On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications

S. Muthukrishnan[1], Viswanath Poosala[1], and Torsten Suel[*2]

[1] Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ 07974.
{muthu,poosala}@research.bell-labs.com.
[2] Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201.
suel@photon.poly.edu.

Abstract. Partitioning a multi-dimensional data set into rectangular partitions subject to certain constraints is an important problem that arises in many database applications, including histogram-based selectivity estimation, load-balancing, and construction of index structures. While provably optimal and efficient algorithms exist for partitioning one-dimensional data, the multi-dimensional problem has received less attention, except for a few special cases. As a result, the heuristic partitioning techniques that are used in practice are not well understood, and come with no guarantees on the quality of the solution. In this paper, we present algorithmic and complexity-theoretic results for the fundamental problem of partitioning a two-dimensional array into rectangular tiles of arbitrary size in a way that minimizes the number of tiles required to satisfy a given constraint. Our main results are approximation algorithms for several partitioning problems that provably approximate the optimal solutions within small constant factors, and that run in linear or close to linear time. We also establish the NP-hardness of several partitioning problems, therefore it is unlikely that there are efficient, i.e., polynomial time, algorithms for solving these problems *exactly*.
We also discuss a few applications in which partitioning problems arise. One of the applications is the problem of constructing multi-dimensional histograms. Our results, for example, give an efficient algorithm to construct the *V-Optimal* histograms which are known to be the most accurate histograms in several selectivity estimation problems. Our algorithms are the first to provide guaranteed bounds on the quality of the solution.

1 Introduction

Many problems arising in databases and other areas require partitioning a multi-dimensional data set into rectangular partitions or tiles such that certain mathematical constraints are satisfied. Often these constraints take the form of minimizing (or maximizing) a metric using a fixed number of partitions or, conversely,

[*] This work was done while the author was at Bell Labs.

minimizing the number of partitions while not exceeding (or falling below) a given value of that metric.

These problems are quite challenging for most interesting metrics and hence one usually resorts to heuristic approaches. Unfortunately, many of these approaches do not provide any guarantees on the quality of the solution and may thus adversely affect the application. In this paper we present algorithmic and complexity-theoretic results on the fundamental problem of partitioning two-dimensional arrays into rectangular partitions tiles[1] of arbitrary sizes. We develop solutions that offer guarantees on the quality of their solutions *and* that run in small polynomial time (near-linear in most cases). We start out with a few examples.

Example 1. Consider the 4×4 array in Fig. $1(a)$. A partitioning with 5 tiles is shown in Figure $1(b)$ such that the maximum sum of the elements that fall within any one tile is at most 57. There are many different ways to tile the array with 5 tiles with different maximum sums. There are also alternative ways to evaluate the partitioning, other than by considering the maximum sum of the elements. For instance, for each tile, we could sum up the squares of the difference between each element in the tile and the average of all the elements in that tile; we could then total all the values thus obtained for the tiles. This value is 204.7 as shown in Fig. $1(b)$. Again different partitions induce different values. □

Example 2. Consider the 4×4 array in Fig. 1. A 3×3 tiling, namely one obtained by partitioning rows into 3 intervals and columns into 3 intervals, is presented in Fig. $1(c)$. For the partition presented there, the maximum sum of the elements that fall within any tile is at most 79. Again different partitions induce different values. □

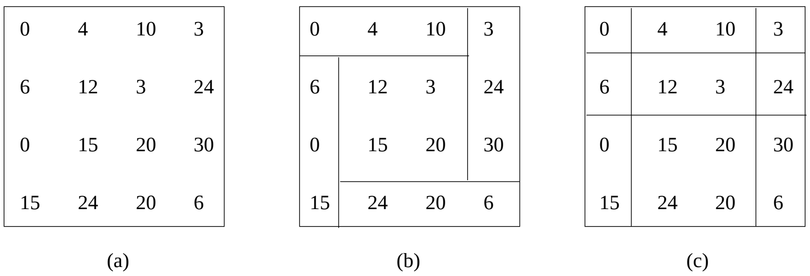

Fig. 1. Partitioning Examples

Example 1 arises in data partitioning for load balancing [23] and histogram-based selectivity estimation [31], while Example 2 arises in constructing grid-files which are well-known index structures [28]. We will list other application scenarios further below, but we briefly describe the histogram context here.

[1] We use the terms tile and partition interchangeably in the rest of the paper.

Example Application: *(Histograms)* Query optimizers require reasonably accurate estimates of query result sizes in order to estimate the costs of various execution plans. Most commercial database management systems use *histograms* to approximate the data in the database in order to perform these estimations. Histograms group attribute values into subsets (*buckets*) and approximate true attribute values and their frequencies based on summary statistics maintained in each bucket [20, 31]. Researchers have proposed the use of *multi-dimensional histograms* for approximating distributions with multiple attributes [27, 32]. This approach involves partitioning the multi-dimensional space of attribute values into rectangular buckets based on various partitioning constraints. This leads to partitioning problems of the sort we consider in this paper. In particular, the well-known V-optimal histogram in two dimensions, which has been shown to minimize selectivity estimation errors for estimating the result sizes of several classes of queries [14, 30], corresponds to the alternative metric, and one of the partitions in Example 1. □

There are many different types of partitions and many different metrics to evaluate the partitions. A generic optimization problem that arises in several application contexts is as follows.

The Partitioning Problem. We are given a two-dimensional array of elements, the type of partition sought, a metric to evaluate the partitions, and a bound δ. The problem is to produce a partitioning of the array of the type sought, with the minimal number of rectangular tiles such that the metric computed on the partition is at most δ. □

The partitioning problem has been extensively studied in many application scenarios in the Database and Algorithms Research communities, in various specialized forms. However, most known results are either heuristics without provable guarantees, or provably efficient algorithms for simple, specific metrics and tilings. For example, little is known about the complexity of the partitioning problem with the alternative metric in Example 1 for different types of partitions, but this problem is fundamental in histogram construction.

Motivated by this state-of-the-art, we formulate the partitioning problem in its generality and study its difficulty for various types of partitions and metrics of relevance in application scenarios. Our contributions are two-fold.

1. *We show that the partitioning problem is NP-hard for many natural metrics and partitionings arising in database applications. Thus, efficient, i.e. polynomial time, algorithms exist if and only if $NP = P$, a central complexity question that remains unresolved.*
   The partitioning problem can be naturally defined on one-dimensional arrays too. These problems can be solved efficiently in small polynomial time [15], but our claim above implies that their natural two-dimensional variants are NP-Hard. Thus, the partitioning problem becomes fundamentally different as we go from one-dimensional to multidimensional arrays.
2. Our main technical results are a number of algorithms that complement the negative results above. *We present very efficient (near-linear time) algorithms for* approximately *solving the partitioning problem in two dimensions*

for fairly general metrics and different partitionings (including most that arise in our database applications); the approximation bounds are guaranteed, and they are small constants.

We define the partitioning problem and state our results more formally in Section 2. Here are a few high-level remarks on our results.

Remark 1. All of our algorithms extend to multiple dimensions with similar (i.e., near-linear) performance bounds. However, this may sound better than it actually is, because they will take time that is linear in the size of the underlying d-dimensional array. However, in many applications, this array may be very sparse, and an algorithm that works in time linear in the number of non-zero entries would be much preferable, while one that works in time linear in the size of the array can be prohibitively expensive. Some of our algorithms can likely be modified to exploit sparseness of the input domain, while for others this is more difficult.

Remark 2. There are other limitations to our framework of partitioning problems. For example, in some application scenarios, tiles may be allowed to overlap, i.e., the problem may be one of *covering* the array rather than partitioning it. This arises, for example, when building certain spatial indices such as R-trees. Our results do not directly apply to such covering problems. Also, there are application scenarios where one may be allowed to permute the rows and columns of the input array. All our results assume that some canonical ordering (such as that given by the natural order of numeric attributes in database applications) is fixed.

Remark 3. Despite the limitations above, the partitioning problems we study are very general, and they arise in a number of important application scenarios within databases. Besides the problems of histogram-based selectivity estimation, grid file construction, and load balancing mentioned earlier, there are applications in database compression, bulk-loading of hierarchical index structures, and partitioning of spatial data, as well as to problems outside databases such as domain partitioning in scientific computation, support of data partitioning in data-parallel languages, and image and video processing. In this paper, we clarify applications to histogram-based selectivity estimation only; discussions about other applications can be found in the full version of this paper.

□

Map. The rest of the paper is organized as follows. We formalize the partitioning problem and relate it to various application contexts in Section 2; we also state our results formally there. In Sections 4, 5, and 6, we present hardness and algorithmic results for different types of partitions with different metrics. In Section 7, we discuss the implications of our results for one applied area in databases, namely, histogram-based selectivity estimation. Section 8 has concluding remarks.

2 Problem Formulation and Overview of Results

2.1 The Partitioning Problem Definition

We are given an $n \times n$ array A containing $N = n^2$ real numbers. A *tile* is any rectangular subarray $A[i \dots j, k \dots l]$. A *partitioning* of array A is a partitioning of A into tiles; by definition of a partition, each element $A[i, j]$ lies within some tile and no two tiles overlap. As mentioned in the previous section, partitioning schemes can be classified based on the *type of partition* and the *metric functions* used within and outside the tiles to evaluate the partition. We define these criteria below and present interesting instantiations for each of them.

Type of Partitioning: There are many possible types of partitionings of a two-dimensional array. The ones we consider here are the common ones that arise in database applications:

1. *Arbitrary*: No restrictions on the arrangement of tiles within A (Figure 2.a).
2. *Hierarchical*: A hierarchical partition is one in which there exists a vertical or horizontal separation of array into two disjoint parts in each of which the partitioning is again hierarchical (Figure 2.b). A hierarchical partitioning is naturally represented by a *hierarchy tree* which is a binary tree in which a node represents a subarray of A and each of its children represent a partition of that subarray of A into two disjoint parts; the root represents A.
3. $p \times p$: Here, the rows and columns of A are partitioned into p disjoint intervals each; the induced p^2 tiles form the $p \times p$ partitioning. This can be thought of as a special case of hierarchical partitioning where the tiling in the subarrays of two sibling nodes of the hierarchy tree are the same along one dimension (Figure 2.c).

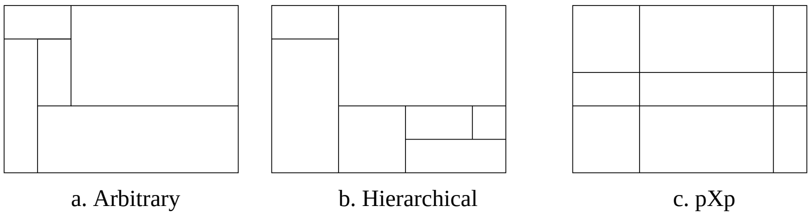

Fig. 2. Tilings

Quality Metrics: Metrics are defined on the tiling using the following three functions.

1. *Elementary Function:* An elementary function h maps the array elements of any tile to real numbers. Common functions include (i) ID, i.e., $h(A[i]) = A[i]$, (ii) AVG_DIFF, i.e., $h(A[i]) = |A[i] - \overline{A}|$ where $\overline{A}$ is the average of the elements in that tile, (iii) GEO_DIFF, i.e., $h(A[i]) = |A[i] - A'|$ where A' is

the geometric median of the elements in the tile, and (iv) SQR_DIFF which is the square of AVG_DIFF.

2. *Heft Function.* A heft function g is defined for a tile r and an elementary function h of the elements in that tile. The common heft functions are (i) SUM, i.e., $\sum_{A[i,j]\in r} h(A[i,j])$, (ii) MAX, i.e., $\max_{A[i,j]\in r} h(A[i,j])$, (iii) RATIO, i.e., $\frac{\max_{A[i,j]\in r} h(A[i,j])}{\min_{A[i,j]\in r} h(A[i,j])}$, etc. SUM and MAX are the common heft functions in our application scenarios. Notice that the heft is an "intra-tile" function. The value evaluated by the heft function in a tile is called its *heft*.
3. *Cumulative Function.* A cumulative function f is defined for the entire set of tiles, for combining heft function g. The common cumulative functions are SUM and MAX, representing the total and maximum heft value of tiles in the given tiling, respectively. Note that the cumulative function is an "inter-tile" function.

Any partition thus has a *metric* that is a combination of these three functions $f-g-h$. For example, the SUM-MAX-ID metric of a partition is the sum over the tiles of the maximum element in each tile. In Section 1, Example 1 has the MAX-SUM-ID metric with the alternative being the SUM-SUM-SQR_DIFF metric, and Example 2 has the MAX-SUM-ID metric. Some combination of the functions are trivial – for example, all partitions are identical under the MAX-MAX-ID and SUM-SUM-ID metrics – but most of the metrics are nontrivial. The value the metric evaluates to on a given array and a partition, is called its *metric value*, or where there is no ambiguity, the *heft* of the partition.

The Optimization Problem: The optimization problem we consider is as follows: *For the given type of partition, metric, and bound δ, determine the partitioning of that type that has the minimum number of tiles with the metric value of the partition being at most δ.* A related problem is one in which we are given a bound p on the number of tiles and the goal is to determine a tiling using at most p tiles with minimum metric value. As far as the NP-Hardness is concerned, these two optimization problems are identical, that is, if one is NP-hard, so is the other. However, this similarity does not extend to efficient approximability of these problems. In this paper, we only consider the first version.

Several partitioning problems arising in real applications mentioned in the Introduction fit naturally in our framework. We present one such application (histograms) in full detail in Section 7 and illustrate how our problem definition and results solve an important problem arising in that application.

2.2 Some Preliminaries

We state some properties of the metrics and hefts that will be used latter. We say that a heft function g is *monotonic* if $g(r) \leq g(R)$ for two tiles r and R, $r \in R$. For example, SUM-ID is monotonic provided the array elements are non-negative. Most of the heft functions that arise in our applications are monotonic. We say that a metric $f-g$ is *superadditive* if the following holds. $f(g(r \bigcup R)) \geq$

$f(g(r), g(R))$, where r and R are any two disjoint tiles. For example, SUM-SUM-SQR_DIFF is superadditive (this requires a nontrivial proof). Some of our results hold only for superadditive metrics.

Our algorithms often identify tiles and are required to compute their hefts. Say t_Q represents the time to determine the heft of any tile. The straightforward way would be to consider each element of the tile and this takes $t_Q = O(N)$ time in the worst case. However, there is a more efficient method in some cases. Consider the MAX heft function: after $O(N)$ time preprocessing, we can compute the heft of any tile in $t_Q = O(1)$ time. We omit the details of this procedure here. For SUM heft functions, the same holds for some elementary functions such as the SQR_DIFF and ID, and not for others, such as the AVG_DIFF, GEO_DIFF etc. There are still other examples of heft functions for which t_Q is $O(\log N)$. In what follows, we state all our bounds in terms of t_Q. For applications arising in databases, t_Q is often $O(1)$ since SUM-SQR_DIFF is the most common heft function (such as in V-optimal histograms).

3 Related Work

Partitioning problems have been studied extensively in various application areas including databases, parallel computing (e.g., load balancing), computational geometry (e.g., clustering), video compression (e.g., block matching) etc. Some related papers from a variety of application areas include [17, 23, 25, 24, 1, 3, 10]. Here we review a selection of related work most relevant to us.

Hardness Results. Hardness results exist only for a simple metric function, namely, MAX-SUM-ID – [18] proved it to be NP-hard for arbitrary partitions, and [11, 6] proved it to be NP-hard for $p \times p$ partition. Our NP-hardness results are inspired by the abovementioned results. However, the basic gadgets in our reductions are different from the ones in [18, 6] and we derive different non-approximability bounds for various metrics.

Algorithmic Results. Dynamic programming has been used to find optimal results for hierarchical partitions in several contexts [2, 26, 18]. However, our sparse hierarchy approach with provable guarantee appears to be new. For $p \times p$ partition MAX-SUM-ID, the best known approximation is by $O(\log n)$ factor [19]. We derived substantially improved bounds for this problem. Many heuristic algorithms have been proposed for this problem [23].

Applications: We focus on prior work in one database application, namely, histograms. Histograms have been studied quite extensively in the literature in the context of approximating single attributes [20, 33, 13, 14, 30, 31, 15]. On the other hand, these has been very little work on multi-dimensional histograms. Muralikrishna *et al* proposed a heuristic hierarchical algorithm for constructing multi-dimensional equidepth histograms [27]. Poosala *et al* extended the definition of multi-dimensional histograms to other bucketization techniques, such as V-Optimal, MaxDiff etc [32] and provided a more sophisticated and general hierarchical partitioning algorithm called *MHIST*. But, neither of these algorithms provide any guarantee on the quality of the partitioning.

4 Hierarchical Tilings

In this section, we consider the hierarchical partitioning problems. Recall that the problem is to produce a hierarchical partition of an $n \times n$ array A with metric value at most δ using the minimum number of tiles. We set $N = n^2$ throughout. Let t_Q be an upper bound on the time taken to calculate the heft value of any tile in A; all our bounds below will be in terms of t_Q. First we focus on exact algorithms, and then approximate ones. *We state our results for only SUM and MAX metrics, but they hold for any superadditive metric with time bound identical to that for the SUM metric.*

4.1 Exact Algorithms

Theorem 1. *For cumulative function MAX, there exists an $O(N^{2.5} + N^2 t_Q)$ time algorithm to solve the hierarchical partitioning problem exactly. For cumulative function SUM, there exists an $O(N^{2.5}(B^*)^2 t_Q)$ time algorithm to solve the hierarchical partitioning problem exactly; here, B^* is the number of tiles in the optimum solution.*

Proof. The proof uses dynamic programming; it is simple, and it has appeared, for example, in [18], for a special case of the heft function. We include it here since some of our algorithms will be derived by suitably modifying this solution. We consider the cumulative function MAX only; the SUM case is a simple modification. Define $E^*(i \cdots j, k \cdots \ell)$ to be smallest number of tiles needed to partition the region $A[i \cdots j, k \cdots \ell]$ with metric value at most δ. If the heft of the tile $A[i \ldots j, k \ldots \ell]$ is at most δ, we have $E^*(i \cdots j, k \cdots \ell) = 1$. Else, we have Equation (I) below:

$$E^*(i \cdots j, k \cdots \ell) = \min_{i \le x < j,\ k \le y < \ell} \left\{ \begin{array}{l} E^*(i \cdots x, k \cdots \ell) \\ +E^*(x+1 \cdots j, k \cdots \ell), \\ E^*(i \cdots j, k \cdots y) \\ +E^*(i \cdots j, y+1 \cdots \ell) \end{array} \right\}$$

We need to calculate $E^* = E^*(1 \cdots n, 1 \cdots n)$. We use the dynamic programming technique and calculate $E^*(i \cdots j, k \ldots \ell)$ for all $1 \le i < j \le n$ and $1 \le k < \ell \le n$. In all, there are $O(n^4)$ possible sub-rectangles and for each we calculate $O(n)$ values and the heft of a single tile. Thus we take $O(n^5 + n^4 t_Q)$ time which is $O(N^{2.5} + N^2 t_Q)$ time. □

4.2 Approximate Algorithms

In this section, we present approximation algorithms for computing hierarchical partitions for the SUM and MAX metrics (as before, they can be modified to work for any superadditive metric); the algorithms in this section are faster than the exact ones in Section 4.1. Our algorithms rely on two natural, well-known ideas, namely, *rounding*, and *pruning*, which we informally describe below. These strategies "sparsify" the dynamic programming, that is, reduce the number of subproblems to be considered; this results in a speedier dynamic programming solution. They are easy to implement, and the technical crux is their analyses.

Rounding is a structured way to limit ourselves to only a subset of all possible tiles. This is achieved by considering only tiles with endpoints at multiples of a small number of parameters. However, there are many ways to perform rounding: round only along one of the dimensions, or possibly both; round so both the endpoints of tiles are multiples of the parameters, or just one of them; etc. Rounding can also be done hierarchically with choice of parameter values and rounding techniques at different levels. A general tradeoff for solving the partitioning problems in terms of all these combinations is difficult to state, but in our solutions, we employ interesting combinations and obtain our bounds.

Rounding to one grid pattern. We fix a parameter L and assume n is a multiple of L – the following description can be easily modified to handle other values of n. We define an *L-grid* as the subset of columns and rows numbered $1, L+1, 2L+1, \cdots$; thus there are n/L grid columns and rows in an L-grid. We refer to each such row (column) as the *L-row* (*L-column* respectively). We define the 1-hierarchical partition to be a hierarchical partition in which the tiles *additionally* satisfy the following two conditions. (1) At least one side has both its endpoints on L-rows on L-columns, that is, the side is of length a multiple of L, and (2) Both its sides have their endpoints *between* two consecutive L-rows or two consecutive L-columns; that is, they are each of length at most L. The 1-hierarchical partitioning problem is to produce a 1-hierarchical partition of an $n \times n$ array A with metric value at most δ using the minimum number of tiles. In what follows we will argue the following two points: (1) the optimal 1-hierarchical partition can be determined efficiently, and (2) the optimal 1-hierarchical partition approximates the hierarchical partition nicely.

Lemma 1. *For cumulative function MAX, there exists an $O(N^{1.6} t_Q)$ time algorithm to solve the* 1*-hierarchical partitioning problem exactly. For cumulative function SUM, there exists an $O(N^{1.6} (B^*)^2 t_Q)$ time algorithm to solve the* 1*-hierarchical partitioning problem exactly; here, B^* is the number of tiles in the optimum solution.*

Proof. We will only show the proof for the MAX function; the proof for the SUM function is similar. Define $E^*(i \cdots j-1, k \cdots \ell-1)$ to be smallest number of tiles in an 1-hierarchical partition of the region $A[i \cdots j-1, k \cdots \ell-1]$ with metric value at most δ. We employ the Equation (I) to compute $E^*(i \cdots j, k \cdots \ell)$, with some changes in the set of values taken by x and y in Equation (I). If $j - i > L$, then x takes *only* the values of L-rows in Equation (I) between i and j. otherwise, x takes all values between i and j; same holds for y values too. As before, the computation is done using the dynamic programming technique to calculate $E^* = E^*(1 \cdots n, 1 \cdots n)$. By choosing $L = n^{2/5}$ (optimized based on the running time calculations), the total running time of this algorithm turns out to be $O(N^{1.6}t_Q)$. $\square$

Lemma 2. *Consider any hierarchical partition of array A with B tiles and metric value at most δ, for an superadditive metric. There exists a 1-hierarchical partition using at most $9B$ tiles with metric value at most δ for any positive integer L.*

The proof of this lemma is omitted for space constraints. The two preceding lemmas together let us conclude that,

Theorem 2. *For cumulative function MAX, there exists an $O(N^{1.6}t_Q)$ time algorithm for solving the hierarchical partitioning problem that returns a hierarchical partition with at most $9B^*$ tiles and metric value at most δ; here, B^* is the minimum number of tiles in a hierarchical partition with metric value at most δ and the metric is superadditive. For the cumulative function SUM, the same holds with running time $O(N^{1.6}(B^*)^2t_Q)$; here, B^* is the number of tiles in the optimum solution.*

This algorithm is faster than the one in Theorem 1, but produces a partitioning with a slightly larger number of tiles. We can improve the running time further by increasing the number of tiles in a structured manner as described below.

Rounding to several grid patterns. We are given a sequence of positive integers $L_0, L_1, L_2, \dots L_k$ such that $L_0 = n$, $L_k = 1$ and L_{i+1} divides L_i for $i > 0$. We define a *k-hierarchical partition* as follows. It is a hierarchical partition in which there are k sets S_i of permissible intervals that can be the side lengths of the tiles. S_1 comprises of intervals $[jL_1, \ell L_1]$ for some integers $j < \ell$. In general, S_i for $1 \le i < k$, comprises the set of intervals $[jL_i, \ell L_i]$ such that for some h, $hL_{i-1} \le jL_i$ and $(h+1)L_{i-1} \ge \ell L_i$. The set S_k comprises intervals in S_i, for $i = k$, as defined above, but additionally, all subintervals thereof. The k-hierarchical partitioning problem is to find the k-hierarchical partition with metric value at most δ and minimum number of tiles. We can solve this problem again using dynamic programming, although the solution is somewhat more involved. The analysis of the running time of this algorithm follows the same line as in Lemma 1. We choose L_i's appropriately to minimize the running time; it turns out to be a geometric progression of values. Also, we claim: *Consider any hierarchical partition of array A with B tiles and cumulative metric value at most δ, for any superadditive metric. There exists a k-hierarchical partition using at most $(2k+1)^2B$ tiles with cumulative metric value at most δ for any positive integer k.* The proof is similar to that of Lemma 2. This lets us conclude,

Theorem 3. *For cumulative function MAX, there exists an $O(N^{1+\epsilon}t_Q)$ time algorithm for solving the hierarchical partitioning problem with at most $B^*O(1/\epsilon^2)$ tiles with metric value at most δ; here, B^* is the the minimum number of tiles in a hierarchical partition with metric value at most δ, ϵ is any chosen positive fraction, and the cumulative metric is superadditive.*

By choosing ϵ appropriately, we can get the running time to be as close to $O(Nt_Q)$ as we desire (which is linear for our applications – See Section 2.2). This is achieved at the expense of more (though not a very large number of) tiles. A different result which may be of interest is the following: we can prove that there is an $O(N(B^*)^2)$ time algorithm that approximates the minimum number of buckets to $O((\log\log n)^2)$ factor, for any superadditive metric.

The other natural idea we explore for approximate algorithms is pruning. Pruning also limits the set of tiles that are examined in dynamic programming. However, this is data-dependent since, effectively, we do not examine those tiles for which the heft is beyond certain prune condition. There is no efficient pruning strategy without rounding, since there are many large tiles that cannot be pruned. We have different pruning conditions for MAX and SUM. Due to space constraints, we omit the description of these conditions which can be found in [29]. Two examples of the results we obtain for MAX metrics are: an $O(N^{1.25}(B^*)^3)$ time algorithm for factor 9 approximation, and $O(N(B^*)^3)$ time algorithm for factor 25 approximation. Further results for MAX, SUM and other superadditive metrics can be found in [29].

5 Arbitrary Partitionings

5.1 NP-Hardness Results

In this subsection, we prove NP-hardness results for several metrics that show that minimizing the partitioning with p tiles that minimizes the heft is NP-hard. In fact, the proof also implies limits on the approximability of the problem for some cases.

For the special case of the MAX-SUM-ID metric, it was shown in [18] that the minimum heft cannot be approximated to within a factor of 1.25. We establish similar results for a a different set of metrics that includes SUM-SUM-SQR_DIFF, MAX-MAX-AVG_DIFF, and MAX-MAX-GEO_DIFF. As in [18] and the earlier work in [9], the proof is based on a reduction from the *Planar 3SAT* problem (shown to be NP-complete in [21]), though a number of changes are needed to adapt the argument to our types of metrics. Similar results can also be shown for several other metrics, but we restrict ourselves to the most important ones (proofs are omitted here).

Theorem 4. *Given a data distribution A and an upper bound on p, it is NP-hard*

– *to find the minimum heft of any rectangular partitioning with p tiles under the SUM-SUM-SQR_DIFF metric, and*
– *to approximate the minimum heft of any rectangular partitioning with p tiles under the MAX-MAX-GEO_DIFF metric to any factor less than 2, and*
– *to approximate the minimum heft of any rectangular partitioning with p tiles under the MAX-MAX-AVG_DIFF metric to any factor less than 3/2.*

5.2 Approximate Algorithms

In view of the hardness results in Section 5.1, we can not anticipate efficient, that is, polynomial time algorithms for exactly solving the partitioning problems with arbitrary partitions for many natural metrics. In this section, we focus on developing efficient approximate algorithms instead. All our approximations are based on the following observation which was presented in [18] for a special metric.

Lemma 3. *Consider any arbitrary partitioning of a two dimensional array A with superadditive cumulative metric at most δ and B tiles. There exists a hierarchical partition of A with cumulative metric at most δ and at most $4B$ tiles.*

Proof. It is shown in [8] that any arbitrary rectangular partition can be converted into a hierarchical one by splitting each tile into at most 4 disjoint tiles. If we apply their procedure to the given arbitrary partition, the resulting hierarchical partition has at most $4B$ tiles; furthermore, the metric value of this hierarchical partition is at most δ by the superadditivity of the metric. □

Using this observation with the results in Section 4, we get the following. (Similar results can be obtained for the SUM cumulative function as well.)

Theorem 5. *For cumulative function MAX, say B^* is the optimal solution to the arbitrary partitioning problem with metric value at most δ. There is an algorithm that finds a partition with metric value at most δ in*

1. $O(N^{2.5} + N^2 T)$ *time; the solution has at most* $4B^*$ *tiles.*
2. $O(N^{1+\epsilon} T)$ *time; the solution has at most* $4B^* O(1/\epsilon^2)$ *tiles.*

Again, by setting ϵ appropriately, we can obtain an algorithm with near linear running time.

Note. Using the ideas in [18], an $O(N^c)$ time algorithm can be obtained that approximates the arbitrary partition using at most twice as many buckets as the optimum; however, the c is rather large (at least 5), and the resulting algorithm is impractical for all but tiny values of N.

6 p×p Partitioning Schemes

In this section, we consider the $p \times p$-partitioning problem defined earlier. Thus, we are given a two-dimensional distribution A, a metric E and a value δ, and we are interested in finding a minimum p, and a $p \times p$ partitioning H, such that $E(H) \leq \delta$. Here, a $p \times p$ partitioning is determined by a set of horizontal dividers (rows) $h_0 = 0 \leq h_1 \leq \ldots \leq h_p = n$ and a set of vertical dividers (columns) $v_0 = 0 \leq v_1 \leq \ldots \leq v_p = n$, and tile $r_{i,j}$ of the partition consists of all entries $A[k,l]$ such that $h_{i-1} < k \leq h_i$ and $v_{j-1} < l \leq v_j$.

We describe algorithms that run in linear or nearly linear time and that compute solutions that are guaranteed to be within a small constant factor of optimal. The algorithms provide an interesting application of the framework for approximating Set Cover for set systems with bounded Vapnik-Chervonenkis (VC) dimension described by Brönnimann and Goodrich [4] (see also the discussion in Section 6.3.)

6.1 NP-Hardness Results

For the special case of the MAX-SUM-ID metric, Charikar, Chekuri, Feder, and Motwani [6] have shown that it is NP-hard to approximate the minimum heft of any $p \times p$ partitioning to within a factor of less than 2. This result can be extended to several other interesting metrics, including SUM-SUM-SQR_DIFF, MAX-MAX-AVG_DIFF, and MAX-MAX-GEO_DIFF. We point out that all the results are obtained by modifications of the hardness proof in [6], which uses a reduction from the *k-Balanced Bipartite Vertex Cover* (k-BBVC) problem. The results are summarized in the following theorem, the proof of which is omitted for space constraints.

Theorem 6. *Given a data distribution A and an upper bound on p, it is NP-hard*

- *to approximate the minimum heft of any $p \times p$ partitioning under the MAX-MAX-AVG_DIFF and MAX-MAX-GEO_DIFF metrics to any factor less than* 2*, and*
- *to find the minimum heft of any $p \times p$ partitioning under the SUM-SUM-SQR_DIFF metric.*

The proof of the first claim follows from some fairly simple modifications of the proof in [6], while the second claim requires an additional accounting argument. Of course, the result also implies the NP-hardness of the problem of minimizing p given an upper bound on the heft, though we do not have any inapproximability result for that case.

6.2 Preliminaries

We denote by X the set of n rows and n columns of the $n \times n$ distribution A. As before, we assume $N = n^2$. For each tile $r_{i,j}$ of a $p \times p$ partitioning H, we define a corresponding subset $R_{i,j}$ of X consisting of all rows and columns that intersect $r_{i,j}$, except for the last intersecting row and column. We also use a weight function w, to be defined later, that assigns a real-valued weight $w(x)$ to each $x \in X$, and define $w(Y) = \sum_{y \in Y} w(y)$ for any subset Y of X.

Definition 1. *Given a weight function w, we say that a $p \times p$ partitioning is* α-good *if every tile $r_{i,j}$ of H satisfies $w(R_{i,j}) \leq \alpha \cdot w(X)$.*

We remark that our α-good partitionings correspond to the ϵ-nets used in [4] and originally introduced in [12], which have found many applications in computational geometry.

6.3 Upper Bounds for MAX Metrics

We now present approximation results for the $p \times p$-partitioning problem where the cumulative metric is the MAX metric, i.e., the heft of a partition is the largest heft of any of the tiles. We also require that the heft function is monotonic, i.e., the heft of a tile does not decrease if we grow its size, which is true for most interesting cases[2].

The Algorithm Suppose that we are given a maximum value δ for the heft of the solution, and assume that there exists a $p_0 \times p_0$ partitioning H_0 with heft at most δ (the value of p_0 will be guessed using binary search). We will show that the following algorithm computes a $p \times p$ partitioning with heft at most δ and $p \leq (2+\epsilon) \cdot p_0$, for any chosen $\epsilon > 0$.

Algorithm *MAX-pxp*.

(1) Set the weights of all elements of X to 1.
(2) Repeat the following three steps:
  (a) Compute an α-good partitioning H, for $\alpha = \frac{1}{(2+\epsilon)p_0}$.
  (b) Find a tile $r_{i,j}$ in H such that $w(R_{i,j}) > \delta$. If none exists, terminate and return H as solution.
  (c) Multiply the weights of all elements of X that are contained in $R_{i,j}$ by $\beta = (1 + \epsilon/2)$.

Analysis of the Algorithm. We now analyze the performance of the algorithm in three steps: (1) We show how Step (2a) can be implemented, and bound the size of the resulting α-good partitioning, (2) we bound the number of iterations in Step (2), and (3) we analyze the running time of each iteration. Theorem 7 then gives the main result of this subsection.

Lemma 4. *There exists an α-good $p \times p$ partitioning with $p = 1/\alpha$. It can be computed in time $O(n)$.*

Proof: Simply set $1/\alpha - 1$ horizontal dividers one after the other, starting at the top, and repeatedly choosing the next divider h_i as the first element where the sum of the weights of all rows encountered after h_{i-1} surpasses $\alpha \cdot w(X_h)$, where X_h is the set of all rows. Choose the vertical dividers v_i in an analogous fashion. □

Lemma 5. *The loop in Step (2) of* MAX-pxp *terminates after $O(p \log n)$ iterations.*

Proof: The proof is similar to that of Lemma 3.4 of [4], which itself follows the arguments in [7, 22, 34].

Note that the weight $w(X)$ is initially $2n$, and that it increases by at most a factor of $\left(1 + \frac{\epsilon}{2 \cdot (2+\epsilon) \cdot p_0}\right)$ in each iteration, since in each iteration we multiply

[2] An exception is the MAX-MAX-AVG_DIFF metric.

the weights of exactly one of the sets $R_{i,j}$ by a factor of $(1+\epsilon/2)$, and this set $R_{i,j}$ has a total weight of at most $\frac{1}{(2+\epsilon)\cdot p_0} \cdot w(X)$ due to the definition of α-goodness. Thus, after k iterations, we can upperbound $w(X)$ by

$$2n\left(1+\frac{\epsilon}{2\cdot(2+\epsilon)\cdot p_0}\right)^k \leq 2n{\cdot}exp\left(\frac{\epsilon\cdot k}{2\cdot(2+\epsilon)\cdot p_0}\right) = exp\left(\frac{\epsilon \dot{k}}{2\cdot(2+\epsilon)\cdot_0} + ln(2n)\right)$$

where $exp()$ denotes the exponential function with basis e. We now consider the weight $w(H_0)$ of the $p_0 \times p_0$ partitioning H_0 of heft at most δ that we assume to exist.[3] Note that any tile $r_{i,j}$ that is selected in Step (2b) has a heft larger than δ, and hence H_0 must cut $r_{i,j}$, due to the monotonicity of the heft function. This implies that at least one element of H_0 is also contained in $R_{i,j}$ and has its weight increased by a factor of $(1+\epsilon/2)$. Thus, we have

$$w(H_0) = \sum_{x_i \in H_0} (1+\epsilon/2)^{z_i}$$

with $\sum z_i >= k$, where z_i denotes the number of times the weight of the corresponding element $x_i \in H_0$ has been multiplied by $(1+\epsilon/2)$. Using the convexity of the exponential function with basis $(1+\epsilon/2)$, we can lower-bound this as

$$w(H_0) \geq 2p_0 \cdot (1+\epsilon/2)^{k/(2p_0)} = exp\left(\frac{ln(1+\epsilon/2)\cdot k}{2p_0} + ln(2p_0)\right).$$

Since H_0 is a subset of X, we must have $w(H_0) \leq w(X)$, which implies that

$$\frac{ln(1+\epsilon/2)\cdot k}{2p_0} + ln(2p_0) \leq \frac{\epsilon\cdot k}{2\cdot(2+\epsilon)\cdot p_0} + ln(2n),$$

which, using the inequality $ln(1+\epsilon/2) \geq \frac{\epsilon}{2+\epsilon}$ for $0 < \epsilon < 1$, can be shown to imply that

$$k \leq p_0 \cdot \frac{ln(n/p_0)}{ln(1+\epsilon/2) - \frac{\epsilon}{2+\epsilon}}.$$

□

Lemma 6. *Each iteration in Step (2) runs in time $O(n + p^2 \cdot t_Q)$, where t_Q is the time needed to compute the heft of a tile.*

Proof: Steps (2a) and (2c) clearly run in time $O(n)$. In Step (2b), we have to compute the heft of at most p^2 tiles to find a tile with heft more than δ. □

Theorem 7. *For any δ and any $\epsilon > 0$, a $p \times p$ partitioning H with heft at most δ and $p \leq (2+\epsilon)p_0$ can be computed in time $O((n + p^2 \cdot t_Q) \cdot p \log n)$, where p_0 is the minimum number such that there exists a $p_0 \times p_0$ partitioning with heft at most δ, and t_Q is the time needed to compute the heft of any tile.*

[3] The weight of H_0 is the sum of the weights of the rows and columns that are horizontal dividers h_i or vertical dividers v_i, respectively, of H_0.

Proof: We perform binary search for the value of p_0, starting at $p_0 = 2$. We run algorithm *MAX-pxp* for each p_0 in the search. If the algorithm does not terminate after the number of iterations stated in Lemma 5, then we know that there is no $p_0 \times p_0$ partitioning with heft at most δ, and we increase p_0 by some small factor $(1+\epsilon')$. The total running time is dominated by the time used to run *MAX-pxp* on the largest p_0; this implies the stated bound. □

As explained before, in many cases the heft of each tile can be computed in $O(1)$ time, by performing $O(N)$ steps of preprocessing. In particular, this is true for the case of the MAX-SUM-ID metric, which is probably the most important of the metrics covered by Theorem 7, and we get the following corollary. Note that for the common case of $p \ll \sqrt{N}$, this gives a linear time bound.

Corollary 1. *For the MAX-SUM-ID metric, a $p \times p$ partitioning H with heft at most δ and $p \leq (2+\epsilon)p_0$ can be computed in time $O(N + p^3 \log N)$, where p_0 is the minimum number such that there exists a $p_0 \times p_0$ partitioning with heft at most δ.*

Discussion. We now discuss the relation of our algorithm to the work in [19] and [4]. A simple reduction of the $p \times p$ partitioning problem to the Set Cover problem was given in [19], resulting in an approximation ratio of $O(\log N)$ using the well known greedy algorithm for Set Cover. This bound can be improved to $O(\log p)$ by using the algorithm for approximating Set Cover for the case of bounded VC-dimension in [4], and observing that the set system generated by the reduction in [19] has VC-dimension 4. By additionally using a construction of an ϵ-net for this set system along the lines of our Lemma 5, one can obtain an approximation ratio of 16 and a running time of $O(N^{3/2} \cdot p \log N)$. In order to get near-linear running time, we describe a modified algorithm that operates directly on the data distribution without materializing the set system used for the reduction to Set Cover, which could be of size $\Theta(N^{3/2})$ in the worst case. The approximation ratio of $(2+\epsilon)$ is then obtained by tightening the analysis of [4] in several places.

Approximating the Error. For the important special case of the MAX-SUM-ID metric, which arises when partitioning data or work evenly among the tiles, we can also get significantly improved results for the problem of approximating the minimal heft of any $p \times p$ partitioning given an upper bound on p. The best previous result in [19] achieved a running time of $O(N^2)$ and an approximation ratio of around 120. We can show the following results.

Theorem 8. *Let δ_0 be the minimum heft of any $p_0 \times p_0$ partitioning. Then in time $O(N + p^3 \log N)$, we can compute*

(a) a $p \times p$ partitioning with heft $\delta \leq 4\delta_0$ and $p \leq (\frac{2}{3}+\epsilon)p_0$, and
(b) a $p \times p$ partitioning with heft $\delta \leq 2\delta_0$ and $p \leq (1+\epsilon)p_0$,

for any chosen $\epsilon > 0$.

Note that these results come quite close to the lower bound of 2 on the approximability shown by Charikar, Chekuri, Feder, and Motwani [6]. These results are based on a fairly simple observation: If we modify Algorithm *MAX-pxp* such that in Step (2b) we search for a tile with heft at least $2\delta_0$ ($4\delta_0$), then we can conclude that the optimum solution H_0 must cut this tile at least 2 (resp., 3) times in order to get a heft of at most δ_0. This means that Step (2c) guarantees a larger increase in the weight of H_0. We can then adjust the choice of the other parameters α and β appropriately to get the result.

Other Extensions. All results in this subsection can also be easily extended to $p \times q$ partitionings with $p \neq q$, and to non-square input distributions. In particular, in the case of a $p \times q$ partitioning of an $n \times m$ data distribution, the running time becomes $O(n + m + pq \cdot t_Q)(p + q)^2 \log(n + m))$, while the approximation ratio remains as before. It is also easy to extend the techniques to d dimensions, resulting in an approximation ratio of $d + \epsilon$ and a running time of $O((n + p^d \cdot t_Q)p \log N)$ for the result in Theorem 7.

Input data in higher dimensions is usually sparse, and thus efficiency crucially depends on exploiting this sparseness. For the algorithms in this subsection, the easiest solution would be to implement the computation of the heft of a tile (represented by the term t_Q) in a way that exploits sparseness; the details depend on the particular metric.

6.4 Upper Bounds for SUM Metrics

We now present approximation results for the case where the cumulative metric is the SUM metric. We again require that the heft function is monotonic. The algorithm follows the approach from the previous subsection. In contrast to the MAX case, we are not aware of any direct reduction of the SUM case to the Set Cover problem, and thus it is surprising that the same approach applies.

The Algorithm Suppose that we are given a maximum value δ_0 for the heft of the solution, and assume that there exists a $p_0 \times p_0$ partitioning H_0 with heft at most δ_0. Then the following algorithm computes a $p \times p$ partitioning with heft at most $2\delta_0$ and $p \leq (4 + \epsilon) \cdot p_0$, for any chosen $\epsilon > 0$.

Algorithm *SUM-pxp*.

(1) Set the weights of all elements of X to 1.
(2) Repeat the following three steps:
   (a) Compute an α-good partitioning H, for $\alpha = \frac{1}{(4+\epsilon)p_0}$.
   (b) If the heft of the partitioning is at most $2\delta_0$, terminate and return H as solution. Otherwise, select a tile $r_{i,j}$ at random such that the probability of picking a tile is proportional to its heft.
   (c) Multiply the weights of all elements of X that are contained in $R_{i,j}$ by $\beta = (1 + \epsilon/2)$.

Sketch of Analysis. The following lemma provides the main insight underlying the analysis.

Lemma 7. *With probability at most $\frac{1}{2}$, the tile chosen in Step (2b) is not cut by H_0.*

Proof: Let U be the set of tiles that are not cut by H_0. Then the hefts of the tiles in U sum up to at most δ_0, since otherwise the monotonicity of the heft function would imply that H_0 has a heft of more than δ_0. Since the sum of the hefts of all the tiles is at least $2\delta_0$, and each tile is chosen with probability proportional to its heft, the probability of choosing a tile from U is at most $\frac{1}{2}$. □

The lemma directly implies that the weight of H_0 is increased in Step (2c) with probability at least $\frac{1}{2}$. This results in a slightly weaker lower bound for $w(H_0)$ as compared to the MAX case in the previous subsection. To deal with this weaker lower bound, we compute an α-good partitioning with $\alpha = \frac{1}{(4+\epsilon)p_0}$ instead of $\frac{1}{(2+\epsilon)p_0}$. The remainder of the analysis is then along the lines of the analysis for the MAX case, and we get the following result.

Theorem 9. *For any δ_0 and any $\epsilon > 0$, a $p \times p$ partitioning H with heft at most $2\delta_0$ and $p \leq (4+\epsilon)p_0$ can be computed in expected time $O((n + p^2 \cdot t_Q) \cdot p \log n)$, where p_0 is the minimum number such that there exists a $p_0 \times p_0$ partitioning with heft at most δ_0, and t_Q is the time needed to compute the heft of any tile.*

Discussion and Extensions. We point out that the algorithm can be easily made deterministic by modifying Steps (2b) and (2c) such that instead of choosing one particular tile at random, we take every tile $r_{i,j}$ and increase the weight of the elements in $R_{i,j}$ by a factor that is proportional to the heft of the tile. Also, the algorithm can be generalized to yield a trade-off between the approximation of the error and the approximation of the number of cuts.

As before, in many cases t_Q can be implemented in $O(1)$ steps by performing appropriate preprocessing. An interesting example is the SUM-SUM-SQR_DIFF metric, which models the case when we wish to form tiles containing similar values. Finally, the result can also be extended to $p \times q$ partitionings with $p \neq q$ and to higher dimensions, resulting in the same bounds as for the MAX case.

7 An Example Application of Results

We focus on one database application where partitioning problems arise; see [29] for the implications of our results in other applications.

Consider a database over relation R with n numerical attributes. This can be visualized as a multidimensional array A with one attribute along each dimension in which each array element contains the number of tuples in the database with the associated attribute values. This is the *joint frequency distribution* of the database. Histograms partition this distribution into rectangular regions (*buckets*) and approximate each region using a small amount of space. Typically, the frequencies in a bucket are approximated by their average.

Histograms are typically used to estimate the result sizes of relational queries. The errors in the estimation depend mainly on the bucketization. A theory of optimal histograms has been developed [31] in which a number of histograms have been identified as being optimal for various queries and operators. These histograms, such as Equidepth, Equiwidth etc can all be considered as special cases of our partitioning problems, and our results apply to them uniformly. Here, we focus on an important class of histograms, namely, the *V-Optimal histogram* which is provably the most accurate in several estimation problems [31].

Definition 2. V-Optimal Histogram *[14]: For a given number of buckets, a V-Optimal histogram is the one with the number of buckets bounded by the specified threshold, but having the least* variance*, where variance is the sum of squared differences between the actual and approximate frequencies. Alternately, for a given total variance, the V-Optimal histogram is one with the least number of buckets with total variance bounded by the specified threshold.*

Note that for joint distributions over two attributes, one version of the problem of constructing the optimal V-optimal histogram is identical to the partitioning problem with arbitrary partitions under the SUM-SUM-SQR_DIFF metric. By applying our general results to this partitioning problem, we derive the following results (further details are in [29].):

1. Identifying the optimal V-Optimal histogram with arbitrary buckets is NP-Hard.
2. The greedy MHIST algorithm presented in the literature [32] can result in *arbitrarily* poor histograms in terms of the buckets, as well as the total variance, whichever is being optimized; we can construct inputs to induce such worst case behavior. In fact, this applies to many other greedy solutions we can design for this problem.
3. We can approximate the minimum number of buckets needed to achieve the threshold variance in the V-Optimal histogram by using results in Section 5.2. The resulting algorithms work in near-linear time and produce small factor approximations.

8 Conclusions

We have considered the complexity of partitioning problems for different partitions and metrics. These problems are fundamental, and they arise in application scenarios such as histogram-based selectivity estimation, constructing grid files, load balancing, and many others. Very little is known about the complexity of these problems except for some special metrics, and heuristics with no proven guarantees on the quality of the solution are used.

In this paper, we show that many natural versions of the partitioning problem are NP-hard and thus it is unlikely they have efficient (polynomial time) *exact* solutions in the worst case. Our main results, however, are positive ones. We present highly efficient (near-linear time) algorithms that approximate the solutions to within small constant factors, for different partitions and metrics.

We applied our general results to solving an important problem arising in query result size estimation: the identification of V-Optimal histograms in two dimensions. Existing greedy algorithms do not offer any quality guarantees for this NP-Hard problem; our approximate solutions to the partitioning problems imply the first known efficient algorithms for this problem with guarantees. We are investigating its impact in practice.

References

1. S. Anily and A. Federgruen. Structured partitioning problems. *Operations Research*, 13, 130–149, 1991.
2. S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. *Proc 37th IEEE Symp. of Foundations of Computer Science (FOCS)*, pages 2–12, 1996.
3. S. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, 37, 38–57, 1988.
4. Brönnimann and Goodrich. Almost optimal set covers in finite VC-dimension. In *Proceedings of the 10th Annual Symposium on Computational Geometry*, 1994.
5. B. Carpentieri and J. Storer. A split-merge parallel block matching algorithm
6. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Personal communication, 1996.
7. K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 452–456, October 1988.
8. F. d'Amore and P. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Proc. Letters*, 44, 255–259, 1992.
9. R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Proc. Letters*, 12, 133–137, 1981.
10. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors*, volume 1. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
11. M. Grigni and F. Manne. On the complexity of the generalized block distribution. Proc. of 3rd international workshop on parallel algorithms for irregularly structured problems (IRREGULAR '96), Lecture notes in computer science 1117, Springer, 319-326, 1996.
12. D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete and Computational Geometry*, 2:127–151, 1987.
13. Y. Ioannidis. Universality of serial histograms. *Proc. of the 19th Int. Conf. on Very Large Databases*, pages 256–267, December 1993.
14. Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. *Proc. of ACM SIGMOD Conf*, pages 233–244, May 1995.
15. H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. *Proc. of the 24rd Int. Conf. on Very Large Databases*, pages 275–286, August 1998.
16. J. Jain and A. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on communications*, 29, 1799–1808, 1981.
17. M. Kaddoura, S. Ranka and A. Wang. Array decomposition for nonuniform computational environments. *Technical Report*, Syracuse University, 1995.

18. S. Khanna, S. Muthukrishnan, and M. Paterson. Approximating rectangle tiling and packing. *Proc Symp. on Discrete Algorithms (SODA)*, pages 384–393, 1998.
19. S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. *Proc. Intl. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 616–626, 1997.
20. R. P. Kooi. *The optimization of queries in relational databases.* PhD thesis, Case Western Reserve University, Sept 1980.
21. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Computing*, 11, 329–343, 1982.
22. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 68–77, October 1987.
23. F. Manne. *Load Balancing in Parallel Sparse Matrix Computations.* Ph.d. thesis, Department of Informatics, University of Bergen, Norway, 1993.
24. F. Manne and T. Sorevik. Partitioning an array onto a mesh of processors. Proc. of *Workshop on Applied Parallel Computing in Industrial Problems.* 1996.
25. C. Manning. *Introduction to Digital Video Coding and Block Matching Algorithms.* http://atlantis.ucc.ie/dvideo/dv.html.
26. J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple method for geometric k-mst problem. *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 402–408, 1996.
27. M. Muralikrishna and David J Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. *Proc. of ACM SIGMOD Conf*, pages 28–36, 1988.
28. J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.
29. S. Muthukrishnan, V. Poosala and T. Suel. On rectangular partitionings in two dimensions: algorithms, complexity and applications. *Manuscript*, 1998.
30. V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. *Proc. of ACM SIGMOD Conf*, pages 294–305, June 1996.
31. V. Poosala. *Histogram-based estimation techniques in databases.* PhD thesis, Univ. of Wisconsin-Madison, 1997.
32. V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. *Proc. of the 23rd Int. Conf. on Very Large Databases*, August 1997.
33. G. P. Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *Proc. of ACM SIGMOD Conf*, pages 256–276, 1984.
34. E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 23–33, June 1988.