# Efficient Array Partitioning

Sanjeev Khanna[1] , S. Muthukrishnan[2] and Steven Skiena[3]

[1] Mathematical Sciences Research Center, Bell Laboratories, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974. sanjeev@research.bell-labs.com
[2] Information Sciences Center, Bell Laboratories, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974. muthu@research.bell-labs.com
[3] Dept. of Computer Science, State University of New York, Stony Brook, NY 11794-4400. skiena@cs.sunysb.edu. This work is partially supported by ONR award 400x116yip01 and NSF Grant CCR-9625669.

**Abstract.** We consider the problem of partitioning an array of $n$ items into $p$ intervals so that the maximum weight of the intervals is minimized. The currently best known bound for this problem is $O(n+p^{1+\epsilon})$ [HNC92] for any fixed $\epsilon < 1$. In this paper, we present an algorithm that runs in time $O(n \log n)$; this is the fastest known algorithm for arbitrary $p$.
We consider the natural generalization of this partitioning to two dimensions, where an $n \times n$ array of items is to be partitioned into $p^2$ blocks by partitioning the rows and columns into $p$ intervals each and considering the blocks induced by this partition. The problem is to find that partition which minimizes the maximum weight among the resulting blocks. This problem is known to be NP-hard [GM96]. Independently, Charikar *et. al.* have given a simple proof that shows that the problem is in fact NP-hard to approximate within a factor of two. Here we provide a polynomial time algorithm that determines a solution at most $O(1)$ times the optimum; the previously best approximation ratio was $O(\sqrt{p})$ [HM96].
Both the results above are proved for the case when the weight of an interval or block is the sum of the elements in it. These problems arise in load balancing for parallel machines and data partitioning in parallel languages. Applications in motion estimation by block matching in video and image compression give rise to the dual problem, that of minimizing the number of dividers $p$ so that the maximum weight of a block is at most $\delta$. We give an $O(\log n)$ approximation algorithm for this problem. All our results for two dimensional array partitioning extend to any higher fixed dimension.

## 1 Introduction

The problem of partitioning a set of items into roughly equal weight subsets is a fundamental one. We study two dual versions of this, namely, ($a$) given $B$, partition a given array into at most $B$ blocks so as to minimize the maximum weight of any block in the partition, and ($b$) given $\delta$, partition a given array into minimum number of blocks such that their individual weight is no larger than $\delta$. The definition of the weight function for a block, the type of partitions allowed, the dimensionality of the arrays, and the relevant version depends upon the application at hand. The problems we consider arise in load balancing for parallel processing, compilers for high-performance parallel languages, and motion estimation in videos by block matching, and hence have been extensively researched in several communities. In this paper, we present algorithms for these problems which are more efficient than the best ones so far, and give improved approximations over those previously known. In what follows, we describe the setting of the problems (Section 1.1), and describe various application scenarios where three such problems arise (Section 1.2). We state our results for such problems in Section 1.3 and present the technical details in sections 2, 3 and 4.

### 1.1 Problems

We begin with the one dimensional version. Consider an array $A[1 \cdots n]$ of non-negative numbers, and a *weight* function $f$ that maps intervals of $A$ to non-negative integers. The function $f$ is trivially assumed to be 0 on empty intervals. The *p-partition* of $A$ is a division of $A$ into $p$ intervals, that is, setting dividers $d_0 = 0 < d_1 \leq d_2 \leq \cdots \leq d_{p-1} \leq d_p = n$. Here the $i$th interval is $[d_{i-1}+1 \cdots d_i]$ if $d_{i-1} \neq d_i$ and is denoted empty otherwise. The MAX norm of a partition is $\max_{i=1}^{i=p} f(A[d_{i-1}+1 \cdots d_i])$. Two weight functions arise commonly in practice: the additive weight function $F(A[i,j]) = \sum_{k=i}^{k=j} A[k]$ and the Hamming weight function $H_c$ for a given parameter $c$, relative to another array $B$ of size $n$, given by $H_c(A[i,j]) = \min_{-c \leq k \leq c} \mathcal{H}(B[i+k, j+k], A[i,j])$ where $\mathcal{H}(X,Y)$ gives the Hamming distance between two segments $X$ and $Y$ of identical length.

**The 1D $p$-partition problem.** Given $p$, find the $p$-partition that minimizes the MAX norm. □

This notion can be naturally extended to a $p \times p$ *partition* in two dimensions as follows. Consider an $n \times n$ array $A$. Divide the rows $[1, n]$ into $p$ intervals given by horizontal dividers $h_0 = 0 < h_1 \leq h_2 \leq \cdots \leq h_{p-1} \leq h_p = n$, and the columns $[1, n]$ into $p$ other intervals given by the vertical dividers $v_0 = 0 < v_1 \leq v_2 \leq \cdots \leq v_{p-1} \leq v_p = n$. This induces $p^2$ *blocks* given by $A[h_{i-1}+1 \cdots h_i, v_{j-1}+1 \cdots v_j]$ for each $i, j$. The MAX norm of a partition is $\max_{i=1,j=1}^{i=p,j=p} f(A[h_{i-1}+1 \cdots h_i, v_{j-1}+1 \cdots v_j])$. Again, the common weight functions on blocks are $F$ and $H_c$ defined analogously as above for intervals.

**The 2D $p \times p$-partition problem.** Given $p$, find the $p \times p$ partition that minimizes the MAX norm. □

**The 2D $\delta$-weight partition problem.** Given $\delta$, find the minimum $p$ for which there exists a $p \times p$ partition of the array with the MAX norm of at most $\delta$. □

**Remarks.** There are many different ways to partition 2D arrays, as discussed in [GM96, KRW95, MS96, MM+96]. Here we consider only the $p \times p$ partition. These problems can be naturally generalized to higher dimensions. Our solutions for the 2D case extend to higher dimensions in a straightforward way. However, the 1D and 2D cases are fundamentally different, and they will be contrasted later.

## 1.2 Application Scenarios

Array partitioning problems arise in load balancing, scheduling, data layout, video compression, etc. We focus on three specific array partitioning problems. Here we briefly describe the application context for each; further details of modeling will be discussed in the journal version.

**One dimensional case under $F$.** This problem was abstracted for load balancing in pipelined, parallel environments in [B88] and studied in [OM95, AF91, HL92, MS95, M93, CN91, HNC92, N91] etc.

**Two dimensional case under $F$.** This problem arises in balanced data distribution as implemented in the Superb environment [ZBG86] and HPF2 [HPF] (High Performance Fortran). See [M93, CM+95] for more applications to particle-in-cell computations and sparse matrix computations.

**Two dimensional case under $H_c$.** This arises in motion-compensated video compression by block matching. Roughly this involves compressing a frame in a video sequence by cutting it into rectangles each of which is encoded in terms of a block in the previous frame. See [MM+96] and then references therein for the precise setting.

## 1.3 Results

We state our results for each of the three problems of our interest.

**1D $p$-partition under $F$.** This problem has been extensively researched. We summarize the previous work and our results in the table below, providing all citations where identical bounds were obtained independently.

| Reference | Bound |
|---|---|
| Bokhari [B88] | $O(n^3 p)$ |
| Anily & Federgruen [AF91] | $O(n^2 p)$ |
| Hansen & Liu [HL92] | $O(n^2 p)$ |
| Manne & Sorevik [MS95] | $O(np \log p)$ |
| Choi & Narahari [CN91] | $O(np)$ |
| Olstad & Manne [OM95] | $O(np)$ |
| Nicol [N91] | $O(n + p^2 \log^2 n)$ |
| Charikar, Chekuri & Motwani [CCM96] | $O(n + p^2 \log^2 n)$ |
| Han, Narahari & Choi [HNC92] | $O(n + p^{1+\epsilon})$, $\epsilon < 1$ |
| This paper | $O(n \log n)$ |

Our result relies on a binary search over a space of $O(n^2)$ items. However, at each test, an approximate median among these items is identified in only $O(n)$ (as opposed to $O(n^2)$) time by exploiting the structure in our search space. In particular, we design and use an algorithm that finds an approximate median of the $O(n^2)$ elements which are organized into $n$ sorted lists in only $O(n)$ time.

Throughout we have made no assumptions on the range of $F$'s. However, improved bounds may be obtained if the $F$'s lie in a restricted range; we omit the details here.

**The 2D $\delta$-weight problem under $H_c$.** A number of algorithms are known for block matching, and in particular, for the 2D $\delta$-weight problem under $H_c$. These essentially work by splitting subareas greedily until each subarea has weight at most $\delta$ and do not provide any guarantees on the number of blocks used. Building on the result of Grigni and Manne [GM96], this problem can in fact be shown to be NP-hard.

Here we provide an $O(\log n)$ approximate polynomial time algorithm. We obtain our result by a rather simple reduction to the classical set cover problem. Our algorithm works for a general class of metrics including $F$ and $H_c$.

**The 2D $p \times p$ partition problem under $F$.** Grigni and Manne [GM96] showed that the 2D problem is NP-hard even when the given array consists of 0/1 entries. Independently, Charikar *et. al.* have given a simple proof that this problem is APX-hard, that is, the problem is in fact NP-hard to approximate within a factor of two. While a number of natural heuristic algorithms are known for this problem (See for example [MS96]), most of them can be shown to be bad (typically $\Omega(\sqrt{p})$) approximations. One such heuristic has been recently shown to have a performance guarantee of $O(\sqrt{p})$ by Halldorsson & Manne [HM96]. This is the currently best known approximation for this problem.

| Reference | Result |
|---|---|
| Grigni & Manne [GM96] | NP-Hardness |
| Charikar *et. al.* [CCM96] | APX-Hardness |
| Halldorsson & Manne [HM96] | $O(\sqrt{p})$ approximation |
| This paper (Section 4) | $O(1)$ approximation |

We observe that using our result for the 2D $\delta$-weight problem above, one can easily obtain an $O((\log n)^2)$ approximation algorithm for the 2D $p \times p$-partitioning problem under $F$. But our main contribution is an $O(1)$-factor approximation for this problem which builds on an inherent connection between "independent" rectangles of large weight within the array and the cost of the optimal solution. Surprisingly, we are able to show that after a suitable preprocessing of the input array, a locally optimal collection of independent rectangles can be used to generate a solution which is at most a constant factor away from the optimal.

## 2 The One Dimensional Case Under $F$

We assume for convenience that $F(A[i]) \neq 0$ for any $i$; this assumption can be easily removed and we omit that detail. Define the Boolean function $M_A(\ell, k, v)$ to be true if and only if there exists a partition of the elements $A[\ell, n]$ into $k$ intervals, such that the MAX norm of these intervals is $< v$. In our analysis below, we count only the complexity of calls to the $F$ oracle; $F$ can be simulated in constant time after linear preprocessing.

**Lemma 1.** *$M_A(l, k, v)$ can be determined using $O(n)$ calls to the $F$ oracle for arbitrary $k$, $l$, and $v$.*

*Proof.* Note that without loss of generality the $(j+1)$st divider can be placed as far to the right of the $j$th divider such that $F$ value of the elements in that interval is $< v$. By incrementally inserting dividers from left to right so as to prevent the total in any interval from exceeding $v$, we find the minimum number of dividers required in $O(n)$ time. If this total exceeds $k$, then $M_A(l,k,v) = false$. Otherwise, $M_A(l,k,v) = true$. □

In the optimal partitioning with $k$ dividers, there will be an interval $A[i,j]$ which will prove the *bottleneck* of the partitioning: an interval is a bottleneck to the partitioning if it is the largest weight interval that results from this partitioning. There are $\binom{n}{2}$ candidates for this bottleneck interval. Performing a binary search on these candidates, using the linear-time oracle of Lemma 1, would yield an $O(n \log n)$ algorithm to search for the $k$-partition. However, this requires a method to efficiently compute the sequence of (approximate) median candidates to support the binary search. Conventional linear-time median-finding is clearly inadequate, since we have only $O(n)$ time to find the median of $\Theta(n^2)$ elements.

We take advantage of the fact that this collection of $\Theta(n^2)$ elements is not arbitrary, but has rather been derived from interval sums over $n$ elements. We partition the $\binom{n}{2}$ intervals $f(A[i,j])$ into $n$ *columns*, where column $c$ consists of the elements $f(A[i,c])$, $1 \le i \le c$. Let $C_c[i] = f(A[i,c])$ denote the $i$th element of column $c$. The *subcolumn* $S_c[i,j]$ comprises elements $C_c[x]$, $i \le x \le j$. These definitions are illustrated in Figure 1(a).

**Lemma 2.** *$C_c[i] \ge C_c[j]$ iff $i \le j$. Further, the median element of any subcolumn $S_c[i,j]$ can be determined with one call to the $F$ oracle.*

*Proof.* The first claim follows since the elements of each column are monotonically non-increasing. The second claim follows since the median of $S_c[i,j]$ is $F(A[\lfloor (i+j)/2 \rfloor, c])$. □

**Theorem 3.** *The 1D p-partition problem under $F$ can be solved in $O(n \log n)$ time.*

*Proof.* As per the above discussion, we effectively perform a binary search over the set of $\binom{n}{2}$ interval values. For each column, we will maintain one subcolumn containing the range of intervals which might include the optimum. Let $U$ be the set of elements representing the union of the elements in all the active subcolumns. A *splitter* for $U$ is an element $m$ such that the rank of $m$ in $U$, say $r_m$, satisfies

$$|U|/c \le r_m \le |U|(c-1)/c$$

for some constant $c$. The following algorithm finds a splitter for the active subcolumns in $O(n)$ time:

1. We find $m$, the median element of the set of $\le n$ median elements of the active subcolumns. Using Lemma 2, this set of median elements can be identified in $O(n)$ time. The median of this collection, $m$, can now be identified in $O(n)$ time using the standard linear-time median finding algorithm.

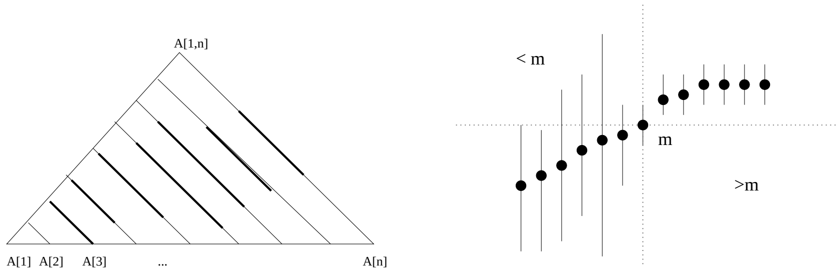


**Fig. 1.** (a) Columns and subcolumns of $A$. (b) The median of medians is not necessarily a splitter.

2. We divide the active subcolumns into two sets according to whether their median is $\leq m$ or not. Let $C_l$ $(C_r)$ denote the set of elements in subcolumns whose medians are $\leq m$ $(> m)$. If $\min(|C_l|, |C_r|) \geq (|C_l| + |C_r|)/8$, we return $m$ as a good splitter.
3. As illustrated by Figure 1(b), this median of medians is not necessarily a good splitter. If not, we recur on the appropriate set of subcolumns (the ones containing the larger number of elements) for the splitter search. Because the set of subcolumns under consideration is halved on each iteration, the total search time remains linear.

If $M_A(1, p, m) = true$, then $m$ is a lower bound on the optimal partitioning. Half of the elements in each subcolumns in $C_l$ may be eliminated, by replacing subcolumn $C_c[i, j]$ with $C_c[\lfloor (i + j)/2 \rfloor, j]$. If $M_A(1, p, m) = false$, then $m$ is an upper bound on the optimal partitioning. Half of the elements in each subcolumns in $C_r$ may be eliminated, by replacing subcolumn $C_c[i, j]$ with $C_c[i, \lfloor (i + j)/2 \rfloor]$. In either case, a constant fraction of the elements are eliminated in each linear-time round, and hence the optimal partition is identified in $O(n \log n)$ time. □

## 3 2D $\delta$-weight partition under $H_c$

We begin by considering the following geometric problem. We say that a rectangle is *stabbed* by a line if the line passes through the interior of the rectangle.

**Stabbing Problem.** Given a set of axis-parallel rectangles in the $[1, n] \times [1, n]$ two dimensional integer grid, determine a set $R$ of grid rows and $C$ of grid columns such that each rectangle is stabbed by one of the rows in $R$ or one of the columns in $C$ and furthermore, $s = \max\{|R|, |C|\}$ is minimized. □

**Lemma 4.** *The stabbing problem is $O(\log n)$-approximable.*

*Proof.* The proof is by reduction to set cover; the details are deferred to the final version. □

**Theorem 5.** *There exists a polynomial time $O(\log n)$ factor approximation for the 2D $\delta$-weight partition problem under $H_c$.*

*Proof.* We reduce this problem to the stabbing problem above. Consider the collection of *all* possibly overlapping *minimal* rectangles where the $F$ value of each rectangle is $> \delta$; rectangles are minimal in the sense that if two rectangles have $F$ value $> \delta$ and one is contained in the other, we retain the smaller one. Now the 2D $\delta$-weight partition problem is precisely the stabbing problem for which a $O(\log n)$ factor approximation exists. $\square$

## 4 2D $p \times p$-partition under $F$

Grigni and Manne [GM96] have shown that the 2D $p \times p$-partition problem under $F$ is NP-Complete. Charikar et al [CCM96] proved that it is NP-Complete even to approximate the solution within a factor of 2. In this section, we present a polynomial time heuristic which provides an $O(1)$ factor approximation.

The following lemma is crucially used in our arguments.

**Lemma 6.** *Let $c$ and $d$ be two positive integers, $c, d \le k$. If there exists a $k \times k$ partitioning such that MAX norm of the blocks is $B$ under $F$, then there exists a $k/c \times k/d$ partitioning with MAX norm $\le cdB$ under $F$.*

*Proof.* Consider a $k \times k$ partitioning with MAX norm $B$ and take every $c$th row as well as every $d$th column. The maximum $F$ value of a block of this $k/c \times k/d$ partitioning is at most $cdB$ since each new block contains $cd$ of the previous blocks. $\square$

This lemma can be combined with the observation that Theorem 5 holds for 2D $\delta$-weight partition problem under $F$ as well, to get the following.

**Theorem 7.** *There exists a polynomial time $O(\log^2 n)$-approximation for the 1D $p \times p$-partition problem under $F$.*

We omit the proof in this extended abstract.

The main result in this section is a substantially improved approximation algorithm; our algorithm computes an $O(1)$ factor approximation.

Let a $\langle W, \ell \rangle$-partition be a $\ell \times \ell$-partition such that the MAX norm of the blocks is at most $W$. We will now show that given an input instance for which a $\langle W, \ell \rangle$-partition exists, we can construct in polynomial time a $\langle O(W), \ell \rangle$-partition. The basic idea behind our algorithm is the notion of independent rectangles:

**Definition 8.** Two axis-parallel rectangles are said to be *independent* if their projections are disjoint along both the $x$-axis and the $y$-axis.

Clearly, no single horizontal or vertical line can stab a pair of independent rectangles. So if an array has a $\langle W, \ell \rangle$-partition, then it may contain at most $2\ell$ independent rectangles of weight strictly greater than $W$. As a result, independent rectangles constitute a useful tool in establishing a lower bound on the optimal solution value. The algorithm presented below builds on this idea to construct a partition whose cost is $O(W)$.

### 4.1 The Algorithm

Let $W$ be the optimal solution value. We assume a knowledge of this value in the presentation below – this value will be determined by performing a binary search over the interval $[0, \sum_{i,j} A[i,j]]$. Observe that $W \geq \max_{ij} A[i,j]$. Our algorithm constitutes of the following five steps:

**Step 1.** We obtain an $\ell \times \ell$ partition of the array such that each row or column within any block in the partition has weight at most $2W$. □

This can done by performing independent horizontal and vertical scans. During the horizontal scan, we keep a running sum of the weight of each row since the most recent vertical partition and set down the next vertical partition when the weight of any one of the rows exceeds $W$. Likewise, we set horizontal partitions based on running sums of the weights of columns during the vertical scan. Since each time a new column (row, respectively) is considered, the weight of the rows (columns, respectively) can increase by at most $W$, it follows that the weight of any row (column, respectively) within any block induced by the vertical and horizontal partitions does not exceed $2W$. Henceforth we consider the array with this $\ell \times \ell$ partition which we refer to as the partition $P$.

**Step 2.** We construct the set $S$ of all *minimal* rectangles whose weight exceeds $W$ and which are entirely contained within the blocks induced by the partition from Step 1. A rectangle is *minimal* if there does not exist another rectangle properly contained in it with weight larger than $W$. □

This can be done by starting from each location within a block and considering rectangles with their top left corner at that location in turn in the order of increasing sides until all minimal rectangles of weight strictly greater than $W$ are discovered.

**Step 3.** We determine a *local* 3-*optimal* set $M \subseteq S$ of independent rectangles. $M$ is a local 3-optimal set if there does not exist $i \in \{1, 2, 3\}$ independent rectangles in $S - M$ which can be added to $M$ by removing at most $(i-1)$ rectangles from $M$ without violating the independence condition. □

Such a set can be easily constructed in polynomial time by repeatedly performing swaps which increase the size of the current independent collection. Each swap takes polynomial time and the procedure terminates in polynomial time since any independent collection can have at most $O(n)$ rectangles.

**Step 4.** We now introduce another partition based on $M$. For each rectangle in $M$, we set two *straddling* horizontal and two *straddling* vertical partitions so as to induce that rectangle. In all, this introduces at most $2M$ horizontal and

$2M$ vertical partitions. The partition $P$ from Step 1 together with this partition induced by rectangles in $M$ is our new partition now. □

**Step 5.** We now have a partition of the input array which uses $h \leq 2M + \ell$ horizontal lines and $v \leq 2M + \ell$ vertical lines. To get a $\ell \times \ell$ partition from this, we simply retain only every $\lceil h/l \rceil$th horizontal line and only every $\lceil v/l \rceil$th vertical line. By Lemma 6, this increases the maximum block weight by at most a factor of $\lceil h/l \rceil \lceil v/l \rceil$.

### 4.2 Analysis: Approximation Guarantee and Correctness

We need to establish two properties of the above algorithm: (a) given a choice $W$ for which the input array has a $\langle W, \ell \rangle$-partition, the weight of any block in the partition constructed by the above algorithm is $O(W)$, and (b) the smallest value $W$ for which the analysis of the algorithm holds, identified via binary search, is upper bounded by the optimum solution value. We begin by establishing the first property above; the following lemma is central to the analysis here.

**Lemma 9.** *Let $b$ be a block contained in some block of the partition $P$ constructed in* Step 1 *above. Then if the weight of block $b$ is at least $27W$, it can be partitioned into 3 independent rectangles, each with weight strictly exceeding $W$.*

*Proof.* Given a block of weight at least $27W$, we construct three independent rectangles of weight exceeding $W$ as follows. First we perform a vertical scan, placing a horizontal cut as soon as the weight of the slab seen thus far exceeds $7W$; we place two horizontal cuts in all. This gives us three slabs each of weight strictly greater than $7W$. Now we perform a horizontal scan from right to left placing the first vertical cut as soon as one of the horizontal slabs exceeds weight $W$. Without loss of generality assume that it is the top slab. Then the top right block has weight greater than $W$ but does not exceed $3W$, and the two lower horizontal slabs to the left of that vertical cut have weight greater than $4W$ each. Now in a similar manner we place a second vertical cut to obtain two independent blocks of weight exceeding $W$ from these two horizontal slabs. Thus we get three independent rectangles of weight greater than $W$ each. □

**Lemma 10.** *The weight of any block in the partition constructed at the end of* Step 4 *is $O(W)$.*

*Proof.* We begin by observing the following easily verifiable properties of the solution: (a) each block of the solution is completely contained in some block of the partition $P$, and (b) given a block $b \in M$ and another block $b' \notin M$, their projections on the $x$-axis or the $y$-axis have either completely disjoint or have a perfect overlap.

Now consider a block $b$ in the solution; using the preceding observations, it is readily seen to fall into one of the following categories: (1) the block $b$ belongs to $M$, or (2) the block $b$ does not belong to $M$ but has a perfect overlap along one of the axes with a block $b' \in M$, or (3) the block $b$ does not belong to $M$ but has a perfect overlap along the $x$-axis with a block $b' \in M$ and a perfect overlap along the $x$-axis with a block $b'' \in M$.

In Case 1, the weight of $b$ is $O(W)$ since the set $S$ as defined in Step 2 has the property that any rectangle $r$ in it has weight at most $3W$. This is because otherwise, we can always remove either a row or a column (of weight at most $2W$) from $r$ to obtain a rectangle $r'$ of weight greater than $W$, contained in $r$, which violates the minimality of the rectangles in $S$.

In Cases 2 and 3, each block has weight at most $27W$; this follows from an application of Lemma 9. We observe that at most two blocks in $M$, say $b'$ and $b''$, may not be independent of a block which falls into these two cases. So if $b$ has weight greater than $27W$, we can replace $b'$ and $b''$ with at least three independent rectangles which are constructible from $b$ (and are contained in $S$). But this contradicts the local 3-optimality of the collection $M$ constructed in Step 3. Hence $b$ must weight at most $27W$. $\square$

**Lemma 11.** *The number of rectangles in $M$ is $2\ell$ for any choice $W$ for which there exists a $\langle W, \ell \rangle$-partition of the input array.*

*Proof.* If $M$ had $x$ rectangles, then each of those rectangles must be stabbed in the optimal solution since the optimal solution value is bounded by $W$ and every rectangle in $M$ has weight strictly greater than $W$. Stabbing $x$ rectangles requires at least $x/2$ horizontal or vertical partitions and hence $x$ must be at most $2\ell$. $\square$

**Lemma 12.** *The weight of any block in the final solution returned in* Step 5 *is at most $O(W)$ for any choice $W$ for which there exists a $\langle W, \ell \rangle$-partition of the input array.*

*Proof.* Lemma 11 tells us that the number of horizontal and vertical partitions at the end of Step 4 is $O(\ell)$ each. This fact, along with an application of Lemma 6, allows us to conclude that the weight of every resulting block in the $\ell \times \ell$ partition is $O(W)$. $\square$

This completes the proof of the first property of our algorithm that it gives a solution of weight $O(W)$ whenever a $\langle W, \ell \rangle$-partition exists. To conclude, we observe that the least value $W$ for which the algorithm either fails to construct the partition $P$ in Step 1 or yields a collection $M$ in Step 3 with more than $2\ell$ rectangles, must exceed the optimum. Thus the binary search procedure works to identify a suitable $W$.

**Theorem 13.** *There exists a polynomial time algorithm that computes an $O(1)$-factor approximation to the two dimensional block partitioning problem.*

## 5 Acknowledgements

Sincere thanks to the referees for bringing [HNC92] to our attention, and to the authors of [CCM96] for giving us a copy of their manuscript. The third author thanks Estie Arkin and Joe Mitchell for helpful discussions.

## References


[AF91] S. Anily and A. Federgruen. Structured partitioning problems. *Operations Research*, 13, 130–149, 1991.

[B88] S. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, 37, 38–57, 1988.

[CM+95] B. Chapman, P. Mehrotra, and H. Zima. High performance Fortran languages: Advanced applications and their implementation. *Future Generation Computer Systems*, 401 – 407, 1995.

[CCM96] M. Charikar, C. Chekuri, and R. Motwani. *Personal Communication*, 1996.

[CN91] H.-A. Choi and B. Narahari. Algorithms for mapping and partitioning chain structured parallel computations. *Proc. Intl. Conf. on Parallel Processing*, Vol I, 625-628, 1991.

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

[GM96] M. Grigni and F. Manne. On the complexity of the generalized block distribution. Proc. of 3rd international workshop on parallel algorithms for irregularly structured problems (IRREGULAR '96), Lecture notes in computer science 1117, Springer, 319-326, 1996.

[HNC92] Y. Han, B. Narahari and H.-A. Choi. Mapping a chain task to chained processors. *Information Processing Letters* 44, 141-148, 1992.

[HM96] M. Halldorsson and F. Manne. *Manuscript*, 1996.

[HL92] P. Hansen and K. Liu Improved algorithms for Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Trans. Computers*, 41, 769–771, 1992.

[KRW95] M. Kaddoura, S. Ranka and A. Wang. Array decomposition for nonuniform computational environments. *Technical Report*, Syracuse University, 1995.

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

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

[MS95] F. Manne and T. Sorevik. Optimal partitioning of sequences. *Journal of Algorithms*, 19, 235 – 249, 1995.

[MS96] F. Manne and T. Sorevik. Partitioning an array onto a mesh of processors. Proc. of *Workshop on Applied Parallel Computing in Industrial Problems.* 1996.

[N91] D. Nicol. Rectilinear partitioning of irregular data parallel computations. *ICASE* Report 91-55, 1991. *J. Parallel and Distributed Computing*, 23, 119-134, 1994.

[OM95] B. Olstad and F. Manne. Efficient partitioning of sequences. *IEEE Transactions on Computers*, 44, 1322 – 1325, 1995.

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