HYPERGRAPH BASED DECLUSTERING FOR MULTI-DISK DATABASES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mehmet Koyutürk

September, 2000

ABSTRACT

HYPERGRAPH BASED DECLUSTERING FOR MULTI-DISK DATABASES

Mehmet Koyutürk
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Cevdet Aykanat
September, 2000

In very large distributed database systems, the data is declustered in order to exploit parallelism while processing a query. Declustering refers to allocating the data into multiple disks in such a way that the tuples belonging to a relation are distributed evenly across disks. There are many declustering strategies proposed in the literature, however these strategies are domain specific or have deficiencies. We propose a model that exactly fits the problem and show that iterative improvement schemes can capture detailed per-relation basis declustering objective. We provide a two phase iterative improvement based algorithm and appropriate gain functions for these algorithms. The experimental results show that the proposed algorithm provides a significant performance improvement compared to the state-of-the-art graph-partitioning based declustering strategy.

Key words: Distributed Databases, Declustering, Hypergraph Partitioning, Max-cut Graph Partitioning.

ÖZET

ÇOK DİSKLİ VERİTABANLARI İÇİN HİPERÇİZGE TABANLI AYRIŞTIRMA

Mehmet Koyutürk
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Cevdet Aykanat
Eylül, 2000

Çok büyük dağınık veritabanlarında, sorguların işlenmesini paralelleştirmek için veri disklere ayrıştırılmaktadır. Ayrıştırma, verinin her ilişkide yer alan öğelerin disklere eşit dağılacakları şekilde yerleştirilmesi anlamına gelir. Literatürde birçok ayrıştırma yöntemi önerilmiş olmasina karşın önerilen yöntemler ya alana özel ya da bazı dezavantajları olan yöntemlerdir. Bu çalışmada probleme tam olarak uyan bir model önerilmiş ve yinelemeli iyileştirme yöntemlerinin her ilişkiyi detaylı olarak değerlendirerek ayrıştırma hedefini gerçekleştirme yetisine sahip olduklarını gösterilmiştir. Ayrıştırma probleminin çözümü için iki aşamalı bir yinelemeli iyileştirme algoritması ve bu probleme uygun kazanç fonksiyonları önerilmiştir. Yapılan deneyler, önerilen algoritmanın en gelişkin ayrıştırma yöntemi olan çizge parçalama yönteminden daha üstün performans sergilediğini göstermektedir.

Anahtar kelimeler: Dağınık veritabanları, ayrıştırma, hiperçizge parçalama, çizge parçalama.

Gül’anama ve Babama...

ACKNOWLEDGMENTS

Derin bilgi ve zekası, insana coşku veren heyecanı, sonsuz anlayışı ve hiç eksik olmayan güleryüzü ile bu teze değer kazandıran danışmanım Cevdet Hoca'ya,

Sağladığı test verileri ve eleştirileriyle katkıda bulunan Sayın Halil Altay Güvenir'e,

Bu tezi okuyup yorumlarıyla güçlendiren Sayın Uğur Güdükbay'a,

Akademik çalışmaya dair kendisinden çok şey öğrendiğim Metin Nafi Gürcan'a,

Bu bölümde yüksek lisans yapmam için beni cesaretlendirerek bu tezin temelini atan Nihan Özyürek'e,

Başımın sıkıştığı her noktada zevkle yardımcı olan üstat Armağan Yavuz'a,

Sadece tezle kalmayıp tüm yüksek lisans hayatım boyunca her konuda destek olan kardeş Bora Uçar'a,

Ve varlığı, sevgisi ve inceliğiliyle beni güçlü kılan yarim Günnur'a,

Sonsuz teşekkürler...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In many database applications like scientific and multimedia databases, very large multi-dimensional or multi-attribute datasets are processed. When the storage space required for such databases is huge and the time required to process a query on such databases is very high, such applications are generally implemented on distributed environments. This is also the case in world wide web applications.

In very large distributed database systems, data placement is an important issue because it directly affects the access time required to process a query. As the disks in the multi-disk system are accessed in parallel, it is desired to access the same amount of data from each disk at a time, so that the I/O times required for access to all disks are balanced. A poor distribution of data over the disks may cause the system to access only a small number of disks among various number of disks resulting in ineffective usage of resources. To use the resources effectively by providing maximum parallelization of query processing, the data is distributed over the disks in such a way that the data that are more likely to be processed together are located into different disks. This operation is known as declustering. Declustering can be applied to any distributed database system with pre-defined relations on the data.

There have been significant amount of research on declustering in the literature. The great amount of work was on mapping function based declustering techniques that scatter the data into disks in such a way that the neighboring data in multi-dimensional space are placed into different disks. Such methods apply only to spatial databases and specific indexing techniques. A promising declustering technique is based on max-cut graph partitioning which outperformed all mapping function based strategies [34]. This method can be applied to any relational database system. However, this method have some deficiencies as a relational database system cannot be fully represented by a graph and the cost model of graph partitioning does not accurately represent the cost function of declustering. We show the flaws of the graph model and provide a model that exactly fits to the physical problem.

We model the declustering problem by representing the relational database system by a hypergraph, where each data item is represented by a vertex and each relation is represented by a net, and we define a cost function for partitioning this hypergraph to fit the cost function of the declustering problem. We adapt the iterative improvement based graph and hypergraph partitioning algorithms to this problem. We propose a two phase algorithm that first obtains an initial partitioning by recursively bipartitioning the hypergraph, then applying a K-way refinement on this partitioning. We provide effective gain models for both phases. Our experimental results show that the model we propose provides significantly better declustering then the graph model which is the most promising strategy in the literature.

We overview and discuss the literature on the declustering problem in Chapter 2. In this chapter, we also show the flaws of the graph model for declustering. We introduce our model and the adaptation of iterative improvement techniques to the problem in chapter 3. In Chapter 4, we report the experimental results showing the performances of the proposed algorithms. We finally discuss our contributions and the directions for future work in Chapter 5.

Chapter 2

Background

In very large database systems, parallel I/O is considered to be the main bottleneck by several researchers [28, 32]. In order to exploit the I/O bandwidth in multi-node database machines and multi-disk database systems, the relations are declustered. Declustering, or horizontal partitioning refers to placing the tuples belonging to a single relation on multiple disks [29]. There have been many research on developing strategies to effectively decluster the data on several disks in order to achieve minimum I/O cost. Many declustering strategies were developed on declustering multidimensional data structures such as cartesian product files, grid files, quad trees and R-trees [7, 12, 26, 28, 30, 32, 34], multimedia databases [2, 5, 27, 31, 33], parallel web servers [20], signature files [8], spatial databases and geographic information systems (GIS) [34, 35].

Most of the efforts on developing declustering strategies were based on mapping functions. These mapping-function based strategies include coordinate modulo declustering (CMD) [28], field-wise exclusive-OR distribution [25], Hilbert curve method [12, 19], lattice allocation method [10] and cyclic allocation scheme [32]. These methods are briefly discussed in Section 2.2.

A remarkable declustering method is using error correcting codes in order to partition binary strings into groups of unsimilar strings [8, 11, 12, 13]. The method is based on the idea of providing the strings in a group have large Hamming distances by grouping the strings in such a way that each group forms an error correcting code. The method was applied to problems such as declustering cartesian product files [11], grid files [7, 12] and signature files [8]. Local Load Balance (LLB) methods [21, 35] define a local window around the data item to be allocated, and map this data item to the disk with minimum load over the local window. This method was applied to parallelizing R-trees [21] and parallelizing geographic information systems (GIS) [35].

Graph theoretical models were applied to declustering problems by several researchers. Berchtold et.al. [2] model the declustering problem as a graph coloring problem by defining the disk assignment graph as an undirected graph with vertices corresponding to buckets and the edges corresponding direct and indirect relationships between buckets in the multidimensional data structure. As graph coloring is an NP-hard problem [36], they exploit some regularities in the specific graph and develop a simple yet efficient coloring algorithm.

Shekhar and Liu [34] introduced the idea of the similarity graph based on the similarity definition of Fang et.al. [14] and developed the max-cut graph-partitioning based declustering technique which outperforms all of the mapping-function based algorithms. The similarity graph partitioning approach will be discussed in detail in Sections 2.3 and 2.4. Moon et.al [30] applied Prim's minimal spanning tree algorithm to the similarity graph and proposed the minimax spanning tree algorithm. The algorithm grows K disjoint minimax spanning trees in round-robin order to obtain K groups of vertices with similar vertices in different groups. The algorithm uses a minimum of maximum cost criterion and selects the vertex that minimizes the maximum of all edge weights between itself and the already selected vertices. This algorithm provides exact storage balance in all disks, i.e. any disk can have a storage load of at most $\lceil \frac{N}{K} \rceil$ data items where N is the number of data items and K is the number of disks.

In the rest of this chapter, we will briefly introduce the mapping function based declustering algorithms in Section 2.2, discuss the similarity graph model in detail in Section 2.3, and finally show the flaws of the similarity graph model in Section 2.4. Before discussing the declustering methods in the literature, it will be appropriate to give the basic definitions on the declustering problem.

2.1 Basic Definitions on Declustering

Declustering problem can be defined in various ways depending on the application. Shekhar and Liu [34] define the problem in a database environment with given data set and a query set. Information on possible queries can be available in many database applications, the possible queries may be predicted using the information on the application or queries may be logged with the assumption that the queries that will be processed in the future will be similar to the recent ones. In some cases, information on queries may not be available and it can be more appropriate to decluster the data items in such a way that the data items sharing a feature are stored in separate disks. This can be the case in some multimedia servers [27, 31] or content-based image retrieval systems [18, 33]. Therefore it will be more convenient to provide a definition of the problem in terms of a set of data items and a set of relations between data items as in the work of Zhou and Williams [37]. The set of relations may refer to the query set or a possible query may be the union of a set of relations in many applications.

Definition 2.1 *A relation q_j on a set D of data items is defined to be a subset of D such that the data items in q_j are likely to be accessed together by the database system. Set of relations Q is the set containing all possible relations q_j on D. Function $f(q_j)$ maps the query set Q to a relative frequency, i.e. the probability that the items in q_j are expected to be accessed together.*

With this definition of a relation on a set of data items, a relation corresponding to a query becomes the set of data items that should be accessed in order to process that query as these data items are obviously likely to be accessed together. The relative frequency of a relation corresponds to the probability of processing the corresponding query.

Definition 2.2 *Given a partitioning of the set of data items D and a relation q_j, retrieval time $t(q_j)$ of relation q_j is defined as the cardinality of the largest set among the sets $q_{j1}, q_{j2}, ..., q_{jK} \subseteq q_j$, where subset q_{ji} of q_j is the set of data items in q_j assigned to disk i.*

This definition of the retrieval time of a relation corresponds to the time required to process a query if we assume that a data item is accessed by the database system in unit time.

Definition 2.3 *Given a set D of data items and a set of relations Q on D, declustering problem is defined as assigning the data items in D to K parts so that the total retrieval time over the set of relations $T(Q) = \sum_{q_j \in Q} f(q_j)t(q_j)$ is minimized.*

The cost function defined above has been used as the performance metric of the declustering methods in the literature, and Shekhar and Liu [34] use the average retrieval time as the metric to measure the quality of a declustering strategy which is equal to the cost defined above divided by the total relative frequency of queries. It is obvious that the retrieval time of a relation q_j cannot be lower than $\lceil \frac{|q_j|}{K} \rceil$ and this number forms the basis of analyzing the performance of an allocation method.

Definition 2.4 *An allocation method is strictly optimal with respect to a relation q_j if and only if $t(q_j) = \lceil \frac{|q_j|}{K} \rceil$.*

Definition 2.5 *An allocation method is strictly optimal with respect to a set Q of relations if and only if it is strictly optimal for every relation $q_j \in Q$.*

The extension of the definition of strict optimality of a relation and a set of relations to a query and a set of queries is straightforward from the above discussion. An allocation method is strictly optimal with respect to a query set if it achieves the minimum possible processing time for all queries in the set, i.e. if it provides maximum parallelism.

2.2 Mapping Function Based Declustering Techniques

Most of the work on declustering in the literature address the mapping-function based declustering techniques. These techniques take advantage of the spatial information on data items (buckets, pages etc.) and scatter the data items across disks in order to ensure that the data items that are more likely to be processed together by a query are stored in different disks. The methods try achieving this objective by maximizing the distance between any pair of data items that are assigned to the same disk in the n-dimensional data space. We will define and discuss these algorithms briefly in this section. In the rest of this chapter, we will denote the number of data items by N, number of disks by K, the number of dimensions in the database by n, a data item $d_i \in D$ in n-dimensional space by vector $\overrightarrow{d_i} = (X_1, X_2, ..., X_n)$ and the function that maps a data item $\overrightarrow{d_i}$ to a disk by $Disk(\overrightarrow{d_i})$.

Definition 2.6 *Coordinate Modulo Declustering (CMD) is the allocation method that maps data item $\overrightarrow{d_i}$ to disk $Disk(\overrightarrow{d_i}) = (\sum_{j=1}^{n} X_j)$ mod K.*

Li et al. [28] show that CMD provides exact storage balance and is strictly optimal for all range queries whose length in some dimension is equal to kK where $k \in Z^+$.

Definition 2.7 *Field-wise exclusive-or distribution method (FX) is the declustering strategy with mapping function $Disk(\overrightarrow{d_i}) = (X_1 \otimes X_2 \otimes ... \otimes X_n)$ mod K.*

Kim and Pramanik [25] showed that when the number of disks and the size of each field are powers of two, the set of partial match queries that CMD is strictly optimal for is a subset of that of FX. The probability that FX will be strictly optimal with respect to a range query is greater than that of CMD [12].

Definition 2.8 *Hilbert Curve Allocation Method (HCAM) is the declustering strategy that imposes a linear ordering on the data items with a space-filling* *curve in n-dimensional space, and traverses the sorted list of data items by assigning data items to disks in a round-robin fashion.*

HCAM was proposed by Faloutsos and Bhagwat [12] in order to apply the good clustering properties of space filling curves to the declustering problem. Hilbert curve visits all points in a d-dimensional grid exactly once and never crosses itself. This property of Hilbert curve ensures that neighboring data items will be close to each other on the linear ordering, and thus assigned to separate disks. It is shown experimentally that HCAM achieves better declustering than CMD, FX and error correcting codes [12].

Definition 2.9 *A lattice allocation method in 2-dimensional space with basis vectors* $\overrightarrow{a} = (a_0, 0)$ *and* $\overrightarrow{b} = (b_0, b_1)$ *where* $b_0 \leq a_0$ *and* a_0, b_0, b_1 *integers is defined by the mapping function:*

$$Disk((X_0, X_1)) = \begin{cases} (X_0 \; mod \; a_0) + X_1 a_0, & if \; 0 \leq X_0 \leq a_0 \;\; and \;\; 0 \leq X_1 \leq b_1 \\ Disk((X_0 - b_0(X_1 \; div \; b_1)) \; mod \; a_0, X_1 \; mod \; b_1), & otherwise \end{cases}$$

Lattice allocation methods are designed to parallelize the set of small range queries. The performance of lattice allocation methods depends on the query distribution [10].

Definition 2.10 *Cyclic allocation methods are mapping functions defined as*

$$Disk(\overrightarrow{d}) = (\sum_{i=1}^{n} H_i \; X_i) \; mod \; K \quad , \; H_1 = 1$$

where H_i, $i = 1..n$ *are constants specified by the allocation scheme.*

Obviously, CMD is a special case of the cyclic allocation methods. It is proved in [32] that for any cyclic allocation, the cost of the query depends only on its shape, not its location. Prabhakar et al. provide methods for determining the H_i values in order to obtain minimum load imbalance while processing an arbitrary query. The reported experimental results show that proposed cyclic allocation methods perform better than HCAM, CMD and

FX [32]. Generalized disk modulo (GDM) method is a cyclic allocation method with $H_2 = 5$.

All methods discussed above are designed for cartesian product files. These methods can be applied to multidimensional data structures like grid files and R-trees by introducing greedy algorithms for decision making in the case of conflicting disk assignments. The conflicting assignments are caused by page sharing which can be defined as the situation that multiple cells in the spatial database are included by one data page or bucket. These greedy algorithms include selecting the disk with minimum number of data items (data balance), choosing the disk that occurs the most often in the conflicting mappings (most frequent) and selecting the disk with minimum total area of assigned data items (area balance) for a data item with conflicting alternatives [30]. However, for grid files, with high degree of page sharing the number of conflicts become very high resulting in poor performance of the mapping function based strategies. Additionally, the mapping function based methods are designed by the assumption that the disks are homogeneous in terms of both their storage and I/O capacities. However, storage and I/O capacities of disks may differ in many situations and these strategies may perform very poor since they do not consider any information about disk capacities. Therefore, mapping function based methods are limited to spatial databases, a number of indexing techniques and homogeneous database environments. Mapping function based techniques cannot be applied to databases with no spatial information on the data items. For instance, an image database with binary signature files recording significant wavelet coefficients cannot be represented by spatial relationships [18]. In the case of spatial databases, Shekhar and Liu provide experimental results obtained on grid files which show that their Max-cut graph partitioning model outperforms a number of mapping function based methods [34]. Therefore we will discuss the max-cut graph partitioning model of Shekhar and Liu as a promising and general declustering strategy in the next section.

2.3 Weighted Similarity Graph Model

Shekhar and Liu proposed an elegant graph model for the declustering problem and provided theoretical analysis of correctness of their model [34]. The model is based on the similarity concept defined by Fang et al. [14]. They define a weighted similarity graph corresponding to a set of data items and a query set and define an objective function that approximates the cost of processing a query in the database system.

Definition 2.11 *Given a set D of data items and a query set Q, weighted similarity graph $WSG(D,Q) = (V,E)$ is defined to be the graph with vertex set $V = D$ and edge set $E = \{e(u,v) \mid u,v \in V$ and $\exists\ q_j \in Q$ s.t. $u,v \in q_j\}$. Each edge $e(u,v) \in E$ is associated with a weight $w(u,v) = \sum_{q_j \in Q_{uv}} f(q_j)$, where $Q_{uv} \subseteq Q$ is the set of all queries such that $u,v \in q_j$ and $f(q_j)$ is the relative frequency of query q_j.*

With this definition of weighted similarity graph, it becomes obvious that the larger the weight of the edges between two vertices of WSG, the more the two corresponding data items in the database are likely to be processed together. Observing this property of WSG, the similarity between two groups, i.e. two subsets of the vertex set of WSG is defined as follows.

Definition 2.12 *Let $WSG(V,E)$ be a weighted similarity graph. Then the similarity between two vertex subsets V_i and V_j of $V(WSG)$ is defined as*

$$s(V_i,V_j) = \sum_{u \in V_i} \sum_{v \in V_j} w(u,v)$$

As the definition of the declustering problem enforces similar data items to be allocated in separate disks, all disks should be similar to each other with respect to the data items they contain. In terms of WSG, the partitioning of WSG should enforce that all K groups defined by the partitioning of the graph should be similar to each other for effective declustering. Shekhar and Liu [34] conclude from this point that the weighted similarity graph should be partitioned in such a way that for a given pair of groups V_i and V_j, the

$s(V_i, V_j)$ values will be as high as possible. Therefore, the objective function of a partition $\Pi(V)$ of a weighted similarity graph $WSG(V, E)$ is defined as maximizing the metric

$$S(\Pi(V)) = \sum_{\forall\ V_i, V_j \subset V\ \ i \neq j} s(V_i, V_j) = \sum_{e(u,v) \in E_c} w(u, v)$$

where E_c is defined as the set of all edges $e(u, v)$ such that $u \in V_i$, $v \in V_j$ $i \neq j$, namely the cutset of the partitioning. Thus the WSG should be partitioned to maximize the cut in order to obtain similar subsets of the vertex of the weighted similarity graph. Shekhar and Liu [34] define max-cut graph partitioning as follows.

Definition 2.13 *Max-Cut partitioning of the weighted similarity graph is defined as: Given a weighted similarity graph $WSG = (V, E)$, the number of disks K, and the disk capacity constraints L_i for each disk i, find a partition $\Pi(V) = (G_1, G_2, ..., G_K)$ among K disks to maximize $S(\Pi(V)) = \sum_{e(u,v) \in E_c} w(u, v)$ which is the total weight of the edges in the cut set, such that $L_i(G_i) = True\ \forall i\ 1 \leq i \leq K$.*

The max-cut graph partitioning method is a heuristic approach for declustering problems. It exploits the concept of obtaining similar groups of data items in order to ensure that similar data items are contained in separate groups. However, Shekhar and Liu provide a number of lemmas and theorems with proofs showing that this heuristic exhibits optimality under special conditions. The following theorem states the condition for obtaining optimal solution via max-cut graph partitioning method. We do not prove the theorem here as the proof is provided in [34].

Theorem 2.1 *If there exists a strictly optimal allocation method for a query set Q, the max-cut graph partitioning method is also strictly optimal with respect to the query set Q.*

As the max-cut graph partitioning problem is NP-complete, Shekhar and Liu propose two heuristics to solve the problem. The first heuristic is named incremental max-cut declustering algorithm (SM-INCR) and aims at allocating data items in order to fulfill the objective of maximizing the cut in a local window around each data item in a greedy manner. The second heuristic named global max-cut graph partitioning (SM-GP) transforms max-cut graph partitioning problem into the well known K-way min-cut graph partitioning problem by inverting the weight of each edge, then applies the modified ratio-cut heuristic of Cheng and Wei [6] which is a move-based two-way partitioning heuristic. If the number of disks K is a power of 2, SM-GP algorithm recursively performs two-way partitioning until K parts are found, else it performs two-way partitioning algorithm to produce a set of $\frac{|V|}{K}$ vertices and a set of remaining vertices, and repeats this procedure $K-1$ times on the set of remaining vertices in order to find K balanced subsets of the vertex set of the WSG. Then the partitioning is improved by applying the two-way partitioning procedure to the selected pairs of K parts.

Shekhar and Liu [34] compared the similarity graph based declustering model with declustering methods HCAM, GDM and LLB with experiments on parallelizing grid files with 16 disks and the results are reported in [34]. The results show that the WSG model outperforms other declustering strategies for all row/column, square and diagonal query sets on uniform and hot-spot data sets. SM-GP provides better quality results than SM-INCR. The experiments are performed with two variations of SM-GP, a general max-cut graph partitioning technique (SM-GP-G) and a technique adapted to query sets (SM-GP-S) and it is reported that SM-GP-S provides better quality results than SM-GP-G on parallelizing grid files. The effect of the number of disks is not included in the experimental study with the assumption that the number of disks does not affect the performance of an allocation method, however we show in Section 2.4 that the performance of the WSG model is degraded by increasing number of disks.

2.4 Flaws of Weighted Similarity Graph Model

Although WSG is an elegant model that finds the optimal allocation if exists, there are some points that the objective function of the model does not fit the actual cost function of the declustering problem and these points may cause the method to make serious errors for hard instances of the problems. In this section, we will exploit the flaws of the model with theoretical analysis and examples to clarify the situations where WSG is more likely to make errors.

In order to accurately model the cost function of the physical problem, the objective function of the model must be proportional to the actual cost function. In other words, a model fits the physical problem if one can say that the higher/lower the objective function of the model the lower the cost function of the model. The objective function of the WSG model is maximizing the cut, so we can define the objective function of max-cut graph partitioning to be the cut i.e. $S(\Pi_K) = \sum_{e(u,v) \in E_c} w(u,v)$. As the objective of the graph model is maximizing this function, one should be able to say that the higher the cut on the WSG of the database system, the lower the cost function or average retrieval time of the system. However, this is not the case for the max-cut graph model. We can see the intuition behind our claim if we define the cost function as the sum of the cost functions for each relation.

For the weighted similarity graph model, a relation $q_j \in Q$ induces a clique of $|q_j|$ vertices which corresponds to the data items in that relation. We can define the cut due to a relation as the cut on this clique. The sum of the cuts due to all relations is clearly equal to the total cut on the WSG. The cost or retrieval time of a relation was defined in Section 2.1. If we compare these two functions, we can observe that the relation between these two functions is not linear, moreover they are not proportional. The relation between the retrieval time of a relation and the cut due to a relation is not linear, so the sum of the retrieval times/cuts over the relations may be inconsistent. For example, for a two disk system, the cut due to a relation q_j can be formulated as $|q_{j1}|(|q_j|-|q_{j1}|)$ although the retrieval time of q_j is equal to $\max(|q_{j1}|, |q_j|-|q_{j1}|)$. The functions are also not proportional, i.e. the statement that "if the cut due to relation q_j is higher than the cut due to relation q_i, the cost of q_j is lower than the cost of q_i" is not always true. For example if we look for an allocation to 4 disks, and if two relations q_i and q_j of size 5 are distributed among these 4 disks as [2 1 1 1] and [2 2 1 0] respectively, the cut due to q_i will be higher than the cut due to q_j although the retrieval times of these two relations are the

q1= {d1, d2, d3}
q2 = {d1, d4}
q3 = {d1, d5}
q4 = {d2, d4}
q5 = {d2, d5}
q6 = {d3, d4}
q7 = {d3, d5}

Database System

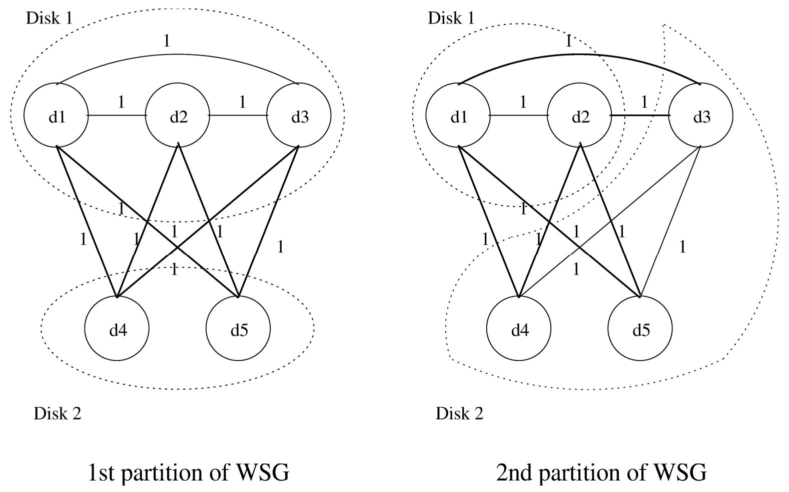

Figure 2.1: Sample partition of WSG with given database information

same. The statement is only true if the sizes of these relations are at most two or exactly equal to two as a relation of size 1 is trivial. This is because of the mathematical fact that the multiplication of two numbers with constant sum is maximum if the difference between them is at most one. The graph model only fits to the case of relations of size two as the graph model can only represent the relations between pairs of vertices. This observation shows the necessity of a model that can represent a relation between a set of vertices independent of the cardinality of the set.

The example of Figure 2.1 illustrates the effect of the nonlinearity of the relation between the retrieval time of a relation and the cut due to a relation. The figure shows a database system with query set $Q = \{q_1, ..., q_7\}$ of 7 queries and a data item set of 5 data items. The relative frequencies of all queries are assumed to be equal and q_1 accesses 3 data items while the other queries access two data items each. There is no strictly optimal allocation for this set of queries. The WSG of this database system and two different partitions of this graph are shown in the figure. The edges in the cutset of each partition are indicated by bold lines. The total cut of the first partition is equal to 6 and the allocation is strictly optimal with respect to all queries other than q_1. The cut corresponding to q_1 is estimated to be zero. In the second partition of the WSG, the total cut is again equal to 6, but now this partitioning is not strictly optimal with respect to two queries, q_6 and q_7. The total actual cost or retrieval time of the first partitioning is 9 while the second one has cost 10.

This difference is the result of the over-estimation of the cost due to q_1. The difference between the cuts corresponding to queries q_6 and q_7 of two partitioning schemes are equal to 1 which is equal to the difference between actual costs of these queries. However, the difference between the cuts corresponding to q_1 is estimated to be equal to 2 while the difference between the actual costs of this query is also 1. Assuming that the second partition was obtained at an instance of the max-cut partitioning of the WSG, the partitioning tool will not move further vertices because the maximum achievable cut on this graph is equal to the cut of this partitioning. Unfortunately, there exists a better partition of this database which is the first partition but this partition is missed by the similarity graph partitioning model.

The weighted similarity graph model can also estimate the cut corresponding to a query lower for lower cost of that query and vice versa. This results in the selection of higher cost partitioning as such partitioning can provide higher cut in the case of conflicts between queries. Table 2.1 shows several examples of this phenomenon. Two different distributions of a query of given size into given number of parts and the actual cost functions and the cost corresponding to that query with such partitioning are displayed in the table. In the first example, a query of size 11 is partitioned into 3 disks. In the first partitioning, the query accesses 5 data items in a part and the cut corresponding to this query is 35. However, with a partitioning that provides a cost of 4 for this query results in a cut of 36, i.e. this partitioning is preferred to the previous one by the max-cut graph partitioning model. This deficiency of max-cut graph

Table 2.1: Examples of false cost estimation of similarity graph model

K	**Query size**	**Distribution**	**Cost**	**Cut**
3	11	[5, 5, 1]	5	35
		[6, 3, 2]	6	36
4	9	[3, 3, 3, 0]	3	27
		[4, 2, 2, 1]	4	28
6	15	[4, 4, 4, 1, 1, 1]	4	88
		[5, 2, 2, 2, 2, 2]	5	90
8	8	[2, 2, 2, 2, 0, 0, 0, 0]	2	24
		[3, 1, 1, 1, 1, 1, 0, 0]	3	25

partitioning model comes from the fact that the max-cut objective models the variance on the distribution of queries to disks rather than the imbalance of the distribution which depends only on the maximum number of data items accessed by a query allocated into one disk. With greater number of disks, the degree of freedom is greater, so the number of deviations which are not related to the maximum of the distribution is greater. As max-cut graph partitioning tries balancing such deviations, the probability of erroneous cost estimation is higher in greater number of disks. This degrades the scalability of the similarity graph model. The correctness of this observation will be shown by experimental findings displayed in Chapter 4.

Chapter 3

Hypergraph Model for Declustering

The deficiencies of the graph partitioning based declustering model source from the same basis as that of graph partitioning based sparse matrix reordering models [3, 4]. Modeling the relation between N items with $N(N-1)/2$ pairs of relations between all pairs of these items masks the conflicts between original relations. Therefore, the problem should be modeled in such a way that each relation defined on a set of a number of items in the item set can be captured by the model. Çatalyürek and Aykanat [4] modeled the sparse matrix reordering problem via hypergraphs, preserving the significance of the relations between rows/columns having non-zeros on the same column/row of a sparse matrix and obtained a significant performance improvement when compared with the graph model. A hypergraph is a generalized version of a graph in which an edge(hyperedge) can define a relation between more than 2 vertices. This property of hypergraph makes it capable of exactly modeling a set of relations on a set of items independent of the cardinality of sets defined by the relations. In other words, a relation of N items can be represented by an hyperedge of N vertices in a hypergraph.

We exploit the accurate modeling ability of hypergraphs and model the problem of declustering very large databases as an hypergraph partitioning problem with a new cost function. In this chapter, we will define the original hypergraph partitioning problem in Section 3.1, then explain the model we propose on the declustering problem in Section 3.2 and finally provide algorithms for solving the proposed hypergraph based declustering problem in Section 3.3.

3.1 Hypergraph Partitioning Problem

The hypergraph partitioning model has been used for solving the VLSI circuit partitioning problem [1, 15] and for reordering sparse matrices for efficient parallelization of iterative sparse matrix problems recently [3, 4].

Definition 3.1 *A hypergraph $H = (V, N)$ is defined as a set of vertices(cells) V and a set of nets(hyperedges) N among those vertices. Every net $n_j \in N$ is a subset of vertices, i.e. $n_j \subseteq V$. Each vertex in a net is called a pin of the net. The size of a net n_j is equal to the number of its pins, $|n_j|$. The set of nets containing a vertex v_i is called the nets of v_i and denoted as $nets(v_i)$. The cardinality of $nets(v_i)$ is called the degree d_i of v_i, i.e. $d_i = |nets(v_i)|$. A hypergraph with all nets having size 2 is a graph.*

The nets or vertices of the hypergraph can be associated with a weight function.

Definition 3.2 *A K-way partitioning $\Pi_K(H)$ of a hypergraph H is a mapping of vertex set V of H to K disjoint groups. A net with at least a pin mapped to a part is said to be connected to that part. The cut-set $N_c(H, \Pi_K)$ is the set of nets that are connected to more than one part. The connectivity Λ_j of net n_j is defined to be the number of parts that n_j is connected to.*

The min-cut hypergraph partitioning problem is similar to the min-cut graph partitioning problem: find a partitioning of the hypergraph that minimizes the number or the total weight of nets in the cut set of the partitioning. Another cost function of the hypergraph partitioning problem is applied to the sparse matrix reordering problem [4] and defined as the difference between total connectivity of the nets and the number of nets in the hypergraph. The hypergraph partitioning problem is solved by the iterative improvement based heuristics frequently used for the graph partitioning problem. The iterative improvement based partitioning algorithms start with an initial partitioning of the hypergraph and swap the parts of two vertices [24] or move a vertex to a different part [15] repeatedly in order to improve the quality of the partitioning. The quality of direct K-way partitioning strictly depends on the initial partition and the time and space consumed by direct K-way partitioning is high. Therefore, an initial partition is found by recursively bipartitioning the hypergraph and the partition is refined by direct K-way partitioning scheme in order to obtain high quality partition while consuming less time and space.

3.2 Hypergraph Based Declustering Model

We model the problem of declustering large databases as a hypergraph partitioning problem with a cost function that matches the I/O cost of processing a query in a multi-disk database system. The data items in the database are represented by the vertices of the hypergraph and the relations among these data items are represented by its nets. This definition of the relational hypergraph of a database system exactly represents the system.

Definition 3.3 *The relational hypergraph $H(D, Q)$ of a database system with data item set D and relation set Q among these data items is the hypergraph with vertex set $V = D$ and net set $N = Q$. Each relation $q_j \in Q$ defines a net $n_j \in N$ with $n_j = q_j$. Each net n_j representing a relation q_j is associated with a weight function $w_j = f(q_j)$ representing the relation's relative frequency.*

The definition of declustering enforces partitioning the relational hypergraph to find a mapping of the vertices of the hypergraph in such a way that the number of pins of each net in the part that contains the maximum number of pins of that net is minimized. We define the partitioning of the relational hypergraph of a database system on this objective.

Definition 3.4 *In a K-way partition $\Pi_K = \{V_1, V_2, ..., V_K\}$ of H, $n_j(k) \subset n_j$ denotes the subset of pins of net n_j that lie in part V_k, i.e. $n_j(k) = n_j \cap V_K$ for k=1,2,...,K. Cardinality $|n_j(k)|$ of set $n_j(k)$ is called the degree of connectivity of n_j to part P_k. $\delta_j = \max_{1 \le k \le K}\{|n_j(k)|\}$ denotes the maximum degree of connectivity of net n_j. $\delta_j^{opt} = \lceil\frac{|n_j|}{K}\rceil$ denotes the strictly optimal degree of connectivity of net n_j. The cost $c_{\Pi_K}(n_j)$ of net n_j due to partition Π_k is defined as the difference between maximum degree of connectivity and strictly optimal degree of connectivity of n_j, i.e.*

$$c_{\Pi_K}(n_j) = \max_{1 \le k \le K}\{|n_j(k)|\} - \lceil\frac{|n_j|}{K}\rceil.$$

Obviously, the maximum cardinality of sets $n_j(k)$ is bounded from below by $\lceil\frac{|n_j|}{K}\rceil$, so the partitioning can achieve this value at its best. Therefore we define the cost of the partitioning with respect to a net as the penalty of exceeding the strictly optimal degree of connectivity of that net. The cost of a net in the partitioning of a relational hypergraph defined above is linearly proportional to the cost of a relation due to an allocation scheme defined in Section 2.1 by the function $c_{\Pi_K}(n_j) = cost(q_j) - \lceil\frac{|q_j|}{K}\rceil$. Therefore this cost function accurately represents the extra time spent to process a query due to the imbalance of the partitioning of the data items accessed by a query if that query accesses the set of data items belonging to a single relation in the database system. If the set of data items to be accessed in order to process a query is the union of some relations, this cost function is an upper bound of the extra time spent. However, if no information is available on what query will access which relations, then this upper bound is the only function representing the extra time spent to process a query. Thus, we can conclude that the cost function defined above accurately matches the I/O cost of a multi-disk database system. The cost of a K-way partitioning of a relational hypergraph is defined similarly to be linearly proportional to the actual cost function of the declustering problem. We call this partitioning scheme as the min-net-imbalance partitioning of a hypergraph.

Definition 3.5 *The K-way min-net-imbalance partitioning of a hypergraph is a mapping $\Pi_K(H)$ defined on the set V of the vertices of the hypergraph which* *minimizes the cost function*

$$C_{\Pi_K}(H) = \sum_{n_j \in N(H)} w_j c_{\Pi_K}(n_j) = \sum_{n_j \in N(H)} w_j \Big(\max_{1 \le k \le K} \{ |n_j(k)| \} - \lceil \frac{|n_j|}{K} \rceil \Big)$$

satisfying the storage capacity constraints $L_i(V_i) = True\ \forall i\ 1 \le i \le K$.

3.3 Algorithms for Partitioning the Relational Hypergraph of a Database System

The min-cut and min-connectivity hypergraph partitioning problems are solved by iterative improvement based multi-level tools like PaToH [3, 4] and hMeTiS [22, 23]. In order to obtain a K-way partitioning of the hypergraph, these tools find a bipartitioning of the original hypergraph and split it into two hypergraphs with vertex sets consisting of the vertices mapped to the first part for the first hypergraph and the second part for the second hypergraph and net sets containing nets with pins as subsets of the corresponding nets in the original hypergraph containing the vertices in the corresponding parts. This procedure is recursively applied to the hypergraphs created by splitting the original hypergraph until K parts are found. This procedure is called recursive bipartitioning and is less time and space consuming than direct K-way partitioning. For this reason, the K-way partitioning of a hypergraph is performed in two phases, the recursive partitioning phase to obtain an initial partition of the vertices followed by a K-way refinement phase in order to increase the quality of the partitioning [3, 4, 22, 23]. This procedure is preferred to direct K-way partitioning methods in terms of both resource usage and partitioning quality.

We propose a two-phase algorithm for partitioning a relational hypergraph as in the case of min-cut partitioning. Our heuristic is based on the iterative improvement based graph/hypergraph partitioning heuristics [15, 24] extensively used in VLSI circuit partitioning and sparse matrix reordering applications. The basics of the iterative improvement based heuristics are explained in Section 3.3.1. In our method, an initial K-way partition of the hypergraph is obtained via recursive bipartitioning with an optimistic cost model and then a fast K-way refinement heuristic is applied to the initial partition. As the cost function of the min-net-imbalance partitioning problem is a non-linear function of the K-way mapping, it cannot be applied directly to the recursive bipartitioning scheme. Therefore, we propose an optimistic cost model that considers the final K-way cost of the partitioning and include a memory concept that relates the independent bipartitioning steps. The details of the method are explained in Section 3.3.2. In order to save time and memory space during the K-way refinement phase, we propose a fast K-way refinement heuristic using the concept of virtual gain to approximate the actual gain of moving a vertex. The proposed method is introduced in Section 3.3.3.

3.3.1 Iterative Improvement Based Partitioning Algorithms

The well-known circuit partitioning problem is the problem of allocating the nodes of the circuit to K parts in order to minimize the sum of the costs of the edges between the nodes in separate parts satisfying the balance constraints on parts. Kernighan and Lin [24] proposed an efficient heuristic for 2-way partitioning of a graph. Their algorithm starts with a balanced random initial partition of the vertices in the graph and tries to improve the quality of the partitioning with respect to the cost function by swapping the parts of selected vertices repeatedly. The gain of swapping a vertex pair is defined as the decrease in the total cost on the cut that will be caused by swapping these vertices. The algorithm searches for the set of ordered swappings that will provide the largest total gain. This is done by repeatedly selecting a pair of unlocked vertices with the highest gain, swapping them temporarily and locking them until all vertices are locked. The vertices are locked in order to prevent infinite loops, i.e. repeated swapping of some set of vertices. All vertices are exhausted in order to climb out local minima of the cost function. After all vertices are exhausted, the point in the swapping process that gives the maximum cumulative gain is selected and the swapping operations before that point are realized. This procedure is named a pass and repeated until an improvement on the cost function cannot be achieved.

```
**procedure** RecursivePartitioning($H$: hypergraph, $K$: number of parts)

**begin**
    **if** $K = 1$ **then**
        stop
    **else**
        find initial disjoint subsets $V_{left}$ and $V_{right}$ of $V(H)$
        **repeat**
            $UL \leftarrow \emptyset$
            $M \leftarrow \emptyset$
            $count \leftarrow 0$
            $cumgain[0] \leftarrow 0$
            Initialize gains of all vertices $gain(v_i)$
            $UL \leftarrow V(H)$
            **while** $UL \neq \emptyset$ **do**
                select $v_i \in UL$ with $gain(v_i) \geq gain(v_j)\ \forall\ v_j \in UL$
                **if** moving $v_i$ does not violate balance constraint **then**
                    $count \leftarrow count + 1$
                    $M[count] \leftarrow v_i$
                    $cumgain[count] \leftarrow cumgain[count - 1] + gain(v_i)$
                    update gains of all $v_j \in UL$ assuming $v_i$ is moved
                $UL \leftarrow UL \setminus v_i$
            select $count$ maximizing $cumgain[count]$
            **for** $i = 1$ **to** $count$ **do**
                move vertex $M[i]$ to the other part
            $passgain \leftarrow cumgain[count]$
        **until** $passgain < 0$
        split $H$ to obtain $H_{left}$ and $H_{right}$ with vertex sets $V_{left}$ and $V_{right}$
        call RecursivePartitioning($H_{left}$, $K/2$)
        call RecursivePartitioning($H_{right}$, $K/2$)
**end**
```

Figure 3.1: KL-FM algorithm for hypergraph partitioning

Fiduccia and Mattheyses [15] improved Kernighan and Lin's heuristic by introducing the concept of single vertex move and implementing the algorithm using buckets for selecting the vertex with maximum gain for partitioning of a hypergraph. Their algorithm improves the bipartitioning by moving a vertex from one part to the other instead of swapping a pair vertices. This approach provides more flexibility for selecting the set of vertices to be moved. The move gain of a vertex during partitioning a hypergraph is the difference between total weight of nets that are connected to that vertex's part with only that vertex and the total weight of the nets that are not connected to the other part. The algorithm proceeds as Kernighan and Lin's algorithm and it establishes balance by starting with a balanced initial partition and permitting the vertices to move to the other side if the move will not exceed a pre-specified imbalance tolerance. The algorithm is implemented by using buckets to store the gains of the vertices with the observation that the maximum possible gain of a vertex for a given hypergraph is bounded by the maximum vertex degree in the hypergraph. The use of buckets provides fast update of the gains after a move and it has been shown by Fiduccia and Mattheyses [15] that the algorithm has a linear time complexity in the order of total number of pins of the hypergraph with such implementation. The iterative improvement based partitioning algorithm is known as KL-FM algorithm and summarized in Figure 3.1.

3.3.2 Initial Recursive Bipartitioning of Relational Hypergraph

To solve the K-way relational hypergraph partitioning problem, we are encouraged to start with an initial partitioning obtained by recursive bipartitioning of the hypergraph because recursive bipartitioning is less time and space consuming when compared to direct K-way partitioning. A partitioning obtained by recursive bipartitioning is more likely to be close to an optimal solution than a random partitioning of the hypergraph, therefore K-way refinement of such partitioning will be performed in significantly less number of passes than that of random partitioning and the quality of the final partitioning will be better for such initial partitioning.

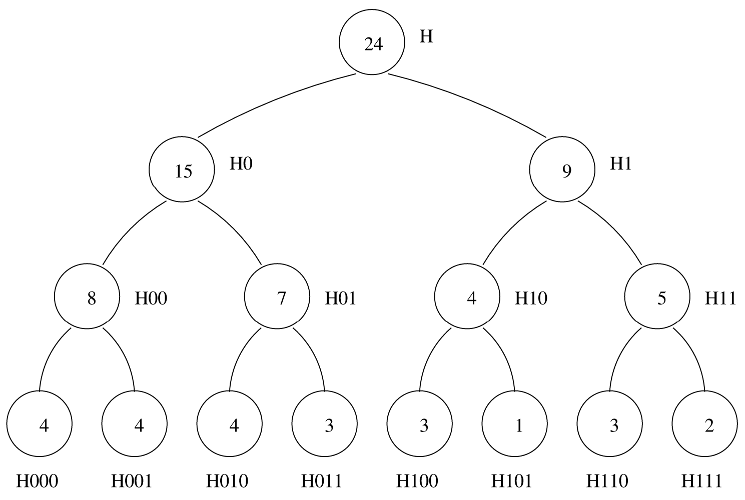

Figure 3.2: Pin distribution of a net of size 24 during recursive bipartitioning

The cost of a K-way partitioning of a relational hypergraph depends on the maximum degree of connectivity of each net with respect to the K parts defined by the partitioning. Thus, the cost function of a 2-way partitioning in the recursion tree of recursive bipartitioning does not fit the cost of the final K-way partitioning as it depends only on the the two parts obtained at that level of the recursion tree.

After bipartitioning a hypergraph at a level of the recursion tree, a net is splitted into two nets in the two child hypergraphs containing the pins of the net in each part. During bipartitioning, it will be inconvenient to take into account the maximum degree of connectivity of a net due to two parts as shown by the example of Figure 3.2. The figure shows a recursion tree of recursive 8-way partitioning of a hypergraph and the numbers in the circles representing the nodes of the tree show the number of pins of a net in the hypergraphs in the recursion tree. The size of the net is 24 in the original hypergraph H, and the number of pins in each part obtained by bipartitioning are 15 and 9. The cost of this net due to the bipartition is $15 - 12 = 3$ as maximum degree of connectivity of this net due to the bipartition is 15 and its ideal degree of connectivity is 12. However this net's maximum degree of connectivity is 4 and strictly optimal degree of connectivity is 3 resulting in a cost of only 1 due to the 8-way partition. The bipartitioning in the first level of the recursion tree overestimates the cost of the net as it does not take into account the degree of freedom gathered by further partitioning of the obtained hypergraphs. No matter if the maximum degree of connectivity of this net to the first-level bipartitioning was 13 or 16, the minimum achievable maximum degree of connectivity would be $\lceil\frac{13}{4}\rceil = \lceil\frac{16}{4}\rceil = 4$ in the further 4-way partitioning of H_0. However if this net had 17 pins in H_0 after the first-level bipartitioning, the best-case maximum degree of it would be $\lceil\frac{17}{4}\rceil = 5$, causing an increment in the cost of the net due to the 8-way partition.

Observing that the cost of a bipartitioning at one level of the recursion depends on the number of parts that will be obtained after further partitioning, we propose a cost model that takes into account the best-case performance of further partitioning. As the expected maximum degree of connectivity at the final level of the K-way partitioning grows with the function $\lfloor\frac{\delta_j}{\kappa}\rfloor$, we define a cost function that depends on the integer division of the maximum degree of connectivity of a net by the number of parts that will be obtained by further partitioning each of the hypergraphs obtained by that bipartitioning step denoted by κ.

Definition 3.6 *The cost $c_{\Pi_2}(n_j, \kappa)$ of net n_j due to bipartition Π_2 in the recursion tree of a K-way partitioning of a relational hypergraph is defined as*

$$c_{\Pi_2}(n_j, \kappa) \;=\; \lceil\frac{\delta_j - \delta_j^{opt}}{\kappa}\rceil \;=\; \lceil\frac{\max\left(|n_j(1)|, |n_j(2)|\right) - \lceil\frac{|n_j|}{K}\rceil * \kappa}{\kappa}\rceil,$$

where κ is the number of parts that will be obtained by further partitioning the children of the hypergraph at that level of the tree. The cost of bipartition Π_2 is the sum of the costs of each net, i.e. $C_{\Pi_2}(N, \kappa) = \sum_{n_j \in N(H)} w_j c_{\Pi_2}(n_j, \kappa)$.

This approach provides the flexibility of moving further vertices to the part containing a number of pins of a net that exceeds the ideal degree of connectivity of that net without increasing the expected cost of that net due to the final K-way partitioning. This definition of the cost function of a bipartitioning leads to simple algorithms for initializing and updating gains of vertices.

In the example of Figure 3.2, hypergraph H_{10} is partitioned in a non-optimal manner, with cost equal to 1 for this net. However, as the maximum degree of connectivity of this net due to K-way partition is 4, this cost is overestimated, i.e. there is no cost of that imbalanced bipartition on the overall cost of the

```
procedure InitBestOfNets($H$:hypergraph, $K$: number of parts)

begin
        for each $n_j \in N(H)$ do
            $bestof[n_j] \leftarrow \lceil \frac{|n_j|}{K} \rceil$
end
```

Figure 3.3: Initialization of the table containing best-case maximum degree of connectivity of each net

```
procedure UpdateBestOfNets($H$: hypergraph, $\kappa$: number of parts for further
partitioning)

begin
        for each $n_j \in N(H)$ do
            $\delta_j \leftarrow \lceil \frac{|n_j|}{\kappa} \rceil$
            if $\delta_j > bestof[n_j]$ then
               $bestof[n_j] \leftarrow \delta_j$
end
```

Figure 3.4: Update of the global table with given resulting hypergraph after a bipartitioning step

partitioning. Thus the cost of a bipartition at a level of the recursion tree should be estimated in such a way that the best-case performance of the other bipartitioning steps are taken into account. Such an effort will provide the flexibility of permitting the algorithm not to consider the nets that are sacrificed by the previous bipartitioning steps. We introduce the concept of the global best-case maximum degree of connectivity in order to take advantage of this observation.

Our algorithm keeps a table containing the minimum achievable maximum degree of connectivity of each net, and each bipartitioning step updates this table with its information on the size of the nets in the two children of the partitioned hypergraph. This table initially contains the strictly optimal degree of connectivity of each net due to K-way partition. After a child hypergraph is created by a bipartitioning step, the best-case maximum degree of connectivity of the child net of each net in this hypergraph is estimated for further partitioning and stored in the global table in order to provide information for other bipartitioning steps. A bipartitioning step respects the minimum achievable maximum degree of connectivity as the strictly optimal degree of connectivity of a net for its own gain estimations. Our algorithm updates the table containing the minimum achievable maximum degree of connectivities of all nets after a bipartitioning step, and uses these updated values to estimate the gains of vertices in the next bipartitioning step. The procedures for maintaining the global table of best-case maximum degree of connectivity are given in Figures 3.3 and 3.4. Procedure *InitBestOfNets* initializes the table with the prior knowledge of strictly optimal degree of connectivity of each net. Then, after a child hypergraph is created as a result of a bipartitioning step, the procedure *UpdateBestOfNets* computes the strictly optimal degree of connectivity with respect to the size of each child net in this new hypergraph and the number of parts this hypergraph will be partitioned into, and if this value exceeds the previously stored best-case maximum degree of connectivity, it updates the table in order to provide the knowledge that the maximum degree of connectivity for that net can be no smaller from now on. The best-case maximum degree of connectivity of net n_j is stored in the variable $bestof[n_j]$.

The estimation of gains in each bipartitioning step depends on κ, the number of parts each resulting part will be partitioned. For the initialization of the gains of vertices during bipartitioning, the estimation is divided into two cases of $\kappa = 1$, i.e. no further partitioning will be performed, and $\kappa > 1$, i.e. the resulting parts will be further partitioned. Figure 3.5 shows the algorithm for initializing gains. Procedure *InitGains* takes the value in the global table of best-case maximum degree of connectivity of nets and estimates the optimal maximum degree of connectivity of each net, which is the maximum value that has the chance of achieving the best-case maximum degree of connectivity of that net. This estimated value is considered to be the strictly optimal degree of connectivity of the corresponding net for the bipartitioning step and it is denoted by δ_j. The case of $\kappa = 1$ is the case that the resulting parts will be the final parts of the partitioning, so each increment in the maximum degree of connectivity of a net causes an increment in the actual K-way cost of that net. However, in the case $\kappa > 1$, the resulting hypergraphs will continue being

```
procedure InitGains($H$: hypergraph, $\kappa$: number of parts for further partitioning )

begin
    for each $v_i \in V(H)$ do
        $gain[v_i] \leftarrow 0$
    for each $n_j \in N(H)$ do
        $\delta_j \leftarrow bestof[n_j] * \kappa$
        $partgain[l] \leftarrow 0$
        $partgain[r] \leftarrow 0$
        if $\kappa = 1$ then
            if $|n_j(l)| = \delta_j$ then
                $partgain[r] \leftarrow partgain[r] - w_j$
            elseif $|n_j(l)| > \delta_j$
                $partgain[r] \leftarrow partgain[r] - w_j$
                $partgain[l] \leftarrow partgain[l] + w_j$
            if $|n_j(r)| = \delta_j$ then
                $partgain[l] \leftarrow partgain[l] - w_j$
            elseif $|n_j(r)| > \delta_j$
                $partgain[l] \leftarrow partgain[l] - w_j$
                $partgain[r] \leftarrow partgain[r] + w_j$
        elseif $\kappa > 1$
            if $|n_j(l)| \geq \delta_j$ then
                if $|n_j(l)| - \delta_j)\ mod\ \kappa = 0$ then
                    $partgain[r] \leftarrow partgain[r] - w_j$
                elseif $|n_j(l)| - \delta_j)\ mod\ \kappa = 1$
                    $partgain[l] \leftarrow partgain[l] + w_j$
            if $(|n_j(r)| \geq \delta_j$ then
                if $|n_j(r)| - \delta_j)\ mod\ \kappa = 0$ then
                    $partgain[l] \leftarrow partgain[l] - w_j$
                elseif $(|n_j(r)| - \delta_j)\ mod\ \kappa = 1$
                    $partgain[r] \leftarrow partgain[r] + w_j$
        for each $v_i \in n_j$ do
            $gain[v_i] \leftarrow gain[v_i] + partgain[partof[v_i]]$
end
```

Figure 3.5: Gain initialization algorithm for recursive bipartitioning

```
**procedure** UpdateGains($H$:hypergraph, $\kappa$:Number of parts for further partitioning, $u$: Vertex being moved)

begin
        $s \leftarrow partof[u]$
        $t \leftarrow 1 - s$
        for each $n_j \in nets[u]$ do
            $\delta_j \leftarrow bestof[n_j] * \kappa$
            $\Delta_{gain}[s] \leftarrow 0$
            $\Delta_{gain}[t] \leftarrow 0$
            if $\kappa = 1$ then
                call EstimateDeltaGains1($H$, $n_j$, $s$, $t$)
            elseif $\kappa = 2$
                call EstimateDeltaGains2($H$, $n_j$, $s$, $t$)
            elseif $\kappa > 2$
                call EstimateDeltaGains$\kappa$($H$, $\kappa$,$n_j$, $s$, $t$)
            for each $v_i \in n_j$ do
                $gain[v_i] \leftarrow gain[v_i] + \Delta_{gain}[partof[v_i]]$
end
```

Figure 3.6: Gain update algorithm for recursive bipartitioning

partitioned, so that each κ increase in the maximum degree of connectivity of that net due to the current bipartitioning will cause at least an increase of 1 in the actual K-way cost. Therefore we divide the maximum degree of connectivity to levels of κ, that is each κ decrease in the maximum degree of connectivity of a net is respected to be a single gain. We initialize and update gains of vertices due to these levels by simple estimations using *mod* function. A net is considered to be critical if the number of pins of the net in a part is a multiple of κ larger than the best-case maximum degree of connectivity of that net. The vertices increasing the number of pins of a critical net on the part that the net is maximally connected to are assigned with a negative gain due to this net. A vertex making a net critical by its move is associated with a positive gain due to that net. So we check for the nets who are critical or one pin away from being critical to estimate the gains of the vertices in the hypergraph.

Figure 3.6 shows the algorithm for updating gains. The procedure *Update-Gains* visits all nets the moved vertex is connected to, estimate the change in

```
**procedure** EstimateDeltaGains1($H$: hypergraph, $n_j$: net, $s$: the part vertex
is moved from, $t$: the part vertex is moved to )

**begin**
        **if** $|n_j(t)| = \delta_j - 1$ **then**
            $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        **elseif** $|n_j(t)| = \delta_j$
            $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
        **if** $|n_j(s)| = \delta_j + w_j$ **then**
            $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        **elseif** $|n_j(s)| = \delta_j$
            $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
**end**
```

Figure 3.7: Estimation of gain changes for $\kappa = 1$

```
**procedure** EstimateDeltaGains2($H$: hypergraph, $n_j$: net, $s$: the part vertex
is moved from, $t$: the part vertex is moved to )

**begin**
        **if** $|n_j(t)| \geq \delta_j$ **then**
            **if** $(|n_j(t)| - \delta_j)$ $mod$ $2 = 0$ **then**
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] + w_j$
            **elseif** $(|n_j(t)| - \delta_j)$ $mod$ $2 = 1$
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] - w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        **elseif** $|n_j(t)| = \delta_j - 1$
            $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        **if** $|n_j(s)| > \delta_j$ **then**
            **if** $(|n_j(s)| - \delta_j)$ $mod$ $2 = 0$ **then**
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] + w_j$
            **elseif** $(|n_j(s)| - \delta_j)$ $mod$ $2 = 1$
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] - w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        **elseif** $|n_j(s)| = \delta_j$
            $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
**end**
```

Figure 3.8: Estimation of gain changes for $\kappa = 2$

```
procedure EstimateDeltaGains$\kappa$ ($H$: hypergraph, $\kappa$: number of parts for further partitioning, $n_j$: net, $s$: the part vertex is moved from, $t$: the part vertex is moved to )

begin
    if $|n_j(t)| \geq \delta_j - 1$ then
        if $(|n_j(t)| - \delta_j + 1)$ $mod$ $\kappa$=0 then
            $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        elseif $(|n_j(t)| - \delta_j + 1)$ $mod$ $\kappa$=1
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] + w_j$
        elseif $(|n_j(t)| - \delta_j + 1)$ $mod$ $\kappa$=2
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] - w_j$
    if $|n_j(s)| \geq \delta_j$ then
        if $(|n_j(s)| - \delta_j)$ $mod$ $\kappa$=0 then
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] + w_j$
        elseif $(|n_j(s)| - \delta_j)$ $mod$ $\kappa$=1
                $\Delta_{gain}[t] \leftarrow \Delta_{gain}[t] - w_j$
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] - w_j$
        elseif $(|n_j(s)| - \delta_j)$ $mod$ $\kappa$=2
                $\Delta_{gain}[s] \leftarrow \Delta_{gain}[s] + w_j$
end
```

Figure 3.9: Estimation of gain changes for $\kappa > 2$

the gains of the vertices in each part connected to the visited net and then updates the gains of all these vertices. The estimation of the change in the gains of the vertices in each part is performed by checking if the decrease or increase in the maximum degree of connectivity of the net causes the net to become critical or a candidate for being critical if the net is not critical and become non-critical if the net is critical. A non-critical net becomes critical if the difference between the maximum and ideal degrees of connectivity of the net becomes an integer multiple of κ, becomes a candidate for being critical if this difference becomes one less or more than an integer multiple of κ. A critical net becomes non-critical but a candidate for being critical if this difference is an integer multiple of κ before the move. Thus a net causes a change in the gains of its pins if the mod of the difference between its maximum and ideal degrees of connectivity due to κ is equal to 0, 1, 2, $\kappa - 1$ or $\kappa - 2$. As these values can intersect for the cases $\kappa = 1$ and $\kappa = 2$, different gain change estimation schemes are needed for the cases $\kappa = 1$, $\kappa = 2$ and $\kappa > 2$. The algorithms for estimating the changes in gains for these cases are shown in Figures 3.7, 3.8 and 3.9 respectively.

In addition to the gain functions defined in this section, a second level gain function can be associated with the vertices of the hypergraph relating to a second level cost function defined for a bipartitioning step in the recursion tree. This cost function may be defined to be the cost function of 2-way partitioning of the hypergraph in the recursion tree, independent of the number of parts the resulting parts will be partitioned. Such a second-level gain will provide the algorithm to select the vertices which tend to provide balance on the degree of connectivity of some nets, making sure that the move of such a vertex makes the maximum degree of connectivity of these nets closer to their strictly optimal degree of connectivity.

The algorithms provided here assume that K is a power of two, i.e. the parts resulting from a bipartitioning step will further be partitioned into same number of parts. However, these algorithms will also apply to the case that K is not a power of two with slight modification. In that case, there will be two parameters κ_l and κ_r denoting the number of parts that will be generated from left and right parts respectively. Then, there will be two strictly optimal degrees of connectivity of a net, δ_{jl} and δ_{jr} and the algorithm will take into account these differing values of κ and δ_j. Therefore, the algorithms and definitions provided in this subsection are for the special case of $\kappa_l = \kappa_r$, and they can be extended to the general case easily.

3.3.3 K-way Refinement of Relational Hypergraph Partitioning

As the cost function of the recursive bipartitioning does not exactly represent the actual cost function of K-way partitioning and uses an optimistic cost function which assumes the best-case performance of the recursive bipartitioning, a direct K-way refinement of the partitioning obtained by the recursive bipartitioning is necessary. However, as the amount of time and memory required for the direct K-way implementation of the iterative improvement based heuristics is huge and the instances in the very large databases domain create hypergraphs with very large number of pins, we propose a K-way refinement heuristic which visits each vertex in the hypergraph in a specified order and decides whether to move the vertex to a part instead of considering all move gains of vertices for each $K-1$ parts as a selection criterion. We introduce the concept of *virtual gain* and propose two virtual gain functions that can be used to determine the order of the selection of the candidate vertices for being moved. Instead of storing and maintaining $K-1$ gain values for each vertex in the hypergraph, we propose a virtual gain that is an approximation to the sum of all these $K-1$ gain values, and compute the actual $K-1$ gains after selecting a vertex with maximum virtual gain.

The K-way refinement consists of several passes on the K-way partitioning. Each vertex is considered once in a pass and moved if it has a positive move gain for one part, moved to improve the storage balance if it has no positive gain but has at least one zero gain, and not moved if it has no non-negative gain. We propose three selection criteria for the order of selection of vertices in each pass. One of these criteria is random selection of vertices which provides faster selection, the other two criteria are based on virtual gains which approximate the sum of the actual move gains of the vertices to each part. Figure 3.10

```
procedure KWayRefine($H$: hypergraph, $K$: number of parts)
begin
    repeat
        $totgain \leftarrow 0$
        $M \leftarrow V(H)$
        while $M \neq \emptyset$ do
            select $v_i \in M$ with maximum virtual gain
            for $k \leftarrow 1$ to $K$, $k \neq partof[v_i]$ do
                call EstimateMoveGain($H$, $K$, $v_i$, $partof[v_i]$, $k$) to obtain
                $gain[v_i, k]$
            repeat
                select $t$ s.t. $gain[v_i, t] = \max_{1 \leq k \leq K,\ k \neq s} gain[v_i, k]$
                if $gain[v_i, t] > 0$ and $L_t[G_t \cup v_i] = True$ then
                    move $v_i$ to part $t$
                    $totgain \leftarrow totgain + gain[v_i, t]$
                    update virtual gains
                else if $gain[v_i, t] > 0$ and$|G_s| > |G_t|$ then
                    move $v_i$ to part $t$
                    update virtual gains
                else
                    $gain[v_i, t] \leftarrow -1$
            until vertex moved or all target candidates explored
            $M \leftarrow M \setminus v_i$
    until $totgain = 0$
end
```

Figure 3.10: K-way refinement phase

shows the K-way refinement algorithm. The procedure *EstimateMoveGain* estimates the gain of moving vertex v_i to part V_t and shown in Figure 3.11.

The actual gain of moving a vertex v_i to a part V_t is equal to the difference between the total weight of the nets of v_i that will have a lower maximum degree of connectivity by the move of v_i and the total weight of the nets that will have higher. A net n_j's maximum degree of connectivity will become smaller by the move of v_i if the number of pins of n_j in the part of v_i is the only part having number of pins of n_j equal to the maximum degree of connectivity of n_j and this number is greater than the strictly optimal degree of connectivity of n_j, i.e. can be reduced further. On the other hand, the maximum degree of connectivity of a net n_j will be increased by the move of v_i to part V_t if n_j has the number of pins in V_t equal to the maximum connectivity of n_j. The gain of moving vertex v_i to the target part V_t can be formulated as

$$gain(v_i, t) = \sum_{n_j \in N_P} w_j - \sum_{n_j \in N_N} w_j$$

where $N_P = \{n_j \in nets(v_i) : |n_j(partof(v_i))| = \delta_j > \delta_j^{opt}$ *and* $\exists$ *no* $k \neq partof(v_i)$ *s.t.* $|n_j(k)| = \delta_j\}$ and $N_N = \{n_j \in nets(v_i) : |n_j(t)| = \delta_j\}$.

The first virtual gain function we propose is called the part non-optimal virtual gain. This virtual gain function approximates the sum of the actual move gains of each vertex to all parts by summing the weights of the nets connected to that vertex which have number of pins in that vertex's part larger than strictly optimal degree of connectivity.

Definition 3.7 *The part non-optimal virtual gain of a vertex v_i is defined as the sum of the weights of the nets of v_i with larger number of pins in v_i's part than strictly optimal degree of connectivity, i.e.* $VG_{PN}(v_i) = \sum_{n_j \in N_{PN}} w_j$ *where* $N_{PN} = \{n_j \in nets(v_i) : |n_j(partof(v_i))| > \delta_j^{opt}\}$.

This virtual gain function is based on the idea that a net with number of pins in a part larger than its ideal degree of connectivity has cost function larger than zero, so the partitioning can be improved by moving one vertex in that part to another part. Clearly this metric overestimates the gain of a vertex, however it is a promising criteria as it guarantees that the vertex being moved will make the maximum number of nets approach to their strictly optimal degrees of connectivity, at least in the vertex's part, if moved.

We propose a second virtual gain function named the part-maximum virtual gain, taking into account only the nets that have the maximum number of pins in the vertex's part as the only part with cardinality equal to the net's maximum degree of connectivity. In other words, the existence of a net connected to a vertex is considered to be a potential gain for that vertex if the cost associated with that net can be reduced by moving that vertex. We define the part-maximum virtual gain as follows.

```
procedure EstimateMoveGain($H$: hypergraph, $K$: number of parts, $v_i$: vertex, $s$: the current part vertex is moved from, $t$: the candidate part vertex is moved to )

begin
        $gain[v_i, t] \leftarrow 0$
        for each $n_j \in nets[v_i]$ do
            if $|n_j(s)| = \delta_j$ and $\delta_j > \delta_j^{opt}$ and $\eta_j = 1$ then
                $gain[v_i, t] \leftarrow gain[v_i, t] + w_j$
            if $|n_j(t)| = \delta_j$ then
                $gain[v_i, t] \leftarrow gain[v_i, t] - w_j$
end
```

Figure 3.11: Estimation of the gain of moving vertex v_i to part t

Definition 3.8 *The part-maximum virtual gain of vertex v_i is defined as the sum of weights of the nets that contain v_i and have the v_i's part as the only part having number of pins of that net equal to the net's maximum degree of connectivity, i.e. $VG_{PM}(v_i) = \sum_{n_j \in N_{PM}} w_j$ where $N_{PM} = \{n_j \in nets(v_i) : |n_j(partof(v_i))| = \delta_j > \delta_j^{opt}$ and $\exists$ no $k \neq partof(v_i)$ s.t. $|n_j(k)| = \delta_j\}$.*

This is a more accurate approximation than the part non-optimal virtual gain function but it still overestimates the actual gain of a vertex. In fact, these two functions provide upper-bounds for the sum of the gains of a vertex, while the part-maximum virtual gain provides a tighter upper-bound. But surprisingly our experimental results show that the part non-optimal virtual gain model provides a significantly better performance than the part-maximum virtual gain model.

For the estimation of actual and virtual gains during the K-way refinement of the partitioning, our algorithm constructs a load table containing the number of pins of each net in each part, i.e. $|n_j(k)|$ $1 \leq k \leq K$, the maximum degree of connectivity of each net, i.e. δ_j and the number of parts with cardinalities equal to the maximum degree of connectivity of each net, i.e. $\eta_j = |\{k : |n_j(k)| = \delta_j)\}|$, $\forall\ n_j \in N(H)$. The table consists of $|N(H)|$ rows corresponding to each net and $K + 2$ columns for the K degrees of connectivity, maximum of them and the number of parts having maximum of them.

```
procedure InitPartNonOptimalVirtualGains($H$: hypergraph, $K$: number of parts)

begin
    for each $v_i \in V(H)$ do
        $vgain[v_i] \leftarrow 0$
    for each $n_j \in N(H)$ do
        for $k \leftarrow 1$ to $K$ do
            if $|n_j(k)| > \delta_j^{opt}$ then
                $partvgain[k] \leftarrow w_j$
            else
                $partvgain[k] \leftarrow 0$
            for each $v_i \in n_j$ do
                $vgain[v_i] \leftarrow vgain[v_i] + partvgain[partof[v_i]]$
end
```

Figure 3.12: Part non-optimal virtual gains initialization algorithm

This table is constructed after the initial recursive bipartitioning phase and maintained during the K-way refinement phase. After a vertex is moved from source part s to target part t, the degrees of connectivity of that vertex's all nets connected to source are decremented and those to target are incremented. The maximum degree of connectivity of each net and the net's degree of connectivity to source and target are checked and the other two entries of the tables are updated if necessary. We initialize and update the virtual gains and estimate the move gains of a selected vertex to each part using this load table.

The procedure for estimating the gain of moving a vertex v_i to part t is shown in Figure 3.11. Figures 3.12 and 3.13 show the algorithms for initializing and updating part non-optimal virtual gains of vertices respectively. The algorithms for initializing and updating the part-maximum virtual gains are shown respectively in Figures 3.14 and 3.15. The algorithm for updating part non-optimal virtual gains is simpler and requires less time than that of part-maximum virtual gain. Thus it seems that the part non-optimal virtual gain model can be preferred to part-maximum virtual gain model in terms of runtime. However, a property of the part-maximum virtual gain model is that the vertex moves are realized at the very beginning of a pass and no further move is performed near the end of the pass as will be discussed in Chapter 4.

```
procedure UpdatePartNonOptimalVirtualGains($H$: hypergraph, $K$: number
of parts, $v_m$:vertex being moved, $t$: target part)
begin
        $s \leftarrow partof[v_m]$
        for each $n_j \in nets[v_m]$ do
            for $k \leftarrow 1$ to $K$ do
                $partvgain[k] \leftarrow 0$
            if $|n_j(s)| = \delta_j^{opt} + 1$ then
               $partvgain[s] \leftarrow -w_j$
            if $|n_j(t)| = \delta_j^{opt}$ then
               $partvgain[t] \leftarrow w_j$
            for each $v_i \in n_j$ do
               $vgain[v_i] \leftarrow vgain[v_i] + partvgain[partof[v_i]]$
end
```

Figure 3.13: Part non-optimal virtual gains update algorithm

This is because the part-maximum virtual gain is a good approximation to the actual gain of a vertex especially for higher values of K. This property may be used in such a way that no vertex is checked after some number of *don't move* decisions and this will bring a significant improvement on the time complexity of the algorithm.

```
**procedure** InitPartMaximumVirtualGains($H$: hypergraph, $K$: number of parts)

**begin**
    **for** each $v_i \in V(H)$ **do**
        $vgain[v_i] \leftarrow 0$
    **for** each $n_j \in N(H)$ **do**
        **for** $k \leftarrow 1$ **to** $K$ **do**
            **if** $|n_j(k)| = \delta_j$ **and** $\delta_j > \delta_j^{opt}$ **and** $\eta_j = 1$ **then**
                $partvgain[k] \leftarrow w_j$
            **else**
                $partvgain[k] \leftarrow 0$
            **for** each $v_i \in n_j$ **do**
                $vgain[v_i] \leftarrow vgain[v_i] + partvgain[partof[v_i]]$
**end**
```

Figure 3.14: Part-maximum virtual gains initialization algorithm

```
**procedure** UpdatePartMaximumVirtualGains($H$: hypergraph, $K$: number of parts, $v_m$:vertex being moved, $t$: target part)

**begin**
    $s \leftarrow partof[v_m]$
    **for** each $n_j \in nets[v_m]$ **do**
        **for** $k \leftarrow 1$ **to** $K$ **do**
            $parvgain[k] \leftarrow (0)$
        **if** $|n_j(s)| = \delta_j$ **and** $\delta_j > \delta_j^{opt}$ **and** $\eta_j = 1$ **then**
            **if** $\exists\ k\ s.t.\ |n_j(k)| = \delta_j - 1$ **then**
                $partvgain[s] \leftarrow -w_j$
            **else**
                $partvgain[s] \leftarrow 0$
        **if** $|n_j(t)| = \delta[n_j]$ **then**
            $partvgain[t] \leftarrow w_j$
        **for** each $v_i \in n_j$ **do**
            $vgain[v_i] \leftarrow vgain[v_i] + partvgain[partof[v_i]]$
**end**
```

Figure 3.15: Part-maximum virtual gains update algorithm

Chapter 4

Experiments and Results

In order to verify the effectiveness of the proposed algorithms, we have implemented these algorithms and tested on a several number of real and synthetic instances with several number of disks. We have chosen the WSG model proposed by Shekhar and Liu [34] to compare the performance of our model as this model can be applied to all kinds of data and it outperforms the previous methods. We use two real and three synthetic instances for performance tests which have been selected to cover a great range of possible data sets. Table 4.1 shows the number of data items, number of relations, total number of pins in the corresponding relational hypergraph, number of edges in the corresponding WSG and the average relation size of each data set.

Data set	Number of Data Items	Number of Relations	Total Relation Size	Average Relation Size	Number of Edges in WSG
Face	844	1024	23632	23.1	254664
House 16-H	1638	1000	43309	43.3	601460
8D HS	3200	6000	147554	24.6	897150
2D Regular HS	4426	2500	67791	27.1	180074
2D Random HS	4426	5000	91626	18.3	410057

Table 4.1: Description of test data

The first of the real data sets we use is a face image data set. We have constructed the face data set by combining three face image databases which have been used for face recognition and facial expression studies in the literature. The first image set is collected from the MIT image database and contains a total of 144 gray scale images of size 120X128 belonging to 16 people [38]. The second image set is obtained from the Pilot European Image Processing Archive and contains a total of 300 gray scale images of size 512X512 belonging to 30 people [9]. The last image set is obtained from the ORL database provided by Cambridge AT&T Laboratories and contains a total of 400 gray scale images of size 92X112 belonging to 40 people [39]. The 844 images are down-sampled to get images of the same size. These 844 images of size 32X32 are used to construct an image database for image retrieval purpose and the image retrieval algorithm described in [18] is applied to this database. A 1024-bit signature file is constructed for each image in the database with 28 1's in a signature corresponding to the 28 highest positive or negative wavelet coefficients of the image. Each bit in the signature files is assumed to be a relation as the set of images having high wavelet coefficients on a specific pixel are more likely to be accessed together. The face image database constitutes a database system with 1024 relations and 844 data items.

The second real data set used in the experiments is the House 16-H database provided by US Census Bureau and used for function approximation research [16]. The database contains 22784 instances of 17 continuous features each. The 22784 points in 17 dimensions are indexed into a grid file data structure [17]. Each page in the grid directory contains a maximum of 16 instances and the grid directory contains approximately 10 cells in each dimension. This results in a high degree of page sharing. The database system is constructed by randomly generating 1000 rectangular range queries where each query defines a rectangle in at most five dimensions. The House 16-H database contains 1000 relations(queries) and 1687 data items(pages).

Three synthetic data sets are constructed by randomly generating hot-spot data in eight and two dimensions. The 8-D HS data set is a 50000 point data in 8 dimensions. The hot-spot data is indexed into a grid directory of 3200 pages containing at most 16 points. 6000 rectangular range queries are generated resulting in a database system of 6000 relations and 3200 data items. The other two synthetic data sets share the same grid directory of 60000 points in two dimensions. The grid directory contains 480 cells in each dimension resulting in 4426 pages containing at most 16 points. The 2-D Regular HS data set is constructed by defining 2500 regular square range queries of size 40 in one dimension on this grid directory. 2-D random HS data set is constructed by generating 4000 random rectangular range and 1000 random diagonal of size at most 80 in one dimension resulting in a database system of 5000 relations and 4426 data items.

The max-cut graph partitioning algorithm is implemented by FM based recursive bipartitioning. The conventional multi-level partitioning tools are not used as the max-cut objective function conflicts by the clustering criteria of these multi-level tools by nature. The total weight on the cut-set of the partition is used as the objective function instead of the ratio cut function proposed by Shekhar and Liu [34] as the balance on the parts is enforced by the max-cut objective itself. After an initial partition is found by recursive bipartitioning, each pair of parts are refined by the same FM based bipartitioning procedure and then a refinement is applied on selected candidate pairs of parts until a pair to be refined cannot be found. This implementation of the max-cut graph partitioning algorithm is similar to the SM-GP-G [34] implementation of the method.

The relational hypergraph based declustering algorithm is implemented as described in Chapter 3. All three proposed K-way refinement algorithms are implemented and named as H-random(random visit order), H-nonopt (part non-optimal virtual gain) and H-max (part-maximum virtual gain). The max-cut graph partitioning and min-net-imbalance hypergraph partitioning algorithms are implemented using C programming language on Linux platform running on a PC with Pentium-III 500 MHz CPU and 512 MB 168-Pin 100 MHz SDRAM. All experiments are performed by running each algorithm 10 times and reporting the averages of the performance metrics on these 10 runs.

Table 4.2 shows the average retrieval times obtained by the algorithms on each of the five data sets for 8, 16, 32 and 64 disks. The average retrieval time is estimated by taking the average of the maximum number of data items

			Graph	Hypergraph		
Data set	K	Ideal	Max-cut	H-random	H-nonopt	H-max
Face	8	3.32	4.15	4.12	**4.11**	4.12
	16	1.92	2.77	2.65	**2.64**	2.65
	32	1.26	2.01	1.82	**1.80**	1.87
	64	1.03	1.57	1.36	**1.34**	1.51
House 16-H	8	5.84	6.82	6.81	**6.79**	6.81
	16	3.18	4.21	4.09	**4.06**	4.16
	32	1.86	2.83	2.65	**2.64**	2.76
	64	1.24	2.06	1.92	**1.90**	2.03
8D HS	8	3.52	4.50	4.44	**4.43**	4.44
	16	2.01	2.98	2.87	**2.86**	2.88
	32	1.31	2.13	1.99	**1.98**	2.02
	64	1.05	1.62	1.48	**1.47**	1.53
2D Regular HS	8	3.83	4.13	**4.09**	4.10	4.10
	16	2.03	2.50	**2.37**	2.39	2.45
	32	1.09	1.70	1.59	**1.58**	1.66
	64	1.00	**1.00**	1.01	1.01	1.04
2D Random HS	8	2.74	3.17	**3.10**	3.10	3.11
	16	1.65	2.07	**1.93**	1.93	1.98
	32	1.17	1.47	1.34	**1.33**	1.42
	64	1.03	1.14	1.11	**1.10**	1.17

Table 4.2: Averages of average retrieval time

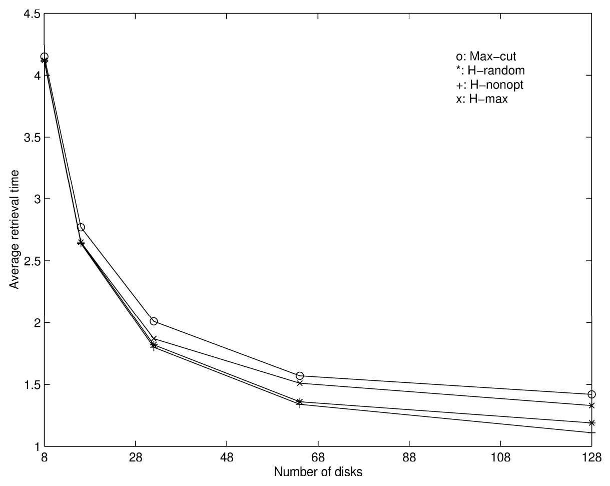

Figure 4.1: Performance of declustering algorithms with respect to number of disks

of a relation in a part over all relations assuming that the retrieval of a data item is performed in one time unit. The numbers displayed in the table are the averages of these average retrieval times over 10 runs with different initial random partitions. The bold numbers show the minimum of average retrieval time over the five algorithms. The table also shows the ideal retrieval time that is defined as the average of the strictly optimal retrieval times of all relations. As seen on the table H-nonopt and H-random show nearly the same performance but H-nonopt provides slightly lower average retrieval time for almost all of the data sets and number of disks. These two models provide significantly lower average retrieval times than the other two models for almost all cases. The performance of H-max is poor when compared to the other two hypergraph model, however it still outperforms the graph model. The poor performance of the part-maximum virtual gain model can be due to the lower number of moves done as this method gives priority to the vertices with highest gain. It is likely that the move of vertices with high gains blocks the gains of the remaining vertices resulting on lower number of moves and the algorithm cannot climb out local minima. However, the part-non-optimal virtual gain model orders the vertices in a more flexible manner and this property gives the opportunity of climbing out local minima for this method. Despite this elegant property of H-nonopt, visiting the vertices in random order can achieve very close average retrieval times to those of H-nonopt.

For all data sets, the performance of the WSG model is close to the performances of the three hypergraph based methods we propose for smaller number of disks. However, as the number of disks increases, performance gap between WSG model and hypergraph based models increases in favor of the hypergraph based models. For example, for the Face data set, the average retrieval time obtained by H-nonopt is only 1% better than the average retrieval time obtained by Max-cut for 8 disks, while it is 4.7%, 10.4% and 14.6% better for 16, 32 and 64 disks respectively. Figure 4.1 shows the performances of each method with the respect to the number of disks. As seen in the graph H-nonopt and H-random show a more scalable performance than Max-cut and H-max. H-max's performance gets poor with increasing number of disks as the performance of the algorithm depends more on the K-way refinement phase for large number of disks.

			Graph	Hypergraph		
Data set	Relations	K	Max-cut	H-random	H-nonopt	H-max
Face	1024	8	850	814	**812**	813
		16	874	748	**738**	746
		32	770	574	**556**	624
		64	553	338	**320**	498
House 16-H	1000	8	981	968	**956**	975
		16	1024	910	**883**	975
		32	977	796	**782**	907
		64	821	678	**657**	783
8-D HS	6000	8	5907	5513	**5454**	5531
		16	5806	5128	**5060**	5225
		32	4955	4104	**4025**	4307
		64	3401	2578	**2520**	2879
2-D Regular HS	2500	8	752	**646**	676	677
		16	1173	**861**	901	1038
		32	1532	1246	**1225**	1437
		64	**0**	22	21	93
2-D Random HS	5000	8	2139	**1793**	1802	1833
		16	2069	**1386**	1407	1625
		32	1501	820	**797**	1228
		64	565	443	**357**	699

Table 4.3: Averages of parallel retrieval overhead

Our experiments approve the theoretical result that the WSG model is able to find the strictly optimal allocation if exists, as in the case of partitioning the 2-D regular HS data set into 64 sets. For this case, WSG provides an average retrieval time of 1.00, which is the minimum achievable retrieval time for a relation. The 2-D regular HS data set has a very regular structure as the queries are distributed uniformly over the data space and thus a strictly optimal allocation scheme exists for enough number of disks. Although the hypergraph model exactly fits the declustering problem so it is likely to find the optimal allocation for all cases theorically, the three proposed hypergraph based methods were not able to find the optimal allocation for this case. This shows that the heuristics we propose and the tool we develop should be improved by further research to exploit the accurate representing ability of the model.

The parallel retrieval overhead, i.e. the total deviation of the retrieval time of all queries from ideal provides a more detailed analysis of the performance of the methods. Table 4.3 shows the cost of each algorithm for all test runs with this definition of parallel retrieval overhead. As a declustering method aims to find an allocation that will be as close as possible to the minimum achievable access time for each relation, this cost function is a proper metric to evaluate a declustering method. We include the number of relations of each data set in the table as the magnitude of parallel retrieval overhead depends on the number of relations in the database system. For the Face data set, H-nonopt provides an overhead that is 4.2%, 15.6%, 27.8% and 40.3% better than the overhead provided by Max-cut for 8, 16, 32 and 64 disks respectively. For 16 disks, the overhead of H-nonopt is 13.8% for House 16-H, 12.8% for 8-D HS, 23.2% for 2-D Regular HS and 32.0% for 2-D Random HS better than that of Max-cut. Note that the overhead due to a relation was almost always equal to 1 for all methods in our experiments. The table also shows that all of the hypergraph based methods obtain a parallel retrieval overhead smaller than the number of relations for all cases, concluding that a strictly optimal allocation is provided for some number of relations in the database system.

Table 4.4 shows the average run time of each algorithm for all test runs. We can observe from the table that the time complexity of Max-cut is stable and it does not grow significantly with increasing number of disks. This is because the graphs partitioned at the lower levels of the recursion tree are very sparse as the earlier bipartitioning steps provide that many of the edges in the original graph are cut by the partition. However, in some situations H-max spends less time than Max-cut although it provides slightly better results that Max-cut. These situations are the cases when the number of edges in the WSG is close to the number of pins in the relational hypergraph, i.e. the relations are different from each other. This is because H-max cannot make a significant improvement on the partition during the K-way refinement phase, so it requires a very small number of passes and moves in this phase. Therefore, H-max can only be preferred to the other methods to save time. The other two hypergraph based methods, H-random and H-nonopt have a significantly higher time complexity than the other two methods. The time spent by this algorithms grows significantly with increasing number of disks, as the time spent on the gain computations of K-way refinement phase depends on the

		Graph	Hypergraph		
Data set	K	Max-cut	H-random	H-nonopt	H-max
Face	8	1.03	0.87	0.96	**0.77**
	16	1.18	1.08	1.41	**0.82**
	32	1.33	1.80	2.37	**0.53**
	64	1.67	3.84	5.50	**0.29**
House 16-H	8	2.29	1.41	1.67	**1.04**
	16	2.56	2.49	3.07	**1.15**
	32	3.24	3.61	5.09	**0.86**
	64	4.32	5.62	7.64	**0.69**
8-D HS	8	**3.98**	5.85	7.08	5.29
	16	**4.52**	12.78	12.78	5.50
	32	**5.39**	26.29	30.10	6.28
	64	7.00	53.69	56.91	**6.92**
2-D Regular HS	8	**1.55**	3.69	3.25	3.13
	16	**1.66**	6.97	5.82	2.91
	32	**1.80**	12.33	11.47	2.31
	64	2.00	7.80	0.93	**0.57**
2-D Random HS	8	**2.38**	6.47	6.10	4.26
	16	**2.99**	13.94	13.98	3.90
	32	3.03	24.82	23.75	**2.20**
	64	3.53	31.69	21.24	**1.12**

Table 4.4: Averages of run time

		Retrieval Time		Cut on WSG	
Data set	K	Max-cut	H-nonopt	Max-cut	H-nonopt
Face	8	4.15	4.11	462574	461243
	16	2.77	2.64	494367	492787
	32	2.01	1.80	509522	508190
	64	1.57	1.34	516415	515478
House 16-H	8	6.82	6.79	1357391	1355250
	16	4.21	4.06	1452516	1449696
	32	2.83	2.64	1498689	1495494
	64	2.06	1.90	1520406	1516894

Table 4.5: Comparison of WSG cut values and retrieval times

number of disks.

Our experiments also approve the theoretical findings on the deficiencies of the WSG model discussed in Section 2.4. To illustrate the fact that a higher cut in the similarity graph does not necessarily provide a lower average retrieval time, we computed the cut on the corresponding similarity graph of the data sets for the partitions found by Max-cut and H-nonopt methods. Table 4.5 displays the average retrieval times and the total cut of the resulting partitions for the experiments on Face and House 16-H data sets. As seen on the table, Max-cut provides a partition with higher cut than that of H-nonopt for all cases, the average retrieval times provided by Max-cut are worser than that of H-nonopt. From this observation, we can conclude that the relational hypergraph based declustering model is a more accurate representation of the problem than the weighted similarity graph model.

The experimental results reported in this chapter show that the proposed model for declustering provides an accurate and reliable model of the problem of partitioning very large databases. In addition, the relational hypergraph based model provides a significant improvement on the performance of the weighted similarity model which does depend on the special properties of the application and is an elegant model that outperforms the various declustering strategies in the literature. Therefore, we can conclude that the proposed model is a promising declustering strategy that can be applied to any database system independent of the application type and indexing technique.

Chapter 5

Conclusion

In this thesis, we have demonstrated the flaws of the similarity graph based declustering model and proposed iterative improvement based algorithms to the problem that exactly models the declustering problem. Allocation of data into multiple number of disks will become more important by the growing amount of information shared in the world and the growing world wide communication. The problem considered in this work is a general problem that can be faced for any kind of database applications and the data may not be regular in many cases. The provided model and methods are designed to deal with all kinds of applications regardless of the irregularity contained in the data. It is also approved by the experimental results that the proposed methods provide a significantly higher declustering performance than the existing general method and this performance can be further improved by developing the methods proposed on the model.

The experimental results show that there is a significant performance difference between the graph and hypergraph models and the objective function of the graph model does not fit the actual cost function of the problem. The basic problem with the proposed model is the high time complexity for some instances on high number of disks. However, one of the proposed K-way refinement schemes requires less run time than the graph model although it outperforms the graph model on most of the instances. Therefore, this K-way refinement scheme can be preferred if the time for allocation process is limited.

However, if allocation is performed only once or very rarely for a database system, then the high-quality K-way refinement schemes can be preferred.

This study covers the case of database systems with relation set of non-identical relative frequencies and disks with non-identical storage capacities. There can be database systems with non-identical disk I/O capacities, and our model can be adapted to such case by slightly modifying the definition of the cost of a net due to a partition. The methods we propose should also be modified accordingly. Another case that can be faced is that the data items in the database can have non-identical access times as in the case of Geographic Information Systems. In such condition, the problem may be defined as a multi-set number partitioning problem. We should note that the algorithms we propose here cannot be easily adapted to this problem and further detailed research is necessary.

The scope of this study was static allocation of the data. The problem of updating the allocation when new data are added to the system, i.e. the dynamic allocation problem can also be solved by the similar model. This problem is also open to research in the future.

Bibliography

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *VLSI Journal*, 19(1-2):1–81, 1995.

[2] S. Berchtold, C. Böhm, B. Braunmüller, D. A. Keim, and H. P. Kriegel. Fast parallel similarity search in multimedia databases. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1997.

[3] Ü. V. Çatalyürek. *Hypergraph models for sparse matrix partitioning and reordering.* PhD thesis, Bilkent University Department of Computer Engineering, November 1999.

[4] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Computing*, 10(7):673–693, 1999.

[5] L. T. Chen and D. Rotem. Declustering objects for visualization. In *19th International Conference on Very Large Data Bases*, pages 85–96, 1993.

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

[7] P. Ciaccia. Parallel independent grid files based on dynamic declustering method using multiple error correcting codes. Technical report, University of Bologna Laboratory for Computer Science, November 1994.

[8] P. Ciaccia, P. Tiberio, and P. Zezula. Declustering of key-based partitioned signature files. *ACM Trans. on Database Systems*, 21(3):295–338, 1996.

[9] E. F. Clark. Pilot Europan Image Processing Archive. http://peipa.essex.ac.uk/ipa/pix/faces/manchester/train/.

[10] M. Coyle, S. Shekhar, and Y. Zhou. Evaluation of disk allocation methods for parallelizing spatial queries on grid files. *Journal of Computer and Software Engineering*, 1995.

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

[12] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18–25, 1993.

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

[14] M. T. Fang, R. C. T. Lee, and C. C. Chang. The idea of declustering and applications. In *Proc. of International Conference on Very Large Databases*, 1986.

[15] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[16] H. A. Güvenir and İ. Uysal. Bilkent University Function Approximation Repository. http://funapp.cs.bilkent.edu.tr/, 2000.

[17] S. Hanan. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[18] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *Computer Graphics Proc.*, pages 277–286, 1995.

[19] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *ACM SIGMOD Conf.*, pages 332–342, 1990.

[20] J. Jensch, R. Lüling, and N. Sensen. A data layout strategy for parallel web servers. In *4th International Euro-Par Conference*, pages 944–952, 1998.

[21] I. Kamel and C. Faloutsos. Parallel R-trees. In *Proceedings of the Int. Conf. on Management of Data*, pages 195–204. ACM SIGMOD, 1992.

[22] G. Karypis, V. Kumar, R. Aggrawal, and S. Shekhar. *hMeTis A Hyper-Graph Partitioning Package Version 1.0.1.* Uni. of Minnesota, Dept. of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[23] G. Karypis, V. Kumar, R. Aggrawal, and S. Shekhar. Hypergraph partitioning using multilevel approach: Applications in VLSI domain. *IEEE Trans. on VLSI Systems*, to appear.

[24] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[25] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Conf.*, pages 173–182, 1988.

[26] N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. In *EDBT Conference*, pages 592–614, 1996.

[27] T. G. Kwon and S. Lee. Load-balanced data placement for variable-rate continuous media retrieval. *Multimedia Database Systems*, pages 185–207, 1996.

[28] J. Li, J. Srivastava, and D. Rotem. Cmd: A multidimensional declustering method for parallel database systems. In *18th International Conference on Very Large Data Bases*, pages 3–14, 1992.

[29] M. Mehta and D. J. De Witt. Data placement in shared-nothing parallel database systems. *The VLDB Journal*, (6):53–72, 1997.

[30] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 434–440, 1996.

[31] H. Pang, B. Jose, and M. S. Krishnan. Resource scheduling in a high-performance multimedia server. *IEEE Transactions on Knowledge and Data Engineering*, 11(2):303–320, 1999.

[32] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Efficient retrieval of multidimensional datasets through parallel I/O. In *Proceedings of the 5th International Conference on High Performance Computing*, 1998.

[33] S. Prabhakar, D. Agrawal, A. El Abbadi, A. Singh, and T. Smith. Browsing and placement of multiresolution images on parallel disks. In *Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems*, pages 102–113, 1997.

[34] S. Shekhar and D. R. Liu. Partitioning similarity graphs: A framework for declustering problems. *Information Systems*, 21(6):475–496, 1996.

[35] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner. Declustering and load balancing methods for parallelizing geographical information systems. *IEEE Trans. on Knowledge and Data Engineering*, to appear.

[36] H. S. Wilf. *Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1996.

[37] S. Zhou and M. H. Williams. Data placement in parallel database systems. In M. Abdelguerfi and K.F. Wong, editors, *Parallel Database Techniques*, chapter 10, pages 203–219. IEEE CS Press, Los Amitos, CA, 1998.

[38] Massachussets Institute of Technology Image Database. ftp://whitechapel.media.mit.edu/pub/images/faceimages.zip.

[39] The ORL Database of Faces.
ftp://ftp.uk.research.att.com/pub/data/att_faces.tar.Z. AT&T Laboratories, Cambridge, 1999.