On Parsimonious Explanations For 2-D Tree- and Linearly-Ordered Data

Howard Karloff[1], Flip Korn[2], Konstantin Makarychev[3], and Yuval Rabani[4]

1 AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. E-mail: howard@research.att.com.
2 AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. E-mail: flip@research.att.com.
3 IBM Research, Box 218, Yorktown Heights, NY 10598. E-mail: konstantin@us.ibm.com.
4 The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel. E-mail: yrabani@cs.huji.ac.il.

Abstract

This paper studies the "explanation problem" for tree- and linearly-ordered array data, a problem motivated by database applications and recently solved for the one-dimensional tree-ordered case. In this paper, one is given a matrix $A = (a_{ij})$ whose rows and columns have semantics: special subsets of the rows and special subsets of the columns are meaningful, others are not. A submatrix in A is said to be meaningful if and only if it is the cross product of a meaningful row subset and a meaningful column subset, in which case we call it an "allowed rectangle." The goal is to "explain" A as a sparse sum of weighted allowed rectangles. Specifically, we wish to find as few weighted allowed rectangles as possible such that, for all i, j, a_{ij} equals the sum of the weights of all rectangles which include cell (i, j).

In this paper we consider the natural cases in which the matrix dimensions are tree-ordered or linearly-ordered. In the tree-ordered case, we are given a rooted tree T_1 whose leaves are the rows of A and another, T_2, whose leaves are the columns. Nodes of the trees correspond in an obvious way to the sets of their leaf descendants. In the linearly-ordered case, a set of rows or columns is meaningful if and only if it is contiguous.

For tree-ordered data, we prove the explanation problem NP-Hard and give a randomized 2-approximation algorithm for it. For linearly-ordered data, we prove the explanation problem NP-Hard and give a 2.56-approximation algorithm. To our knowledge, these are the first results for the problem of sparsely and exactly representing matrices by weighted rectangles.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.332

1 Introduction

This paper studies two related problems of "explaining" data parsimoniously. In the first part of this paper, we focus on providing a top-down "hierarchical explanation" of "tree-ordered" matrix data. We motivate the problem as follows. Suppose that one is given a matrix $A = (a_{ij})$ of data, and that the rows naturally correspond to the leaves of a rooted tree T_1, and the columns, to the leaves of a rooted tree T_2. For example, T_1 and T_2 could represent hierarchical IP addresses spaces with nodes corresponding to IP prefixes. Each node of either T_1 or T_2 is then said to correspond to the set of rows (or columns, respectively) corresponding to its leaf descendants. Say `128.*` (i.e., the set of all 2^{24} IP addresses beginning with "128", which happens to correspond to the `.edu` domain) is a node in T_1 and `209.85.225.*` (i.e., the set of all 2^8 IP addresses beginning with `209.85.225`, which is `www.google.com`'s

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 332–343
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

domain) is a node in T_2. Then `(128.*, 209.85.225.*)` could, say, represent the amount of traffic flowing from all hosts in the `.edu` domain (e.g., `128.8.127.3`) to all hosts in the `www.google.com` domain (e.g., `209.85.225.99`). It is easy to relabel the rows or columns so that each internal node of T_1 or T_2 corresponds to a contiguous block of rows or columns.

We need a few definitions. Let us say a *rectangle* in an $m \times n$ matrix A is a set $Rect(i_1, i_2, j_1, j_2) = \{i : i_1 \le i \le i_2\} \times \{j : j_1 \le j \le j_2\}$, for some $1 \le i_1 \le i_2 \le m$, $1 \le j_1 \le j_2 \le n$. Certain rectangles are *allowed*; others are not. Let $\mathcal{R}$ denote the set of allowed rectangles. Say a set of $w(R)$-weighted rectangles R *represents* $A = (a_{ij})$ if for any cell (i, j), the sum of $w(R)$ over cells that contain (i, j) is a_{ij}.

Returning to the Internet example, a pair (u, v), u a node of T_1, v a node of T_2, corresponds to a rectangle. Say that a rectangle is allowed, relative to T_1 and T_2, if it is the cross product of the set of rows corresponding to some node u in T_1 and the set of columns corresponding to some node v in T_2. In this scenario, we attempt to "explain" or "describe" the matrix by writing it as a sum of weighted allowed rectangles. Formally, we wish to assign a weight w_R to each allowed rectangle R such that the set of weighted rectangles represents A.

Of course there is always a solution: one can just assign weights to the 1×1 rectangles. But this is a trivial description of the matrix. Usually more concise explanations are preferable. For this reason we seek an "explanation" with as few nonzero terms as possible. More precisely, we seek to assign a weight w_R to each allowed rectangle R such that the set of weighted rectangles represents A, and such that the number of nonzero weights w_R assigned is minimized. (We define problems formally in Section 3.)

Here is a 1-dimensional example. Suppose that a media retailer sells items in exactly four categories: action-movie DVD's, comedy DVD's, books, and CD's. The retailer builds a hierarchy with four leaves, one for each of the categories of items. A node "DVD's" is the parent of leaves "action-movie DVD's" and "comedy DVD's". There is one more node, a root labeled "all", with children "DVD's", "books", and "CD's".

Suppose that one year, sales of action-movie DVD's increased by \$6000 and sales of the other three categories increased by \$8000 each. One could represent the sales data by giving those four numbers, one for each leaf of the hierarchy, yet one could more parsimoniously say that there was a general increase of \$8000 for all (leaf) categories, in addition to which there was a decrease of \$2000 for action-movie DVD's. This is represented by assigning \$8000 to node "all" and \$-2000 to "action-movie DVD's". While many different linear combinations may be possible, simple explanations tend to be most informative. Therefore, we seek an answer minimizing the explanation size (the number of nonzero terms required in the explanation).

Here is a definition of TREE×TREE. An instance consists of an $m \times n$ matrix $A = (a_{ij})$, along with two rooted trees, a tree T_1 whose leaf set is the set of rows of the matrix, and a tree T_2 whose leaf set is the set of columns. Let $L_i(v)$ be the leaf descendants of node v in tree T_i, $i \in \{1, 2\}$. Now $\mathcal{R}$ is just the set $\{L_1(u) \times L_2(v) \ : \ u$ is a node in T_1 and v is a node in $T_2\}$. The goal is to find the smallest set of weighted rectangles which represents A. We prove this problem NP-hard and give a randomized 2-approximation algorithm for it. APX-hardness is not known.

The second problem, ALLRECTS, is motivated by the need to concisely describe or explain linearly-ordered data. Imagine that one has two ordered parameters, such as horizontal and vertical location, or age and salary. No trees are involved now. Instead we allow any interval of rows (i.e., $\{i : i_1 \le i \le i_2\}$ for any $1 \le i_1 \le i_2 \le m$) and any interval of columns (i.e., $\{j : j_1 \le j \le j_2\}$ for any $1 \le j_1 \le j_2 \le n$). For example, $[800, 1000] \times [500, 1500]$ could be used to represent a geographical region extending eastward from 800 to 1000 miles and northward from 500 to 1500 miles, and $[35.0, 45.0] \times [80000, 95000]$ could be used to represent the subset of people 35-44 years old and earning a salary of \$80000-\$95000. Then we can use the former "rectangles" to summarize the change (say, in population counts) with respect to location, or use the latter with respect to demographic attributes age and salary.

Hence in AllRects the set $\mathcal{R}$ of allowed rectangles is the cross product between the set of row intervals and the set of column intervals. As a linear combination of how few arbitrary rectangles can we write the given matrix? We prove this problem NP-hard and give a 2.56-approximation algorithm for it. Again, APX-hardness is unknown.

2 Related Work

To our knowledge, while numerous papers have studied similar problems, none proposes any algorithm for either of the two problems we study. One very relevant prior piece of work is a polynomial-time exact algorithm solving the 1-dimensional version of Tree×Tree (more properly called the "tree" case in 1-d, since only one tree is involved) [1]. Here, as in the media-retailer example above, we have a sequence of integers and a tree whose leaves are the elements of the sequence. Indeed, we use this algorithm heavily in constructing our randomized constant-factor approximation algorithm for the tree×tree case.

Relevant to our work is [4] by Bansal, Coppersmith, and Schieber, which (in our language) studies the 1-d (exact) problem in which all intervals are allowed and all must have *nonnegative* weights, proves the problem NP-hard, and gives a constant-factor approximation algorithm.

Also very relevant is a paper by Natarajan [13], which studies an "inexact" version of the problem: instead of finding weighted rectangles whose sum of weights is a_{ij} exactly, for each matrix cell (i, j), these sums approximate the a_{ij}'s. (Natarajan's algorithm is more general and can handle any arbitrary set $\mathcal{R}$ of allowed rectangles; however, the algorithm is very slow.) More precisely, in the output set of rectangles, define a'_{ij} to be the sum of the weights of the rectangles containing cell (i, j). Natarajan's algorithm ensures, given a tolerance $\Delta > 0$, that the L_2 error $\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (a'_{ij} - a_{ij})^2}$ is at most Δ. (Natarajan's algorithm cannot be used for $\Delta = 0$.) The upper bound on the number of rectangles produced by Natarajan's algorithm is a factor of approximately $18 \ln(||A||_2/\Delta)$ (where $||A||_2$ is the square root of the sum of squares of the entries of A) larger than the optimal number used by an adversary who is allowed, instead, only L_2-error $\Delta/2$. Furthermore, Natarajan's algorithm is very slow, much slower than our algorithms. See the full version of our paper for details.

Frieze and Kannan in [9] show how to inexactly represent a matrix as a sum of a small number of rank-1 matrices, but their method is unsuitable to solve our problem, as not only is there no way to restrict the rank-1 matrices to be rectangles, the error is of L_1 type rather than L_∞. In other words, the *sum* of the mn errors is bounded by Δmn, rather than individual errors' being bounded by Δ.

Our problem may remind readers of compressed sensing, the decoding aspect of which requires one to seek a solution x with fewest nonzeroes to a linear system $Hx = b$. The key insight of compressed sensing is that when H satisfies the "restricted isometry property" [16, 6, 8], as do almost all random matrices, the solution x of minimum L_1 norm is also the sparsest solution. The problem with applying compressed sensing to the problems mentioned herein, when the matrix A is $m \times n$, is that the associated matrix H, which has mn rows and a number of columns equal to the number of allowed rectangles, is anything but random. On a small set of test instances, the authors found the solutions of minimum L_1 norm (using linear programming) and discovered that they were far from sparsest.

Other authors have studied other ways of representing matrices. Applegate et al. [2] studied the problem of representing a *binary* matrix, starting from an all-zero matrix, by an *ordered* sequence of rectangles, each of whose entries is all 0 or all 1, in which a_{ij} should equal the entry of the *last* rectangle which contains cell (i, j). Anil Kumar and Ramesh [3] study the same model in which only all-1 rectangles are allowed (in which case the order clearly doesn't matter). Two papers [14, 11] study the Gale-Berlekamp switching game and can be thought of as a variant of our problem over $\mathbb{Z}_2$.

3 Formal Definitions and Examples

Given an $m \times n$ matrix $A = (a_{ij})$ and $1 \le i_1 \le i_2 \le m$, $1 \le j_1 \le j_2 \le n$, recall that $Rect(i_1, i_2, j_1, j_2) = \{(i, j) | i_1 \le i \le i_2, j_1 \le j \le j_2\}$. Define $Rects = \{Rect(i_1, i_2, j_1, j_2) | 1 \le i_1 \le i_2 \le m, 1 \le j_1 \le j_2 \le n\}$. For each of the two problems, we are given a subset $\mathcal{R} \subseteq Rects$; the only difference between the two problems we discuss is the definition of $\mathcal{R}$. The goal is to find a smallest subset $OPT_2(A)$ of $\mathcal{R}$, and an associated weight $w(R)$ (positive or negative) for each rectangle R, such that every cell (i, j) is covered by rectangles whose weights sum to a_{ij}, that is,

$$a_{ij} = \sum_{R: R \in OPT_2(A) \text{ and } R \ni (i,j)} w(R), \tag{1}$$

the "2" in "$OPT_2(A)$" referring to the fact that A is 2-dimensional.

While the algorithm for the tree×tree case appears (in Section 4) before that for the arbitrary-rectangles case (in Section 5), here we define AllRects, the latter, first, since it's easier to define. As mentioned above, we call the case of $\mathcal{R} = Rects$ AllRects.

Example. Since the matrix

$$A = \begin{bmatrix} 2&2&2&2\\5&3&1&2\\6&4&1&3\\5&5&2&2 \end{bmatrix} = 2\begin{bmatrix} 1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1 \end{bmatrix} + 3\begin{bmatrix} 0&0&0&0\\1&1&0&0\\1&1&0&0\\1&1&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0\\0&0&0&0\\1&1&1&1\\0&0&0&0 \end{bmatrix} - 2\begin{bmatrix} 0&0&0&0\\0&1&1&0\\0&1&1&0\\0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&0 \end{bmatrix},$$

A can be written as a linear combination with $w(\{1,2,3,4\} \times \{1,2,3,4\}) = 2$, $w(\{2,3,4\} \times \{1,2\}) = 3$, $w(\{3\} \times \{1,2,3,4\}) = 1$, $w(\{2,3\} \times \{2,3\}) = -2$, and $w(\{2\} \times \{3\}) = 1$. Hence $|OPT_2(A)| \le 5$.

We need some notation in order to define Tree×Tree, in which we are also given trees T_1 and T_2. We use R_i to denote the row vector in the ith row of the input matrix, $1 \le i \le m$. For a node $u \in T_1$, let $S^1_u = \{R_l \ : \ l \text{ is a leaf descendant in } T_1 \text{ of } u\}$. Similarly, we use C_j to denote the column vector in the jth column of the input matrix, $1 \le j \le n$. For a node $v \in T_2$, let $S^2_v = \{C_l \ : \ l \text{ is a leaf descendant in } T_2 \text{ of } v\}$. Note that, since T_1 and T_2 are trees, $\{S^1_u | u \in T_1\}$ and $\{S^2_v | v \in T_2\}$ are laminar.

In this notation, in Tree×Tree, $\mathcal{R} = \{S^1_u | u \in T_1\} \times \{S^2_v | v \in T_2\}$.

Example. Using trees T_1, T_2 having a root with four children (and no other nodes) apiece, we may use any single row or all rows, and any single column or all columns. For example, since the matrix

$$\begin{aligned} A = \begin{bmatrix} 5&3&4&5\\3&0&2&4\\2&2&1&3\\3&3&2&3 \end{bmatrix} &= 3\begin{bmatrix} 1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1 \end{bmatrix} + 2\begin{bmatrix} 1&1&1&1\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} - 1\begin{bmatrix} 0&0&0&0\\0&0&0&0\\1&1&1&1\\0&0&0&0 \end{bmatrix} - 1\begin{bmatrix} 0&0&1&0\\0&0&1&0\\0&0&1&0\\0&0&1&0 \end{bmatrix} \\ &- 2\begin{bmatrix} 0&1&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} - 3\begin{bmatrix} 0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0\\0&0&0&1\\0&0&0&0\\0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&0&0 \end{bmatrix}, \end{aligned}$$

we can write A as a sum with $w(\{1,2,3,4\} \times \{1,2,3,4\}) = 3$, $w(\{1\} \times \{1,2,3,4\}) = 2$, $w(\{3\} \times \{1,2,3,4\}) = -1$, $w(\{1,2,3,4\} \times \{3\}) = -1$, $w(\{1\} \times \{2\}) = -2$, $w(\{2\} \times \{2\}) = -3$, $w(\{2\} \times \{4\}) = 1$, and $w(\{3\} \times \{4\}) = 1$. Since there are eight matrices, $|OPT_2(A)| \le 8$.

Note that we use the same notation, $OPT_2(A)$, for the optimal solutions of both All-Rects and Tree×Tree.

4 Approximation Algorithm for Tree×Tree

We defer the (interesting) proof of NP-Hardness of Tree×Tree to the full version of the paper. Our algorithm will rely upon the exact algorithm, due to Agarwal et al. [1], for the case in which the matrix has just one column (that is, the 1-dimensional case).

▶ **Definition 1.** Given a fixed rooted tree T_1 with m leaves, and an m-vector $V = (v_i)$, let $OPT_1(V)$ denote a smallest set of intervals $I = \{i : i_1 \leq i \leq i_2\} \subseteq [1, m]$ and associated weights $w(I)$, each I corresponding to a node of T_1, such that for all i, $v_i = \sum_{I: I \in OPT_1(V) \text{ and } I \ni i} w(I)$.

Clearly $|OPT_1(V)|$ equals $|OPT_2(V')|$, where V' is the $m \times 1$ matrix containing V as a column. The difference is that $OPT_1(V)$ is a set of vectors while $OPT_2(V')$ is a set of rectangles. We emphasize that V is a vector and that the definition depends on T_1 and not T_2 by putting the "1" in "$OPT_1(V)$". The key point is that [1] showed how to compute $OPT_1(V)$ exactly.

In order to charge the algorithm's cost against $OPT_2(A)$, we need to know some facts about $OPT_2(A)$. Recall that $OPT_2(A)$ is a smallest subset of $\mathcal{R}$ such that there are weights $w(R)$ such that equation (1) holds.

▶ **Definition 2.**

1. For each rectangle R and associated weight w_R, let R'_{w_R} denote the $m \times n$ matrix which is 0 for every cell (i, j), except that $(R'_{w_R})_{ij} := w_R$ if $(i, j) \in R$.
2. Given a vertex v of T_2, let D_v be the set of all $R \in OPT_2(A)$ such that R has column set exactly equal to S^2_v.
3. Now let $K_v = \sum_{R \in D_v} R'_{w_R}$. By definition of D_v, all columns j of K_v for $j \in D_v$ are the same. Let V_v be column j of K_v for any $j \in D_v$.

▶ **Lemma 3.** *The column vectors (V_v) satisfy the following:*

1. *For all leaves l in T_2, the vector C_l equals the sum of V_v over all ancestors v of l in T_2.*
2. *For all leaves l' and l'' in T_2 with a common ancestor u, the vector $C_{l'} - C_{l''}$ equals the sum of V_v over all vertices v on the path from u down to l' (not including $v = u$) minus the sum of V_v over all vertices v on the path from u down to l'' (not including $v = u$).*
3. *The union, over all vertices $v \in T_2$, of $OPT_1(V_v) \times \{S^2_v\}$ (which obviously has size $|OPT_1(V_v)|$), with the corresponding weights, is an optimal solution for* Tree×Tree *on A.*
4. *$|OPT_2(A)| = \sum_{v \in T_2} |OPT_1(V_v)|$.*

Proof. The nodes v which correspond to sets of columns containing column C_l are exactly the ancestors in T_2 of l. Hence, Part 1 follows.

Part 2 is an immediate corollary of Part 1. Clearly, by Part 1, the union over all vertices $v \in T_2$ of $OPT_1(V_v) \times \{S^2_v\}$ is a feasible solution for Tree×Tree on A. It is also optimal, and here is a proof. The size of the optimal solution $OPT_2(A)$ equals the sum, over vertices $v \in T_2$, of the number of rectangles in $OPT_2(A)$ having column set S^2_v. Fix a vertex $v \in T_2$. Since the weighted sum of the rectangles in $OPT_2(A)$ with column set S^2_v is V_v, and each has a row set S^1_u for some $u \in T_1$, the number of such rectangles must be at least $OPT_1(V_v)$. If the number of rectangles with column set S^2_v strictly exceeded $OPT_1(V_v)$, we could replace all rectangles in $OPT_2(A)$ having column set S^2_v by a smaller set of weighted rectangles having column set S_v^2, each of whose columns is the same, and summing to V_v in each column; since the new set and the old set have the same weighted sum, the new solution would still sum to A, and have better-than-optimal size, thereby contradicting optimality of $OPT_2(A)$. Part 3 follows.

Part 4 follows from Part 3. ◀

Lemma 3 will be instrumental in analyzing the algorithm.

While the algorithm is very simple to state, it was nontrivial to develop and analyze. In the algorithm, we use the algorithm by Agarwal et al. [1] to obtain $OPT_1(V)$ given a vector V.

Algorithm for TREE×TREE

1. For every internal node u in the tree T_2, pick a random child u^* of u and let $c(u) = u^*$. Let $path(u)$ be the random path going from u to a leaf: $u \mapsto c(u) \mapsto c(c(u)) \mapsto \cdots \mapsto l(u)$, where we denote the last node on the path, the leaf, by $l(u)$.
2. Where $root$ denotes the root of T_2, for every node u in T_2, in increasing order by depth, do:
   - If u is the root of T_2, then
     - Output $OPT_1(C_{l(root)}) \times \{S^2_{root}\}$ with the corresponding weights (those of the optimal solution for $C_{l(root)}$).
   - Else
     - Let $p(u)$ be the parent of u.
     - Output $OPT_1(C_{l(u)} - C_{l(p(u))}) \times \{S_u^2\}$ with the corresponding weights.

▶ **Theorem 4.** *The expected cost of the algorithm is at most* $2|OPT_2(A)|$.

In the main part of the paper we prove a weaker guarantee for exposition: the expected cost of the algorithm is at most $4|OPT_2(A)|$. We defer the improvement to the full version of the paper. The algorithm can be easily derandomized using dynamic programming.

Proof. Every column C_u is covered by rectangles with sum

$$(C_u - C_{l(p(u))}) + (C_{l(p(u))} - C_{l(p(p(u)))}) + \cdots + C_{l(root)} = C_u.$$

Thus the algorithm produces a valid solution. We now must estimate the expected cost of the solution. The total cost incurred by the algorithm is

$$|OPT_1(C_{l(root)})| + \sum_{u \neq root} |OPT_1(C_{l(u)} - C_{l(p(u))})|.$$

Assume, without loss of generality, that all nodes in the tree either have two or more children or are leaves. Denote the number of children of a node v, the degree of v, by $d(v)$. Denote by $\mathbf{1}$ the indicator function. Observe that for the root node we have

$$|OPT_1(C_{l(root)})| = \left| OPT_1\Big(\sum_{v \in path(root)} V_v \Big) \right| \leq \sum_{v \in path(root)} |OPT_1(V_v)|;$$

for a nonroot vertex u, we have by Lemma 3 (2), keeping in mind that $l(\cdot)$, $c(\cdot)$, and $path(\cdot)$ are random,

$$\begin{aligned} |OPT_1(C_{l(u)} - C_{l(p(u))})| &= \left| OPT_1\Big(\sum_{v \in path(u)} V_v - \sum_{v \in path(c(p(u)))} V_v \Big) \right| \\ &\leq \Big(\sum_{v \in path(u)} |OPT_1(V_v)| + \sum_{v \in path(c(p(u)))} |OPT_1(V_v)| \Big) \cdot \mathbf{1}(u \neq c(p(u))). \end{aligned}$$

Here we used the triangle inequality for the function $|OPT_1(\cdot)|$.

Consider the second sum in the right-hand side. For every child u' of $p(u)$, the random node $c(p(u))$ takes value u' with probability $1/d(p(u))$. Thus

$$\begin{aligned}
&\mathbb{E}\Big[\sum_{v\in path(c(p(u)))} |OPT_1(V_v)|\cdot \mathbf{1}(u\neq c(p(u)))\Big]\\
&\quad= \frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u} \mathbb{E}\Big[\Big(\sum_{v\in path(c(p(u)))} |OPT_1(V_v)|\Big) \mid c(p(u))=u'\Big]\\
&\quad= \frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u} \mathbb{E}\Big[\sum_{v\in path(u')} |OPT_1(V_v)|\Big].
\end{aligned}$$

$\Pr\Big(u\neq c(p(u))\Big)$ equals $(d(p(u))-1)/d(p(u))$. Denote this expression by α_u. The total expected size of the solution returned by the algorithm is bounded by

$$\mathbb{E}\Big[\sum_{v\in path(root)} |OPT_1(V_v)|\Big] + \sum_{u\neq root}\alpha_u \mathbb{E}\Big[\sum_{v\in path(u)} |OPT_1(V_v)|\Big] \tag{2}$$

$$\begin{aligned}
&+\textstyle\sum_{u\neq root}\frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u}\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big]\\
&= \mathbb{E}\Big[\textstyle\sum_{v\in path(root)}|OPT_1(V_v)|\Big] + \sum_{u\neq root}\alpha_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big]\\
&+\textstyle\sum_{u'\neq root}\Big(\sum_{u\neq root}\frac{\mathbf{1}(u' \text{ is a sibling of } u)}{d(p(u'))}\Big)\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big].
\end{aligned}\tag{3}$$

Notice that, for a fixed $u'\neq root$,

$$\sum_{u\neq root}\frac{\mathbf{1}(u' \text{ is a sibling of } u)}{d(p(u'))} = \frac{d(p(u'))-1}{d(p(u'))} = \alpha_{u'} < 1. \tag{4}$$

Hence, the total cost of the solution is bounded by

$$\sum_u \mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big] + \sum_{u'\neq root}\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big] \leq 2\sum_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big].$$

Finally, observe that node v belongs to $path(v)$ with probability 1; it belongs to the $path(p(v))$ with probability at most $1/2$; it belongs to the path $path(p(p(v)))$ with probability at most $1/4$, etc. It belongs to $path(u)$ with probability 0 if u is not an ancestor of v. Thus

$$\begin{aligned}
2\sum_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big] &= 2\sum_v |OPT_1(V_v)|\cdot\Big(\sum_u \Pr\Big(v\in path(u)\Big)\Big)\\
&\leq 2\sum_v|OPT_1(V_v)|\cdot\Big(1+1/2+1/4+\cdots\Big)\\
&< 4\sum_v|OPT_1(V_v)| \leq 4|OPT_2(A)|.
\end{aligned}$$

We have proven that the algorithm finds a 4-approximation. A slightly more careful analysis, in the full version of the paper, shows that the approximation ratio of the algorithm is at most 2. ◀

What is the running time of the 2-approximation algorithm? The time needed to run the 1-dimensional algorithm of [1] is $O(dn)$ where there are n leaves in each tree and the *smaller* of the two depths is d. One can verify that the running time of our 2-approximation algorithm is a factor $O(n)$ larger, or $O(dn^2)$. In most applications at least one of the trees would have depth $O(\log n)$, giving $O(n^2\log n)$ in total.

5 Approximation Algorithm For AllRects

5.1 The 1-Dimensional Problem

First we consider the one-dimensional case, for which we will give a $(23/18+\varepsilon)$-approximation algorithm; $23/18 < 1.278$. We are given a sequence $a_1, a_2, \dots, a_n$ of numbers and we need to find a collection of closed intervals $[i, j]$ with arbitrary real weights w_{ij} so that every integral point $k \in \{1, \dots, n\}$ is covered by a set of intervals with total weight a_k. That is, for all k,

$$\sum_{i,j:k\in[i,j]} w_{ij} = a_k. \tag{5}$$

Our goal is to find the smallest possible collection. We shall use the approach of Bansal, Coppersmith, and Schieber [4] (in their problem all $a_i \geq 0$ and all $w_{ij} > 0$). Set $a_0 = 0$ and $a_{n+1} = 0$. Observe that if $a_k = a_{k+1}$, then in the optimal solution every interval covering k also covers $k+1$. On the other hand, since every rectangle covering both k and $k+1$ contributes the same weight to a_k and a_{k+1}, if $a_k \neq a_{k+1}$, then there should be at least one interval that either covers k but not $k+1$, or covers $k+1$ but not k. By the same reason, the difference $a_{k+1} - a_k$, which we denote by $\Delta_k = a_{k+1} - a_k$, equals the difference between the weight of intervals with the left end-point at $k+1$ and the weight of rectangles with the right endpoint at k:

$$\Delta_k = \sum_{j:k+1\leq j} w_{k+1,j} - \sum_{i:i\leq k} w_{ik}. \tag{6}$$

Note that if we find a collection of rectangles with weights satisfying (6), then this collection of intervals is a valid solution to our problem, i.e., then equality (5) holds. Define a directed graph on vertices $\{0, \dots, n\}$. For every interval $[i, j]$, we add an arc going from $i-1$ to j. Then the condition (6) can be restated as follows: The sum of weights of arcs outgoing from k minus the sum of weights of arcs entering k equals Δ_k. Our goal is to find the smallest set of arcs with non-zero weights satisfying this property. Consider an arbitrary solution and one of the weakly connected components S. The sum $\sum_{k\in S} \Delta_k = 0$, since every arc is counted twice in the sum, once with the plus sign and once with the minus sign. Since S is a connected component the number of arcs connecting nodes in S is at least $|S| - 1$. Thus a lower bound on the number of arcs or intervals in the optimal solution is the minimum of

$$\sum_{t=1}^{M} (|S_t| - 1) = n + 1 - M$$

among all partitions of the set of items $\{0, \dots, n\}$ into M disjoint sets $S_1, \dots, S_M$ such that $\sum_{k\in S_t} \Delta_k = 0$ for all t. On the other hand, given such a partition $(S_1, \dots, S_M)$, we can easily construct a set of intervals. Let k_t be the minimal element in S_t. For every element k in $S_t \setminus \{k_t\}$, we add an interval $[k_t + 1, k]$ with weight $-\Delta_k$. We now verify that these intervals satisfy (6). If k belongs to S_t and $k \neq k_t$, then there is only one interval in the solution with right endpoint at k. This interval is $[k_t + 1, k]$ and its weight is $-\Delta_k$. The solution does not contain intervals with left endpoint at $k+1$ (since $k \neq k_t$). Thus (6) holds as well. If k belongs to S_t and $k = k_t$, the solution does not contain intervals with the right endpoint at k, but for all $k' \in S_t$ there is an interval $[k+1, k']$ with weight $-\Delta_{k'}$. The total weight of these intervals equals

$$\sum_{k'\in S_t;k'\neq k} -\Delta_{k'} = -\sum_{k'\in S_t} \Delta_{k'} + \Delta_k = \Delta_k.$$

Condition (6) again holds.

Thus the problem is equivalent to the problem of partitioning the set of items $\{0,\ldots,n\}$ into a family of M sets $\{S_1,\ldots,S_M\}$ satisfying the condition $\sum_{k\in S_t}\Delta_k = 0$ for all t, so as to minimize $\sum_t(|S_t|-1) = (n+1)-M$. Notice that the sum of all Δ_k equals 0. Moreover, every set with the sum of Δ_k equal to 0 corresponds to an instance of the 1-dimensional rectangle covering problem. We shall refer to the problem as Zero-Weight Partition.

We now describe the approximation algorithm for Zero-Weight Partition which is a modification of the algorithm of Bansal, Coppersmith, and Schieber [4] designed for a slightly different problem (that of minimizing setup times in radiation therapy).

▶ Remark. For Zero-Weight Partition, our algorithm gives a slightly better approximation guarantee than that of [4]: $23/18 \approx 1.278$ vs $9/7 \approx 1.286$. The difference between algorithms is that the algorithm of Bansal, Coppersmith, and Schieber [4] performs either the first and third steps (in terms of our algorithm; see below), or the second and third steps; while our algorithm always performs all three steps.

In the first step the algorithm picks all singleton sets $\{k\}$ with $\Delta_k = 0$ and pairs $\{i,j\}$ with $\Delta_i = -\Delta_j$. It removes the items covered by any of the chosen sets. At the second step, with probability 2/3 the algorithm enumerates all triples $\{i,j,k\}$ with $\Delta_i+\Delta_j+\Delta_k = 0$ and finds the largest 3-set packing among them using the $(3/2+\varepsilon)$-approximation algorithm due to Hurkens and Schrijver [10], i.e., it finds the largest (up to a factor of $(3/2+\varepsilon)$) disjoint family of triples $\{i,j,k\}$ with $\Delta_i+\Delta_j+\Delta_k = 0$. Otherwise (with probability 1/3), the algorithm enumerates all quadruples $\{i,j,k,l\}$ having $\Delta_i+\Delta_j+\Delta_k+\Delta_l = 0$ and finds the largest 4-set packing among them using the $(2+\varepsilon)$-approximation algorithm due to Hurkens and Schrijver [10]. At the third, final, step the algorithm covers all remaining items, whose sum of Δ_k's is zero, with one set.

Before we start analyzing the algorithm, let us consider a simple example. Suppose that $(a_1,a_2,a_2,a_4,a_5,a_6) = (15,8,10,17,18,15)$. First we surround the vector with two 0's: $(a_0,a_1,a_2,a_2,a_4,a_5,a_6,a_7) = (0,15,8,10,17,18,15,0)$. Then compute the vector of Δ_k's: $(\Delta_0,\Delta_1,\Delta_2,\Delta_2,\Delta_4,\Delta_5,\Delta_6) = (15-0, 8-15, 10-8, 17-10, 18-17, 15-18, 0-15) = (15,-7,2,7,1,-3,-15)$. Notice that $(-15)+7+(-2)+(-7)+(-1)+3+15 = 0$. We partition the set into sets of weight 0: $\{\Delta_0,\Delta_6\},\{\Delta_1,\Delta_3\},\{\Delta_2,\Delta_4,\Delta_5\}$. This partition corresponds to the following solution of the 1-dimensional problem: interval $[1,6]$ with weight 15, interval $[2,3]$ with weight -7, interval $[3,4]$ with weight -1, interval $[3,5]$ with weight 3.

▶ **Lemma 5.** *For every positive $\varepsilon > 0$, the approximation ratio of the algorithm when using ε is at most $23/18 + O(\varepsilon)$, with $23/18 < 1.278$.*

Proof. First, observe that the partitioning returned by the algorithm is a valid partitioning, i.e., every item belongs to exactly one set and the sum of Δ_k's in every set equals 0. We show that the first step of the algorithm is optimal. That is, there exists an optimal solution that contains exactly the same set of singletons and pairs as in the partition returned by the algorithm. Suppose that the optimal solution breaks one pair $\{i,j\}$ ($\Delta_i = -\Delta_j$) and puts i in S and j in T. Then we can replace sets S and T with two new sets $\{i,j\}$ and $S\cup T\setminus\{i,j\}$. The new solution has the same cost as before; the sum of Δ_k's in every set is 0, but the pair $\{i,j\}$ belongs to the partitioning. Repeating this procedure several times, we can transform an arbitrary optimal solution into an optimal solution that contains the same set of singletons and pairs as the solution obtained by the approximation algorithm.

For the sake of the presentation let us assume that $\varepsilon = 0$ (that is, we assume that the approximation algorithms due to Hurkens and Schrijver [10], we use in our algorithm, have approximation guarantees at most 3/2 and 2). Let p_k be the number of sets of size k in the optimal solution. The cost of the optimal solution is $p_2 + 2p_3 + 3p_4 + 4p_5 + \cdots$, because the objective function charges $|S| - 1$ to a set of size $|S|$. Our approximation algorithm also finds p_1 singleton sets and p_2 pairs. Then with probability $2/3$, it finds $s_3 \geq (2/3)p_3$ triples and covers the remaining $3 \cdot (p_3 - s_3) + 4p_4 + 5p_5 + \cdots$ vertices with one set; and with probability $1/3$, it finds $s_4 \geq p_4/2$ quadruples and covers the remaining $3p_3 + 4 \cdot (p_4 - s_4) + 4p_4 + 5p_5 + \cdots$ vertices with one set. Thus the expected cost of the solution returned by the algorithm equals

$$\frac{2}{3}\Big(p_2 + 2 \cdot \frac{2p_3}{3} + 3 \cdot \frac{p_3}{3} + 4p_4 + \sum_{k \geq 5} kp_k - 1\Big) + \frac{1}{3}\Big(p_2 + 3 \cdot \frac{p_4}{2} + 3p_3 + 4 \cdot \frac{p_4}{2} + \sum_{k \geq 5} kp_k - 1\Big)$$
$$= p_2 + \frac{23}{9}p_3 + \frac{23}{6}p_4 + \sum_{k \geq 5} kp_k - 1. \quad (7)$$

Therefore, the approximation ratio of the algorithm, assuming that $\varepsilon = 0$, is

$$\frac{p_2 + \frac{23}{9}p_3 + \frac{23}{6}p_4 + \sum_{k \geq 5} kp_k - 1}{p_2 + 2p_3 + 3p_4 + \sum_{k \geq 5}(k-1)p_k} \leq \max\left\{\frac{1}{1}, \frac{\frac{23}{9}}{2}, \frac{\frac{23}{6}}{3}, \frac{5}{4}, \frac{6}{5}, \dots\right\} = \frac{23}{18}.$$

It is easy to verify that if $\varepsilon > 0$, the approximation ratio of the algorithm is at most $23/18 + O(\varepsilon)$. ◀

In the full version of the paper we prove that finding the exact solution of the problem is NP-hard.

5.2 The 2-Dimensional Case

We now consider the 2-dimensional case (which does not appear in [4]). We are given an $m \times n$ matrix $A = (a_{ij})$ $(1 \leq i \leq m,\ 1 \leq j \leq n)$ and we need to cover it with the minimum number of weighted rectangles $Rect(i_1, i_2, j_1, j_2)$ (for arbitrary i_1, i_2, j_1, j_2); we use $w(i_1, i_2, j_1, j_2)$ for the weight of $Rect(i_1, i_2, j_1, j_2)$. We assume that $a_{ij} = 0$ for i and j outside the rectangle $\{1, \dots, m\} \times \{1, \dots, n\}$.

By analogy to the 1-dimensional case, define $\Delta_{ij} = a_{i,j} - a_{i,j+1} + a_{i+1,j+1} - a_{i+1,j}$. Call a pair (i, j) with $0 \leq i \leq m$, $0 \leq j \leq n$, with $\Delta_{ij} \neq 0$ an *array corner*. Imagine that the matrix is written in an $m \times n$ table, and Δ_{ij}'s are written at the grid nodes. The key point is that every rectangle covers exactly one, two, or four of the cells $(i+1, j+1)$, (i, j), $(i, j+1)$, $(i+1, j)$ bordering a grid point, and that those covering two or four of those cells cannot affect Δ_{ij}. This means that only rectangles having a corner at the intersection of the ith and jth grid line contribute to Δ_{ij}. (This is why the definition of Δ_{ij} was "by analogy" to the 1-d case.) This means that the number of rectangles in the optimal solution must be at least one quarter of the number of array corners, the "one-quarter" arising from the fact that each rectangle has exactly four corners and can hence be responsible for at most four of the array corners.

It is easy now to give a 4-approximation algorithm, which we sketch without proof, based on this observation. Build a matrix M, initially all zero, which will eventually equal the input matrix A. Until no more array corners exist in $A - M$, find an array corner (i, j) with $i < m$ and $j < n$. (As long as array corners exist, there must be one with $i < m$ and $j < n$.) Let $\Delta \neq 0$ be Δ_{ij}. Add to M a rectangle of weight Δ with upper left corner at (i, j) and extending as far as possible to the right and downward, eliminating the array corner at (i, j) in $A - M$.

It is easy to see that (1) when the algorithm terminates, $M = A$, and that (2) the number of rectangles used is at most the number of array corners in A, and hence at most $4|OPT_2(A)|$.

Now we give, instead, a more sophisticated, $23/9+\varepsilon < 2.56$-approximation algorithm for the 2D problem. The idea is to make more efficient use of the rectangles. Instead of using only one corner of each (in contrast to the adversary, who might use all four), now we will use two. In fact, we will deal separately with different horizontal (between-consecutive-row) grid lines, using a good 1-dimensional approximation algorithm to decide how to eliminate the array corners on that grid line. Every time the 1-d algorithm tells us to use an interval $[j_1, j_2]$, we will instead inject a rectangle which starts in column j_1 and ends in column j_2, and extends all the way to the bottom. Because we use 2 of each rectangle's 4 corners, we pay a price of a factor of 4/2 over the 1-d approximation ratio of $23/18 + O(\varepsilon)$. Hence we will get $23/9 + O(\varepsilon)$.

Here are the details. Fix i and consider the restriction of the zero-weight partition problem to the ith horizontal grid line, i.e., the 1-dimensional zero-weight partition problem with $\Delta_j = \Delta_{ij}$. Denote by OPT^i the cost of the optimal solution. The number of rectangles touching the ith horizontal grid line from above or below is at least OPT^i, since only these rectangles contribute Δ_{ij}'s. Every rectangle touches only two horizontal grid lines, thus the total number of rectangles is at least $\sum_{i=1}^m OPT^i/2$.

All rectangles generated by our algorithm will touch the bottom line of the table; that is why we lose a factor of 2. Note that if we could solve the 1-dimensional problem exactly we would be able to find a covering with $\sum_{i=1}^m OPT^i$ rectangles and thus get a 2 approximation. For each horizontal grid line i, the algorithm solves the 1-dimensional problem (with $\Delta_j = \Delta_{ij}$) and finds a set of intervals $[j_1, j_2]$ with weights $w_{j_1 j_2}$. These intervals are the top sides of the rectangles generated by the algorithm. All bottom sides of the rectangles lie on the bottom grid line of the table. That is, for every interval $[j_1, j_2]$ the algorithm adds the rectangle $Rect(i, m, j_1, j_2)$ to the solution and sets its weight $w(i, m, j_1, j_2)$ to be $w_{j_1 j_2}$.

The total number of rectangles in the solution output by the algorithm is $\sum_{i=1}^m ALG_i$, where ALG_i is the cost of the solution of the 1-dimensional problem. Thus the cost of the solution is at most $2 \cdot (23/18 + O(\varepsilon))$ times the cost of the optimum solution. We now need to verify that the set of rectangles output by the algorithm is indeed is a solution.

Subtract the weight of each rectangle from all a_{ij}'s covered by the rectangle. We need to prove that the residual matrix

$$a'_{ij} = a_{ij} - \sum_{i_1, j_1, j_2 : (i,j) \in Rect(i_1, m, j_1, j_2)} w(i_1, m, j_1, j_2)$$

equals zero. Observe that $\Delta'_{ij} = a'_{i+1,j+1} + a'_{ij} - a'_{i+1,j} - a'_{i,j+1} = 0$ for all $0 \le i \le m-1$ (i.e., all rows i, possibly, except for the bottom line) and $0 \le j \le n$. Assume that not all a'_{ij} equal to 0. Let $a'_{i_0 j_0}$ be the first nonzero a'_{ij} with respect to the lexicographical order on (i,j). Then $a'_{i_0-1,j_0-1} = a'_{i_0-1,j_0} = a'_{i_0,j_0-1} = 0$. Thus $a'_{i_0 j_0} = 0$. We have proven the following theorem.

▶ **Theorem 6.** *For every positive ε, there exists a polynomial-time approximation algorithm for* AllRects *with approximation guarantee at most $23/9 + O(\varepsilon)$, with $23/9 = 2.5555.....$*

5.3 A Simplified Algorithm

Because of the dependence on ε, the running time of the previous algorithm can be large when ε is small. A simpler algorithm for the 1-dimensional case—namely, just use pairs and triples—can be shown to give ratio 4/3 for the 1-d case, and hence $8/3 = 2.6666...$ in 2-d, only slightly worse than 23/9. For the simplified 1-d algorithm, the running time is $O(n + k^2 \log k)$, if there are k Δ's. To run the 2-d algorithm, the running time becomes

$O(n^2+\sum_{i=1}^{n} k_i^2 \log k_i)$, where there are k_i corners on the ith row. Since the number of corners is $\Theta(OPT)$, the running time is at most $O(n^2)$ plus $O(\max_{k_1+k_2+\cdots+k_n=OPT} \sum_i k_i^2 \log k_i)$. Since $f(x) = x^2 \log x$ is convex, this quantity is maximized by making as many k_i's equal to n as possible. A simple proof then shows that the time is $O(n^2 + OPT \cdot (n \log n))$.

6 Acknowledgment

We thank Divesh Srivastava for initial conversations which inspired this work.

References

1 D. Agarwal, D. Barman, D. Gunopulos, N. Young, F. Korn, and D. Srivastava, "Efficient and Effective Explanation of Change In Hierarchical Summaries," Proc. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007, 6-15.

2 D. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. "Compressing Rectilinear Pictures and Minimizing Access Control Lists," SODA 2007, 1066-1075.

3 V. S. Anil Kumar and H. Ramesh, "Covering Rectilinear Polygons With Axis-Parallel Rectangles," STOC 1999, New York, 445-454.

4 N. Bansal, D. Coppersmith, and B. Schieber. "Minimizing Setup and Beam-On Times in Radiation Therapy," APPROX 2006, 27-38.

5 S. Bu, V. S. Lakshmanan, and R. T. Ng. MDL Summarization with Holes. In *VLDB '05: Proceedings of the 31st international conference on Very Large Databases*, pages 433–444, VLDB Endowment, 2005.

6 E. Candes and T. Tao. Decoding By Linear Programming. In *IEEE Transactions on Information Theory* 51 (12), 2005, 4203-4215.

7 G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data. In *Proc. of ACM SIGMOD '04*, Paris, France, 2004.

8 D. Donoho. For Most Large Underdetermined Systems of Linear Equations the Minimal ℓ^1-norm Solution is also the Sparsest Solution, In *Communications on Pure and Applied Mathematics* 59 (6), June 2006, 797-829.

9 A. Frieze and R. Kannan. Quick Approximation to Matrices and Applications. In *Combinatorica* 19 (2), 175-220, 1999.

10 C. Hurkens and A. Schrijver. "On the Size of Systems of Sets Every t of Which Have an SDR, With an Application to the Worst-Case Ratio of Heuristics for Packing Problems," *SIAM J. Discrete Math.* 2(1), 1989, 68-72.

11 M. Karpinski and W. Schudy. Linear Time Approximation Schemes for the Gale-Berlekamp Game and Related Minimization Problems. In *STOC '09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 313–322, New York, NY, USA, 2009. ACM.

12 V.S. Lakshmanan, R.T. Ng, C. Xing Wang, X. Zhou, and T. Johnson. The Generalized MDL Approach for Summarization. In *VLDB*, pages 766–777, 2002.

13 B. K. Natarajan. "Sparse Approximate Solutions To Linear Systems," *SIAM J. Comp.* 24(2), 1995, 227-234.

14 R. M. Roth and K. Viswanathan. On the Hardness of Decoding the Gale-Berlekamp Code. *IEEE Transactions on Information Theory*, 54(3):1050–1060, 2008.

15 S. Sarawagi. Explaining Differences in Multidimensional Aggregates. In *Proc. of the 25th International Conference on Very Large Databases (VLDB)*, pages 42–53, Scotland, UK, 1999.

16 Wikipedia page `http://en.wikipedia.org/wiki/Restricted_isometry_property`.