On Approximating Rectangle Tiling and Packing

Sanjeev Khanna* S. Muthukrishnan† Mike Paterson‡

Abstract

Our study of tiling and packing with rectangles in two-dimensional regions is strongly motivated by applications in database mining, histogram-based estimation of query sizes, data partitioning, and motion estimation in video compression by block matching, among others.

An example of the problems that we tackle is the following: given an $n \times n$ array A of positive numbers, find a tiling using at most p rectangles (that is, no two rectangles must overlap, and each array element must fall within some rectangle) that minimizes the maximum weight of any rectangle; here the *weight* of a rectangle is the sum of the array elements that fall within it. If the array A were one-dimensional, this problem could be easily solved by dynamic programming. We prove that in the two-dimensional case it is NP-hard to approximate this problem to within a factor of 1.25. On the other hand, we provide a near-linear time algorithm that returns a solution at most 2.5 times the optimal.

Other rectangle tiling and packing problems that we study have similar properties: while it is easy to solve them optimally in one dimension, the two-dimensional versions become NP-hard. We design efficient approximation algorithms for these problems.

1 Introduction

For motivation we begin with a simple example involving two-dimensional histograms.

Example. Cost-based query optimizers rely on estimating the result sizes of operators (e.g., natural join, selection) to evaluate complex query execution plans. Some statistics on the underlying data are used to efficiently *estimate* the approximate result size of operators. The popular method is to approximate the data distribution using *histograms*, that is, to partition the data into ranges and approximate each region by an appropriate "average" value: the fewer the data ranges, the faster is the estimation, and the better the accuracy in estimation. An example is the classical *equi-depth histogram* for two numerical attributes of a database. Consider the rectangular region indexed by the domain of these two attributes, and which stores the frequency of tuples in the database with each combination of attribute values. The equi-depth histogram is a partition (without overlap) of this two-dimensional space into a prespecified number of rectangular tiles of nearly equal sums of frequencies. This is an instance of the rectangle tiling problem that we consider here.

Other applications for rectangle tiling and packing include data partitioning, load balancing in parallel scientific applications, motion estimation in videos by block matching, and database mining applications among others. All our problems are natural extensions to two dimensions of problems well studied and thoroughly understood on one-dimensional arrays. From an algorithmic point of view, all our problems would have polynomial time optimal solutions by dynamic programming in one-dimension. For two-dimensional regions, we show that these problems become NP-hard and sometimes are NP-hard even to approximate to within certain constant factors. Our main results are highly efficient (often linear time) algorithms with provably small approximation ratios for these problems. We state our problems precisely in Section 1.1, give a brief overview of application areas where they occur (Section 1.2), and then describe our results in detail (Section 1.3).

1.1 Problems.

The three problems `RTILE`, `DRTILE`, and `RPACK`, that are the primary focus of this extended abstract are described next; a few others are given in Section 5.

(a) Tiling Problems. Given an $n \times n$ array[1] A of non-negative numbers, partition it into at most p rectangular subarrays (that is, the rectangles cover A without overlap) such that the maximum weight of any rectangle is minimized; here, the *weight* of a rectangle is the sum of all the array elements which fall within it. We call this the *rectangle tiling* (`RTILE`) problem. We also consider its dual, namely, to find a tiling as above of an $n \times n$ array A in which every rectangle has weight at most W and which minimizes the number of rectangles. We call this the *dual rectangle tiling* (`DRTILE`) problem.

(b) Packing Problem. Given l weighted axis-parallel rectangles in an $n \times n$ grid, find a disjoint subset of at

*Math. Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ. sanjeev@research.bell-labs.com

†Information Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ. muthu@research.bell-labs.com

‡Dept. of Computer Science, Univ. of Warwick, Coventry, CV4 7AL, UK. msp@dcs.warwick.ac.uk. This work was partly supported by ESPRIT LTR Project no. 20244 — ALCOM-IT.

[1] All our results extend easily to $n \times m$ arrays.

most k rectangles of largest total weight.[2] We call this the *rectangle packing* (`RPACK`) problem.[3]

Some remarks about our problems. The tilings that we allow in `RTILE` and `DRTILE` are arbitrary. For restricted cases of tilings, there are known results. For examples, see [MM+96] for quad-tree tilings, and see [KMS97] for tiling with p vertical and q horizontal lines to obtain pq blocks. The `RTILE` problem in one dimension has a rich history as surveyed in [KMS97]; a dynamic programming solution of $O(np)$ time is known and the best known general bound is $O(\min\{n+p^{1+\epsilon}, n\log n\})$. The `DRTILE` problem in one dimension can be solved optimally by a simple greedy strategy in linear time.

The `RPACK` problem in two dimensions is very general; it captures many interesting problems as special cases. One that we consider allows *all* axis-parallel subrectangles of a rectangular array; the weight of a rectangle is the sum of array elements in it — the array elements are allowed to take negative values. The one-dimensional version of this is a generalization of Bentley's Maximum Subarray Sum to k regions; in Bentley's problem, one must find the interval of a one-dimensional array with largest sum [Be84]. We focus on the general two-dimensional `RPACK` problem in this abstract; a special case is discussed in Section 5. We point out that two-dimensional bin packing problems have been studied in the literature (e.g., [KR96]), but they differ significantly from our problems: algorithms for bin packing have the flexibility to move rectangles around in the space whereas in our case all rectangles are fixed in space. Our problems are similar in spirit to geometric covering and packing studied by Hochbaum et al (e.g., [H97, HW85]). The problem they study which is most relevant to our work is a special case of the `DRTILE` problem when all the given objects are squares of identical sizes. The authors obtain polynomial time approximation schemes for this special case of our problem.

1.2 Application Scenarios.

The problems of tiling and packing arrays with rectangles arise naturally in a number of application areas. We describe some scenarios where these problems are found exactly in the form we study here.

Two-dimensional histograms. As mentioned earlier, this problem arises in building histograms over two attributes. A histogram is an approximate compact statistic of the database contents and it is used for estimating result sizes of operators. Although other such estimators exist (such as sampling [LNS90] and parametric techniques [SL+93]), histograms are the most commonly used form of statistics in practice and they are used in DB2, Informix, Ingres, Oracle, Sybase, etc. (See page 29 of [P97] for a list and more information.) Using histograms over a single numerical attribute has been extensively researched in the database community, and a number of histograms have been identified: *MaxDiff*, *V-Optimal*, *Equi-depth*, etc. (See [P97] for a fine taxonomy, and [I93, IP95] for examples of their accuracy and optimality.) More recently, there has been an effort to capture the *joint frequency distributions* in the database, and histograms over multiple numerical attributes are being studied. This was initiated in [MD88] and recent work is found in [PI97, P97]. Both the `RTILE` and `DRTILE` problems are instances of constructing what are known as equi-depth histograms on two numerical attributes. Intuitively, the maximum weight of a tile controls the accuracy of estimation and the number of tiles affects the efficiency of estimation. Our two tiling problems respectively optimize one of these parameters while keeping the other bounded.

Data partitioning. For data stored in two-dimensional arrays, high-performance computing languages allow the user to specify a partitioning and distribution of data onto a logical set of processors. The goal is to balance the load. See [FJ+88, M93] for technical discussions. Examples of languages which support such data partitioning include the Superb environment [ZBG86] and High Performance Fortran HPF2 [HPF]. The general version of this problem with p processors is precisely the `RTILE` problem.

Database Mining. Discovering *association rules* in databases, introduced in [AIS93], is a promising approach to mining. Association rules are of the form $\mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are predicates over a set of attributes. Such an association rule means that the tuples that satisfy $\mathcal{X}$ tend to satisfy $\mathcal{Y}$. We are interested in finding *optimized-gain* rules, that is ones where the number of tuples that both $\mathcal{X}$ and $\mathcal{Y}$ satisfy exceeds a certain percentage of the number of tuples that satisfy $\mathcal{X}$.[4] The `RPACK` problem is that of finding *k clustered two-dimensional optimized-gain* rules [LSW97, FM+97][5]. It

[2] This is a packing problem with the additional constraint that at most k items be packed.

[3] The dual of the `RPACK` problem is to find a minimum cardinality subset of disjoint rectangles with total weight at least W. Since it is NP-hard to find even a feasible solution for this problem (see [FPT81]), it cannot be approximated to any factor and hence is not considered any further.

[4] There are optimized-confidence and optimized-support rules as well; they will not be discussed here.

[5] An example: (Balance $\in [10^4, 10^5]$) AND (Age $\in [40, 50]$) $\rightarrow$ (Carloan=yes). Here the ranges in Balance and Age make the association rule clustered.

is clustered since we look for rectangular regions. The k regions are disjoint since presumably the goal is to target the k "favorable" regions for marketing, mail-order or advertising purposes, where each region will be targeted differently or by different departments within a company. Finally, in our definition of the RPACK problem, we have considered the general case of practical interest when only a subset of all the induced rectangles from the two-dimensional domain (of the two numerical attributes) may be of interest for mining. See [LSW97, FM+97] for more on this application.

1.3 Results and Techniques.

We summarize our results for each of the three problems.

1. **The RTILE problem.** We show that it is NP-hard not only to compute the optimum solution but even to approximate it to within a factor of $5/4 = 1.25$. The hardness result holds even when each array element is an integer in the range $[1, 4]$. On the other hand, our main result is an $O(n^2 + p \log n)$ (near-linear time) algorithm for this problem with an approximation ratio of 2.5, which holds even in the presence of large array elements. Observe that a running time of $O(n^2 + p \log n)$ is near-linear because the input is of size $O(n^2)$ and $p = O(n^2)$.

   Let the total sum of all the items in array A be W. A lower bound of W/p on the maximum weight is immediate, and if there are no large elements in the array our algorithm returns a tiling with maximum weight $O(W/p)$. We use the natural approach of partitioning the problem into two disjoint parts and assigning each part a proportionate number of rectangles for tiling, but the potential presence of large elements complicates this strategy.

2. **The DRTILE Problem**. Since this is the dual of the RTILE problem, it follows that it is NP-hard to solve it exactly. We show a series of approximation algorithms trading off running time versus the quality of approximation. The best approximation we achieve is a factor of 2 with high polynomial running time. At the other end of the spectrum, we have a near-linear time algorithm with a larger constant approximation factor.

   The analysis of our approximation algorithm for the RTILE problem proves that a strong relationship exists between approximate solutions to these two dual problems. This relationship forms the basis of our efficient $O(1)$-approximation for the DRTILE problem. Our alternative algorithms are based on binary space partitioning (BSP) [FKN80], and use the fact that an arbitrary tiling can be converted into a *structured hierarchical* one with a relatively small increase in the number of rectangles. While a straightforward BSP-based dynamic programming can be used to get a 4-approximation, a more sophisticated dynamic programming yields a 2-approximation, albeit at the expense of a very high running time.

   REMARK 1. *Our analysis of the* RTILE *approximation algorithm establishes that the solution value changes only by a constant factor when the total number of tiles used is changed by a constant factor. This strong relationship enables a BSP-based approach to get a constant-factor approximation to the* RTILE *problem. Without such a relationship, the BSP approach would not be applicable since the number of tiles allowed is fixed.*

3. **The RPACK problem.** We show that the RPACK problem is NP-hard even in the restrictive setting where each rectangle in the input intersects at most three other rectangles. We design an $O(\log n)$-approximation algorithm for the general problem that runs in time $O(n^2 p \log n)$. The main idea underlying the algorithm is a decomposition into constant-width problems which we show can be well-approximated quite efficiently. While the decomposition scheme itself is similar to the PTAS paradigm used by Baker [B83], it requires a partition of the input into disjoint classes of geometrically "well-structured" rectangles such that each class is amenable to our decomposition strategy. The geometry of two dimensions leads to $O(\log n)$ disjoint classes and the algorithm chooses the best solution among all the classes.

   Even the special case of the RPACK problem that allows all axis-parallel subrectangles in an array can be shown to be NP-hard. On the other hand, a BSP-based dynamic programming approach gives a 2-approximation for this problem.

1.4 General Comments on Our Results.

Our rectangle packing and tiling problems seem quite natural; we expect them to find applications in scenarios besides the ones we have described here. Most of our algorithms are simple and easy to implement. Our algorithms for the RTILE and DRTILE problems are variations of what practitioners would normally implement, namely, hierarchical partitioning schemes. The key insight our algorithms for these problems can supply to practitioners is that even simple heuristics can give provably good performance guarantees.

1.5 Map.

Section 2 shows the hardness result and the approximation algorithms for the RTILE problem. In Section 3, we present our results for the dual DRTILE problem. Section 4 sketches the hardness of the RPACK problem and gives our $O(\log n)$-approximation algorithm. Finally, in Section 5 we briefly describe other related problems and our results for them.

2 The RTILE Problem

2.1 Hardness of Approximation.

We begin with a proof of the APX-hardness of the RTILE problem.

THEOREM 2.1. *The* RTILE *problem is NP-hard, even in the case where the element weights are non-negative integers bounded by a constant. Furthermore, it is NP-hard to determine the optimal value to within a factor of* 5/4.

Proof. The proof reduces PLANAR-3SAT to RTILE. An instance of PLANAR-3SAT is a 3CNF formula with the extra property that the following graph G_F is planar. The bipartite graph G_F has the variables and clauses of F as its two vertex sets, and an edge corresponding to each occurrence of a variable or its negation in a clause. It was shown in [L82] that the problem PLANAR-3SAT is NP-complete.

For any formula F in 3CNF for which G_F is planar we construct an array A_F of integers in the range $[1, 4]$, and an integer p_F, which gives an instance $(A_F, p_F, 4)$ of RTILE such that F is satisfiable if and only if A_F can be tiled by p_F tiles of weight at most 4. For each variable in F, we construct a closed loop of orthogonally adjacent elements, mostly of value 2. The loops are disjoint but, for each clause C, the three loops corresponding to the variables occurring in C approach closely together in a special *clause gadget* which we describe below. Because G_F is a planar graph there are no topological obstructions to this construction.

Figure 1 shows a portion of the array A_F. The heavy lines are for the explanation only. They mark out three paths, each corresponding to part of a variable loop. For the path on the left it is clear that any tile of weight 4 can cover at most two elements of the path. Since the target number p_F will require this to be achieved, there are just two choices, depending on whether odd or even pairs are to be covered. The bottom-right path shows a minor variation where the sequence 2112 occurs. Here the two 1's must be tiled with one or other of the 2's. This device allows us freedom to alter the relative parity of the *physical* layout of any path.

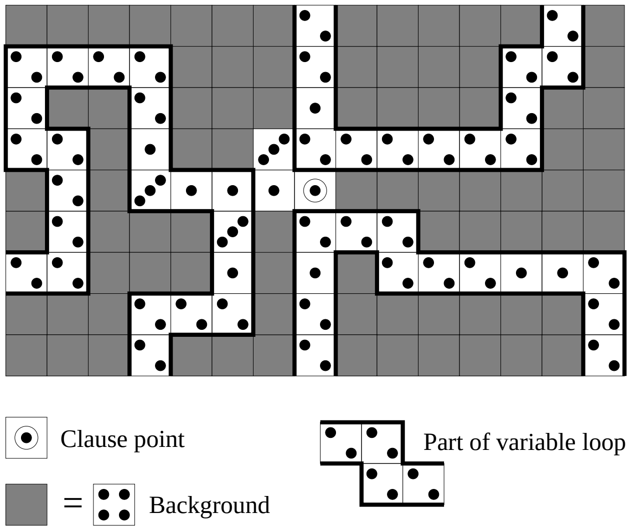

Figure 1: Clause gadget and variable loops

This portion of A_F illustrates the modeling of a clause gadget. The element marked with a circle has value 1. It can be checked that this can be tiled by just extending one of the tiles of the tiling for a variable loop, *provided that* this tiling has the appropriate parity. If none of the three loops has the right parity then no 4-tiling is possible, without using an extra tile.

If F has a satisfying assignment then, with the corresponding choices of parity for each variable loop, each clause point may be tiled by extending one of the tiles in a variable loop. If F is not satisfiable then any choice of parity for the variable loops leaves at least one clause point uncovered. Any tiling with p_F tiles which is not of the form discussed above has at least one tile with weight 5 or more. ∎

2.2 Approximation Algorithm.

In the remainder of this section, we focus on designing an efficient constant-factor approximation algorithm for the RTILE problem. If W_0 denotes the total weight of the given array A_0, p denotes the number of tiles, and M the maximum weight of any element in the array, then $\max\{W_0/p, M\}$ is clearly a lower bound for the minimum weight in a tiling. As we shall see, it is always possible to find a tiling which is within a constant factor of this simple lower bound.

Our algorithm uses a straightforward recursive partitioning strategy. Let $w = W_0/p$ and $M' = \max\{w, M\}$, so w is the average weight of a tile and M' is the simple lower bound on the maximum weight.

DEFINITION 1. (Thick Cut) *For any rectangular array, consider partitions into two subarrays defined by vertical*

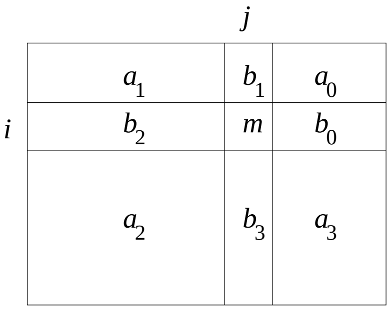

Figure 2: Subarray with no thick cut

or horizontal lines. Such a partition for which each subarray has weight at least w is a thick cut*.*

THEOREM 2.2. *There exists a 5/2-approximation algorithm for the* RTILE *problem.*

Proof. The algorithm begins by simply using thick cuts repeatedly to partition the array into subarrays. When no further thick cuts are possible we analyse the "end-game" and complete the tiling. Provided that we use at most $\lfloor W/w \rfloor$ tiles for a subarray of weight W, the total number of tiles used will be at most p.

Let A be a subarray, of weight W say, remaining after the partitioning, so A has no thick cut and the weight of A is at least w. Let i be the largest row index of A such that the total weight of rows of index less than i is less than w. It must be that the total weight of rows of index greater than i (if any) is also less than w, otherwise there would be a thick cut. Define the column index j similarly. We can depict A as in Figure 2, where the a_r's and b_r's denote the total weights of the corresponding regions and m is the value of $A_{i,j}$.

If $W < 5w/2$ then using just a single tile for A is within the 5/2-approximation bound, since $5w/2 \leq 5M'/2$.

Otherwise, we may assume $W \geq 5w/2$ and so at least two tiles can be allocated to A. There are four obvious ways to partition A into two rectangles with a cut. Provided that $b_r + a_r + b_{r+1} + a_{r+1} + b_{r+2} + m < 5M'/2$ for at least one value of r ($0 \leq r \leq 3$ and subscripts are taken modulo 4), we have a tiling with two tiles of maximum weight less than the 5/2-approximation bound.

Otherwise, we have $b_r + a_r + b_{r+1} + a_{r+1} + b_{r+2} + m \geq 5M'/2$ for $0 \leq r \leq 3$, and adding these four inequalities yields

$$2(a_0 + a_1 + a_2 + a_3) + 3(b_0 + b_1 + b_2 + b_3) + 4m \geq 10M'.$$

Hence,

$$\begin{aligned} 3W &= 3(a_0 + a_1 + a_2 + a_3 + b_0 + b_1 + b_2 + b_3 + m) \\ &\geq 10M' - m \geq 9M', \end{aligned}$$

and so $W \geq 3M'$. In this case at least three tiles can be allocated to A.

There is an obvious way to partition A with two parallel cuts into three rectangles of maximum weight less than $5M'/2$ unless $b_0 + b_2 + m \geq 5M'/2$ and $b_1 + b_3 + m \geq 5M'/2$. In this case

$$W \geq b_0 + b_1 + b_2 + b_3 + m \geq 5M' - m \geq 4M'.$$

This case then allows four tiles for A.

In a similar way, there is a 4-tiling with maximum weight less than $5M'/2$ unless $b_r + m \geq 5M'/2$ for all r. The latter case implies

$$W \geq b_0 + b_1 + b_2 + b_3 + m \geq 10M' - 3m \geq 7M'.$$

It is clear though that if at least five tiles can be used, there is a tiling with maximum weight less than M'. ∎

The following theorem shows an $O(n^2 + p \log n)$ bound on the time complexity of the algorithm outlined in the previous theorem.

THEOREM 2.3. *The algorithm of Theorem 2.2 can be implemented to run in time $O(n^2 + p \log n)$.*

Proof. Assume access to an oracle that, given the end-points of any rectangular region in the array, gives in constant time the total weight of all its elements. Then we claim that the initial partitioning can be created in $O(p \log n)$ time. To see this, first observe that each partition step can be done in $O(\log n)$ time since finding a thick cut only requires a binary search which can be done in $O(\log n)$ oracle calls. Next we claim that the final tiling can be done in time $O(\log n)$ per subarray; this follows since these partitions can be determined by a straightforward binary search. Thus all the steps associated with the algorithm can be performed in $O(p \log n)$ time. Finally, the constant time oracle used above can be implemented by an $O(n^2)$ time pre-processing step using standard arguments. Let $A = \{a_{ij}\}_{1 \leq i,j \leq n}$ be the input array. For $1 \leq k, l \leq n$, define $S(k,l) = \sum_{1 \leq j \leq l} a_{kj}$. Using the prefix sum algorithm, these sums can be implicitly computed after $O(n)$ time preprocessing for each row in the given array. For $1 \leq k, l \leq n$, define $T(k,l) = \sum_{1 \leq i \leq k} S(i,l)$. It is easy to see that the table of $S()$ entries can be used to compute all $T(k,l)$ entries in $O(n^2)$ time (just compute them left to right, and top to bottom). The final step is to show that the array T allows us to implement the desired oracle in constant time. Suppose we want the sum of the elements contained in the rectangular region determined by (i_1, j_1) (top left corner) and (i_2, j_2) (bottom right corner). This sum is given by $T(i_2, j_2) + T(i_1, j_1) - T(i_1, j_2) - T(i_2, j_1)$. The claimed result now follows. ∎

The proof of the following theorem is based on the ideas already described, and hence is omitted.

THEOREM 2.4. *There exists an $O(n^2 + p \log n)$ time algorithm which gives a 9/4-approximation for the* RTILE *problem on $\{0, 1\}$-arrays.*

3 The DRTILE Problem

The following theorem is an immediate corollary of Theorem 2.1.

THEOREM 3.1. *The* DRTILE *problem is NP-hard, even in the case where the element weights are non-negative integers bounded by a constant.*

We design three different approximation algorithms for this problem. While each one gives a constant-factor approximation, they reflect a tradeoff between the time complexity and the quality of the approximation guarantee achieved. The first two algorithms use dynamic programming, while the third algorithm is based on the partitioning approach and uses the strong relation that exists between the average tile weight and the maximum tile weight in an optimal solution, as shown in the analysis of the RTILE problem.

3.1 Hierarchical Binary Tilings.

We begin by formally defining a nicely structured class of tilings that underlies the dynamic programming approach that we use in this section.

DEFINITION 2. (Hierarchical Binary Tiling) *A tiling of a region is called a* hierarchical binary tiling (HBT) *if it consists of a single tile, or there exists a vertical or horizontal line such that the line does not intersect the interior of any rectangle, and the two distinct regions on either side are also hierarchical binary tilings.*

The *hierarchical binary W-tiling problem* is to find an HBT such that the maximum weight of any rectangle is at most W and the number of rectangles is a minimum. We show how to calculate the minimum number r of rectangles in any such partition using dynamic programming; a solution, namely an HBT with r rectangles none of whose weights is more than W, can be found easily from the dynamic program. For any region, we calculate the minimum over all possible horizontal and vertical lines of the number of rectangles used in the two regions obtained by splitting with that line. At the base level, any region which has total weight at most W can be tiled with a single rectangle. The total running time of this algorithm is bounded by $O(n^5)$, since there are $O(n^4)$ subrectangles and, for each subrectangle, the problem can be solved in $O(n)$ time using previously computed solutions and the recursive definition.

That lets us conclude:

LEMMA 3.1. *The hierarchical binary W-tiling problem can be solved in $O(n^5)$ time.*

Now we relate the hierarchical binary W-tiling problem to the DRTILE problem.

LEMMA 3.2. *If an array has a solution to* DRTILE *with p rectangles, then there is a hierarchical binary W-tiling with at most $4p$ rectangles.*

Proof. The proof follows from the results in [dAF92] for constructing binary space partitions (BSPs) for axis-parallel rectangles in two dimensions. Given a two-dimensional region with n axis-parallel rectangles, an *orthogonal binary space partition* (OBSP) recursively partitions the region into two disjoint regions using a vertical or a horizontal line, repeating this process until each region intersects at most one of the rectangles. If a rectangle is intersected by a line, then it is split into two disjoint rectangles given by the partitioning. It is clear from the definitions that an OBSP of rectangles gives a hierarchical binary tiling. In [dAF92], it is shown that there exists an OBSP in which each original rectangle is split into at most four parts. Thus, there exists a hierarchical binary tiling of at most $4p$ rectangles, and the lemma directly follows. ∎

Combining the two preceding lemmas, we have the following theorem:

THEOREM 3.2. *There exists a $O(n^5)$-time 4-approximation algorithm for the* DRTILE *problem.*

Proof. The proof follows from the observation that any tiling with p rectangles can be converted to a hierarchical binary tiling of at most $4p$ rectangles by using the results of [dAF92] on binary space partitions. Since an optimal hierarchical binary W-tiling can be found in $O(n^5)$ time, the theorem follows. ∎

We next show that working with a relaxation of the notion of HBTs can lead to a better approximation at the expense of significantly increased time complexity. The relaxation is referred to as a *two-hierarchical binary tiling* and is defined as follows:

DEFINITION 3. (Two-Hierarchical Binary Tiling) *A tiling of a region is called a* two-hierarchical binary tiling

(2HBT) *if it consists of a single tile, or there exists a vertical or horizontal line such that the line intersects the interior of at most two rectangles, situated at either end, and the two distinct regions on either side are also two-hierarchical binary tilings.*

THEOREM 3.3. *There exists an $n^{O(1)}$ time 2-approximation algorithm for the* DRTILE *problem.*

Proof Sketch. As before, let the W-2HBT problem be to find a 2HBT with maximum weight at most W with the minimum number of rectangles. It can be shown that the W-2HBT problem can be solved by dynamic programming in $n^{O(1)}$ time; the details are omitted. On the other hand, given n axis-parallel rectangles in two dimensions, the BSP of [dAF92] can be shown to produce a 2HBT of size at most $2n$. Combining the two, we get the claimed result. ∎

Unfortunately, the $O(1)$ in the exponent is very high, so this is not a practical algorithm.

3.2 Partitioning Approach.

While the dynamic programming approach taken above leads to strong performance guarantees, it has a rather high time complexity even for the 4-approximation. We now sketch how the partitioning approach for the RTILE problem in the previous section can be used to obtain a near-linear time approximation algorithm for the DRTILE problem.

Let p^* be the number of rectangles in the optimum solution and W_0 be the total weight of the array. Clearly, $p^* \geq W_0/W$ and $W \geq M$ where M is the largest entry in the array. We first solve the RTILE problem with $p = \lceil W_0/W \rceil$. From the analysis in Theorem 2.2, we know that the weight of each resulting rectangle is at most $2.5 \max\{W_0/p, M\}$ (observe that the solution is again a HBT). Those regions for which the weight is at most W are not processed further, while the others are processed until their weight drops to W or less. By an end-game analysis in a manner similar to the analysis for the RTILE problem, we can derive the following approximation guarantee for the DRTILE problem; the details are omitted:

THEOREM 3.4. *There exists an $O(n^2 + p \log n)$ time 5-approximation algorithm for the* DRTILE *problem. If the array has only $\{0, 1\}$-entries, the approximation guarantee improves to 9/4.*

4 The RPACK Problem

The following theorem shows that even a very restricted version of the RPACK problem is NP-hard.

THEOREM 4.1. *The* RPACK *problem is NP-hard in two dimensions, even in the case that each rectangle in the family intersects at most three other rectangles.*

Proof. The proof is a reduction from PLANAR-3SAT as in Theorem 2.1, and is similar to the proof of Theorem 2 in [FPT81]. ∎

Our main result in this section is an $O(\log n)$-approximation for the RPACK problem. The algorithm uses a decomposition strategy that requires the input rectangles to have a certain geometric structure. We ensure the existence of such a structure by partitioning the input into many distinct classes and independently solving the problem on each such class. Both steps, namely the partitioning of the input into disjoint classes and the decomposition strategy, rely on the geometry of our problem.

To motivate the main algorithm, we begin with a very simple $O(\log^2 n)$-approximation algorithm.

LEMMA 4.1. *Given a family $\mathcal{F}$ of weighted rectangles in the plane such that every rectangle intersects at most c disjoint rectangles, one can compute a c-approximation to the problem of finding a largest-weight disjoint collection.*

Proof. We use a greedy algorithm: repeatedly choose the largest-weight rectangle such that it is disjoint from each of the previously chosen rectangles. We claim that the weight of this selection is within a factor c of the optimal solution. Let $O = \{O_1, O_2, ..., O_k\}$ be the rectangles chosen by an optimal solution and let $G = \{G_1, G_2, ..., G_l\}$ denote those chosen by the greedy solution above. Define a map $f : O \to G$ such that $f(O_i) = G_j$ if and only if G_j is the first rectangle chosen by the greedy algorithm that intersects O_i. Clearly, $\texttt{weight}(G_j) \geq \texttt{weight}(O_i)$. Also, at most c rectangles from O may map to any fixed G_j. Thus $\sum_{i=1}^{k} \texttt{weight}(O_i) \leq c \sum_{j=1}^{l} \texttt{weight}(G_j)$. The lemma follows. ∎

THEOREM 4.2. *There is a simple polynomial time approximation algorithm that returns a solution of p rectangles with total weight at least $\Omega(1/\log^2 n)$ of the optimal.*

Proof. The algorithm consists of the following simple steps:

1. Partition the set of given rectangles into $\lceil \log n \rceil^2$ classes (i, j), $1 \leq i \leq \lceil \log n \rceil$ and $1 \leq j \leq \lceil \log n \rceil$. The class (i, j) comprises all rectangles with width $\in [2^{i-1} + 1, 2^i]$, and height $\in [2^{j-1} + 1, 2^j]$.
2. In each class of rectangles, we compute a constant-factor approximation to the optimal solution for

this class. Since rectangles in each class satisfy the premise of Lemma 4.1 for $c = 9$ (it is easy to see that this is tight), we can get a 9-approximation.

3. Pick the largest-weight solution among all the classes.

By the pigeon-hole principle, the largest-weight class must contribute a proportion $\Omega(1/\log^2 n)$ of the total weight. Thus we get an $O(\log^2 n)$-approximation. ■

We next show how the total number of classes can be reduced to $O(\log n)$ by using somewhat more involved ideas. We begin by showing that the packing problem can be efficiently approximated if the problem has "constant width", i.e., any vertical line intersects only a constant number of rectangles.

LEMMA 4.2. *Given a family $\mathcal{F}$ of n weighted rectangles in the plane such that every vertical line intersects at most c disjoint rectangles, one can compute in $O(np)$ time a c-approximation to the problem of finding a largest-weight disjoint collection of size p.*

Proof. The algorithm is as follows:

1. Project each rectangle onto the x-axis; assign each projection the weight of the corresponding rectangle.

2. The projections define an interval graph G. We use dynamic programming to compute the optimum solution as follows:

   (a) We maintain a two-dimensional array $S[i, j]$ where $1 \leq i \leq n$ and $0 \leq j \leq p$. The array entry $S[i, j]$ denotes the optimal weight obtainable by a collection of j intervals such that each interval is contained in $[1, i]$.

   (b) We iteratively compute $S[i + 1, j]$ as follows. Let $\mathcal{J}_i$ denote the set of intervals ending at location i and let w_U and l_U respectively denote the weight and the left end-point of an interval U. Then $S[i+1, j] = \max_{U \in \mathcal{J}_{i+1}} \{S[i, j], w_U + S[l_U, j - 1]\}$.

   (c) Output $S[n, p]$.

3. Output the p rectangles associated with projections in I, breaking any ties arbitrarily.

Clearly, every independent set in G maps to a disjoint collection of the same weight in $\mathcal{R}$. On the other hand, we now establish that given any disjoint collection C in $\mathcal{R}$, there exists an independent set of size p whose weight is at least $1/c$ times the weight of C in the graph G. To see this, consider the following level-assignment scheme for C : Sweep a vertical line from left to right; the topmost rectangle touching the sweep line at any instant is assigned level 1. Remove all rectangles that have been assigned a level, and assign the next level in an identical fashion. Repeat until there are no more rectangles remaining. Clearly, at most c distinct levels will be assigned in this manner and, by the pigeon-hole principle, one of the levels must contain at least a fraction $1/c$ of the weight of the collection C.

Thus the solution computed by the above algorithm must be at least a c-approximation. The dynamic programming above trivially takes $O(n^2p)$ time because it takes $O(n)$ time to evaluate each of the np terms $S[i, j]$. A more careful accounting gives $O(np)$ running time since, for each fixed j, the calculation of $S[i, j]$ for all i takes only $O(n)$ time. ■

THEOREM 4.3. *There is an $O(n^2p + np^2 \log n)$ time approximation algorithm that returns a solution of at most k rectangles with total weight at least $\Omega(1/\log n)$ of the optimal.*

Proof. The algorithm comprises the following steps:

1. Partition the set of all the given rectangles into $\lceil \log n \rceil$ classes such that the class i consists of all rectangles with height $\in [2^{i-1} + 1, 2^i]$.

2. For each class i of rectangles, do the following steps:

   (a) Superimpose a collection $\mathcal{L}$ of horizontal lines such that each consecutive pair of lines is exactly distance 2^{i+1} apart, the topmost line is exactly distance 2^{i+1} above the topmost rectangle of the class and the lowest line is just below the lowest rectangle in the class. This gives us a collection of slabs indexed, say, 1 through q.

   (b) Discard any rectangles that intersect any lines in the above collection. Each slab satisfies the premise of Lemma 4.2 with $c = 3$ and hence for any $i \in [1, p]$, we can compute a 3-approximate solution for rectangles in each slab.

   (c) Let $T_j[i]$ be the weight of the solution computed above for slab j with an allocation of i rectangles. We use dynamic programming to find an allocation $i_1, ..., i_q$ of rectangles such that $\sum_{j=1}^{q} i_j = p$, and $\sum_{j=1}^{q} T_j[i_j]$ is maximum possible over all such allocations. Since the slabs induce a disjoint partition, a union of solutions across the slabs also constitutes a solution to our problem.

(d) Shift the collection $\mathcal{L}$ down by one unit and repeat this process. Terminate when the topmost line touches the topmost rectangle in the class.

(e) Output the best solution computed over all 2^{i+1} iterations.

3. Output the best solution computed for any class.

We begin with an analysis of Step 2 for a fixed class, say, class i. Let OPT_i denote an optimal solution restricted to class i rectangles. We first claim that, in one of the iterations in Step 2 above, the weight of the intersected rectangles is at most half the weight of rectangles in OPT_i. To see this, simply observe that, among the 2^{i+1} shifts used in this step, a rectangle is intersected by a line in $\mathcal{L}$ in at most 2^i shifts.

This observation combined with Step 2(c) implies that Step 2 outputs a solution of weight at least 1/6 of OPT_i. The approximation in the theorem now follows by observing that, for some i, we must have $\mathrm{OPT}_i \geq \mathrm{OPT}/\lceil \log n \rceil$. The running time follows from two observations. There are $\Theta(n)$ iterations in Step (e) each of which takes $O(np)$ time in Step (b), giving the $O(n^2p)$ term. The $O(np^2 \log n)$ term comes from summing $O(n^2p)$ for Step (c) over all iterations. ∎

5 Concluding Remarks

Some related rectangle packing and covering problems are as follows. A simple case of the `RPACK` problem is that *all* possible rectangular, axis-parallel subrectangles (subarrays) of a $n \times n$ array are given, with their weights being the sum of the numbers within; a crucial point is that the array elements may be positive or negative. This problem arises in database mining situations where each array entry denotes the optimized-gain relative to some threshold, and hence some entries may be negative. The problem as before is to pick at most k non-intersecting rectangles of largest total weight. The following theorem summarizes our results for this problem, we refer to it as the `ARRAY RPACK` problem:

THEOREM 5.1. *The* `ARRAY RPACK` *problem is NP-hard but can be approximated to within a factor of two in polynomial time.*

The NP-hardness proof above is somewhat involved and we defer its details to the final version. The approximation algorithm, on the other hand, is based on 2HBTs. Rastogi and Shim [RS97] have independently obtained a 4-approximation based on the HBT approach.

A further variation of this scenario is when our goal is to pick at most k *possibly overlapping* rectangles of largest total weight where the array elements in the common portions are counted only once. A greedy approach works in this case to give a constant factor approximation.

Motion-compensated video compression by block matching is an application area where a variant of the `RTILE` problem arises. Since successive frames of a video sequence are strongly correlated, frames can be compressed by reference to the preceding ones. The most popular method for this is *block matching* [JJ81], which involves tiling the current frame with blocks, then replacing each by a reference to the block in the previous frame from which it differs least (say, in Hamming metric) and appending the vector of the differences. One problem here is to put a limit on the maximum difference for each tile and to minimize the number of tiles used. That gives the `DRTILE` problem with a caveat: the weight of a tile is no longer the sum of the elements in it, but rather it is given by an oracle that computes the appropriate Hamming distance for each tile. The HBT-based algorithms from Section 3 carry through for this problem and we have 4- and 2-approximations respectively. See [S94] for background on block matching, [M94] for an extensive bibliography, and see [MM+96] for a detailed formulation of this problem.

Acknowledgements

Sincere thanks to Rajeev Rastogi and Kyesuk Shim for formulating a version of the `RPACK` problem and sharing it with us graciously. Thanks to Vishy Poosala for numerous discussions on multi-dimensional histograms, and to Jon Sharp for a careful reading of a preliminary version of this paper.

References

[AIS93] R. Agrawal, T. Imielinski and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 914-925, 1993.

[BNR95] V. Bafna, B. Narayanan and R. Ravi. Nonoverlapping local alignments (Weighted independent sets of axis-parallel rectangles). *Proceedings of the Workshop on Algorithms and Data Structures* (WADS '95), LNCS 955, 506-517, 1995.

[B83] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Proc. 24th FOCS*, 265-273, 1983.

[Be84] J. Bentley. Programming Pearls. *Comm. ACM*, 865-871, 1984.

[dAF92] F. d'Amore and P. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Proc. Letters*, 255-259, 1992.

[FPT81] R. Fowler, M. Paterson and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Proc. Letters*, 133-137, 1981.

[FKN80] H. Fuchs, Z. Kedem and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 124-133, 1980.

[FM+97] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama. Optimized two dimensional optimization rules. *Proc. SIGMOD*, 1997.

[H97] D. Hochbaum. Chapter 9. *Approximating algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[HW85] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of ACM*, 130-136, 1985.

[FJ+88] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker. *Solving problems on concurrent processors*, Vol 1, Prentice-Hall, 1988.

[HPF] High Performance Fortran Forum Home Page. `http://www.crpc.rice.edu/HPFF/home.html`.

[KR96] C. Kenyon and E. Remila. Approximate strip packing. *Proc. 37th FOCS*, 31-36, 1996.

[I93] Y. Ioannidis. Universality of serial histograms. *Proc. 19th Intl. Conf. on Very Large Databases*, 256-267, 1993.

[IP95] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. *Proc. ACM SIGMOD*, 233-244, 1995.

[JJ81] J. Jain and A. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on communications*, 29, 1799–1808, 1981.

[KMS97] S. Khanna, S. Muthukrishnan and S. Skiena. Efficient array partitioning. *Proc. 24th ICALP*, 616-626, 1997.

[L82] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Computing* 11, 329–343, 1982.

[LNS90] R. Lipton, J. Naughton and D. Schneider. Practical selectivity estimation through adaptive sampling. *Proc. ACM SIGMOD*, 1-11, 1990.

[LSW97] B. Lent, A. Swami and J. Widom. Clustering association rules. *Proc. 13th Intl Conf on Data Engineering*, 1997.

[M93] F. Manne. *Load Balancing in Parallel Sparse Matrix Computations.* Ph.D. thesis, Department of Informatics, University of Bergen, Norway, 1993.

[M94] C. Manning. *Introduction to Digital Video Coding and Block Matching Algorithms.* `http://atlantis.ucc.ie/dvideo/dv.html`.

[MM+96] G. Martin, S. Muthukrishnan, R. Packwood and I. Rhee. Fast algorithms for variable size block matching motion estimation with minimal error. Submitted, 1996.

[MD88] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. *Proc. SIGMOD*, 28-36, 1988.

[P97] V. Poosala. *Histogram-based estimation techniques in databases.* Ph.D. dissertation, U. Wisconsin, 1997.

[PI97] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. *Proc. Very Large Data Bases*, 1997.

[RS97] R. Rastogi and K. Shim. Personal communication, 1997.

[S94] Block Matching Methods. `http://www.ee.princeton.edu/~santanu/journal/paper/node3.html`.

[SL+93] W. Sun, Y. Ling, N. Rishe and Y. Deng. An instant and accurate size estimation method for joins and selections in a retrieval-intensive environment. *Proc. ACM SIGMOD*, 79-88, 1993.

[UZ+96] M. Ujaldon, E. Zapata, B. Chapman and H. Zima. Vienna Fortran/HPF extensions for sparse and irregular problems and their compilation. Manuscript, 1996.

[ZBG86] H. Zima, H. Bast and M. Gerndt. Superb: A tool for semi-automatic MIMD/AIMD parallelization. *Parallel Computing*, 1–18, 1986.

On Approximating Rectangle Tiling and Packing

Sanjeev Khanna[*] S. Muthukrishnan[†] Mike Paterson[‡]

Abstract

Our study of tiling and packing with rectangles in two-dimensional regions is strongly motivated by applications in database mining, histogram-based estimation of query sizes, data partitioning, and motion estimation in video compression by block matching, among others.

An example of the problems that we tackle is the following: given an $n \times n$ array A of positive numbers, find a tiling using at most p rectangles (that is, no two rectangles must overlap, and each array element must fall within some rectangle) that minimizes the maximum weight of any rectangle; here the *weight* of a rectangle is the sum of the array elements that fall within it. If the array A were one-dimensional, this problem could be easily solved by dynamic programming. We prove that in the two-dimensional case it is NP-hard to approximate this problem to within a factor of 1.25. On the other hand, we provide a near-linear time algorithm that returns a solution at most 2.5 times the optimal.

Other rectangle tiling and packing problems that we study have similar properties: while it is easy to solve them optimally in one dimension, the two-dimensional versions become NP-hard. We design efficient approximation algorithms for these problems.

1 Introduction

For motivation we begin with a simple example involving two-dimensional histograms.

Example. Cost-based query optimizers rely on estimating the result sizes of operators (e.g., natural join, selection) to evaluate complex query execution plans. Some statistics on the underlying data are used to efficiently *estimate* the approximate result size of operators. The popular method is to approximate the data distribution using *histograms*, that is, to partition the data into ranges and approximate each region by an appropriate "average" value: the fewer the data ranges, the faster is the estimation, and the better the accuracy in estimation. An example is the classical *equi-depth histogram* for two numerical attributes of a database. Consider the rectangular region indexed by the domain of these two attributes, and which stores the frequency of tuples in the database with each combination of attribute values. The equi-depth histogram is a partition (without overlap) of this two-dimensional space into a prespecified number of rectangular tiles of nearly equal sums of frequencies. This is an instance of the rectangle tiling problem that we consider here.

Other applications for rectangle tiling and packing include data partitioning, load balancing in parallel scientific applications, motion estimation in videos by block matching, and database mining applications among others. All our problems are natural extensions to two dimensions of problems well studied and thoroughly understood on one-dimensional arrays. From an algorithmic point of view, all our problems would have polynomial time optimal solutions by dynamic programming in one-dimension. For two-dimensional regions, we show that these problems become NP-hard and sometimes are NP-hard even to approximate to within certain constant factors. Our main results are highly efficient (often linear time) algorithms with provably small approximation ratios for these problems. We state our problems precisely in Section 1.1, give a brief overview of application areas where they occur (Section 1.2), and then describe our results in detail (Section 1.3).

1.1 Problems.

The three problems `RTILE`, `DRTILE`, and `RPACK`, that are the primary focus of this extended abstract are described next; a few others are given in Section 5.

(a) Tiling Problems. Given an $n \times n$ array[1] A of non-negative numbers, partition it into at most p rectangular subarrays (that is, the rectangles cover A without overlap) such that the maximum weight of any rectangle is minimized; here, the *weight* of a rectangle is the sum of all the array elements which fall within it. We call this the *rectangle tiling* (`RTILE`) problem. We also consider its dual, namely, to find a tiling as above of an $n \times n$ array A in which every rectangle has weight at most W and which minimizes the number of rectangles. We call this the *dual rectangle tiling* (`DRTILE`) problem.

(b) Packing Problem. Given l weighted axis-parallel rectangles in an $n \times n$ grid, find a disjoint subset of at

[*]Math. Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ. `sanjeev@research.bell-labs.com`

[†]Information Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ. `muthu@research.bell-labs.com`

[‡]Dept. of Computer Science, Univ. of Warwick, Coventry, CV4 7AL, UK. `msp@dcs.warwick.ac.uk`. This work was partly supported by ESPRIT LTR Project no. 20244 — ALCOM-IT.

[1]All our results extend easily to $n \times m$ arrays.

most k rectangles of largest total weight.[2] We call this the *rectangle packing* (RPACK) problem.[3]

Some remarks about our problems. The tilings that we allow in RTILE and DRTILE are arbitrary. For restricted cases of tilings, there are known results. For examples, see [MM+96] for quad-tree tilings, and see [KMS97] for tiling with p vertical and q horizontal lines to obtain pq blocks. The RTILE problem in one dimension has a rich history as surveyed in [KMS97]; a dynamic programming solution of $O(np)$ time is known and the best known general bound is $O(\min\{n+p^{1+\epsilon}, n\log n\})$. The DRTILE problem in one dimension can be solved optimally by a simple greedy strategy in linear time.

The RPACK problem in two dimensions is very general; it captures many interesting problems as special cases. One that we consider allows *all* axis-parallel subrectangles of a rectangular array; the weight of a rectangle is the sum of array elements in it — the array elements are allowed to take negative values. The one-dimensional version of this is a generalization of Bentley's Maximum Subarray Sum to k regions; in Bentley's problem, one must find the interval of a one-dimensional array with largest sum [Be84]. We focus on the general two-dimensional RPACK problem in this abstract; a special case is discussed in Section 5. We point out that two-dimensional bin packing problems have been studied in the literature (e.g., [KR96]), but they differ significantly from our problems: algorithms for bin packing have the flexibility to move rectangles around in the space whereas in our case all rectangles are fixed in space. Our problems are similar in spirit to geometric covering and packing studied by Hochbaum et al (e.g., [H97, HW85]). The problem they study which is most relevant to our work is a special case of the DRTILE problem when all the given objects are squares of identical sizes. The authors obtain polynomial time approximation schemes for this special case of our problem.

1.2 Application Scenarios.

The problems of tiling and packing arrays with rectangles arise naturally in a number of application areas. We describe some scenarios where these problems are found exactly in the form we study here.

Two-dimensional histograms. As mentioned earlier, this problem arises in building histograms over two attributes. A histogram is an approximate compact statistic of the database contents and it is used for estimating result sizes of operators. Although other such estimators exist (such as sampling [LNS90] and parametric techniques [SL+93]), histograms are the most commonly used form of statistics in practice and they are used in DB2, Informix, Ingres, Oracle, Sybase, etc. (See page 29 of [P97] for a list and more information.) Using histograms over a single numerical attribute has been extensively researched in the database community, and a number of histograms have been identified: *MaxDiff*, *V-Optimal*, *Equi-depth*, etc. (See [P97] for a fine taxonomy, and [I93, IP95] for examples of their accuracy and optimality.) More recently, there has been an effort to capture the *joint frequency distributions* in the database, and histograms over multiple numerical attributes are being studied. This was initiated in [MD88] and recent work is found in [PI97, P97]. Both the RTILE and DRTILE problems are instances of constructing what are known as equi-depth histograms on two numerical attributes. Intuitively, the maximum weight of a tile controls the accuracy of estimation and the number of tiles affects the efficiency of estimation. Our two tiling problems respectively optimize one of these parameters while keeping the other bounded.

Data partitioning. For data stored in two-dimensional arrays, high-performance computing languages allow the user to specify a partitioning and distribution of data onto a logical set of processors. The goal is to balance the load. See [FJ+88, M93] for technical discussions. Examples of languages which support such data partitioning include the Superb environment [ZBG86] and High Performance Fortran HPF2 [HPF]. The general version of this problem with p processors is precisely the RTILE problem.

Database mining. Discovering *association rules* in databases, introduced in [AIS93], is a promising approach to mining. Association rules are of the form $\mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are predicates over a set of attributes. Such an association rule means that the tuples that satisfy $\mathcal{X}$ tend to satisfy $\mathcal{Y}$. We are interested in finding *optimized-gain* rules, that is ones where the number of tuples that both $\mathcal{X}$ and $\mathcal{Y}$ satisfy exceeds a certain percentage of the number of tuples that satisfy $\mathcal{X}$.[4] The RPACK problem is that of finding *k clustered two-dimensional optimized-gain* rules [LSW97, FM+97][5]. It

[2] This is a packing problem with the additional constraint that at most k items be packed.

[3] The dual of the RPACK problem is to find a minimum cardinality subset of disjoint rectangles with total weight at least W. Since it is NP-hard to find even a feasible solution for this problem (see [FPT81]), it cannot be approximated to any factor and hence is not considered any further.

[4] There are optimized-confidence and optimized-support rules as well; they will not be discussed here.

[5] An example: (Balance $\in [10^4, 10^5]$) AND (Age $\in [40, 50]$) $\to$ (Carloan=yes). Here the ranges in Balance and Age make the association rule clustered.

is clustered since we look for rectangular regions. The k regions are disjoint since presumably the goal is to target the k "favorable" regions for marketing, mail-order or advertising purposes, where each region will be targeted differently or by different departments within a company. Finally, in our definition of the RPACK problem, we have considered the general case of practical interest when only a subset of all the induced rectangles from the two-dimensional domain (of the two numerical attributes) may be of interest for mining. See [LSW97, FM+97] for more on this application.

1.3 Results and Techniques.

We summarize our results for each of the three problems.

1. **The RTILE problem.** We show that it is NP-hard not only to compute the optimum solution but even to approximate it to within a factor of $5/4 = 1.25$. The hardness result holds even when each array element is an integer in the range $[1, 4]$. On the other hand, our main result is an $O(n^2 + p \log n)$ (near-linear time) algorithm for this problem with an approximation ratio of 2.5, which holds even in the presence of large array elements. Observe that a running time of $O(n^2 + p \log n)$ is near-linear because the input is of size $O(n^2)$ and $p = O(n^2)$.

   Let the total sum of all the items in array A be W. A lower bound of W/p on the maximum weight is immediate, and if there are no large elements in the array our algorithm returns a tiling with maximum weight $O(W/p)$. We use the natural approach of partitioning the problem into two disjoint parts and assigning each part a proportionate number of rectangles for tiling, but the potential presence of large elements complicates this strategy.

2. **The DRTILE problem**. Since this is the dual of the RTILE problem, it follows that it is NP-hard to solve it exactly. We show a series of approximation algorithms trading off running time versus the quality of approximation. The best approximation we achieve is a factor of 2 with high polynomial running time. At the other end of the spectrum, we have a near-linear time algorithm with a larger constant approximation factor.

   The analysis of our approximation algorithm for the RTILE problem proves that a strong relationship exists between approximate solutions to these two dual problems. This relationship forms the basis of our efficient $O(1)$-approximation for the DRTILE problem. Our alternative algorithms are based on binary space partitioning (BSP) [FKN80], and use the fact that an arbitrary tiling can be converted into a *structured hierarchical* one with a relatively small increase in the number of rectangles. While a straightforward BSP-based dynamic programming can be used to get a 4-approximation, a more sophisticated dynamic programming yields a 2-approximation, albeit at the expense of a very high running time.

   REMARK 1. *Our analysis of the* RTILE *approximation algorithm establishes that the solution value changes only by a constant factor when the total number of tiles used is changed by a constant factor. This strong relationship enables a BSP-based approach to get a constant-factor approximation to the* RTILE *problem. Without such a relationship, the BSP approach would not be applicable since the number of tiles allowed is fixed.*

3. **The RPACK problem.** We show that the RPACK problem is NP-hard even in the restrictive setting where each rectangle in the input intersects at most three other rectangles. We design an $O(\log n)$-approximation algorithm for the general problem that runs in time $O(n^2 p \log n)$. The main idea underlying the algorithm is a decomposition into constant-width problems which we show can be well-approximated quite efficiently. While the decomposition scheme itself is similar to the PTAS paradigm used by Baker [B83], it requires a partition of the input into disjoint classes of geometrically "well-structured" rectangles such that each class is amenable to our decomposition strategy. The geometry of two dimensions leads to $O(\log n)$ disjoint classes and the algorithm chooses the best solution among all the classes.

   Even the special case of the RPACK problem that allows all axis-parallel subrectangles in an array can be shown to be NP-hard. On the other hand, a BSP-based dynamic programming approach gives a 2-approximation for this problem.

1.4 General Comments on Our Results.

Our rectangle packing and tiling problems seem quite natural; we expect them to find applications in scenarios besides the ones we have described here. Most of our algorithms are simple and easy to implement. Our algorithms for the RTILE and DRTILE problems are variations of what practitioners would normally implement, namely, hierarchical partitioning schemes. The key insight our algorithms for these problems can supply to practitioners is that even simple heuristics can give provably good performance guarantees.

1.5 Map.

Section 2 shows the hardness result and the approximation algorithms for the `RTILE` problem. In Section 3, we present our results for the dual `DRTILE` problem. Section 4 sketches the hardness of the `RPACK` problem and gives our $O(\log n)$-approximation algorithm. Finally, in Section 5 we briefly describe other related problems and our results for them.

2 The `RTILE` Problem

2.1 Hardness of Approximation.

We begin with a proof of the APX-hardness of the `RTILE` problem.

THEOREM 2.1. *The* `RTILE` *problem is NP-hard, even in the case where the element weights are non-negative integers bounded by a constant. Furthermore, it is NP-hard to determine the optimal value to within a factor of* $5/4$.

Proof. The proof reduces `PLANAR-3SAT` to `RTILE`. An instance of `PLANAR-3SAT` is a 3CNF formula with the extra property that the following graph G_F is planar. The bipartite graph G_F has the variables and clauses of F as its two vertex sets, and an edge corresponding to each occurrence of a variable or its negation in a clause. It was shown in [L82] that the problem `PLANAR-3SAT` is NP-complete.

For any formula F in 3CNF for which G_F is planar we construct an array A_F of integers in the range $[1, 4]$, and an integer p_F, which gives an instance $(A_F, p_F, 4)$ of `RTILE` such that F is satisfiable if and only if A_F can be tiled by p_F tiles of weight at most 4. For each variable in F, we construct a closed loop of orthogonally adjacent elements, mostly of value 2. The loops are disjoint but, for each clause C, the three loops corresponding to the variables occurring in C approach closely together in a special *clause gadget* which we describe below. Because G_F is a planar graph there are no topological obstructions to this construction.

Figure 1 shows a portion of the array A_F. The heavy lines are for the explanation only. They mark out three paths, each corresponding to part of a variable loop. For the path on the left it is clear that any tile of weight 4 can cover at most two elements of the path. Since the target number p_F will require this to be achieved, there are just two choices, depending on whether odd or even pairs are to be covered. The bottom-right path shows a minor variation where the sequence 2112 occurs. Here the two 1's must be tiled with one or other of the 2's. This device allows us freedom to alter the relative parity of the *physical* layout of any path.

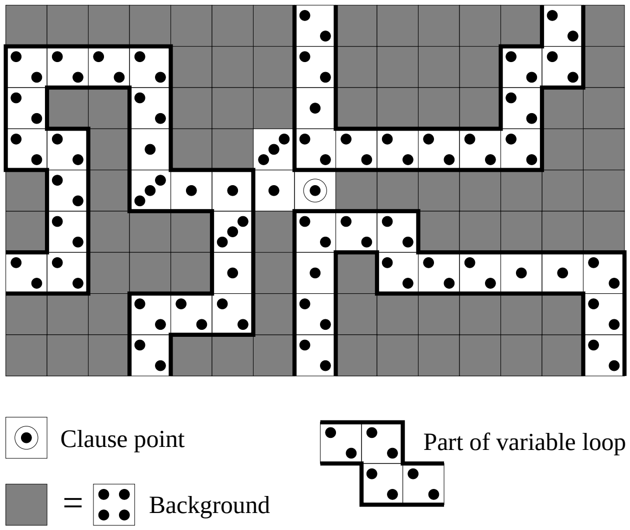

Figure 1: Clause gadget and variable loops

This portion of A_F illustrates the modeling of a clause gadget. The element marked with a circle has value 1. It can be checked that this can be tiled by just extending one of the tiles of the tiling for a variable loop, *provided that* this tiling has the appropriate parity. If none of the three loops has the right parity then no 4-tiling is possible, without using an extra tile.

If F has a satisfying assignment then, with the corresponding choices of parity for each variable loop, each clause point may be tiled by extending one of the tiles in a variable loop. If F is not satisfiable then any choice of parity for the variable loops leaves at least one clause point uncovered. Any tiling with p_F tiles which is not of the form discussed above has at least one tile with weight 5 or more. ∎

2.2 Approximation Algorithm.

In the remainder of this section, we focus on designing an efficient constant-factor approximation algorithm for the `RTILE` problem. If W_0 denotes the total weight of the given array A_0, p denotes the number of tiles, and M the maximum weight of any element in the array, then $\max\{W_0/p, M\}$ is clearly a lower bound for the minimum weight in a tiling. As we shall see, it is always possible to find a tiling which is within a constant factor of this simple lower bound.

Our algorithm uses a straightforward recursive partitioning strategy. Let $w = W_0/p$ and $M' = \max\{w, M\}$, so w is the average weight of a tile and M' is the simple lower bound on the maximum weight.

DEFINITION 1. (Thick Cut) *For any rectangular array, consider partitions into two subarrays defined by vertical*

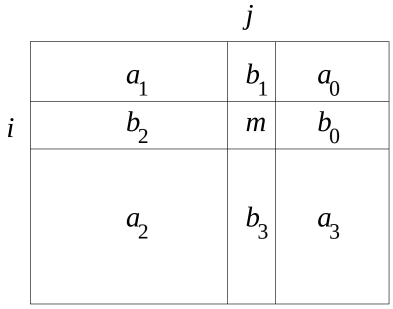

Figure 2: Subarray with no thick cut

or horizontal lines. Such a partition for which each subarray has weight at least w is a thick cut.

THEOREM 2.2. *There exists a 5/2-approximation algorithm for the* RTILE *problem.*

Proof. The algorithm begins by simply using thick cuts repeatedly to partition the array into subarrays. When no further thick cuts are possible we analyse the "end-game" and complete the tiling. Provided that we use at most $\lfloor W/w \rfloor$ tiles for a subarray of weight W, the total number of tiles used will be at most p.

Let A be a subarray, of weight W say, remaining after the partitioning, so A has no thick cut and the weight of A is at least w. Let i be the largest row index of A such that the total weight of rows of index less than i is less than w. It must be that the total weight of rows of index greater than i (if any) is also less than w, otherwise there would be a thick cut. Define the column index j similarly. We can depict A as in Figure 2, where the a_r's and b_r's denote the total weights of the corresponding regions and m is the value of $A_{i,j}$.

If $W < 5w/2$ then using just a single tile for A is within the 5/2-approximation bound, since $5w/2 \leq 5M'/2$.

Otherwise, we may assume $W \geq 5w/2$ and so at least two tiles can be allocated to A. There are four obvious ways to partition A into two rectangles with a cut. Provided that $b_r + a_r + b_{r+1} + a_{r+1} + b_{r+2} + m < 5M'/2$ for at least one value of r ($0 \leq r \leq 3$ and subscripts are taken modulo 4), we have a tiling with two tiles of maximum weight less than the 5/2-approximation bound.

Otherwise, we have $b_r + a_r + b_{r+1} + a_{r+1} + b_{r+2} + m \geq 5M'/2$ for $0 \leq r \leq 3$, and adding these four inequalities yields

$$2(a_0 + a_1 + a_2 + a_3) + 3(b_0 + b_1 + b_2 + b_3) + 4m \geq 10M'.$$

Hence,

$$\begin{aligned} 3W &= 3(a_0 + a_1 + a_2 + a_3 + b_0 + b_1 + b_2 + b_3 + m) \\ &\geq 10M' - m \geq 9M', \end{aligned}$$

and so $W \geq 3M'$. In this case at least three tiles can be allocated to A.

There is an obvious way to partition A with two parallel cuts into three rectangles of maximum weight less than $5M'/2$ unless $b_0 + b_2 + m \geq 5M'/2$ and $b_1 + b_3 + m \geq 5M'/2$. In this case

$$W \geq b_0 + b_1 + b_2 + b_3 + m \geq 5M' - m \geq 4M'.$$

This case then allows four tiles for A.

In a similar way, there is a 4-tiling with maximum weight less than $5M'/2$ unless $b_r + m \geq 5M'/2$ for all r. The latter case implies

$$W \geq b_0 + b_1 + b_2 + b_3 + m \geq 10M' - 3m \geq 7M'.$$

It is clear though that if at least five tiles can be used, there is a tiling with maximum weight less than M'. ∎

The following theorem shows an $O(n^2 + p \log n)$ bound on the time complexity of the algorithm outlined in the previous theorem.

THEOREM 2.3. *The algorithm of Theorem 2.2 can be implemented to run in time $O(n^2 + p \log n)$.*

Proof. Assume access to an oracle that, given the end-points of any rectangular region in the array, gives in constant time the total weight of all its elements. Then we claim that the initial partitioning can be created in $O(p \log n)$ time. To see this, first observe that each partition step can be done in $O(\log n)$ time since finding a thick cut only requires a binary search which can be done in $O(\log n)$ oracle calls. Next we claim that the final tiling can be done in time $O(\log n)$ per subarray; this follows since these partitions can be determined by a straightforward binary search. Thus all the steps associated with the algorithm can be performed in $O(p \log n)$ time. Finally, the constant time oracle used above can be implemented by an $O(n^2)$ time pre-processing step using standard arguments. Let $A = \{a_{ij}\}_{1 \leq i,j \leq n}$ be the input array. For $1 \leq k, l \leq n$, define $S(k,l) = \sum_{1 \leq j \leq l} a_{kj}$. Using the prefix sum algorithm, these sums can be implicitly computed after $O(n)$ time preprocessing for each row in the given array. For $1 \leq k, l \leq n$, define $T(k,l) = \sum_{1 \leq i \leq k} S(i,l)$. It is easy to see that the table of $S()$ entries can be used to compute all $T(k,l)$ entries in $O(n^2)$ time (just compute them left to right, and top to bottom). The final step is to show that the array T allows us to implement the desired oracle in constant time. Suppose we want the sum of the elements contained in the rectangular region determined by (i_1, j_1) (top left corner) and (i_2, j_2) (bottom right corner). This sum is given by $T(i_2, j_2) + T(i_1, j_1) - T(i_1, j_2) - T(i_2, j_1)$. The claimed result now follows. ∎

The proof of the following theorem is based on the ideas already described, and hence is omitted.

THEOREM 2.4. *There exists an $O(n^2 + p \log n)$ time algorithm which gives a 9/4-approximation for the* RTILE *problem on $\{0, 1\}$-arrays.*

3 The DRTILE Problem

The following theorem is an immediate corollary of Theorem 2.1.

THEOREM 3.1. *The* DRTILE *problem is NP-hard, even in the case where the element weights are non-negative integers bounded by a constant.*

We design three different approximation algorithms for this problem. While each one gives a constant-factor approximation, they reflect a tradeoff between the time complexity and the quality of the approximation guarantee achieved. The first two algorithms use dynamic programming, while the third algorithm is based on the partitioning approach and uses the strong relation that exists between the average tile weight and the maximum tile weight in an optimal solution, as shown in the analysis of the RTILE problem.

3.1 Hierarchical Binary Tilings.

We begin by formally defining a nicely structured class of tilings that underlies the dynamic programming approach that we use in this section.

DEFINITION 2. (Hierarchical Binary Tiling) *A tiling of a region is called a* hierarchical binary tiling (HBT) *if it consists of a single tile, or there exists a vertical or horizontal line such that the line does not intersect the interior of any rectangle, and the two distinct regions on either side are also hierarchical binary tilings.*

The *hierarchical binary W-tiling problem* is to find an HBT such that the maximum weight of any rectangle is at most W and the number of rectangles is a minimum. We show how to calculate the minimum number r of rectangles in any such partition using dynamic programming; a solution, namely an HBT with r rectangles none of whose weights is more than W, can be found easily from the dynamic program. For any region, we calculate the minimum over all possible horizontal and vertical lines of the number of rectangles used in the two regions obtained by splitting with that line. At the base level, any region which has total weight at most W can be tiled with a single rectangle. The total running time of this algorithm is bounded by $O(n^5)$, since there are $O(n^4)$ subrectangles and, for each subrectangle, the problem can be solved in $O(n)$ time using previously computed solutions and the recursive definition.

That lets us conclude:

LEMMA 3.1. *The hierarchical binary W-tiling problem can be solved in $O(n^5)$ time.*

Now we relate the hierarchical binary W-tiling problem to the DRTILE problem.

LEMMA 3.2. *If an array has a solution to* DRTILE *with p rectangles, then there is a hierarchical binary W-tiling with at most $4p$ rectangles.*

Proof. The proof follows from the results in [dAF92] for constructing binary space partitions (BSPs) for axis-parallel rectangles in two dimensions. Given a two-dimensional region with n axis-parallel rectangles, an *orthogonal binary space partition* (OBSP) recursively partitions the region into two disjoint regions using a vertical or a horizontal line, repeating this process until each region intersects at most one of the rectangles. If a rectangle is intersected by a line, then it is split into two disjoint rectangles given by the partitioning. It is clear from the definitions that an OBSP of rectangles gives a hierarchical binary tiling. In [dAF92], it is shown that there exists an OBSP in which each original rectangle is split into at most four parts. Thus, there exists a hierarchical binary tiling of at most $4p$ rectangles, and the lemma directly follows. ∎

Combining the two preceding lemmas, we have the following theorem:

THEOREM 3.2. *There exists a $O(n^5)$-time 4-approximation algorithm for the* DRTILE *problem.*

Proof. The proof follows from the observation that any tiling with p rectangles can be converted to a hierarchical binary tiling of at most $4p$ rectangles by using the results of [dAF92] on binary space partitions. Since an optimal hierarchical binary W-tiling can be found in $O(n^5)$ time, the theorem follows. ∎

We next show that working with a relaxation of the notion of HBTs can lead to a better approximation at the expense of significantly increased time complexity. The relaxation is referred to as a *two-hierarchical binary tiling* and is defined as follows:

DEFINITION 3. (Two-Hierarchical Binary Tiling) *A tiling of a region is called a* two-hierarchical binary tiling

(2HBT) *if it consists of a single tile, or there exists a vertical or horizontal line such that the line intersects the interior of at most two rectangles, situated at either end, and the two distinct regions on either side are also two-hierarchical binary tilings.*

THEOREM 3.3. *There exists an $n^{O(1)}$ time 2-approximation algorithm for the* DRTILE *problem.*

Proof Sketch. As before, let the W-2HBT problem be to find a 2HBT with maximum weight at most W with the minimum number of rectangles. It can be shown that the W-2HBT problem can be solved by dynamic programming in $n^{O(1)}$ time; the details are omitted. On the other hand, given n axis-parallel rectangles in two dimensions, the BSP of [dAF92] can be shown to produce a 2HBT of size at most $2n$. Combining the two, we get the claimed result. ∎

Unfortunately, the $O(1)$ in the exponent is very high, so this is not a practical algorithm.

3.2 Partitioning Approach.

While the dynamic programming approach taken above leads to strong performance guarantees, it has a rather high time complexity even for the 4-approximation. We now sketch how the partitioning approach for the RTILE problem in the previous section can be used to obtain a near-linear time approximation algorithm for the DRTILE problem.

Let p^* be the number of rectangles in the optimum solution and W_0 be the total weight of the array. Clearly, $p^* \geq W_0/W$ and $W \geq M$ where M is the largest entry in the array. We first solve the RTILE problem with $p = \lceil W_0/W \rceil$. From the analysis in Theorem 2.2, we know that the weight of each resulting rectangle is at most $2.5 \max\{W_0/p, M\}$ (observe that the solution is again a HBT). Those regions for which the weight is at most W are not processed further, while the others are processed until their weight drops to W or less. By an end-game analysis in a manner similar to the analysis for the RTILE problem, we can derive the following approximation guarantee for the DRTILE problem; the details are omitted:

THEOREM 3.4. *There exists an $O(n^2 + p \log n)$ time 5-approximation algorithm for the* DRTILE *problem. If the array has only $\{0, 1\}$-entries, the approximation guarantee improves to $9/4$.*

4 The RPACK Problem

The following theorem shows that even a very restricted version of the RPACK problem is NP-hard.

THEOREM 4.1. *The* RPACK *problem is NP-hard in two dimensions, even in the case that each rectangle in the family intersects at most three other rectangles.*

Proof. The proof is a reduction from PLANAR-3SAT as in Theorem 2.1, and is similar to the proof of Theorem 2 in [FPT81]. ∎

Our main result in this section is an $O(\log n)$-approximation for the RPACK problem. The algorithm uses a decomposition strategy that requires the input rectangles to have a certain geometric structure. We ensure the existence of such a structure by partitioning the input into many distinct classes and independently solving the problem on each such class. Both steps, namely the partitioning of the input into disjoint classes and the decomposition strategy, rely on the geometry of our problem.

To motivate the main algorithm, we begin with a very simple $O(\log^2 n)$-approximation algorithm.

LEMMA 4.1. *Given a family $\mathcal{F}$ of weighted rectangles in the plane such that every rectangle intersects at most c disjoint rectangles, one can compute a c-approximation to the problem of finding a largest-weight disjoint collection.*

Proof. We use a greedy algorithm: repeatedly choose the largest-weight rectangle such that it is disjoint from each of the previously chosen rectangles. We claim that the weight of this selection is within a factor c of the optimal solution. Let $O = \{O_1, O_2, ..., O_k\}$ be the rectangles chosen by an optimal solution and let $G = \{G_1, G_2, ..., G_l\}$ denote those chosen by the greedy solution above. Define a map $f : O \rightarrow G$ such that $f(O_i) = G_j$ if and only if G_j is the first rectangle chosen by the greedy algorithm that intersects O_i. Clearly, $\texttt{weight}(G_j) \geq \texttt{weight}(O_i)$. Also, at most c rectangles from O may map to any fixed G_j. Thus $\sum_{i=1}^{k} \texttt{weight}(O_i) \leq c \sum_{j=1}^{l} \texttt{weight}(G_j)$. The lemma follows. ∎

THEOREM 4.2. *There is a simple polynomial time approximation algorithm that returns a solution of p rectangles with total weight at least $\Omega(1/\log^2 n)$ of the optimal.*

Proof. The algorithm consists of the following simple steps:

1. Partition the set of given rectangles into $\lceil \log n \rceil^2$ classes (i, j), $1 \leq i \leq \lceil \log n \rceil$ and $1 \leq j \leq \lceil \log n \rceil$. The class (i, j) comprises all rectangles with width $\in [2^{i-1} + 1, 2^i]$, and height $\in [2^{j-1} + 1, 2^j]$.

2. In each class of rectangles, we compute a constant-factor approximation to the optimal solution for

this class. Since rectangles in each class satisfy the premise of Lemma 4.1 for $c = 9$ (it is easy to see that this is tight), we can get a 9-approximation.

3. Pick the largest-weight solution among all the classes.

By the pigeon-hole principle, the largest-weight class must contribute a proportion $\Omega(1/\log^2 n)$ of the total weight. Thus we get an $O(\log^2 n)$-approximation. ∎

We next show how the total number of classes can be reduced to $O(\log n)$ by using somewhat more involved ideas. We begin by showing that the packing problem can be efficiently approximated if the problem has "constant width", i.e., any vertical line intersects only a constant number of rectangles.

LEMMA 4.2. *Given a family $\mathcal{F}$ of n weighted rectangles in the plane such that every vertical line intersects at most c disjoint rectangles, one can compute in $O(np)$ time a c-approximation to the problem of finding a largest-weight disjoint collection of size p.*

Proof. The algorithm is as follows:

1. Project each rectangle onto the x-axis; assign each projection the weight of the corresponding rectangle.
2. The projections define an interval graph G. We use dynamic programming to compute the optimum solution as follows:
   (a) We maintain a two-dimensional array $S[i,j]$ where $1 \leq i \leq n$ and $0 \leq j \leq p$. The array entry $S[i,j]$ denotes the optimal weight obtainable by a collection of j intervals such that each interval is contained in $[1,i]$.
   (b) We iteratively compute $S[i+1,j]$ as follows. Let $\mathcal{J}_i$ denote the set of intervals ending at location i and let w_U and l_U respectively denote the weight and the left end-point of an interval U. Then $S[i+1,j] = \max_{U \in \mathcal{J}_{i+1}} \{S[i,j], w_U + S[l_U, j-1]\}$.
   (c) Output $S[n,p]$.
3. Output the p rectangles associated with projections in I, breaking any ties arbitrarily.

Clearly, every independent set in G maps to a disjoint collection of the same weight in $\mathcal{R}$. On the other hand, we now establish that given any disjoint collection C in $\mathcal{R}$, there exists an independent set of size p whose weight is at least $1/c$ times the weight of C in the graph G. To see this, consider the following level-assignment scheme for C : Sweep a vertical line from left to right; the topmost rectangle touching the sweep line at any instant is assigned level 1. Remove all rectangles that have been assigned a level, and assign the next level in an identical fashion. Repeat until there are no more rectangles remaining. Clearly, at most c distinct levels will be assigned in this manner and, by the pigeon-hole principle, one of the levels must contain at least a fraction $1/c$ of the weight of the collection C.

Thus the solution computed by the above algorithm must be at least a c-approximation. The dynamic programming above trivially takes $O(n^2 p)$ time because it takes $O(n)$ time to evaluate each of the np terms $S[i,j]$. A more careful accounting gives $O(np)$ running time since, for each fixed j, the calculation of $S[i,j]$ for all i takes only $O(n)$ time. ∎

THEOREM 4.3. *There is an $O(n^2 p + np^2 \log n)$ time approximation algorithm that returns a solution of at most k rectangles with total weight at least $\Omega(1/\log n)$ of the optimal.*

Proof. The algorithm comprises the following steps:

1. Partition the set of all the given rectangles into $\lceil \log n \rceil$ classes such that the class i consists of all rectangles with height $\in [2^{i-1}+1, 2^i]$.
2. For each class i of rectangles, do the following steps:
   (a) Superimpose a collection $\mathcal{L}$ of horizontal lines such that each consecutive pair of lines is exactly distance 2^{i+1} apart, the topmost line is exactly distance 2^{i+1} above the topmost rectangle of the class and the lowest line is just below the lowest rectangle in the class. This gives us a collection of slabs indexed, say, 1 through q.
   (b) Discard any rectangles that intersect any lines in the above collection. Each slab satisfies the premise of Lemma 4.2 with $c = 3$ and hence for any $i \in [1,p]$, we can compute a 3-approximate solution for rectangles in each slab.
   (c) Let $T_j[i]$ be the weight of the solution computed above for slab j with an allocation of i rectangles. We use dynamic programming to find an allocation $i_1, ..., i_q$ of rectangles such that $\sum_{j=1}^{q} i_j = p$, and $\sum_{j=1}^{q} T_j[i_j]$ is maximum possible over all such allocations. Since the slabs induce a disjoint partition, a union of solutions across the slabs also constitutes a solution to our problem.

(d) Shift the collection $\mathcal{L}$ down by one unit and repeat this process. Terminate when the topmost line touches the topmost rectangle in the class.

(e) Output the best solution computed over all 2^{i+1} iterations.

3. Output the best solution computed for any class.

We begin with an analysis of Step 2 for a fixed class, say, class i. Let OPT_i denote an optimal solution restricted to class i rectangles. We first claim that, in one of the iterations in Step 2 above, the weight of the intersected rectangles is at most half the weight of rectangles in OPT_i. To see this, simply observe that, among the 2^{i+1} shifts used in this step, a rectangle is intersected by a line in $\mathcal{L}$ in at most 2^i shifts.

This observation combined with Step 2(c) implies that Step 2 outputs a solution of weight at least 1/6 of OPT_i. The approximation in the theorem now follows by observing that, for some i, we must have $\mathrm{OPT}_i \geq \mathrm{OPT}/\lceil\log n\rceil$. The running time follows from two observations. There are $\Theta(n)$ iterations in Step (e) each of which takes $O(np)$ time in Step (b), giving the $O(n^2p)$ term. The $O(np^2 \log n)$ term comes from summing $O(n^2p)$ for Step (c) over all iterations. ∎

5 Concluding Remarks

Some related rectangle packing and covering problems are as follows. A simple case of the `RPACK` problem is that *all* possible rectangular, axis-parallel subrectangles (subarrays) of a $n \times n$ array are given, with their weights being the sum of the numbers within; a crucial point is that the array elements may be positive or negative. This problem arises in database mining situations where each array entry denotes the optimized-gain relative to some threshold, and hence some entries may be negative. The problem as before is to pick at most k non-intersecting rectangles of largest total weight. The following theorem summarizes our results for this problem, we refer to it as the `ARRAY RPACK` problem:

THEOREM 5.1. *The* `ARRAY RPACK` *problem is NP-hard but can be approximated to within a factor of two in polynomial time.*

The NP-hardness proof above is somewhat involved and we defer its details to the final version. The approximation algorithm, on the other hand, is based on 2HBTs. Rastogi and Shim [RS97] have independently obtained a 4-approximation based on the HBT approach.

A further variation of this scenario is when our goal is to pick at most k *possibly overlapping* rectangles of largest total weight where the array elements in the common portions are counted only once. A greedy approach works in this case to give a constant factor approximation.

Motion-compensated video compression by block matching is an application area where a variant of the `RTILE` problem arises. Since successive frames of a video sequence are strongly correlated, frames can be compressed by reference to the preceding ones. The most popular method for this is *block matching* [JJ81], which involves tiling the current frame with blocks, then replacing each by a reference to the block in the previous frame from which it differs least (say, in Hamming metric) and appending the vector of the differences. One problem here is to put a limit on the maximum difference for each tile and to minimize the number of tiles used. That gives the `DRTILE` problem with a caveat: the weight of a tile is no longer the sum of the elements in it, but rather it is given by an oracle that computes the appropriate Hamming distance for each tile. The HBT-based algorithms from Section 3 carry through for this problem and we have 4- and 2-approximations respectively. See [S94] for background on block matching, [M94] for an extensive bibliography, and see [MM+96] for a detailed formulation of this problem.

Acknowledgements

Sincere thanks to Rajeev Rastogi and Kyesuk Shim for formulating a version of the `RPACK` problem and sharing it with us graciously. Thanks to Vishy Poosala for numerous discussions on multi-dimensional histograms, and to Jon Sharp for a careful reading of a preliminary version of this paper.

References

[AIS93] R. Agrawal, T. Imielinski and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 914-925, 1993.

[BNR95] V. Bafna, B. Narayanan and R. Ravi. Nonoverlapping local alignments (Weighted independent sets of axis-parallel rectangles). *Proceedings of the Workshop on Algorithms and Data Structures* (WADS '95), LNCS 955, 506-517, 1995.

[B83] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Proc. 24th FOCS*, 265-273, 1983.

[Be84] J. Bentley. Programming Pearls. *Comm. ACM*, 865-871, 1984.

[dAF92] F. d'Amore and P. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Proc. Letters*, 255-259, 1992.

[FPT81] R. Fowler, M. Paterson and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Proc. Letters*, 133-137, 1981.

[FKN80] H. Fuchs, Z. Kedem and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 124-133, 1980.

[FM+97] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama. Optimized two dimensional optimization rules. *Proc. SIGMOD*, 1997.

[H97] D. Hochbaum. Chapter 9. *Approximating algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[HW85] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of ACM*, 130-136, 1985.

[FJ+88] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker. *Solving problems on concurrent processors*, Vol 1, Prentice-Hall, 1988.

[HPF] High Performance Fortran Forum Home Page. `http://www.crpc.rice.edu/HPFF/home.html`.

[KR96] C. Kenyon and E. Remila. Approximate strip packing. *Proc. 37th FOCS*, 31-36, 1996.

[I93] Y. Ioannidis. Universality of serial histograms. *Proc. 19th Intl. Conf. on Very Large Databases*, 256-267, 1993.

[IP95] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. *Proc. ACM SIGMOD*, 233-244, 1995.

[JJ81] J. Jain and A. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on communications*, 29, 1799–1808, 1981.

[KMS97] S. Khanna, S. Muthukrishnan and S. Skiena. Efficient array partitioning. *Proc. 24th ICALP*, 616-626, 1997.

[L82] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Computing* 11, 329–343, 1982.

[LNS90] R. Lipton, J. Naughton and D. Schneider. Practical selectivity estimation through adaptive sampling. *Proc. ACM SIGMOD*, 1-11, 1990.

[LSW97] B. Lent, A. Swami and J. Widom. Clustering association rules. *Proc. 13th Intl Conf on Data Engineering*, 1997.

[M93] F. Manne. *Load Balancing in Parallel Sparse Matrix Computations.* Ph.D. thesis, Department of Informatics, University of Bergen, Norway, 1993.

[M94] C. Manning. *Introduction to Digital Video Coding and Block Matching Algorithms.* `http://atlantis.ucc.ie/dvideo/dv.html.`

[MM+96] G. Martin, S. Muthukrishnan, R. Packwood and I. Rhee. Fast algorithms for variable size block matching motion estimation with minimal error. Submitted, 1996.

[MD88] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. *Proc. SIGMOD*, 28-36, 1988.

[P97] V. Poosala. *Histogram-based estimation techniques in databases.* Ph.D. dissertation, U. Wisconsin, 1997.

[PI97] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. *Proc. Very Large Data Bases*, 1997.

[RS97] R. Rastogi and K. Shim. Personal communication, 1997.

[S94] Block Matching Methods. `http://www.ee.princeton.edu/~santanu/journal/paper/node3.html`.

[SL+93] W. Sun, Y. Ling, N. Rishe and Y. Deng. An instant and accurate size estimation method for joins and selections in a retrieval-intensive environment. *Proc. ACM SIGMOD*, 79-88, 1993.

[UZ+96] M. Ujaldon, E. Zapata, B. Chapman and H. Zima. Vienna Fortran/HPF extensions for sparse and irregular problems and their compilation. Manuscript, 1996.

[ZBG86] H. Zima, H. Bast and M. Gerndt. Superb: A tool for semi-automatic MIMD/AIMD parallelization. *Parallel Computing*, 1–18, 1986.