Approximation Algorithms for Array Partitioning Problems

S. Muthukrishnan[*] Torsten Suel[†]

Abstract

We study the problem of optimally partitioning a two-dimensional array of elements by cutting each coordinate axis into p (resp., q) intervals, resulting in $p \times q$ rectangular regions. This problem arises in several applications in databases, parallel computation, and image processing. Our main contribution are new approximation algorithms for these NP-Complete problems that improve significantly over previously known bounds. The algorithms are fast and simple, work for a variety of measures of partitioning quality, generalize to dimensions $d > 2$, and achieve almost optimal approximation ratios. We also extend previous NP-Completeness results for this class of problems.

1 Introduction

Many problems arising in diverse areas such as parallel computation, databases, and image processing require the partitioning of 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, minimizing the number of partitions while not exceeding a given value of that metric. As a simple example, in parallel computation, we may want to partition a computational space among a fixed number of nodes such that no node receives significantly more work than the others.

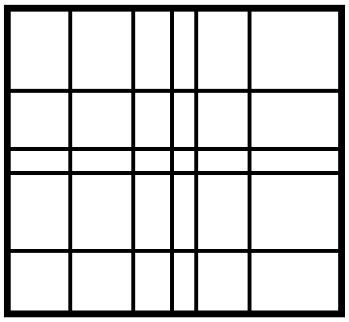

Figure 1: A 5×6 partitioning.

In the paper, we study one important class of such partitioning problems, where the input consists of a two-dimensional (or higher-dimensional) array of real values, and the partitioning is obtained by dividing the y-axis into p intervals and the x-axis into q intervals, resulting in $p \times q$ tiles. We refer to such a partitioning as a $p \times q$ partitioning; see Figure 1 for an example. For simplicity, we will usually focus on the two-dimensional case; however, our results also apply to higher dimensions. We start out by motivating the problem with two example applications.

[*]Department of Computer Science, Rutgers University, Piscataway, NJ 08854. Email: `muthu@cs.rutgers.edu`.
[†]CIS Department, Polytechnic University, Brooklyn, NY 11201. Email: `suel@poly.edu`.

Example 1 (Load Balancing): *In the context of load partitioning in parallel computation, we are typically interested in minimizing total completion time, which is determined by the time the last processor finishes its work. In many scenarios, the workload can be represented by a two- or higher-dimensional array, such as a matrix that needs to be partitioned, or a two- or three-dimensional space in atmospheric, chemical, or astrophysical simulations. The input array to our partitioning problem is thus a two- or higher-dimensional array specifying the amount of work associated with each matrix element or each unit of space in the simulation. We note that the array may often be sparse, particularly in higher dimensions.*

Thus, given a fixed number of nodes, we would like to partition the array into tiles such that no node receives a disproportionate amount of the work. Formally, we are interested in a partitioning that minimizes a metric defined as the maximum, over all tiles in the partitioning, of the sum of the elements in the tile [22, 28]. See the left part of Figure 1, where we partition an array with a 3×3 *partitioning such that no tile sums up to more than* 9*. The total amount of work is* 65 *and thus* $\lceil 65/9 \rceil = 8$ *provides a trivial lower bound for the maximum sum of a tile (or* $65/9 = 7.222$ *if we have non-integer values in the input).*

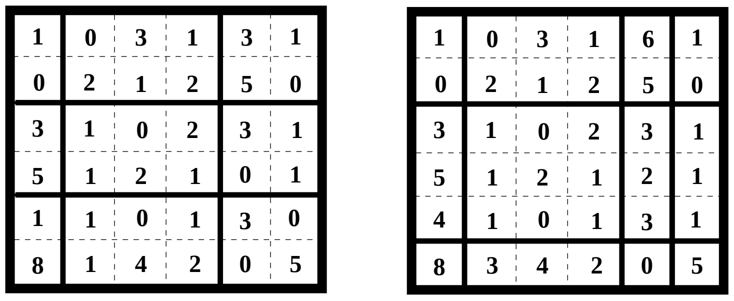

Figure 2: A 3×3 partitioning (left) and a 3×4 partitioning (right) of two different 6×6 arrays.

Example 2 (Data Approximation): *A slightly different scenario arises in some applications in databases or image processing. For example, database systems maintain* histograms *based on rectangular regions to approximate multi-dimensional data distributions; see, e.g., [1, 15, 16, 17, 20, 26, 30, 31, 32]. These histograms are then used by the query optimizer to obtain reasonably accurate estimates of query result sizes, in order to estimate the costs of different execution plans. In the case of an employee database, we may be given a two-dimensional array that contains for each age and for each salary level the number of tuples in the database with matching attribute values. To succinctly approximate this array by a histogram, we are interested in a partitioning that tries to group clusters of points with similar frequencies into tiles. In image processing, we may be interested in partitioning images into tiles representing areas with fairly uniform color.*

There are several formulations of this problem in the literature, but one popular choice called V-Optimal histograms [31] involves minimizing the following well-known metric based on the variance in each tile. The metric is defined as the sum, over all elements in the array, of the square of the difference between its value and the average value in its tile. In the right part of Figure 1, we show a 3×4 *partitioning of an array that attempts to minimize this metric by grouping similar values into tiles. Each uniform tile contributes a value of zero to the metric, and the values contributed by the* 12 *tiles are* $1/2 + 22/4 + 1/2 + 1/2 + 2 + 4 + 6/9 + 0 + 0 + 2 + 0 + 0$ *(in row major order) for a total of* 15.666*. This is in fact the best possible* 3×4 *partitioning, though it may not be obvious.*

In this paper, we focus on the problem of partitioning an input array into tiles while minimizing a given metric, defined on the set of all possible partitionings, that measures the quality of a partitioning. In general, there are many different metrics of possible interest. We present algorithms and complexity results that hold for fairly general classes of metrics that satisfy certain conditions defined later, and that include almost all of the metrics studied in the literature. The two metrics discussed in the examples, which we refer to as MAX-SUM-ID and SUM-SUM-SQRDEV, respectively, are probably the most commonly studied ones. They are also interesting because they are quite different in the intuition they provide to the algorithm designer, and they capture many aspects of the types of metrics that we consider. We note that a number of heuristic approaches for these problems have been proposed in different application areas. Most of these approaches are designed with a particular metric in mind, and many do not provide any provable bounds on the quality of the solution.

Discussion of Rectangular Partitionings: Rectangular partitionings, i.e., partitionings where each tile is a simple rectangle, are desirable in many applications due to the simple and regular structure of the tiles and the fact that they tend to preserve a significant amount of the locality in the underlying computational domain. Nonetheless, there have been many approaches that use other types of shapes, such as triangles, hexagons, rectangles with holes or even more complex shapes. From an algorithmic point of view these problems tend to be quite different from the rectangular case, as are problems that allow a *covering*, rather than partitioning, of the array.

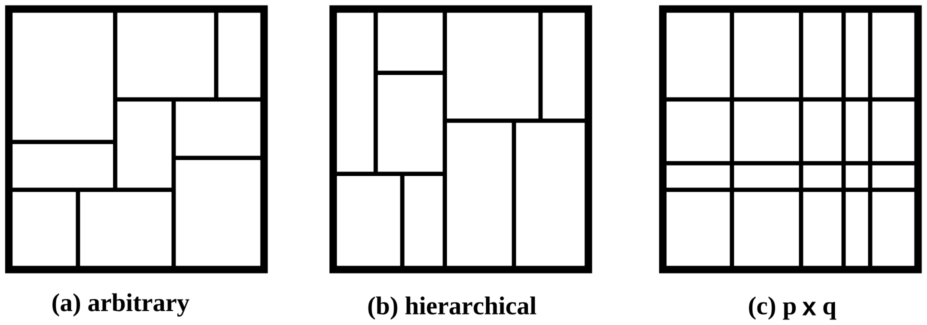

Figure 3: Families of rectangular partitionings.

There are several different families of rectangular partitionings that are commonly studied. In Figure 1 we show three common families: (a) arbitrary, allowing any partitioning into rectangular tiles, (b) hierarchical, obtained through recursive cuts, and (c) the $p \times q$ partitionings considered in this paper. Thus, $p \times q$ partitionings are a subset of the class of hierarchical partitionings, which themselves are a subset of the arbitrary partitionings. We note that there are obviously worst-case input distributions, e.g., for the metrics in the two examples, for which any $p \times q$ partitioning requires a much larger number of tiles to achieve the same value of the metric, when compared to hierarchical and arbitrary partitionings. Another concern is how to choose the values of p and q, though our algorithms are efficient enough to allow search heuristics for finding the best values of p and q on top of them.

However, there are some good reasons why $p \times q$ partitionings are preferred in certain scenarios. In the context of parallel computing, $p \times q$ partitionings result in a very simple communication pattern since every node has exactly 4 horizontal and vertical neighbors. This simplifies the problem, e.g., for compilers that automatically parallelize code, and in fact $p \times q$ partitionings are part of the High Performance Fortran 2.0 standard [14]. Another advantage, important for example in grid files [29] and certain classes of histograms [1, 12], is that the partitioning is uniquely defined by $p-1$ coordinates along the y-axis and $q-1$ coordinates along the x-axis. Thus, tiles can be indexed very efficiently by these coordinates. This simple structure also enables optimizations for cases where many tiles are empty or almost empty, and so a larger number of tiles may be acceptable in some applications.

2 Problem Definition

We now formally define the problem we are considering, for the case of a two-dimensional array. Generalization to the multi-dimensional case is straightforward.

Tiles and Partitionings: We are given an $m \times n$ array $A[1 \ldots m, 1 \ldots n]$ containing $N = m \cdot n$ nonnegative real numbers. Note that the array is indexed starting from 1 rather than 0. A *tile* is any rectangular subarray $A[i \ldots j, k \ldots l]$ with $0 < i \leq j \leq m$ and $0 < k \leq l \leq n$. A *partitioning* of array A is a partitioning of A into tiles; by definition of a partitioning, each element $A[i, j]$ lies within some tile and no two tiles overlap. We restrict ourselves to $p \times q$ partitionings and assume $p \geq q$ without loss of generality. Formally, a $p \times q$ partitioning is determined by a set of horizontal dividers (rows) $h_0 = 0 \leq h_1 \leq \ldots \leq h_p = m$ and a set of vertical dividers (columns) $v_0 = 0 \leq v_1 \leq \ldots \leq v_q = n$. The partitioning consists of $p \cdot q$ tiles $r_{i,j}$, $0 \leq i < p$ and $0 \leq j < q$, where tile $r_{i,j}$ is the set of all entries $A[k, l]$ such that $h_{i-1} < k \leq h_i$ and $v_{j-1} < l \leq v_j$.

Metrics: A quality metric assigns a value to a particular partitioning. Formally, a quality metric is defined by an *elementary function* $e()$ that computes a score for each element in the array, a *heft function* $h()$ that combines the scores of the elements in a tile to compute the *heft* of a tile, and a *cumulative function* $f()$ that combines the hefts of the tiles to obtain the overall value of a partitioning.[1] We are interested in finding partitionings that minimize the value of the quality metric.

We have already encountered two metrics. In Example 1, $e()$ is the identity function, $h()$ is the sum function, and $f()$ is the maximum function over all tiles; we refer to this as the MAX-SUM-ID metric. In Example 2, $e()$ is the square of the deviation from the average value in the tile, and $h()$ and $f()$ are both the sum function; we refer to this as the SUM-SUM-SQRDEV metric. In general, we use the notation *f-h-e* to denote a metric with elementary function $e()$, heft function $h()$, and cumulative function $f()$. Common choices for $e()$ in applications are the identity function (denoted ID), or the deviation from the arithmetic average or geometric median in the tile, or the square of the deviation from the arithmetic average (denoted SQRDEV). Common choices for $h()$ are sum (SUM), maximum (MAX), or sometimes the ratio between the largest and smallest score in the tile, while $f()$ is almost always[2] the sum or maximum.

We say that a heft function $h()$ (including a particular choice of $e()$) is *monotonic* if we have $h(r) \leq h(r')$ for any two tiles r, r' with $r \subset r'$ and any input A. Both of the functions in the examples are monotonic, assuming that all entries are nonnegative in the first example. We say that a metric $f()$ (including a particular choice of $e()$ and $h()$) is *superadditive* if $f(P) \geq f(P')$ for any input A and any partitioning P' that is obtained by refining a partitioning P, i.e., by subdividing one or more tiles of P. In other words, a superadditive metric has the property that applying additional cuts will never degrade the quality of the partitioning. Note that this definition is independent of the types of partitionings ($p \times q$, hierarchical, arbitrary) that we allow. Both metrics in the examples are superadditive.

Tile Heft Queries: All of the algorithms we describe access the input array using only a single simple query operation that, given the boundaries of a candidate tile, returns the heft of this tile. We note that after appropriate preprocessing of the array, this operation can usually be implemented in a highly efficient manner for most of the common heft functions. For example, the heft functions in the two examples can be evaluated in constant time by using $O(N)$ extra space to precompute appropriate prefix sums; see, e.g., [17] for details. Moreover, by performing an amount of preprocessing proportional to the number of non-zero entries, we can build standard range search data structures for very sparse inputs that allow queries to be efficiently evaluated (e.g., in logarithmic time for the MAX-SUM-ID metric in two dimensions).

For this reason, we introduce a term t_Q to represent the cost of a heft query, and state all running times in terms of t_Q. Thus, an implementor can decide what preprocessing to perform for maximum efficiency.

[1] We use the term *heft* to avoid confusion with the term *weight* used in our algorithms, which has a completely different meaning.

[2] In fact, in all applications we are aware of.

The algorithms typically run in time sublinear in the input size N if we ignore preprocessing, and thus we can afford to repeatedly run algorithms with different values of p and q, for example.

Problem Statement: We are interested in the following two dual optimization problems:

(1) *Given an input array A and numbers p and q, find a $p \times q$ partitioning P of A that minimizes $f(P)$.*

(2) *Given an input array A and a real value δ, find a $p \times q$ partitioning P with $f(P) \leq \delta$ that minimizes the number of tiles $p \cdot q$.*

As we will see, these problems are NP-Complete, at least for the most interesting classes of metrics, and thus we are interested in approximation results. In some applications, we may also be interested in minimizing the number of dividers rather than the number of tiles. All our upper bounds can be stated either way, with a t approximation in the dividers typically translating to a t^2 approximation in the number of tiles.

3 Contributions of this Paper

We study the algorithmic complexity of $p \times q$ partitioning problems. Our results include new approximation algorithms for several classes of quality metrics that improve significantly over previous results. In particular, we consider metrics where the heft function $h()$ is monotonic and the cumulative function $f()$ is the sum or the maximum function; this covers all applications we are currently aware of. We also extend known results about NP-Completeness to additional classes of metrics not covered by the known proof. Our main contributions are as follows:

(1) **Algorithms for $f() =$ MAX:** Given an array A and a bound δ, we can compute a $p \times q$ partitioning with value at most δ using a number of tiles that is at most $(4 + \epsilon)$ times the optimum number, for any $\epsilon > 0$. The best previous result [19] achieved an approximation ratio of $O(\log^2 N)$ in the number of tiles, or $O(\log N)$ in the number of dividers. Given an array A and a bound on the number of tiles, we can compute a partitioning with value at most 4 times the optimum value for the case of the MAX-SUM-ID metric. The best previous result [19] achieved an approximation ratio of around 120. In fact, if we allow slightly more tiles, then we can approximate the optimum value by a factor of at most 2. The algorithms run in time $O((m + n + p \cdot q \cdot t_Q) \cdot p \log(N))$.

(2) **Algorithms for $f() =$ SUM:** Given an array A and a bound δ, we can compute a $p \times q$ partitioning with value at most 2δ using a number of tiles that is at most $(16 + \epsilon)$ times the optimum number of tiles for a partitioning of value δ. The algorithm runs in time $O((m + n + p \cdot q \cdot t_Q) \cdot p \log(N))$. We are not aware of any previous non-trivial bounds for this problem.

(3) **Generalizations:** All the algorithms can be extended to higher dimensions, resulting in approximation ratios linear in the dimension d.

(4) **NP-Completeness results:** We show that it is NP-Complete to compute an optimal partitioning for the important SUM-SUM-SQRDEV metric in Example 2. We also extend the known result for the MAX-SUM-ID metric in Example 1 [8, 11] that it is NP-Complete to compute a partitioning with a value less than two times the optimum to several new metrics.

A preliminary version of the results in this paper appeared as part of [27]. Our algorithms are based on the framework of Brönnimann and Goodrich in [6], and exploit an interesting relationship between $p \times q$ partitioning problems and the construction of small ϵ-nets. In particular, the partitioning problem for the MAX-SUM-ID metric can be reduced to a Set Cover problem with small VC dimension. It is shown in [6] that such Set Cover problems can be efficiently approximated using ϵ-nets and an elegant analysis provided by Clarkson in [9]. We show how to translate and expand these ideas to obtain very simple and fast algorithms for the above general classes of metrics.

4 Related Work

Applications: Rectangular partitioning problems have been studied extensively in application areas including databases (e.g., histograms, grid files, index compression), parallel computing (e.g., load balancing), computer graphics (e.g., spatial data structures), and video compression (e.g., block matching); see [18, 23, 3, 5, 10] for some discussion. Following are a few examples that explicitly study $p \times q$ partitionings.

In databases, several authors [1, 12] have used $p \times q$ partitionings to construct histograms in two or more dimensions. In this case, the metric used is often either MAX-SUM-ID for *V-Optimal* histograms [31] or SUM-SUM-SQRDEV for *Equi-Depth* histograms [32]. In particular, the algorithm for updating a histogram in [1] uses the structure of $p \times q$ partitionings by running an algorithm "along the rows and columns" in a way that is slightly similar to our approach (though maybe not related in any formal way). Another scenario in databases is the construction of *Grid File* index structures for multi-dimensional data, introduced in [29], where the goal is to limit the number of items in each tile.

A significant amount of work on $p \times q$ partitionings has been performed in the area of parallel computing, often under the term *Generalized Block Distribution*. In particular, [22, 28] propose heuristic algorithms for $p \times q$ partitionings, discussed further below. The class of $p \times q$ partitionings is particularly important in the context of data-parallel programming languages and environments such as High Performance Fortran [14] and Vienna Fortran [7], and is in fact part of the HPF 2.0 language specification.

NP-Completeness: For the three-dimensional case, Nicol [28] proved that it is NP-Complete to find the best partitioning under the MAX-SUM-ID metric, using a reduction from *Monotonic 3-SAT*. Subsequently, [8, 11] independently proved the problem to be NP-Complete in two dimensions, based on a reduction from the *Balanced Complete Bipartite Subgraph* problem. In fact, the argument implies that given a bound on the number of tiles, the optimal value of the MAX-SUM-ID metric cannot be approximated to within less than a factor of 2. We are not aware of previous results for other types of metrics.

Algorithmic Results: Several authors [22, 24, 28] have independently proposed heuristic algorithms for the MAX-SUM-ID metric based on the following simple approach: Perform alternate scans along the different axes, and during each scan choose the best possible cuts along this axis given the current cuts along the other axes. The algorithm terminates when no changes in the partitioning occur anymore. The approach appears to perform well in many cases, and it is shown in [4] that one such algorithm approximates the minimum value of the metric within a factor of $O(\sqrt{p})$ for a $p \times p$ partitioning. This bound is also tight for this class of algorithms.

In [19], an algorithm is given that achieves a constant approximation factor of about 120 in the value of the metric. For the dual problem of minimizing the number of tiles, a reduction to *Set Cover* is given in [19] that achieves a $O(\log^2 N)$ factor approximation (or a $O(\log N)$ approximation of the number of dividers as stated in [19]). Almost all of the algorithms are proposed for the case of the MAX-SUM-ID metric, though some of the results also extend to other MAX metrics. We are not aware of any previous provable results for SUM metrics.

Techniques: Our algorithms are inspired by and closely related to the algorithms for Set Cover with bounded VC dimension in [6], which are based on the use of ϵ-nets [13]. The analysis of the algorithms uses an argument presented by Clarkson in [9] and is also similar to those in [21, 33].

5 Approximation Algorithms

In this section, we describe our algorithms, which are all based on the same basic approach. We first focus on the case of $f() = $ MAX, and later extend the result to $f() = $ SUM. As mentioned before, the algorithms are closely related to the framework for approximating Set Cover for set systems with bounded Vapnik-Chervonenkis (VC) dimension described by Brönnimann and Goodrich [6], and their analysis is based on an argument given by Clarkson in [9]. However, in order to get the best bounds on approximation ratio and running time, and to keep them as simple as possible, we give the algorithms and their analysis directly in the context of $p \times q$ partitionings rather than trying to cast them into the notation in [6]. A brief discussion of the relation of the algorithms to the results in [6] is given in Subsection 5.3.

5.1 Preliminaries

As before, we assume an $m \times n$ array A with $N = m \cdot n$, and with array coordinates indexed from 1. We denote by X^p the set of m rows, and by X^q the set of n columns, of the $m \times n$ array A. For each tile $r_{i,j}$ of a $p \times q$ partitioning H, we define a corresponding subset $R^p_{i,j}$ of X^p consisting of all rows that intersect $r_{i,j}$, but excluding the last intersecting row of the tile. Similarly, we define a subset $R^q_{i,j}$ of X^q consisting of all columns that intersect $r_{i,j}$, but excluding the last intersecting column. Recall that $p \geq q$.

We also use a weight function w, to be defined later, that assigns a real-valued weight $w(x)$ to each $x \in X^p \cup X^q$, and define $w(Y) = \sum_{y \in Y} w(y)$ for any subset Y of $X^p \cup X^q$. In particular, we define the row weight of a tile $r_{i,j}$ as $w(R^p_{i,j})$, and the column weight as $w(R^q_{i,j})$. We define the row weight of a partitioning H as $w(H^p)$, where $H^p = \{h_i \mid 0 < i \leq p\}$ is the set of horizontal dividers of H, excluding h_0. Similarly, we define the column weight of H as $w(H^q)$, where $H^q = \{v_j \mid 0 < j \leq q\}$ is the set of vertical dividers of H, excluding v_0. Note that these weights are different from the heft of a tile, or the value of a partitioning under a given quality metric.

Definition 5.1 *Given a weight function w, we say that a partitioning H is* (α_p, α_q)-good *if every tile $r_{i,j}$ of H satisfies $w(R^p_{i,j}) \leq \alpha_p \cdot w(X^p)$ and $w(R^q_{i,j}) \leq \alpha_q \cdot w(X^q)$.*

We remark that (α_p, α_q)-good partitionings correspond to the ϵ-nets used in [6] and originally introduced in [13], which have found many applications in approximation algorithms and computational geometry.

5.2 Approximating the Number of Tiles for MAX Metrics

We now present approximation results for the $p \times q$-partitioning problem where the cumulative function is MAX, i.e., the heft of a partition is the largest heft of any of the tiles. We also require the heft function to be monotonic, i.e., the heft of a tile does not decrease if we grow its size, which is true for most interesting cases.[3]

The Algorithm: Suppose that we are given a maximum value δ for the metric $f()$ and thus for the heft of each tile of the solution, and assume that for some p_0 and q_0 there exists a $p_0 \times q_0$ partitioning H_0 with value at most δ. (The values of p_0 and q_0 will be guessed using binary search.) We will show that the following very simple algorithm computes a $p \times q$ partitioning with heft at most δ, $p \leq \lceil (2+\epsilon) \cdot p_0 \rceil$, and $q \leq \lceil (2+\epsilon) \cdot q_0 \rceil$, for any chosen $\epsilon > 0$.

Algorithm *MAX-pxq*.

```
(1) Set the weights of all elements of $X^p \cup X^q$ to 1.
(2) Repeat the following three steps:
    (a) Compute an $(\alpha_p, \alpha_q)$-good $p \times q$ partitioning $H$, for $p = \lceil (2+\epsilon)p_0 \rceil$, $\alpha_p = 1/p$ and $q = \lceil (2+\epsilon)q_0 \rceil$, $\alpha_q = 1/q$.
    (b) Find a tile $r_{i,j}$ in $H$ such that $f(r_{i,j}) > \delta$. If none exists, terminate and return $H$ as solution.
    (c) Multiply the weights of all elements in $R^p_{i,j} \cup R^q_{i,j}$ by $\beta = (1 + \epsilon/2)$.
```

[3]An interesting exception is the case considered in Section 6 where $f() = $ MAX, $h() = $ MAX, and $e()$ is the difference between the value of the element and the average value in the tile.

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 number of tiles of the resulting (α_p, α_q)-good partitioning, (2) we bound the number of iterations in Step (2), and (3) we analyze the running time of each iteration. Theorem 1 then gives the main result of this subsection.

Lemma 5.1 *For any p and q, there exists a $(1/p, 1/q)$-good $p \times q$ partitioning.*

Proof: Simply set $p-1$ 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 $1/p \cdot w(X^p)$. Note that this divider is the last row of the newly defined row of tiles, and that the weight of this last row is not counted towards the weights of the tiles, according to our definition of $R^p_{i,j}$. Choose the vertical dividers v_i in an analogous fashion. ■

Lemma 5.2 *The loop in Step (2) of* MAX-pxq *terminates after $O(p \log N)$ iterations.*

Proof: The analysis is similar to that of Lemma 3.4 of [6], which itself is based on the argument in [9]. The basic idea is to upperbound the total weight $w(X^p)$ of all the rows, and to lowerbound the total weight $w(H^p_0)$ of the horizontal dividers of the (unknown) optimal partitioning H_0, during the course of the algorithm. Since H^p_0 is a subset of X^p, clearly $w(X^p) \geq w(H^p_0)$. This implies that the algorithm has to terminate before the lower bound for $w(H^p_0)$ becomes larger than the upper bound for $w(X^p)$; the same approach is used for the columns and vertical dividers.

Note that the weight $w(X^p)$ is initially m, 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 the weights of exactly one of the sets $R^p_{i,j}$ by a factor of $(1+\epsilon/2)$, and this set $R^p_{i,j}$ has a total weight of at most $\frac{1}{\lceil(2+\epsilon)\cdot p_0\rceil} \cdot w(X^p)$ due to the definition of (α_p, α_q)-goodness. Thus, after k iterations, we can upperbound $w(X^p)$ as follows:

$$w(X^p) = m\left(1 + \frac{\epsilon}{2\cdot(2+\epsilon)\cdot p_0}\right)^k \leq m \cdot \exp\left(\frac{\epsilon \cdot k}{2\cdot(2+\epsilon)\cdot p_0}\right) = \exp\left(\frac{\epsilon\cdot k}{2\cdot(2+\epsilon)\cdot p_0} + \ln(m)\right)$$

where $\exp()$ denotes the exponential function with basis e. Similarly, we get

$$w(X^q) \leq \exp\left(\frac{\epsilon \cdot k}{2\cdot(2+\epsilon)\cdot q_0} + \ln(n)\right).$$

Now consider the $p_0 \times q_0$ partitioning H_0 of heft at most δ that we assume to exist. 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 in each iteration at least one horizontal or one vertical divider of H_0 has its weight increased by a factor of $(1+\epsilon/2)$. Now suppose that after k iterations, we have had at least $k/2$ increases in the horizontal dividers; the other case is symmetric. Then we have

$$w(H^p_0) = \sum_{h_i \in H^p_0} (1+\epsilon/2)^{z_i}$$

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

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

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

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

which, using the inequality $\ln(1+\epsilon/2) > \frac{\epsilon}{2+\epsilon}$ for $0 < \epsilon < 1$, implies that

$$k \leq p_0 \cdot \frac{\ln(m/p_0)}{\ln(1+\epsilon/2) - \frac{\epsilon}{2+\epsilon}} = O(p \log N).$$

Thus, if the horizontal dividers were increased at least $k/2$ times in k iterations, then the process must terminate after at most $O(p \log N)$ iterations. Similarly, if the vertical dividers were increased at least $k/2$ times, then we get

$$k \leq q_0 \cdot \frac{\ln(n/q_0)}{\ln(1+\epsilon/2) - \frac{\epsilon}{2+\epsilon}} = O(q \log N).$$

■

Lemma 5.3 *Each iteration in Step (2) runs in time $O(n + m + p \cdot q \cdot t_Q)$, where t_Q is the time needed to compute the heft of a tile.*

Proof: If we assume that the weights $w(x)$ can be presented in a single machine word of $O(\log N)$ bits, then Steps (2a) and (2c) clearly run in time $O(n+m)$. In Step (2b), we have to compute the heft of at most $p \cdot q$ tiles to find a tile with heft more than δ, requiring time $p \cdot q \cdot t_Q$.

Note that the number of bits needed for the weights may increase linearly with the number of iterations. However, since the weights $w(x)$ are of the form β^i for some i, we can implement Step (2c) by simply increasing the exponent i of the weight. We can then obtain the actual weight by a lookup into a precomputed table of the β^i as needed. For Step (2a) it actually suffices to approximate the sum of the rows between two dividers to within a small constant factor; this results in an extra $(1+\epsilon')$ factor in the number of dividers, which can be absorbed into the current $(1+\epsilon)$ factor. This approximation can be easily achieved by using $O(\log n)$-bit floating point numbers for the β^i and being careful about the order of the additions. (In fact, discussion in [6] for a similar scenario suggests that this may not be much of a problem in practice.) ■

Theorem 1 *For any δ and any $\epsilon > 0$, a $p \times q$ partitioning H with heft at most δ, $p \leq (2+\epsilon)p_0$, and $q \leq (2+\epsilon)q_0$ can be computed in time $O((n + m + p \cdot q \cdot t_Q) \cdot p \log N)$, where $p_0 \cdot q_0$ is the minimum number of tiles 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.*

Proof: We perform binary search for the value of p_0, starting at $p_0 = 2$. For each choice of p_0 we perform a binary search for q_0, starting with $q_0 = 2$, and for each such pair (p_0, q_0), we run algorithm *MAX-pxq*. If the algorithm does not terminate after the number of iterations stated in Lemma 5.2, then we know that there is no $p_0 \times q_0$ partitioning with heft at most δ, and we increase q_0, and eventually p_0 once q_0 becomes larger than p_0, by some small factor $(1+\epsilon')$. The total running time is dominated by the time used to run *MAX-pxq* on the largest p_0 and q_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 1, and we get the following corollary. Note that, for the case of $p = O((N/\log N)^{1/3})$, this gives a linear time bound.

Corollary 1.1 *In the case of the MAX-SUM-ID metric, for any δ and any $\epsilon > 0$, a $p \times q$ partitioning H with heft at most δ, $p \leq (2+\epsilon)p_0$, and $q \leq (2+\epsilon)q_0$ can be computed in time $O(N + p^2 \cdot q \log N)$, where $p_0 \cdot q_0$ is the minimum number of tiles such that there exists a $p_0 \times q_0$ partitioning with heft at most δ.*

5.3 Relationship to Set Cover

We now discuss the relation of our algorithm to the work in [19] and [6]. 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 $O(\log N)$ for the value of p, and thus of $O(\log^2 N)$ for the minimum number of tiles, using the well known greedy algorithm for Set Cover. This bound can be improved to $O(\log^2 p)$ by using the algorithm for approximating Set Cover for the case of bounded VC-dimension in [6], 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 5.1, one can obtain an approximation ratio of 16 for p (and thus 256 for the number of tiles) and a running time of $O(N^{3/2} \cdot p \log N)$. In order to get near-linear running time, we have described 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 $(2 + \epsilon)$ for p (or $(4 + \epsilon)$ for the number of tiles) is then obtained by tightening the analysis of [6] in several places.

5.4 Approximating the Metric for MAX-SUM-ID

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 value of any $p \times q$ partitioning given p and q. 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 2 *Let δ_0 be the minimum heft of any $p_0 \times q_0$ partitioning under the MAX-SUM-ID metric. Then, in time $O(N + p^2 \cdot q \log N)$, we can compute*

(a) a $p \times q$ partitioning with heft $\delta \leq 4\delta_0$, $p \leq (\frac{2}{3} + \epsilon)p_0$, and $q \leq (\frac{2}{3} + \epsilon)q_0$, and

(b) a $p \times q$ partitioning with heft $\delta \leq 2\delta_0$, $p \leq (1 + \epsilon)p_0$, and $q \leq (1 + \epsilon)q_0$,

for any chosen $\epsilon > 0$.

Proof: These results are based on a fairly simple observation: If we modify Algorithm *MAX-pxq* such that in Step (2b) we search for a tile with heft more than $4\delta_0$, then we can conclude that the optimum solution H_0 must cut this tile at least 3 times in order to get a heft of at most δ_0. (Note that both horizontal and vertical cuts are counted.) This means that Step (2c) guarantees a larger increase in the weight of H_0 since at least three dividers of H_0 are increased. We can then adjust the choice of the parameters p, q, and α in the algorithm to $p = \lceil (2/3 + \epsilon)p_0 \rceil$, $q = \lceil (2/3 + \epsilon)q_0 \rceil$, and $\beta = (1 + 3\epsilon/2)$ to get the result.

Similarly, if we search for a tile with heft more than $2\delta_0$, then H_0 must cut this tile at least 2 times in order to get a heft of at most δ_0. In this case, we adjust the choice of the parameters to $p = \lceil (1 + \epsilon)p_0 \rceil$, $q = \lceil (1 + \epsilon)q_0 \rceil$, and $\beta = (1 + \epsilon)$ to get the result. We note that other choices of parameters, like $6\delta_0$, $p = \lceil (1/2 + \epsilon)p_0 \rceil$, and $q = \lceil (1/2 + \epsilon)q_0 \rceil$, are also possible. ■

Thus, these results come quite close to the limit of 2 on the approximability shown in [8, 11]. In fact, we can choose $\epsilon = 1/(2p_0)$ to get a weight of at most $2\delta_0$ with a $(p_0 + 1) \times (q_0 + 1)$ partitioning in part (b); this results in an additional factor of p^2 in the number of iterations of Algorithm *MAX-pxq*, and an additional factor of p in the running times of Steps (2a) and (2c) in each iteration. (In practice, the running times are probably not this bad.) While these result are limited to MAX-SUM-ID, the argument could potentially be adapted to other metrics, provided one can argue that a tile with heft by some multiple higher than δ requires a certain number of cuts to be brought down to δ. We note that this would for example not be the case for a metric such as MAX-SUM-SQRDEV (i.e., SUM-SUM-SQRDEV with $f() = \text{MAX}$) which illustrates one of the challenges in metrics involving variance within tiles.

5.5 Extensions to Higher Dimensions and Sparse Data

All results in this subsection can be very easily extended to the case of d dimensions, with the approximation factor growing with the dimension. For example, we get an approximation ratio of $d + \epsilon$ and a running time of $O((n + p^d \cdot t_Q) \cdot p \log N)$ for the result in Theorem 1. The modifications in the algorithm and proofs are straightforward. For example, in the case of an $p \times q \times r$ partitioning, we define X^p, X^q, and X^r, and $(\alpha_p, \alpha_q, \alpha_r)$-goodness. We choose $p = \lceil (3 + \epsilon)p_0 \rceil$, $q = \lceil (3 + \epsilon)q_0 \rceil$, and $r = \lceil (3 + \epsilon)r_0 \rceil$ in the algorithm. After k iterations in the proof of Lemma 5.2, the dividers along at least one of the three dimensions must have been increased at least $k/3$ times, allowing us to lower-bound the total weight of those dividers as before.

Input data in higher dimensions is often sparse, and thus we would like to implement the algorithm to run in time proportional to the number of non-zero elements rather than the total size of the array A. For MAX-SUM-ID and similar metrics, this can be done by inserting all non-zero elements into a standard geometric data structure for range search that can then efficiently answer heft queries; the details depend on the particular metric. For example, for MAX-SUM-ID, a cost of $O(\log^{d-1} t)$ per heft query, where t is the number of non-zero elements, can be obtained with a data structure based on range trees. The structure takes space $O(t \cdot \log^{d-1} t)$ and can be built in time $O(t \cdot \log^{d-1} t)$. We refer to Section 3 of [2] for an excellent survey of theoretical results and practical approaches to this problem. Additional optimizations could exploit the fact that the heft queries in each iteration have a regular structure corresponding to a $p \times q$ partitioning, and that we really only need to find one tile with heft more than δ in each iteration.

5.6 Upper Bounds for SUM Metrics

We now present approximation results for the case where the cumulative metric is the SUM metric. We again require the heft function to be 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 may be 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 $p_0 \times q_0$ partitioning H_0 with heft at most δ_0. Then the following algorithm computes a $p \times q$ partitioning with heft at most $2\delta_0$, $p \leq (4 + \epsilon) \cdot p_0$, and $q \leq (4 + \epsilon) \cdot q_0$, for any chosen $\epsilon > 0$.

Algorithm *SUM-pxq*.

(1) Set the weights of all elements of $X^p \cup X^q$ to 1.

(2) Repeat the following three steps:

  (a) Compute an (α_p, α_q)-good $p \times q$ partitioning H, for $p = \lceil (4 + \epsilon)p_0 \rceil$, $\alpha_p = 1/p$ and $q = \lceil (4 + \epsilon)q_0 \rceil$, $\alpha_q = 1/q$.

  (b) If the value 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 in $R^p_{i,j} \cup R^q_{i,j}$ by $\beta = (1 + \epsilon/2)$.

The following lemma provides the main insight underlying the analysis.

Lemma 5.4 *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 value 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 $w(H_0^p)$ and $w(H_0^q)$ as compared to the MAX case in the previous subsection. To deal with this weaker lower bound, we use the factor $(4+\epsilon)$ instead of $(2+\epsilon)$ in the number of dividers in Step (2a). The remainder of the analysis is then along the lines of the analysis for the MAX case, and we get the following result.

Theorem 3 *For any δ_0 and any $\epsilon > 0$, a $p \times q$ partitioning H with heft at most $2\delta_0$, $p \leq (4+\epsilon)p_0$, and $q \leq (4+\epsilon)q_0$ can be computed in expected time $O((n+m+p\cdot q\cdot t_Q)\cdot p\log N)$, where $p_0 \cdot q_0$ is the minimum number of tiles 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.*

Discussion and Extensions. We point out that the algorithm can be made deterministic by modifying Steps (2b) and (2c) as follows: Instead of choosing one particular tile at random, we compute for each "row of tiles" i the fraction w_i of the total value of the current partitioning that is due to the hefts in this row of tiles. We then multiply the horizontal dividers in each row of tiles i by $(1 + w_i \cdot \epsilon/2)$, and repeat the same thing for the "stacks of tiles" along the columns.

Note that this approach again gives a trade-off between the approximation of the error and the approximation of the number of dividers. For example, we can modify the algorithm to approximate the value of the partitioning by a factor of 1.5 and the number of dividers by $(6+\epsilon)$. As before, in many cases t_Q can be implemented in $O(1)$ steps by performing appropriate preprocessing, including the SUM-SUM-SQRDEV metric. The result also extends to higher dimensions as described before.

6 NP-Completeness Results

We now extend the known NP-Completeness results. For the special case of the MAX-SUM-ID metric, Charikar, Chekuri, Feder, and Motwani [8] and Grigni and Manne [11] have shown using very similar constructions that it is NP-Complete to approximate the minimum heft of any $p \times p$ partitioning to within a factor of less than 2. We extend this result to several other metrics using a construction very similar to those in [8, 11], with reductions from the following two NP-Complete problems. (We note that these two problems are closely related and can be easily reduced to each other as shown, e.g., in [11].)

- **Balanced Bipartite Vertex Cover (k-BBVC):** Given a bipartite graph G and an integer k, does there exist a vertex cover containing exactly k vertices from each side of the partition?
- **Balanced Complete Bipartite Subgraph (k-BCBS):** Given a bipartite graph G and an integer k, does G contain a bipartite clique containing exactly k vertices from each side of the partition?

Our first result concerns the following types of MAX metrics. Let DEVAVG and DEVGEO be the elementary functions defined as (the absolute amount of) the deviation between an element and the arithmetic average and geometric median, respectively, in its tile.[4] Also recall that SQRDEV is the square of DEVAVG, i.e., the elementary function in SUM-SUM-SQRDEV. Consider metrics with these elementary functions, with the heft function being either MAX or SUM, and with the cumulative function being MAX. As an example, if the heft function is MAX and the elementary function is DEVAVG then we are trying to find a partitioning that minimizes the maximum deviation of any element from the average in its tile. For these metrics we show the following.

[4]The geometric median of a set of numbers is defined as $\frac{max+min}{2}$ where max and min are the maximum and minimum elements of the set.

Theorem 4 *Given a data distribution A and a value for p, it is NP-Complete to approximate the minimum heft of any $p \times p$ partitioning within a factor*

(a) less than 2 for $e()$ = DEVAVG or $e()$ = DEVGEO, $h()$ = MAX, and $f()$ = MAX,

(b) less than 4 for $e()$ = SQRDEV, $h()$ = MAX, and $f()$ = MAX,

(c) less than 4 for $e()$ = DEVAVG or $e()$ = DEVGEO, $h()$ = SUM, and $f()$ = MAX, and

(b) less than 8 for $e()$ = SQRDEV, $h()$ = SUM, and $f()$ = MAX.

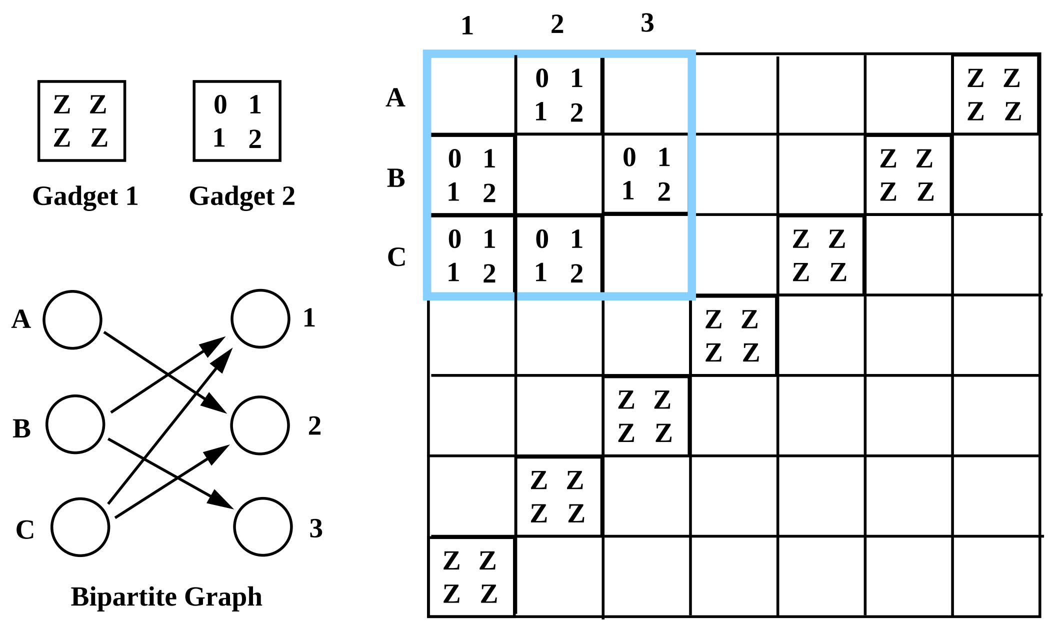

Figure 4: Reduction from the k-BBVC problem to a $p \times p$ partitioning problem, for a bipartite graph on 6 vertices. All elements not shown have a value of 0, and Z is some fairly large number.

Proof: We reduce the k-BBVC problem on a graph with l vertices on each side to a $p \times p$ partitioning problem with $p = k + 2l + 1$ on an $n \times n$ array with $n = 4l + 2$, as follows. As shown in Figure 4, we place $2l + 1$ copies of Gadget 1 along the diagonal, where Z is some fairly large number. Note that under all of the above metrics, this means that a good partitioning must cut each axis into $2l + 1$ intervals of size 2 in order to separate the Z values from the 0 values surrounding them, since any tile containing both a Z and a 0 would result in a very large value of the metric. Next, we use copies of Gadget 2, one for each edge, to construct an adjacency matrix for the bipartite graph in the upper left corner, of size $2l \times 2l$. At this point, shown in Figure 6, we can now choose k more cuts along each axis. Note that there exists a balanced bipartite vertex cover with k vertices on each side of the partition iff we can stab all copies of Gadget 2 using these $2k$ cuts.

Suppose $e()$ = DEVAVG or $e()$ = DEVGEO, $h()$ = MAX, and $f()$ = MAX. If we manage to stab all copies of Gadget 2 using these k cuts, then the maximum heft of any tile is at most $1/2$. If we cannot stab one of the copies of Gadget 2, then that copy is a tile with heft 1. The argument is essentially the same for the other metrics. Finally, it is straightforward to show that these problems are contained in the class NP. ■

We note that metrics with $e()$ = DEVAVG are not always monotonic and so our upper bounds do not apply in this case. For the other metrics, the above result shows that we cannot hope to generalize our results on approximating the value of the metric in Subsection 5.4 to arbitrary MAX metrics. In fact, it is not difficult to construct other, somewhat artificial, monotonic metrics for which no constant factor approximation is possible. Thus, the upper bound results for general MAX metrics in Subsection 5.2 are necessarily limited to approximations of the number of tiles, since the achievable approximation in the value of the metric depends on the particular metric.

For the result on the SUM-SUM-SQRDEV metric, we use a reduction from k-BCBS. The construction is almost identical, except that the value 2 in Gadget 2 is replaced by a 0. Again, Gadget 1 enforces cuts into intervals of size 2 along both axes. Note that if we cut the modified Gadget 2 once, there is no reduction in the value of the metric. However, if we manage to cut it twice, then there is a reduction of the value from 1 to 0 for that particular 2×2 box, and this is the only way to further reduce the value of the metric. Thus, using k cuts along each axis, we are able to cut k^2 copies of Gadget 2 twice iff there exists a k-BCBS subgraph in the input graph. This gives up the following result:

Theorem 5 *Given a data distribution A and a value for p, it is NP-Complete to compute a $p \times p$ partitioning that minimizes the SUM-SUM-SQRDEV metric.*

Note however that both the BBVC and BCBS problems are in polynomial time if we allow k_1 vertices on one side and k_2 vertices on the other side of the partition where $k = k_1 + k_2$. Thus, we have not shown the NP-Hardness of minimizing the total number of dividers for a given value of the metric in the case where we can allocate dividers to the axes as desired. Concerning the problem of minimizing the number of tiles $p \cdot q$, a simple argument from [11] can be used to show that this problem is NP-Hard even if we can allocate dividers to the axes as desired.

7 Concluding Remarks

In this paper, we have presented new algorithms for $p \times q$ partitionings that significantly improve upon previous results. We have also extended known NP-Completeness results to additional metrics.

The new algorithms are simple and fast and we believe that they should perform well in practice. For best results, we recommend that the new algorithms be combined with the previously described heuristics in [4, 22, 24, 28]. These heuristics start from an initial configuration and attempt to greedily improve the partitioning; thus, by using the output of our algorithms as initial configuration we are guaranteed to get partitionings that are at least as good as our algorithms. In fact, intuitively we expect that our algorithms, while achieving provable bounds, are probably not good at picking “exactly the right cut” (i.e., recognizing the precise position of ridges or edges in the data distributions) and thus a local heuristic might result in significant improvements. An experimental evaluation of these issues remains to be done.

There are also several theoretical open problems. For MAX metrics, there are still small constant factor gaps between our upper bounds and the inapproximability results, but it seems our analysis cannot be strengthened enough to narrow these gaps. For SUM metrics, we have only proven a somewhat awkward result that approximates both the number of partitions and the value of the metric. This is due to the very different nature of these metrics, for which combining two tiles can result in a dramatically higher value of the metric.

Another open problem is to show an inapproximability result for the SUM-SUM-SQRDEV metric. We currently can only show simple NP-Completeness but would intuitively expect it to be at least as difficult to approximate SUM-SUM-SQRDEV as MAX-SUM-ID. It would also be interesting to show inapproximability results for the number of tiles, or stronger inapproximability results for higher dimensions.

Finally, it would be of interest to investigate how the approach in this paper can be adapted to the incremental maintenance of partitionings, or the incremental construction of partitionings such as in the *self-tuning histograms* in [1]. In particular, the algorithms in [1] also use only a single query operation to access the input distribution, though in their case the queries are determined by an outside user access pattern. There are other similarities between our algorithms and those in [1], which are based on gradient descent methods in machine learning [25].

Acknowledgements

We thank Vishy Poosala for collaboration and many discussions in the early phases of this work.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proc. of the ACM SIGMOD Conference*, pages 181–192, 1999.

[2] P. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*. *Contemporary Mathematics*, 223, 1–56, AMS Press, 1999.

[3] S. Anily and A. Federgruen. Structured partitioning problems. *Operations Research*, 13, 130–149, 1991.

[4] B. Aspvall, M. Halldorsson, and F. Manne. Approximations for the general block distribution of a matrix. *Theoretical Computer Science*, 262(1), 145–160, 2001.

[5] S. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, 37, 38–57, 1988.

[6] H. Brönnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14, 463–479, 1995.

[7] B. M. Chapman, P. Mehrotra, and H. P. Zima. Programming in Vienna Fortran. *Scientific Programming*, 1(1), 1992.

[8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Personal communication, 1996.

[9] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. *Journal of the ACM*, 42(2), 488–499, 1995.

[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. In *Proc. of 3rd Int. Workshop on Parallel Algorithms for Irregularly Structured Problems*, Springer LNCS 1117, pages 319–326, 1996.

[12] D. Gunopulos, G. Kollios, V. J. Tsotras, C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *Proc. of the ACM SIGMOD Conference*, pages 463–474, 2000.

[13] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete and Computational Geometry*, 2, 127–151, 1987.

[14] High Performance Fortran Forum. *High Performance Fortran language specification*, Version 2.0, January 1997.

[15] Y. Ioannidis. Universality of serial histograms. In *Proc. of the 19th Int. Conf. on Very Large Data Bases*, pages 256–267, 1993.

[16] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proc. of the ACM SIGMOD Conference*, pages 233–244, 1995.

[17] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. of the 24rd Int. Conf. on Very Large Data Bases*, pages 275–286, 1998.

[18] M. Kaddoura, S. Ranka, and A. Wang. Array decomposition for nonuniform computational environments. *Technical Report*, Syracuse University, 1995.

[19] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *Proc. Int. Colloquium on Automata, Languages, and Programming*, pages 616–626, 1997.

[20] R. P. Kooi. *The Optimization of Queries in Relational Databases*. Ph.D. Thesis, Case Western Reserve University, Sept 1980.

[21] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 285–318, 1988.

[22] F. Manne and T. Sorevik. Partitioning an array onto a mesh of processors. In *Proc. of the Workshop on Applied Parallel Computing in Industrial Problems*, 1996.

[23] C. Manning. *Introduction to Digital Video Coding and Block Matching Algorithms.* http://atlantis.ucc.ie/dvideo/dv.html.

[24] A. Mingozzi, S. Ricciardelli, and M. Spadoni. Partitioning a matrix to minimize the maximum cost. Discrete Applied Mathematics, 62, pages 221–248, 1995.

[25] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[26] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proc. of the ACM SIGMOD Conference*, pages 28–36, 1988.

[27] S. Muthukrishnan, V. Poosala and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. In *Proc. of the Int. Conf. on Database Theory*, pages 236–256, 1999.

[28] D. M. Nicol. Rectilinear partitioning of irregular data parallel computations. *Journal of Parallel and Distributed Computing*, 23(2), 119–134, 1994.

[29] 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, 1984.

[30] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. of the ACM SIGMOD Conference*, pages 294–305, 1996.

[31] V. Poosala. *Histogram-Based Estimation Techniques in Databases*. Ph.D. Thesis, Univ. of Wisconsin-Madison, 1997.

[32] G. P. Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. of the ACM SIGMOD Conference*, pages 256–276, 1984.

[33] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. of the 4th Annual Symp. on Computational Geometry*, pages 23–33, 1988.