A New Approximation Algorithm for Multidimensional Rectangle Tiling

Katarzyna Paluch *

1 Institute of Computer Science, University of Wroclaw, Poland
2 Max-Planck-Institute für Informatik, Saarbrücken, Germany

Abstract. We consider the following tiling problem: Given a \( d \)-dimensional array \( A \) of size \( n \) in each dimension, containing non-negative numbers and a positive integer \( p \), partition the array \( A \) into at most \( p \) disjoint rectangular subarrays called rectangles so as to minimise the maximum weight of any rectangle. The weight of a subarray is the sum of its elements.

In the paper we give a \( \frac{4d+2}{d+2} \)-approximation algorithm that is tight with regard to the only known and used lower bound so far.

1 Introduction

In some applications including databases, load balancing and video compression we would like to partition data into sets of roughly the same weight. We consider the following tiling problem: Given a \( d \)-dimensional array \( A \) of size \( n \) in each dimension, containing non-negative numbers and a positive integer \( p \), partition the array \( A \) into at most \( p \) disjoint rectangular subarrays called rectangles so as to minimise the maximum weight of any rectangle. The weight of a subarray is the sum of its elements.

The problem, restricted to two dimensions, was introduced by Khanna et al in [4], where it is shown that a \( 5/4 \)-approximation for this problem is NP-hard. Successive approximation algorithms were constructed for this problem, beginning with the one having factor 5/2 by Khanna et al [4], through factors 7/3 ([8],[11]), 9/4 ([7]), 11/5 ([1]) and ending with the one having factor 17/8([10]).

The multidimensional version was first considered by Smith and Suri in [12], where they give an algorithm with approximation ratio \( \frac{4d^2}{d+2} \), that runs in time \( O(n^d + p \log n^d) \) and the constant is of the order of \( d! \). Next, Sharp in [11] gave a \( (d^2 + 2d - 1)/(2d - 1) \)-approximation algorithm that runs in time \( O(n^d + 2^d p \log n^d) \).

In this paper we give a \( \frac{4d+2}{d+2} \)-approximation algorithm that runs in time \( O(n^d + 2^d p \log n^d) \). Additionally, this algorithm is tight with regard to the only known and used lower bound so far.

* Partially done while at University of Dortmund and supported by Emmy Noether DFG grant KR 2332/1-2.
The general approach has a similar spirit as that in [10]. We also classify the arrays and subarrays into types. In the multidimensional case, however, there are many kinds of subarrays with a short type (having length 2) that are difficult to partition (whereas in a twodimensional case there is only one kind of such subarrays). As previously, we also have to consider arbitrarily large subarrays i.e. having arbitrarily long type. Fortunately, subarrays that are difficult to partition display a regular structure that can be handled by appropriate linear programs. Curiously, linear programs describing large difficult subarrays disintegrate into small linear programs that can be treated independently and in this respect they are much easier to analyze than linear programs describing large difficult subarrays in a twodimensional version, where they cannot be decomposed into small linear programs.

Organization of the paper In Section 2 we give some basic notions and notation. In Section 3 we introduce the notion of a simple subarray and show the way in which we use linear programs. In Section 4 we define the classification into types of arrays and subarrays and show which subarrays having short type are difficult to partition. In Section 5 we give the algorithm and prove its correctness. Also in that section Lemma 7 explains why large linear programs disintegrate into smaller ones.

2 Preliminaries

Let $M$ denote the value of the element(s) of maximal weight in $A$ and $w(S)$ the weight of a subarray $S$.

In any partition of $A$, clearly, at least one rectangle must have weight greater or equal $W = \max\{\frac{w(A)}{d}, M\}$. Thus $W$ is a simple lower bound for the maximum rectangle weight in an optimal solution.

For convenience sake we can rescale the array $A$ by dividing all its elements by $W$ and thus assume that we deal only with arrays of elements from the interval $[0, 1]$ and that the lower bound on the optimal solution is equal to 1.

To represent subarrays we will use the notation $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$. Individual elements will be represented by $(a_1, a_2, \ldots, a_d)$.

**Definition 1.** We say that the array or subarray is $\alpha$-partitioned if it is partitioned into rectangles having weight not greater than $\alpha$. If we additionally require that the number of tiles used does not exceed $\lfloor w(A) \rfloor$ (resp. $\lceil w(A) \rceil$, resp.) then we say that the array is well $\alpha$-partitioned (nearly well $\alpha$-partitioned, resp.).

From [7] we have

**Fact 1** If we partition the input array $A$ into a number of disjoint subarrays $A_1, \ldots, A_l, A_{l+1}$ and well $\alpha$-partition each $A_i$ ($1 \leq i \leq l$) and nearly well $\alpha$-partition $A_{l+1}$, then we will get the solution within $\alpha$ of the optimal one.

**Definition 2.** A slice in dimension $f$ of an array or a subarray is its subarray consisting of the elements having the same index in dimension $f$.

From now on, we will assume that $\alpha = \frac{\alpha + 2}{2}$. 
3 Simple Subarrays, Their Complexity and Difficulty

The key role in the analysis of the possible partitions is played by simple subarrays, into which we will appropriately decompose $A$.

**Definition 3.** Let $\beta \leq \alpha$.
A subarray $S$ is called $\beta$-simple (or simple if we know which $\beta$ we mean) if in every dimension there exists one slice that separates two subarrays, each having weight less than $\beta$ (i.e., $S$ is a disjoint sum of two subarrays having weight less than $\beta$ and the slice). The element that is the common part of all the separating slices of a simple subarray is called its center.

Suppose we have a simple subarray $S = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ that has a center $(c_1, c_2, \ldots, c_d)$. $\beta$ is arbitrary.

In each of the dimensions $S$ can have complexity 0, 1 or 2. In dimension $i$ it has complexity 0 iff $a_i = c_i = b_i$, it has complexity 1 iff $a_i = c_i < b_i$ or $a_i < c_i = b_i$ and it has complexity 2 iff $a_i < c_i < b_i$. The overall complexity of $S$ is the sum of its complexities in all dimensions.

One of the interpretations of the complexity is reflected in the following fact.

**Fact 2** If we have a simple subarray $S$ that has complexity $p$ and cut off one rectangle that contains only its center, then the rest of $S$ can be covered by $p$ rectangles and no fewer.

**Proof.** We can do it as follows. In the first step: if $a_1 < c_1$ we cut off a rectangle $[a_1, c_1 - 1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ and also if $b_1 > c_1$ a rectangle $[c_1 + 1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$.

In the $i$th step ($i \leq d$): if $a_i < b_i$, a rectangle $[c_1, c_1] \times [c_2, c_2] \times \ldots \times [c_{i-1}, c_{i-1}] \times [a_i, c_i-1] \times [a_i, b_i]$ and if $a_i > b_i$ a rectangle $[c_i, c_i] \times [c_{i+1}, c_{i+1}] \times \ldots \times [a_i, b_i]$.

We will use vectors $v = (v_1, v_2, \ldots, v_d)$ such that $v_2 \{-1, 1, 0, 2\}$ to point certain subarrays of $S$. Namely, we will say that $v$ cuts off a subarray $[a'_1, b'_1] \times [a'_2, b'_2] \times \ldots \times [a'_d, b'_d]$, where $[a'_i, b'_i] = [c_i, c_i]$ if $v_i = 0$, $[a'_i, b'_i] = [c_i, b_i]$ if $v_i = 1$, $[a'_i, b'_i] = [a_i, c_i]$ if $v_i = -1$, $[a'_i, b'_i] = [a_i, b_i]$ if $v_i = 2$.

Of course, it does not make much sense to put $v_i = -1$ if $a_i = c_i$ or $v_i = 1$ when $c_i = b_i$. Therefore we will say that $v$ is valid if in every dimension $i$ it holds: if $v_i = -1$, then $a_i < c_i$; if $v_i = 1$, then $c_i < b_i$ and if $v_i = 2$, then $a_i < c_i < b_i$.

Let $|v| = \sum |v_i|$.

A subarray is going to be described as $m$-difficult if it cannot be $\alpha$-partitioned into $m$ rectangles.

**Lemma 1.** A simple subarray $S$ that has complexity $p$ is $m$-difficult iff every vector $v$ such that $|v| = p - m + 1$ cuts off a subarray having weight greater than $\alpha$. 
Proof. If some vector $v$ such that $|v| = p - m + 1$ cuts off a subarray having weight at most $\alpha$, then we can $\alpha$-partition $S$ by taking one rectangle cut off by $v$ and covering the rest of $S$ by $m - 1$ rectangles of weight less than $\beta$ in the way similar as that in the proof of Fact 2. We are able to do so, because in each dimension $i$ a separating slice placed in $\alpha_i$ separates two subarrays having weight less than $\beta$.

In the other direction. Suppose we have some $\alpha$-partition of $S$ into $m$ rectangles. We can show that one of them must contain a rectangle that is cut off by a valid vector $v$ such that $|v| \geq p - m + 1$.

We can calculate the minimal weight of simple subarrays that are $m$-difficult using linear programming.

Let us explain it by an example. Suppose we have a $3$-difficult simple $5 \times 5$ array $A$, whose center is $(3, 3)$. Then the distribution of weight on this array can be described by the following array

\[
\begin{array}{ccc}
x_1 & x_2 & x_3 \\
x_4 & s & x_5 \\
x_6 & x_7 & x_8 \\
\end{array}
\]

where $s$ denotes the weight of the center, $x_2 = a_{1,3} + a_{2,3}, x_7 = a_{4,3} + a_{5,3}$ and $x_1 = a_{1,1} + a_{1,2} + a_{2,1} + a_{2,2}$ and the remaining variables denote analogous sums of elements.

The complexity of array $A$ is 4 and by Lemma 1 we know that each vector $v$, such that $|v| = 2$ cuts off a rectangle of weight greater than $\alpha$. Hence, the lower bound on the minimal weight of $A$ is the solution to the following linear program.

\[
\begin{align*}
\text{minimize} & \quad s + \sum_{i=1}^{8} x_i \\
\text{subject to} & \quad s \leq 1 \\
& \quad s + x_1 + x_2 + x_4 \geq \alpha \\
& \quad s + x_2 + x_3 + x_5 \geq \alpha \\
& \quad s + x_5 + x_7 + x_8 \geq \alpha \\
& \quad s + x_1 + x_6 + x_7 \geq \alpha \\
& \quad s + x_2 + x_7 \geq \alpha \\
& \quad s + x_4 + x_5 \geq \alpha \\
\end{align*}
\]

Let us notice that in estimating the lower bound of this subarray we can use only five variables $s, x_2, x_4, x_5$ and $x_7$ and without loss of generality assume that variables $x_1, x_3, x_6$ and $x_8$ are equal to 0. It is so because if the array, whose weight is distributed as above is 3-difficult, then so is the array, whose weight is distributed as follows, because all of the inequalities from the above linear program are satisfied as well.

\[
\begin{array}{ccc}
0 & x_2 + \frac{x_5}{2} + \frac{x_7}{2} & 0 \\
x_4 + \frac{x_5}{2} + \frac{x_7}{2} & s & x_5 + \frac{x_7}{2} + \frac{x_8}{2} \\
0 & x_7 + \frac{x_5}{2} + \frac{x_8}{2} & 0 \\
\end{array}
\]
We can generalize this reasoning and state the following lemma.

Lemma 2. The minimal weight of a simple \( m \)-difficult subarray \( S \) of complexity \( p \) is greater than the solution of the following linear program

\[
\begin{align*}
\text{minimize} & \quad s + \sum_{i=1}^{p} x_i \\
\text{subject to} & \quad s \leq 1 \\
& \quad s + x_{i_1} + x_{i_2} + \ldots + x_{i_{p-m+1}} \geq \alpha \\
& \quad \text{for all } 1 \leq i_1 < i_2 < \ldots < i_{p-m+1} \leq p
\end{align*}
\]

and thus it is greater than \( 1 + p \frac{\alpha-1}{p-m+1} \).

Proof. Similarly as above we can assume that the weight of the whole subarray is concentrated in \( p \) elements and the center (i.e. only those elements are non-zero) and by Lemma 1 the weight of every rectangle cut off by a vector \( v \) such that \( |v| = p - m + 1 \) is greater than \( \alpha \). One can easily check that the sum of the variables in the above linear program is minimal when all the inequalities are satisfied with equality and thus when \( s = 1 \) and \( x_1 = x_2 = \ldots = x_p = \frac{\alpha-1}{p-m+1} \).

We will say that an element \( \{e_1, e_2, \ldots, e_d\} \) is a satellite of the center if there exists exactly one \( i \) such that \( e_i \neq c_i \). Using this terminology we can say that the weight of a simple \( m \)-difficult subarray is minimal when its weight is concentrated in its center and the \( p \) satellites of the center.

4 Blocks

We look at the array \( A \) as a sequence of its slices in dimension one, which will be called sheets. We will distinguish two classes of them: those with weight at least 1 (\( > \)-sheets) and those with weight less than 1 (\( < \)-sheets). In [6] we have analogous notions of \( < \) and \( > \)-columns which actually are slices in a twodimensional array. There we have:

Lemma 3 ([6]).

1. A \( > \)-column can be well 2-partitioned.
2. A subarray consisting solely of \( < \)-columns having weight at least 1 can be well 2-partitioned.

This remains true in the case of \( < \)-sheets and \( \alpha \)-partitioning. Therefore we can notice that a group of adjacent \( < \)-sheets can be treated like a single \( > \)-column if its overall weight is at least 1 (\( < \)-sheets are elements of a \( > \)-column) and thus can be not only \( \alpha \)-partitioned but even well 2-partitioned and otherwise like a single \( < \)-sheet (its elements are the sums of the appropriate elements of \( < \)-sheets). Without loss of generality we will assume that every array consists of alternating \( < \) and \( > \)-sheets and begins and ends with a \( < \)-sheet. It will sometimes be achieved by inserting artificial sheets of weight 0.

Further we are going to identify more classes of sheets.
**Definition 4.** For every natural number $m$, a sheet of type $m$, also referred to as an $m$-sheet, denotes a $>\$-sheet having weight from the interval $[m, m + 1)$.

The type of an array or a subarray is going to be described by the types of its sheets given in the order of their occurrence. A sheet of type $m$ will be represented by a natural number $m$ and a $<$-sheet by a symbol $\$. The type of a subarray or an array will be given in the form $(\) m_1 \, m_2 \ldots \, m_n (\)$. In the algorithm the array $A$ will be processed from the leftmost sheet to the rightmost one, however not completely in the sheet-wise manner but in the block-wise one.

A block is a subarray of type $om$. Among blocks of type $om$ we will distinguish two subclasses:

- $m$ - simple subarrays that can be $\alpha$-partitioned into $m$ rectangles
- $\\_m$ - simple subarrays that cannot be $\alpha$-partitioned into $m$ rectangles

Now we will show some applications of Lemma 2.

**Lemma 4.** Every simple $>\$-sheet can be well $\alpha$-partitioned.

*Proof.* The complexity of a simple $>\$-sheet is at most $2d - 2$. Suppose its weight falls in the interval $[m, m + 1)$, which means that if we want to well $\alpha$-partition it we can use at most $m$ rectangles. From Lemma 2 we know that a simple $>\$-sheet that cannot be $\alpha$-partitioned into $m$ rectangles has weight greater than $1 + (2d-2)\frac{d}{2d-m-1}$. One can easily check that $1 + (2d-2)\frac{d}{2d-m-1} < m + 1$ holds only for $m \in (d-1, d]$. Therefore a simple $>\$-sheet that cannot be $\alpha$-partitioned into $m$ rectangles has weight greater than $m + 1$.

As every $>\$-sheet $S$ can be easily decomposed into simple $>\$-sheets, we can well $\alpha$-partition $S$ by well $\alpha$-partitioning each simple $>\$-sheet separately.

**Corollary 1.** Every $>\$-sheet can be well $\alpha$-partitioned.

This means that if the array consisted solely of $>\$-sheets we would have a very easy $\frac{d}{2}\$-approximation for $d$-dimensional arrays. Also, blocks of type $\_m$ can be from the definition well $\alpha$-partitioned (because they contain an $m$-sheet that has weight from the interval $[m, m + 1)$ and thus for well $\alpha$-partitioning we are allowed to use $m$ rectangles). Let us now examine which blocks of type $\\_m$ can be well $\alpha$-partitioned and which cannot.

The complexity of a block of type $\\_m$ is at most $2d - 1$. Therefore by Lemma 2, its weight is greater than $1 + (2d-1)\frac{d}{2d-m}$. Next we solve the following inequality

$$1 + (2d-1)\frac{d}{2d-m} \geq m + 1$$

and get that it is true for $m \in [d - \sqrt{d/2}, d + \sqrt{d/2}]$. Thus, for the above $m$, blocks of type $\\_m$ can be well $\alpha$-partitioned, because in this case we can use $m + 1$ rectangles and from Lemma 4 an $m$-sheet can be well partitioned (we use at most $m$ rectangles for it) and we use 1 rectangle for a $<$-sheet. If $m$ falls outside the above interval, then blocks of type $m$ indeed cannot be well $\alpha$-partitioned. This means that if we want to get an $\alpha$-approximation we cannot restrict ourselves to subarrays consisting of only 2 sheets but are forced to examine larger subarrays.
5 Algorithm

We will process the array in the block-wise manner starting from the leftmost. As we have seen in the previous section, some blocks of type \(\text{Lim} \) cannot be well \(\alpha\)-partitioned. Therefore, if we encounter such a block, we will extend it to the neighbouring one and see whether they together can be well \(\alpha\)-partitioned. If it still turns out impossible, we will extend it further until the subarray finally can be well \(\alpha\)-partitioned or we have reached the end of the array. In the other case we will be allowed to nearly well \(\alpha\)-partition the subarray.

If a subarray \(S\) consists only of simple blocks, then its type \(T\) has the form \(\phi_1 m_1 \phi_2 m_2 \ldots \phi_k m_k\), where each \(\phi_i\) denotes either \(\cdot\) or \(\cup\). To types of this form we will ascribe a natural number \(N(T)\):

\[
N(T) = \sum_{i=1}^{k} m_i + \sum_{i=1}^{k} [\phi_i = \cup] - 1.
\]

Thus, if \(T = \cup 4\), then \(N(T) = 4\) and if \(T = \cup 4 \cup 3\), then \(N(T) = 8\).

A subarray that is not simple is complex. In the algorithm we will check whether a given block is 1-simple.

We say that two elements \((e_1, e_2, \ldots, e_d)\) and \((e'_1, e'_2, \ldots, e'_d)\) coincide if there exists at least one \(i\) such that \(e_i = e'_i\).

\(S := \emptyset\)
while extension of \(S\) to the next block possible
extend \(S\) to the next block
if \(S\) is a block of type \(\cdot m\), well \(\alpha\)-partition it, else
if \(S\) ends with a complex block, well \(\alpha\)-partition \(S\) as shown in Lemma 14 else
if \(S\) of type \(T\) has weight at least \(N(T) + 1\), well \(\alpha\)-partition \(S\) as shown in Lemma 5, else
if \(S\) contains a subarray of type \(m\) that can be \(\alpha\)-partitioned into \(m - 1\) rectangles or a subarray of type \(\cup m\) that can be \(\alpha\)-partitioned into \(m\) rectangles, then well \(\alpha\)-partition \(S\) as shown in Lemma 12, else
if the centers of some two neighbouring blocks coincide, well \(\alpha\)-partition \(S\) as shown in Lemma 13
nearly-well \(\alpha\)-partition \(S\) together with the last \(\cdot\)-sheet.

In the algorithm if \(S\) ends with a complex block, then it is relegated for well \(\alpha\)-partitioning to Lemma 14. Therefore, if \(S\) has to be extended, then we know that it consists solely of simple blocks.

Lemma 5. If subarray \(S\) of type \(T\) consists only of simple blocks and its weight is at least \(N(T) + 1\), then we can well \(\alpha\)-partition it.

Proof. We can do it by \(\alpha\)-partitioning each block of type \(\text{Lim} \) into \(m + 1\) rectangles (\(m\) rectangles for an \(m\)-sheet and one for a sheet denoted by \(\cup\)) and by \(\alpha\)-partitioning each block of type \(\cdot m\) into \(m\) rectangles. This way we will use \(N(T) + 1\) rectangles.

5.1 Minimal Weight of \(S\)

This whole subsection is to prove...
Lemma 6. In the algorithm the subarray $S$ of type $T$ has to be extended has weight greater than $N(T) + \frac{1}{2}$.

From Lemma 2 we have

Fact 3 The minimal weight of a block of type $\cup m$ is greater than $m + \frac{3}{2}$.

In the following facts and lemmas we will assume that $S'$ is a subarray of $S$ that has to be extended by the algorithm.

Fact 4 The weight of a subarray $S'$ of type $\cup m$ is greater than $1 + 2d \frac{a-1}{2d-m} \geq m + 1$.

Proof. The minimal weight of $S'$ is equal to the minimal weight of an $(m+1)$-difficult subarray having complexity $2d$ and thus by Lemma 2 it is greater than $1 + 2d \frac{a-1}{2d-m}$ and the inequality $1 + 2d \frac{d/2}{2d-m} \geq m + 1$ implies $d^2 - 2dm + m^2 \geq 0$.

Next we prove a technical lemma that will prove very useful and will mean that often a large linear program describing the weight of a difficult subarray can be decomposed into smaller linear programs.

Lemma 7. If we have a block of type $\cup m$ and it contains one element $e$ having value $b$ that does not coincide with the center and $m \geq 3$, then the minimal weight of such a block is greater than $b + (2d - 1) \frac{a-1}{2d-m}$. In other words, it is greater than the minimal weight of a block of type $\cup m$ plus $b$.

Proof. The linear program from Lemma 2 has the same solution as the following linear program

minimize $s + \sum_{i=1}^{p} x_i$

subject to $s \leq 1$

$s + x_{i \mod p} + x_{i+1 \mod p} + x_{i+2 \mod p} + \ldots + x_{i+p-m \mod p} \geq \alpha$

for $1 \leq i \leq p$

It means that if the vectors to which the above $p$ inequalities correspond cut off rectangles that do not contain element $e$, then the lemma is proved. Since $e$ does not coincide with the center, for each $i$, either $e_i < c_i$ or $e_i > c_i$, which means that a vector $v$ cuts off a rectangle that contains $e$ iff for each $i$ it holds: $e_i = 2$ or $(e_i = 1$ and $e_i > c_i)$ or $(e_i = -1$ and $e_i < c_i)$. This in turn means that there exist $d$ satellites of the center denoted by some variables $x_{1}, \ldots, x_{d}$ such that vector $v$ cuts off a rectangle that contains an element $e$ if the rectangle cut off by this $v$ contains all these $d$ satellites. Therefore if no inequality in the linear program contains all the variables that represent these $d$ satellites, we are done.

If we have a block of type $\cup m$ and $m \geq 3$, then its complexity is $p = 2d - 1$ and $p - m + 1 = 2d - m \leq 2d - 3$, which means that no inequality in the above linear program contains all the variables $x_1, x_3, x_5, \ldots, x_{2d-1}$. Therefore we can rename the variables $x_i$ so that a vector $v$ cuts off a rectangle that contains $e$ iff it cuts off a rectangle that contains satellites denoted by $x_1, x_3, x_5, \ldots, x_{2d-1}$.
Lemma 8. The weight of a subarray $S'$ of type $\lim_1 \cup m_2$, such that $m_2 \geq 3$ is greater than the minimal weight of $\lim_1 \cup$ plus the minimal weight of $\lim_2$ and thus it is greater than $m_1 + m_2 + \frac{1}{2}$.

Proof. We use 2 variables for the center and $2d + 2d - 1$ variables that represent the satellites of the centers. One satellite of the center of $\lim_1 \cup$ is contained in the subarray $\lim_2$, however it does not coincide with the center of $\lim_2$ and one satellite of the center of $\lim_2$ falls in the subarray $\lim_1 \cup$ and also it does not coincide with the center of $\lim_1$. Thus by Lemma 7 the linear programs connected with these two subarrays can be considered separately and thus the minimal weight of $S$ is greater than the sum of the solutions of these linear programs.

Similarly we can prove

Lemma 9. For $m_1, m_2 \geq 3$, the weight of a subarray $S'$ of type $\sim m_1 \cup m_2$ is greater than $m_1 + m_2 + 1$.

From Lemmas 8 and 9 we get

Lemma 10. If the subarray $S'$ has type $T = \phi_1 m_1 \phi_2 m_2 \ldots \phi_k m_k$ and each $m_i \geq 3$, then the weight of $S'$ is greater than $N(T) + \frac{1}{2}$.

To finish the proof of Lemma 7, we need to show it is also true if $S$ contains $m$-sheets such that $m \leq 2$. It is done by enumerating the inequalities that the linear program contains and proving dual programming that the sum of the variables is minimized when all of them are satisfied with equality.

The immediate corollary of Lemma 6 is

Corollary 2. In the algorithm each $S$ that we encounter that is not ended with a complex subarray has weight at least $N(T)$.

Proof. If $S$ is a simple block, then its weight is at least $N(T)$. If $S$ consists of more blocks, then it has the form $S'B$ and $S'$ was relegated for an extension by the algorithm. Thus by Lemma 6 the weight of $S'$ is greater than $N(T') + \frac{1}{2}$ and the weight of a block of type $\sim m$ is at least $m$ and of a block of type $lim$ greater than $m + \frac{1}{2}$.

This means that if we want to well $\alpha$-partition $S$, we can use $N(T)$ rectangles.

Lemma 11. A subarray of type $\sim m$ that is contained in $S$ can be $\alpha$-partitioned into $m$ rectangles.

Proof. If it cannot be $\alpha$-partitioned into $m$ rectangles, then it is $m$-difficult and then the weight of $S$ is at least $N(T) + 1$.

Lemma 12. If in $S$ there exists a subarray of type $m$ that can be $\alpha$-partitioned into $m - 1$ rectangles or a subarray of type $m \cup$ that can be $\alpha$-partitioned into $m$ rectangles, then $S$ can be partitioned into $N(T)$ rectangles.
Lemma 13. If in $S$ the centers of two neighbouring blocks coincide, then $S$ can be well $\alpha$-partitioned.

Lemma 14. If $S$ ends with a complex block, then it can be well $\alpha$-partitioned.

The running time of the algorithm is mostly spent in searching for separating slices in simple subarrays and it can be estimated similarly as the time spent by procedure Heavy-Search, Heavy-Cut and Medium-Tile in [11].

6 The Algorithm is Tight

Suppose that we have an array $A$ that is 1-simple and has complexity $2d$ and it has only $2d + 1$ non-zero elements and these are: the center, that has weight 1 and the $2d$ satellites that have weight $\frac{1}{2d}$. Then the overall weight of $A$ is equal to $1 + d$. Let $p = 1 + d$. Thus the lower bound $W = 1$. We can easily see that one rectangle in the partition must contain the center and at least $d$ satellites, therefore its weight is at least $1 + \frac{d}{2d}$.

References