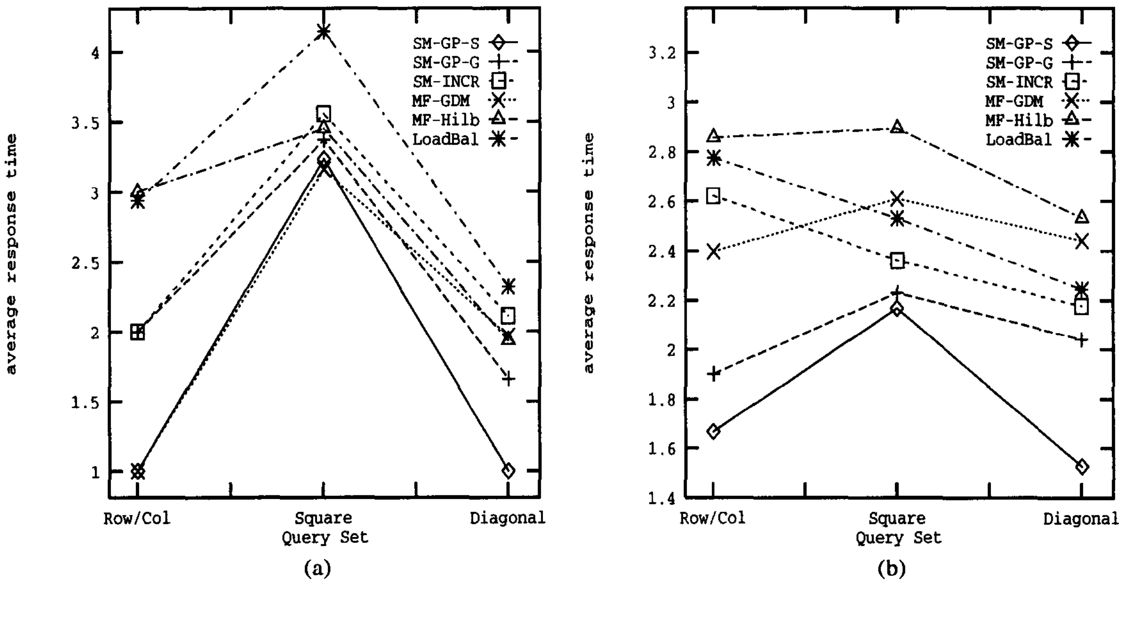


Fig. 8: Average response time v.s. query set (UU data set)

Figure 8 (b) shows that similarity-based methods, including global max-cut graph partitioning and the incremental max-cut methods, outperform mapping-function and load-balancing methods. The performance of SM-GP-S is the best, followed by that of SM-GP-G, SM-INCR and the others. The incremental max-cut technique, SM-INCR, outperforms the load-balancing-based method, LoadBal. Figure 8 (b) shows that MF-GDM and MF-Hilb perform worse than the other methods because of high page-sharing, even though the data set is uniformly distributed. Although mapping function based methods have been modified to resolve conflicts in the disk allocation of pages that are shared by multiple cells of the grid file. They do not perform well in the case of poorly partitioned initial grid directory or high ratio of splits and merges during updates.

6.3.4. Experiment on a Larger Data Set

We examine the scalability of max-cut declustering for a larger data set in this subsection. Incremental max-cut declustering decomposes the data set into subsets of data items within a local window. Thus the method can handle larger data sets. The scalability of global max-cut graph partitioning technique for large data sets needs further work and is proposed as future work. The experiment is conducted on a uniformly distributed data set, using an initial 160x160 grid directory and an initial 128x128 grid directory, which result in a 160x160 grid directory (25600 data pages) with no page sharing, and a 304x305 grid directory (16737 data pages) with page sharing, respectively. We compare SM-INCR, MF-GDM, MF-Hilb and LoadBal using square range queries and 16 disks. All four methods perform static declustering at the the end, after all the data is inserted and the final grid has been created.

The result is shown in Figures 9 and 10 for the experiment on a 160x160 grid with no page sharing, and 304x305 grid with page sharing, respectively. Without page sharing, the performance of MF-GDM is the best for square-shaped range queries, followed closely by that of SM-INCR. SM-INCR outperforms MF-Hilb and LoadBal. With page sharing, the incremental max-cut technique, SM-INCR, outperforms the others, as shown in Figure 10.

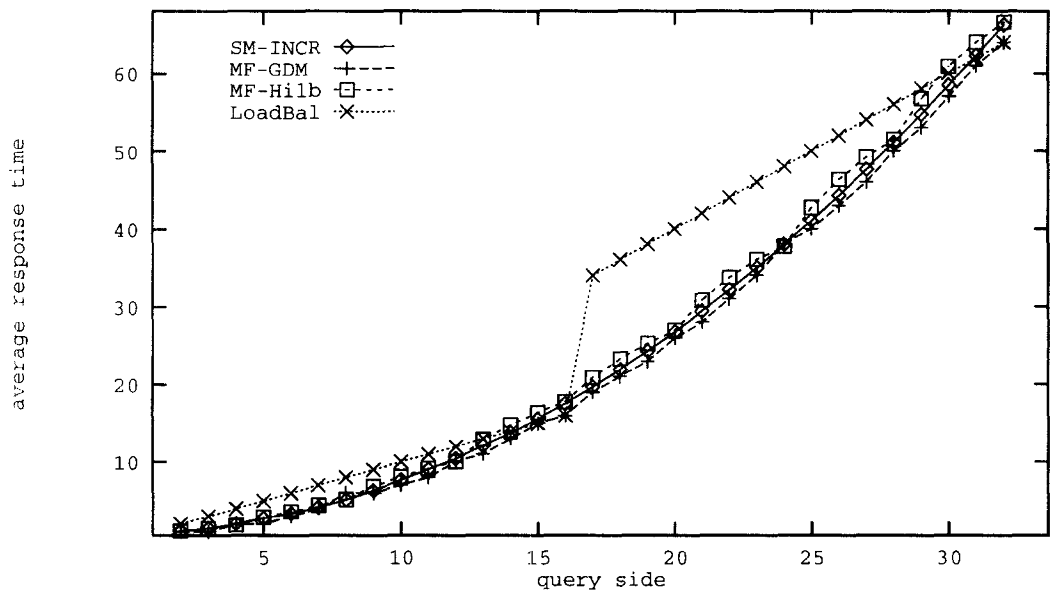


Fig. 9: Average response time v.s. query side r (Query = rxr, Grid size = 160x160)

6.4. Summary of Observations

In this subsection, we summarize our observations.

- Similarity-based max-cut declustering methods can adapt to data distribution, query distribution and page sharing, and they outperform mapping-function based methods. Mapping-function based methods do not adapt well to the high page-sharing typical of non-uniform data distributions, or even to uniform data distributions where the initial grid directory structure is not well partitioned.

- Similarity-based max-cut declustering methods are competitive with mapping-function based methods for many query sets, even if there is no page sharing in the grid-directory. The incremental max-cut method outperforms the load-balancing methods and provides the best trade-off between the parallel response time for queries and the cost of declustering.

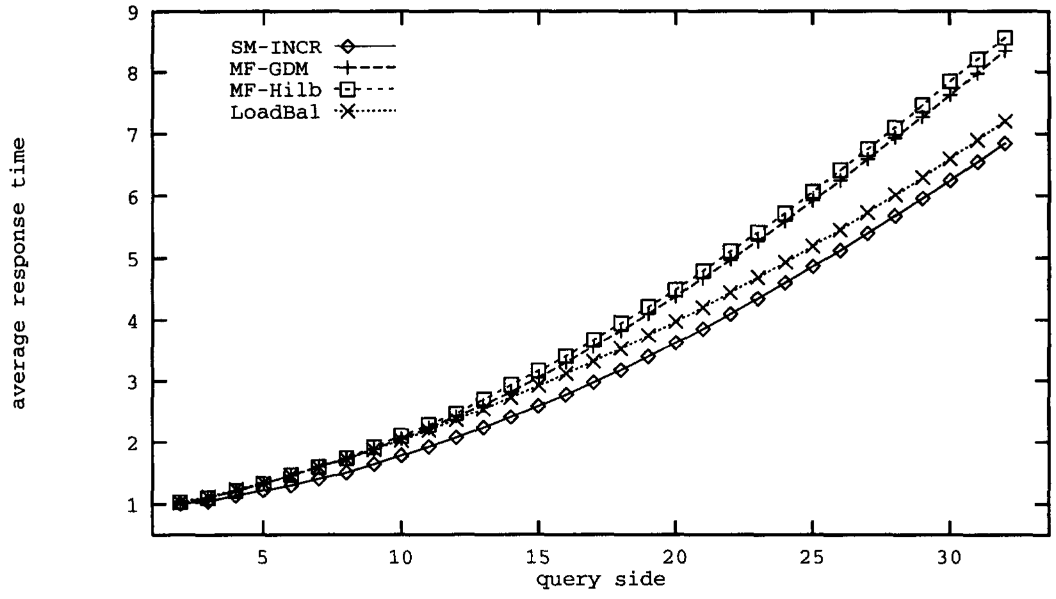


Fig. 10: Average response time v.s. query side r (Query = rxr, Grid size = 304x305)

- There is a trade-off between the declustering quality and the overhead of declustering methods. The global max-cut graph partitioning technique is likely to achieve better declustering. However, it also takes more computation than incremental declustering methods. It may be used when data-distributions and query-distributions change infrequently. Further study of the trade-off between the declustering quality and the overhead is needed.

7. CONCLUSIONS

In this work, we have presented a new similarity-based technique, max-cut declustering, for allocating data items to multiple disks. Our method uses the max-cut declustering approach to maximize the chances that a pair of data-items frequently accessed together by queries are allocated to distinct disks. Max-cut declustering can take advantage of the available information about query distribution, data distribution, data sizes, and partition size-constraints. It is flexible and applicable to alternative data semantics. An incremental max-cut method based on the ideas of max-cut similarity and load-balancing is also proposed to handle declustering for dynamic allocations as well as for static declustering for large data sets.

Our analysis demonstrates that the max-cut declustering approach can achieve a strictly optimal allocation for a query set, if there exists any other declustering method which will achieve the strictly optimal allocation for that query set. It also shows that the max-cut declustering approach can achieve strictly optimal allocation for row/column queries on uniform data distribution, etc. Furthermore, we demonstrate that the max-cut declustering scheme can achieve optimal allocation for the set of binary queries.

Experiments in the context of parallelizing grid files show that our method is competitive with existing effective declustering methods for uniformly distributed data sets, and that it outperforms competing methods for non-uniform data. In general, it is beneficial to use the max-cut declustering approach when the application domain is unstructured, as with non-uniform data distribution or non-equiprobable query distribution, etc. If the application domain has uniform data distribution and a well partitioned structure, then mapping-function based methods are competent choices, since max-cut declustering requires higher complexity. Experiments also show that the proposed incremental max-cut method outperforms the load-balancing based declustering method and provides the best trade-off between the parallel response time for queries and the cost of declustering.

In this paper, data items are assumed to be atomic, i.e., a data item will not be split across multiple disks. In future work, we plan to relax this assumption to model the situation of data items split across disks. In the future, we also plan to evaluate our scheme for other applications beyond grid-files that have data items of different sizes. In addition, Scalability of declustering techniques for very large data sets is an important issue. Incremental max-cut declustering reduces the size of the data set to be allocated by incrementally declustering subsets of data items within a local window. Thus the method can handle very large data sets. The scalability of the global max-cut graph partitioning technique for very large data sets needs further study. Besides, the effect of buffering is not investigated in this paper. Further evaluation is needed to consider the effect of buffering. Finally, We would like to characterize the trade-off between the computational overhead of the max-cut approach and declustering performance, particularly in the context of incremental declustering arising from small changes in data distribution and query distribution.

Acknowledgements — This work was done while the author was pursuing the Ph.D. degree in the Department of Computer Science, University of Minnesota. This research was supported by the Federal Highway Authority (FHWA), Minnesota Dept. of Transportation and the Center for Transportation Studies at the University of Minnesota. We would like to thank Prof. C.K. Cheng (University of California, San Diego) and Dr. L.T. Liu (AT&T Bell Labs) for helping with the ratio-cut program. Besides, we would like to thank several individuals who helped in the implementation and discussion of the grid file, including Y. Zhou and M. Coyle.

REFERENCES

[1] E.R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal Alg. Disc. Meth.*, **3**(4):541–550 (1982).

[2] W. J. Camp, S. J. Plimpton et *al.*. Massively parallel methods for engineering and science problems. *Communication of ACM*, **37**(4):31–41 (1994).

[3] C.C. Chang and C.Y. Cheng. Performance of two-disk partition data allocations. *BIT*, **27**(3):306–314 (1987).

[4] L. T. Chen and D. Rotem. Declustering objects for visualization. In *Proc. of Intl Conference on Very Large Data Bases*, pp. 85–96, Dublin, Ireland (1993).

[5] C.K. Cheng and Y.C. Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Trans. on Computer-Aided Design*, **10**(12):1502–1511 (1991).

[6] D.J. DeWitt et *al.*. The gamma database machine project. *IEEE Trans. on Knowledge and Data Engineering*, **2**(1) (1990).

[7] H. C. Du. Disk allocation methods for binary cartesian product files. *BIT*, **26**(2):138–147 (1986).

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

[9] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proc. of Intl Symposium on Databases in Parallel and Distributed Systems*, pp. 18–25 (1993).

[10] C. Faloutsos and I. Kamel. Beyond uniformity and independence : Analysis of R-trees using the concept of fractal dimension. In *Proc. Symp. on Principles of Database Systems*, *SIGMOD-SIGACT PODS*, pp. 4–13, Minneapolis, MN (1994).

[11] C. Faloutsos and D. Metaxas. Disk allocation methods using error correcting codes. *IEEE Trans. on Computers*, **40**(8):907–914 (1991).

[12] M.T. Fang, R.C.T. Lee, and C.C. Chang. The idea of declustering and its applications. In *Proc. of Intl Conference on Very Large Databases, VLDB*, pp. 181–188, Kyoto, Japan (1986).

[13] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. of 19th Design Automation Conference*, pp. 175–181, Las Vegas, Nevada (1982).

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco (1979).

[15] S. Ghandeharizadeh and D.J. DeWitt. A mutltiuser performance analysis of alternative declustering strategies. In *Proc. of the 6th Intl Conference on Data Engineering, IEEE*, pp. 466–475, Los Angeles, CA (1990).

[16] S. Ghandeharizadeh and D.J. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machine. In *Proc. of Intl Conference on Very Large Databases, VLDB*, pp. 481–492, Brisbane, Australia (1990).

[17] S. Ghandeharizadeh, D.J. DeWitt, and W. Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. of Intl Conference on Management of Data, ACM SIGMOD*, pp. 29–38, San Diego, CA (1992).

[18] B. Himmatsingka and J. Srivastava. Performance evaluation of grid based multi-attribute record declustering methods. In *Proc. of the Tenth Intl Conference on Data Engineering, IEEE*, pp. 356–365, Housten, Texas (1994).

[19] H.V. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. of Intl Conference on Management of Data, ACM SIGMOD*, pp. 332–342, Atlantic, NJ (1990).

[20] I. Kamel and C. Faloutsos. Parallel R-trees. In *Proc. of Intl Conference on Management of Data, ACM SIGMOD*, pp. 195–204, San Diego, CA (1992).

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. In *Proc. Intl Conference on Parallel Processing, ICPP*, **3**, pp. 113–122, Oconomowoc, Wisconsin (1995).

[22] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, **49**(2):291–307 (1970).

[23] K. Kim and V.K. Prasanna. Latin squares for parallel array access. *IEEE Trans. on Parallel and Distributed Systems*, **4**(4):361–370 (1993).

[24] M.H. Kim and S. Pramanik. Optimal file distribution for partial match queries. In *Proc. of SIGMOD Conference on Management of Data, ACM*, pp. 173–182, Chicago, IL (1988).

[25] V. Kouramajian, R. Elmasri, and A. Chaudhry. Declustering techniques for parallelizing temporal access structures. In *Proc. of the Tenth Intl Conference on Data Engineering, IEEE*, pp. 232–242, Housten, Texas (1994).

[26] R. Krishnamurthy and K.-Y. Whang. Multilevel grid files. *IBM Research Report*, Yorktown Heights (1985).

[27] J. Li, J. Srivastava, and D. Rotem. CMD: A multidimensional declustering method for parallel database systems. In *Proc. of Intl Conference on Very Large Data Bases*, pp. 3–14, Vancouver, B.C., Canada (1992).

[28] D.-R. Liu. *Database Design for Spatial Network Management Systems: Clustering and Declustering Techniques*. Ph.D. dissertation, Department of Computer Science, University of Minnesota (1995).

[29] J. Nievergelt, H. Hinteberger, and K.D. Sevcik. The grid file: An adaptable, symmetric multi-key file structure. *ACM Trans. on Database Systems*, **9**(1):38–71 (1984).

[30] D. Rotem, G.A. Schloss, and A. Segev. Data allocation for multidisk databases. *IEEE Trans. on Knowledge and Data Engineering*, **5**(5):882–887 (1993).

[31] B. Seeger and P.A. Larson. Multi-disk B-trees. In *Proc. of Intl Conference on Management of Data, ACM SIGMOD*, pp. 436–445, Denver, Colorado (1991).

[32] S. Shekhar, S. Ravada, et al.. al. Load-balancing in high performance GIS: Partitioning polygonal maps. In *Proc. of 4th Symposium on Spatial Databases, SSD'95*, pp. 197–215, Portland, Maine (1995).

[33] G. Weikum, P. Zabback, and P. Scheuermann. Dynamic file allocation in disk arrays. In *Proc. of Intl Conference on Management of Data, ACM SIGMOD*, pp. 406–415, Denver, Colorado (1991).

[34] C.W. Yeh, C.K. Cheng, and T.T. Y. Lin. A general purpose multiple way partitioning algorithm. In *Proc. of 28th ACM/IEEE Design Automation Conference*, pp. 421–426, San Francisco, CA (1991).

[35] Y. Zhou, S. Shekhar, and M. Coyle. Disk allocation methods for parallelizing grid files. In *Proc. of the Tenth Intl Conference on Data Engineering, IEEE*, pp. 243–252, Housten, Texas (1994).