

A $2\frac{1}{8}$ -Approximation Algorithm for Rectangle Tiling

Katarzyna Paluch*

Institute of Computer Science
University of Wrocław
abraka@ii.uni.wroc.pl

Abstract. We study the following problem. Given an $n \times n$ array A of nonnegative numbers and a natural number p , partition it into at most p rectangular tiles, so that the maximal weight of a tile is minimized. A tile is any rectangular subarray of A . The weight of a tile is the sum of the elements that fall within it. In the partition the tiles must not overlap and are to cover the whole array. We give a $2\frac{1}{8}$ -approximation algorithm, which is tight with regard to the only known and used lower bound. Although the proof of the ratio of the approximation is somewhat involved, the algorithm itself is quite simple and easy to implement. Its running time is linear in the size of the array, but can be also made to be near-linear in the number of non-zero elements of the array.

1 Introduction

We study the following problem:

The RTILE problem. Given an $n \times n$ array A of nonnegative numbers and a natural number p , partition it into at most p rectangular tiles, so that the maximal weight of a tile is minimized. A tile is any rectangular subarray of A . The weight of a tile is the sum of the elements that fall within it. In the partition the tiles must not overlap and are to cover the whole array.

Previous work. Khanna, Muthukrishnan and Paterson showed in [5] that the problem is NP -hard to approximate to within a factor of $\frac{5}{4}$. They also gave the first approximation algorithm with ratio $2\frac{1}{2}$. Next, the factor of approximation was improved to $2\frac{1}{3}$ independently by Sharp [11] and by Lorys, Paluch [7]. The best approximation result up till now is by Berman, DasGupta, Muthukrishnan and Ramaswami [1], who delivered a $2\frac{1}{5}$ -approximation algorithm. In the meantime the $2\frac{1}{4}$ -approximation was proved by Lorys, Paluch in [8]. The problem has applications in load balancing, in parallel computing environments, data compression, and query optimization in databases ([5], [11], [1]).

New results. Our main result is a $2\frac{1}{8}$ approximation for the RTILE problem. The proof of the $2\frac{1}{8}$ approximation is somewhat involved. Some of the ideas evolved from those used in [8], in particular we use an extended but similar classification of arrays into types. In [8] however, it was enough to consider subarrays which had short types (of length at most 8). To obtain this result, however, we have to prove some properties for

* Partially supported by KBN grant 8 T11C 044 19.

subarrays of arbitrarily long type. To do this we adopt a more orderly approach and create more refined apparatus. In case of large troublesome arrays we sort of "dig out" their regularities and notice that then proving some of their properties reduces to solving certain classes of (arbitrarily large) linear programs which describe their locally recursive structure. The approximation is also tight with regard to the used lower bound, which is the only known and used lower bound so far. In particular, we show that for every $\epsilon > 0$ there exists such an array A and the number of rectangles p , that any partition of A into p rectangles will contain a tile of weight equal to $2\frac{1}{8} - \epsilon$ times the value of the lower bound for this instance of the problem. The algorithm itself is quite simple and easy to implement. Its running time is linear in the size of the array, but can be also made to be near-linear in the number of non-zero elements of the array.

2 Preliminaries

Obviously the value of the maximum weight of a tile in the partition cannot be lower than the average weight of a tile or the maximal element in the array. Thus, if $w(A)$ denotes the weight of A , then

$$W = \max\left\{\frac{w(A)}{p}, \max\{a_{ij} : 1 \leq i, j \leq n\}\right\}$$

is the simple lower bound on the maximum weight in an optimal solution. We show how to find the tiling, in which the weight of each tile does not exceed $2\frac{1}{8}W$.

Since the weights of rectangles are considered in their relation to the value of the lower bound W , it will be convenient to rescale the array A by dividing its elements by W . After rescaling the lower bound is naturally equal to exactly 1. Now, if we partition the array A into some number of disjoint subarrays A_1, A_2, \dots, A_m and partition each of them separately using at most $\lfloor w(A_i) \rfloor$ tiles for a subarray A_i of weight $w(A_i)$, then we will not exceed the allowed number of tiles p . Moreover, since $p \geq \lceil \frac{w(A)}{W} \rceil$, we can use $\lceil w(A) \rceil$ tiles. Thus we are allowed to use $\lceil w(A_i) \rceil$ tiles in one case.

Further on we will say that a subarray A_i has been *well-partitioned*, if it has been partitioned into at most $\lfloor w(A_i) \rfloor$ tiles. Clearly the subarray of weight less than 1 cannot be well-partitioned. A subarray A_i partitioned into $\lceil w(A_i) \rceil$ tiles will be called *nearly-well-partitioned* and partitioned into $\lfloor w(A_i) \rfloor - 1$ tiles will be called *extra-partitioned*. For abbreviation we will also use the notion of *f-partitioning*, which will mean that in the partition the maximum weight of a tile does not exceed f . f will be further on referred to as a *factor*. f will always have value at least 2.

The first stage of the approach to the partition begins by looking at the columns of the array and dividing them into two classes: those having weight less than 1 (<-columns) and those having weight at least 1 (>-columns). A group of adjacent columns having weight less than 1, whose total weight is greater than 1 can be treated like a single >-column, because any statement concerning a >-column holds also for such a group (it suffices to look at the columns of a group as the elements of a >-column). Similarly a group of adjacent <-columns, whose total weight is less than 1 can be treated like a single <-column. Thus we can assume, that in the array A there are no adjacent <-columns (every group of adjacent <-columns will be merged into one column). We

will also assume that in the array every $>$ -column is succeeded by a $<$ -column and the first column of the array is a $<$ -column. It will sometimes be achieved by putting artificial columns of weight 0. From [7] it is known, that any $>$ -column can be well 2-partitioned and that any two-column subarray consisting of a $<$ - and a $>$ -column (*a block*) can be well- $2\frac{1}{3}$ -partitioned, which was enough to yield $2\frac{1}{3}$ -approximation. When we looked closer at blocks, i.e. two-column subarrays consisting of a $<$ - and a $>$ -column, it turned out that only one specified kind of them (a "prototype" of a *complex*) cannot be well-2-partitioned, while the rest can (be well- 2-partitioned). In order to improve the approximation we must deal with such blocks. We might try to extend them to the next block and check, whether now it could be well-partitioned with a better factor. If then the factor proved unsatisfactory we might try to extend the subarray further and so on. The process naturally has to end when the extension is no longer possible, that is when we have reached the end of the array and there are no more blocks, i.e. we are left with one $<$ -column or nothing. In that case we are allowed to nearly-well-partition the remaining subarray, that is a subarray that successively defied well- f -partitioning together with the last $<$ -column, if there is such left. Nearly-well-partitioning usually allows a better factor than well-partitioning. Aiming at f - approximation the algorithm can be described as follows:

```

while possible
  subarray  $S :=$  the first block from the left
  if  $S$  can be well- $f$ -partitioned, partition it
  else
    while  $S$  cannot be well- $f$ -partitioned and extension possible
      extend  $S$  to the next block
    if  $S$  can be well- $f$ -partitioned , partition it
    else nearly-well- $f$ -partition  $S$  together with the remaining part
nearly-well- $f$ -partition the remaining (which must be a  $<$ -column)
    
```

3 Classification into Types

Definition 1. For every natural m , a column of type m , also referred to as an m -column, denotes a $>$ -column having weight from the interval $[m, m + 1)$. A block of type $- m$ denotes a block consisting of a $<$ -column and a m -column that can be f -partitioned into m tiles. A block of type $\sqcup m$ denotes a block consisting of two columns as above that cannot be f -partitioned into m tiles.

Based on this, we characterize the type of the whole array by describing the type of every block contained in it. To be precise, we do it in the following way. Recall that we have assumed that the whole array consists of alternating $<$ - and $>$ -columns and begins with a $<$ -column, we take every $<$ -column, beginning from the leftmost, and put it into one block with the $>$ -column on the right, describing its type. If the array's last column is a $<$ -column, we denote it by $..$. Examples of types of the array: $\sqcup 2 \sqcup 2 \sqcup 3$ or $\sqcup 2 . 3 .$. Notice, that the type of the array does not contain information about the types of the blocks, in which a $<$ -column is on the right of a $>$ -column. For example, if the type of the array is $\sqcup 2 \sqcup 3$, we do not know whether the second and a third column are a block of type $2 \sqcup$ or $2 .$

The subarrays we will consider, will consist of columns of the array. The type of the subarray is given in the same way, except for two things. The subarray may begin with a $>$ -column, then we first give the type of this column. If the subarray ends with a $<$ -column, then we denote it by \sqcup , if it is the part of a block of type $\sqcup m$ in the whole array.

\diamond denotes \sqcup or \cdot .

For blocks of type $\sqcup m$ we have the following lemma:

Lemma 1. *If a subarray consisting of a $<$ -column and an m -column cannot be $(2+x)$ -partitioned into m tiles, then its weight is greater than, correspondingly: $m + 1 + (m - 1)x$, if $m \geq 3$; $2\frac{1}{2} + \frac{3}{2}x$, if $m = 2$ and $2 + x$, if $m = 1$.*

Proof. Here we give the proof only for $m = 2$.

A block cannot be $(2 + x)$ -partitioned into 2 parts iff

1. the vertical partition into 2 parts is impossible, which implies that the column of type 2 has weight greater than $2 + x$
2. the horizontal partition is impossible, which means, that there exists such a row b , that both the weight of b together with the part of the subarray above it is greater than $2 + x$ and so the weight of b together with the part below it.

The minimal weight of the block that cannot be $(2+x)$ -partitioned into 2 tiles can be estimated using linear programming. Consider Fig. 1 and the following linear program:

$$\begin{aligned}
 &\text{minimize} && \sum_{i=1}^6 x_i \\
 &\text{subject to} && x_2 + x_5 + x_6 < 1 \\
 & && x_1 \leq 1 \\
 & && x_1 + x_3 + x_4 \geq 2 + x \\
 & && x_1 + x_2 + x_3 + x_5 \geq 2 + x \\
 & && x_1 + x_2 + x_4 + x_6 \geq 2 + x
 \end{aligned}$$

x_5	x_3
x_2	x_1
x_6	x_4

Fig. 1.

We get that the weight of such a block is minimal when $x_1 = 1$, $x_2 = x_3 = x_4 = \frac{1}{2} + \frac{x}{2}$ and $x_5 = x_6 = 0$, and is equal to $2\frac{1}{2} + \frac{3}{2}x$. □

Let us observe that every block of type $\diamond m$ can always be 2-partitioned into $m + 1$ tiles, since every m -column can be well-2-partitioned ([7]), i.e. 2-partitioned into m tiles. Therefore the above lemma implies that all blocks, except for blocks of type $\sqcup 2$, can be well-2-partitioned.

4 Definitions

We assign every type T a certain natural number $N(T)$ in the way described below.

Definition 2. $N(T)$ is defined as follows. For types of columns:

$$\begin{aligned} N(-) &= -1 \\ N(\sqcup) &= 0 \\ N(m) &= m - 1 \end{aligned}$$

For more complex types:

$$N(T_1T_2) = N(T_1) + N(T_2) + 1.$$

For almost all types $N(T)$ is defined in such a way that it is the smallest number n , for which we are guaranteed from the definition of types of blocks, that every subarray of type T can be f -partitioned into $n + 1$ tiles. For example $N(-m) = m - 1$ and $N(\sqcup m) = m$ and indeed we know that a block of type $-m$ can be f -partitioned into $N(-m) + 1 = m$ tiles and a block of type $\sqcup m$ can be 2-partitioned (hence also f -partitioned) into $m + 1$ tiles. Also $N(\sqcup 2_4) = 7$ and we can f -partition every subarray of this type by 2-partitioning the first block into 3 tiles and f -partitioning the second one into 4 tiles. The above does not hold for types ended with $..$ (For example $N(2_) = 1$, however we are not guaranteed that a subarray of this type can be f -partitioned into 2 tiles.)

$N(T)$ is also to denote that every subarray of type T , encountered in the course of carrying out the algorithm, will always have weight at least $N(T)$. For some subarrays, it is immediate from Lemma 1 and the definition of types, like every subarray of type $\sqcup 2_2$ has weight greater than $2\frac{1}{2} + \frac{3}{2}x + 2$ (the second 2 comes from the fact that a 2-column has weight at least 2) or every subarray of type $\sqcup 2 \sqcup 2$ has weight greater than $5 + 3x = N(\sqcup 2 \sqcup 2) + 3x$. However, $N(\sqcup 2 \sqcup 2 \sqcup 2 \sqcup 2) = 11$ and from Lemma 1, it only implies that every subarray of this type has weight greater than $10 + 3x$.

The notions of well- and nearly-well-partitioning are related with the actual weight of the subarray. Sometimes, however, it would be convenient to have analogous notions defined with regard to $N(T)$.

Definition 3. To well-tile a subarray of type T means to partition it into $N(T)$ tiles. Similarly, to nearly-well-tile a subarray of type T means to partition it into $N(T) + 1$ tiles.

We will often take advantage of the following facts:

Fact 1 If we have a subarray S of type $T = T_1T_2$, then in order to well-tile S we can well-tile its subarray of type T_1 and nearly-well-tile its subarray of type T_2 or the other way round. Similarly, in order to nearly-well-tile S we can nearly-well-tile both its subarray of type T_1 and its subarray of type T_2 .

Proof. $N(T) = N(T_1) + N(T_2) + 1$ □

Fact 2 Every subarray of type $\diamond k_1 \diamond k_2 \dots \diamond k_n$ or of type $\diamond k_1 \diamond k_2 \dots \diamond k_n \sqcup$ can always be nearly-well-2-tiled.

We will also quite often use expressions of the form "A can be well- f -tiled or has weight at least w ". To this end we have

Definition 4. For a subarray A of type T we will say that

$$well(A) \geq_f w$$

if A can be well- f -tiled or has weight at least $N(T) + w$.

Similarly, we will say that

$$nearly(A) \geq_f w$$

if A can be nearly-well- f -tiled or has weight at least $N(T) + w$.

Let us notice that if we have a subarray A of type $\diamond k_1 \diamond k_2 \dots \diamond k_n$ and we are able to state that $well(A) \geq_f 1$, then we are in a good situation in the sense that assuming A has indeed weight at least $N(T)$, it means that A can be well- f -partitioned. It is so, because either we are able to well- f -tile the subarray (i.e f -partition it into $N(T)$ tiles) or it has weight at least $N(T) + 1$, which means in turn that for well-partitioning we are allowed to use $N(T) + 1$ tiles and thus able to well-2-partition it (block-wise).

Fact 3 If a subarray A of type T has weight at least $N(T)$, can be nearly- f -tiled and $well(A) \geq_f 1$, then A can be well- f -partitioned.

Often the knowledge about the subsubarrays can be combined to tell something about the subarray.

Lemma 2. Suppose we have a subarray A of type $T = T_1 T_2$ that consists of two subarrays B and C correspondingly of type T_1 and T_2 . Suppose also that both B and C can be nearly- f -tiled. Then it holds

$$well(B) \geq_f w_1 \text{ and } well(C) \geq_f w_2 \Rightarrow well(A) \geq_f w_1 + w_2 - 1.$$

Proof. It follows from Fact 1. □

5 Subarrays of Type $\sqcup 2 \diamond k$

Suppose we have a subarray A of type $T = \sqcup 2 \sqcup k$ and $k \geq 3$. From Lemma 1 we know that its weight is at least $2\frac{1}{2} + \frac{3}{2}x + k + 1 + (k-1)x$, which is greater than $N(T) = 2 + k + 1$. Let B denote the beginning complex $\sqcup 2$ and C the subarray of type $\sqcup k$. By the same lemma, we also have that $well(B) \geq_{2+x} \frac{1}{2} + \frac{3}{2}x$ and $well(C) \geq_{2+x} 1 + (k-1)x$. Thus by Fact 2 and Lemma 2 we get, that $well(A) \geq_{2+x} \frac{1}{2} + \frac{3}{2}x + (k-1)x$. Next we calculate that $well(A) \geq_{2+x} 1$ for $x \geq \frac{1}{2k+1}$, which by Fact 3 means that for $k \geq 3$,

we can always well- $2^{\frac{1}{2k+1}}$ -partition A , thus for $k \geq 4$, it means that we are able to well- $2^{\frac{1}{9}}$ -partition it. Since we aim at a $2^{\frac{1}{8}}$ -approximation, from this time on by writing $well(A) \geq w$ we will mean $well(A) \geq 2^{\frac{1}{8}} w$ and the same for $nearly(A)$. Similarly by well-, nearly-well-tiling we will mean well- $2^{\frac{1}{8}}$ -, nearly-well- $2^{\frac{1}{8}}$ -tiling. A complex will mean a $2^{\frac{1}{8}}$ -complex. We will further prove, that for every subarray A that the algorithm could not well- $2^{\frac{1}{8}}$ -partition, it holds $well(A) \geq \frac{5}{8}$. Thus if A is followed by B such that $well(B) \geq 1^{\frac{3}{8}}$, then $well(AB) \geq 1$ and as all other conditions of Fact 3 and Lemma 1 will be satisfied, it will mean that AB can be well- $2^{\frac{1}{8}}$ -partitioned.

6 Two Neighbouring Complexes and Degenerate Subarrays

Let us now look at the subarray C of type $T = \sqcup 2 \sqcup 2$. By Lemma 1 we know that its weight is at least $5^{\frac{3}{8}} = N(T) + \frac{3}{8}$. By the same lemma and by Lemma 2, we also know that $well(C) \geq \frac{3}{8}$. We notice that the situation seems to be worse now than in the case of only one complex, because we do not state that $well(C) \geq \frac{5}{8}$ and hence cannot say that if C is followed by some block D such that $well(D) \geq 1^{\frac{3}{8}}$, then CD can always be well- $2^{\frac{1}{8}}$ -partitioned. We have to examine the structure of the subarray of type $\sqcup 2 \sqcup 2$ somewhat closer. Before doing this however, we introduce degenerate subarrays. We later prove that degenerate subarrays can always be well- $2^{\frac{1}{8}}$ -partitioned and also that whenever some subarray we could not well- $2^{\frac{1}{8}}$ -partition is followed by a degenerate subarray, then the whole can be well- $2^{\frac{1}{8}}$ -partitioned. It will turn out that in the non-degenerate and difficult to partition subarrays the boundaries of $\sqcup 2$, $2 \sqcup$ or $\sqcup 1 \sqcup$ and \succ -columns (i.e. 1- and 2-columns) form "nonoverlapping crosses". More precisely one type of a cross arises in the part of type $\sqcup 2 \sqcup$ if the boundaries of the complexes $\sqcup 2$ and $2 \sqcup$ are in the same row, it is formed by these boundaries and the 2-column. The second type of a cross is the one in the part of type $\sqcup 1 \sqcup$, which is formed by the boundary of this part and the 1-column. The crosses non-overlap, if the boundaries of overlapping regions (except for $\sqcup 2$ and $2 \sqcup$ in the part of type $\sqcup 2 \sqcup$) fall into different rows, that is, for example, if in the part of type $2 \sqcup 1 \sqcup$ the boundaries of $2 \sqcup$ and $\sqcup 1 \sqcup$ are in different rows or if in the part of type $2 \sqcup 2$ the boundaries of $2 \sqcup$ and $\sqcup 2$ are also in different rows. Degenerate subarrays are mostly those whose structure contradicts in some way the idea of "non-overlapping crosses".

Definition 5. *The following kinds of subarrays will be called degenerate (compare Fig. 2):*

- $\sqcup 1 \sqcup 2$ with the boundaries of subarrays $\sqcup 1 \sqcup$ and $\sqcup 2$ in the same row,
- $\sqcup 2 \sqcup k$ with the boundaries of the first complex and the "right" complex $2 \sqcup$ in different rows,
- $\sqcup 2 \sqcup k$ with the part above (or under) the boundaries of the complexes $\sqcup 2$ and $2 \sqcup$ greater than $2^{\frac{1}{8}}$,
- $\sqcup 1 \sqcup 1 \sqcup 2$ with the boundaries of two subarrays of type $\sqcup 1 \sqcup$ in the same row.

Also, those of type $\sqcup T$ and T , where T is

- $2 \sqcup 2$ with the boundaries of subarrays $\sqcup 2$, $2 \sqcup$ in the same row,

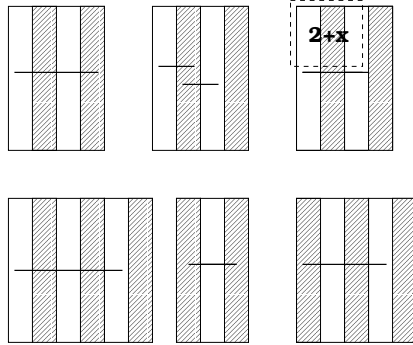


Fig. 2.

- $2 \sqcup 1 \sqcup k$ with the boundaries of $2 \sqcup$ and $1 \sqcup 1$ in the same row.

We define also some degenerate subarrays of type $\sqcup 2 \sqcup 3$ and $\sqcup 2 \sqcup 3 \sqcup 2$.

Lemma 3. For every degenerate subarray D of type T , it holds

$$well(D) \geq 1\frac{3}{8}.$$

Since we will further prove, that for every subarray S (of type $\diamond k_1 \diamond k_2 \dots \diamond k_n$) $well(S) \geq \frac{5}{8}$, by the above lemma, we get that the appearance of a degenerate subarray D , causes that we are able to well- $2\frac{1}{8}$ -partition SD .

Assume now we have a subarray S of type $\sqcup 2 \sqcup 2$, that is not degenerate. We will prove

Fact 4 For every non-degenerate subarray S of type $\sqcup 2 \sqcup 2$, it holds

$$well(S) \geq \frac{21}{32}.$$

Proof. Let us try to give the conditions, under which it cannot be well-tiled. First we can notice that the part $2 \sqcup$ should be a complex - a so called "right" complex, because if it were not, then it could be $2\frac{1}{8}$ -partitioned into 2 tiles and the whole could be $2\frac{1}{8}$ -partitioned into 5 tiles, that is the whole could be well-tiled. Another condition is that the sum of the three elements contained in the boundaries of $\sqcup 2$ and $2 \sqcup$ in the part $\sqcup 2 \sqcup$ should be greater than $2\frac{1}{8}$. If it were not, then we could well-tile the subarray, as we know that the part above and under this tile has weight not greater than $2\frac{1}{8}$ (because the subarray is not degenerate), and the rest (the 2-column) can easily be 2-partitioned into 2 tiles.

Let us now estimate the minimal weight of such a subarray. Since it is not degenerate its structure is as shown in Fig. 3: the boundaries of the first complex $\sqcup 2$, the "right" one $2 \sqcup$ and the second one $\sqcup 2$ are represented by variables correspondingly s_1, x_2 and s_1, x_4 and s_2, x_6 . Variables x_1, x_3, x_5 and x_7 denote the sums of the values of the elements accordingly above or below the appropriate boundary.

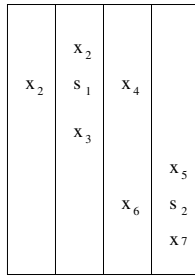


Fig. 3.

We do not need more variables in \prec -columns and can assume that the whole weight of \prec -columns is concentrated in x_2, x_4, x_6 , i.e. in the boundaries. We calculate:

$$\begin{aligned}
 &\text{minimize } s_1 + s_2 + \sum_{i=1}^7 x_i \\
 &\text{subject to} \quad \begin{aligned}
 &s_1 \leq 1 \\
 &s_1 + x_1 + x_2 \geq 2\frac{1}{8} \\
 &s_1 + x_2 + x_3 \geq 2\frac{1}{8} \\
 &s_1 + x_1 + x_4 \geq 2\frac{1}{8} \\
 &s_1 + x_3 + x_4 + x_6 \geq 2\frac{1}{8} \\
 &s_1 + x_1 + x_3 \geq 2\frac{1}{8} \\
 &s_1 + x_2 + x_4 \geq 2\frac{1}{8} \\
 &s_2 \leq 1 \\
 &s_2 + x_4 + x_5 + x_6 \geq 2\frac{1}{8} \\
 &s_2 + x_6 + x_7 \geq 2\frac{1}{8} \\
 &s_2 + x_5 + x_7 \geq 2\frac{1}{8},
 \end{aligned}
 \end{aligned}$$

which turns out to be $5\frac{21}{32}$. □

7 The Algorithm in Greater Detail

In the description of the algorithm in Section 2, we start from the leftmost block and try to well- $2\frac{1}{8}$ -partition it. From Lemma 1, we know, that unless it is a complex, its weight is at least $N(T) + 1$ and if it is a complex, its weight is greater than $N(T) + \frac{11}{16} = 2\frac{11}{16}$. Therefore in new terminology, if we want to well- $2\frac{1}{8}$ -partition it, we in fact want to well- or nearly-tile it, depending on whether its weight is less than $N(T) + 1$ or at least $N(T) + 1$. Further on we will prove that every subarray encountered in the course of carrying out the algorithm, will have weight at least $N(T)$. Thus we will always be interested in either well- or nearly-tiling it. From Fact 2 we know that nearly-tiling is easy. As for well-tiling, we have that it is enough to well-tile some subarray and nearly-tile the preceding and succeeding part. Therefore in order to well-tile some subarray, we will try well-tiling its *small* subarrays i.e. those of type $\diamond k$, $k\diamond$, $\diamond 1\diamond$, $\diamond 2\diamond$ and $\diamond 3\diamond$.

$S := \emptyset$
 while extension of S to the next block possible
 extend S to the next block
 if S of type T has weight at least $N(T) + 1$, nearly-well-tile S and $S := \emptyset$
 else if any small subarray can be well-tiled
 or if S is ended with a degenerate subarray,
 well-tile S and $S := \emptyset$
 nearly-well- $2\frac{1}{8}$ -partition S together with the last $<$ -column.

Theorem 1. *The above algorithm is a $2\frac{1}{8}$ approximation algorithm for the RTILE problem.*

To prove that the above is a $2\frac{1}{8}$ -approximation algorithm for rectangle tiling, we should show, that it is correct and also executable, that is whenever the instruction to well-tile, nearly-well-tile or nearly-well- $2\frac{1}{8}$ -partition appears, we are capable of carrying it out.

To prove, that it is correct it is enough to show, that every time we well-tile S (of type T) in the algorithm, S indeed has weight at least $N(T)$.

In the following, we show, that if the subarray of type T that does not qualify for well-tiling according to the algorithm and therefore has to be extended, has weight greater than $N(T) + \frac{1}{2} + \frac{1}{8}$.

We will say that a subarray is *difficult* if no small subarray of it can be well-tiled and if it does not contain a degenerate subarray.

The following lemma concerns subarrays arbitrarily large and is in some sense the most essential and difficult part of $2\frac{1}{8}$ approximation.

Lemma 4. *Every difficult subarray S of type $T = \sqcup 2(B \sqcup 2)^n (n \geq 0)$, where B denotes either an empty type or types $\sqcup 1, \sqcup 1 \sqcup 1$ or $\sqcup 3$, has weight greater than $N(T) + \frac{5}{8}$.*

Every difficult subarray S of type $T = 2A \sqcup 2(B \sqcup 2)^n (n \geq 0)$, where B is as above and A denotes either an empty type or a subarray of type $\sqcup 1$, has weight greater than $N(T) + 1$.

We will say that a subarray is *proper*, if it is of type $\diamond k_1 \diamond k_2 \dots \diamond k_n$ or $\diamond k_1 \diamond k_2 \dots \diamond k_n$ - and its every subarray of type $T_i = \diamond k_1 \dots \diamond k_i$ consisting of the beginning $2i$ columns has weight less than $N(T_i) + 1$.

The corollary of Lemma 4 is

Corollary 1. (a) *For every proper subarray S*

$$well(S) \geq \frac{5}{8}.$$

(b) *For every proper subarray S of type $\diamond k_1 \diamond k_2 \dots \diamond k_n$ -*

$$nearly(S) \geq 1.$$

From the above corollary we get, that every subarray that the algorithm tries to well-tile, has weight at least $N(T)$. First, it is true for every block. Next, suppose we have a

subarray S of type T , that we could not well-tile, then by Corollary 1(a), its weight is greater than $N(T) + \frac{5}{8}$. If we extend it to the next block B of type T' , then the weight of SB is greater than $N(T) + \frac{5}{8} + N(T') + \frac{11}{16} > N(T) + N(T') + 1 = N(TT')$.

If we are able to well-tile any small subarray, then by Facts 1,2 and Corollary 1(b), we are able to well-tile the whole subarray S we are dealing with at the moment.

If the subarray S is followed by a degenerate subarray D , then by Lemma 3, Corollary 1(a) and Lemma 2, we have that $well(SD) > 1$.

Another implication of Corollary 1(b) is that we are always in a position to nearly-well- $2\frac{1}{8}$ -partition the subarray at the end of running the algorithm, as either it can be nearly-tiled i.e. nearly-well- $2\frac{1}{8}$ -partitioned as it has weight at least $N(T)$ and we use $N(T) + 1$ tiles or it has weight greater than $N(T) + 1$, but then in order to nearly-well- f -partition it, we can use $N(T) + 2$ tiles.

Theorem 2. *The approximation with a better factor using the lower bound used in this paper is impossible.*

Proof. Let us consider the subarrays of the type $\sqcup 2(\sqcup 1 \sqcup 2)^n$ having weight $N(T) + 1$, which for n will be equal to $5n + 3$. Let them have the following structure drawn in Fig. 4:

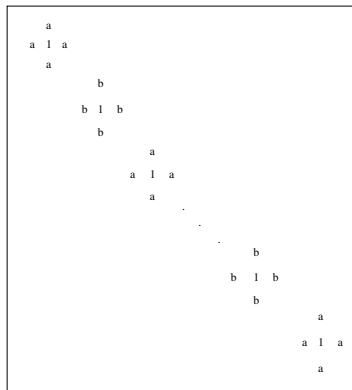


Fig. 4.

There are as many ones encircled by a as the columns of type 2 and as many ones encircled by b as columns of type 1. Additionally $b = \frac{a}{3}$. Let $a = \frac{9n+6}{16n+12}$. It can be noticed, that if the subarray is to be partitioned into $5n + 3$ tiles, then one tile will have to contain at least one 1 and two a or at least one 1, one a and three b . The weight of such a tile amounts to $2 + \frac{2n}{16n+12}$. □

References

1. P. Berman, B. DasGupta, S. Muthukrishnan, S. Ramaswami, *Efficient Approximation Algorithms for Tiling and Packing Problems with Rectangles*, J. Algorithms 41(2): 443-470 (2001).
2. M. Charikar, C. Chekuri, R. Motwani, Unpublished.
3. M. Grigni, F. Manne, *On the complexity of generalized block distribution*, Proc. 3rd intern. workshop on parallel algorithms for irregularly structured problems (IRREGULAR'96), Springer, 1996, LNCS 1117, 319-326.
4. Y. Han, B. Narahari, H.-A. Choi, *Mapping a chain task to chained processors*, Information Processing Letters 44, 141-148, 1992.
5. S. Khanna, S. Muthukrishnan, M. Paterson, *On approximating rectangle tiling and packing*, Proc. 19th SODA (1998), 384-393.
6. S. Khanna, S. Muthukrishnan, S. Skiena, *Efficient array partitioning*, Proc. 24th ICALP, 616-626, 1997.
7. K. Lorys, K. Paluch, *Rectangle Tiling*, Proc. 3rd APPROX, Springer, 2000, LNCS 1923, 206-213.
8. K. Lorys, K. Paluch, *A new approximation algorithm for RTILE problem*, Theor. Comput. Sci. 2-3(303): 517-537 (2003).
9. G. Martin, S. Muthukrishnan, R. Packwood, I. Rhee, *Fast algorithms for variable size block matching motion estimation with minimal error*.
10. S. Muthukrishnan, V. Poosala, T. Suel, *On rectangular partitioning in two dimensions: algorithms, complexity, and applications*, Proc. 7th ICDT, 1999, 236-256.
11. J. Sharp, *Tiling Multi-dimensional Arrays*, Proc. 12th FCT, Springer, 1999, LNCS 1684, 500-511.
12. A. Smith, S. Suri, *Rectangular Tiling in Multidimensional Arrays*, J. Algorithms 37(2): 451-467 (2000).