Information and Computation 206 (2008) 1334–1345

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

On the complexity of sequential rectangle placement in IEEE 802.16/WiMAX systems☆

Amos Israeli [a], Dror Rawitz [b,*,1], Oran Sharon [a]

[a] *Department of Computer Science, Netanya Academic College, Netanya 42100, Israel*
[b] *School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel*

ARTICLE INFO

Article history:
Received 17 July 2007
Revised 23 June 2008
Available online 26 July 2008

Keywords:
Approximation algorithms
IEEE 802.16/WiMAX systems
Scheduling
Sequential rectangle placement

ABSTRACT

We study the problem of scheduling transmissions on the downlink of IEEE 802.16/WiMAX systems that use the OFDMA technology. These transmissions are scheduled using a matrix whose dimensions are frequency and time, where every matrix cell is a time slot on some carrier channel. The IEEE 802.16 standard mandates that: (i) every transmission occupies a rectangular set of cells, and (ii) transmissions must be scheduled according to a given order. We show that if the number of cells required by a transmission is not limited (up to the matrix size), the problem of maximizing matrix utilization is very hard to approximate. On the positive side we show that if the number of cells of every transmission is limited to some constant fraction of the matrix area, the problem can be approximated to within a constant factor. As far as we know this is the first paper that considers this *sequential rectangle placement problem*.

1. Introduction

1.1. Background and motivation

The IEEE 802.16/WiMAX system [6] is an emerging standard for Wireless Local Loop (WLL) systems [13] that are designed to enable residential and business subscribers Broadband Wireless Access (BWA) to core networks, e.g., the public telephone network and the Internet. An IEEE 802.16 system consists of a Base Station (BS) and Subscriber Systems (SSs) as depicted in Fig. 1. The wireless link from the BS to the SSs (from the SSs to the BS, respectively) is called the downlink (uplink, respectively). One of the options to realize the physical layer in 802.16 is Orthogonal Frequency Division Multiplexing Access (OFDMA), which is a form of multicarrier modulation. In OFDMA, the transmission bandwidth on the downlink is divided into several subchannels that are used by the BS to transmit to the SSs in parallel. The transmission time over each subchannel is further divided into time slots. A predefined number of slots from all the subchannels together are grouped into periods called *subframes*. The same holds respectively on the uplink.

Notice that the downlink subframe is actually a time/frequency matrix, which for the sake of brevity is called from now on simply the *matrix*. The matrix' frequency dimension is equal to the number of the OFDMA subchannels, while the time dimension is equal to the number of time slots in each downlink subframe. An example of a matrix is given in Fig. 2.

☆ A preliminary version of this paper appeared in the 15th Annual European Symposium on Algorithms, 2007 [7].

* Corresponding author.
E-mail addresses: amos@netanya.ac.il (A. Israeli), rawitz@eng.tau.ac.il (D. Rawitz), oran@netanya.ac.il (O. Sharon).

[1] Part of the work on this paper was done while the author was a postdoc at the Caesarea Rothschild Institute, University of Haifa.

0890-5401/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2008.07.002

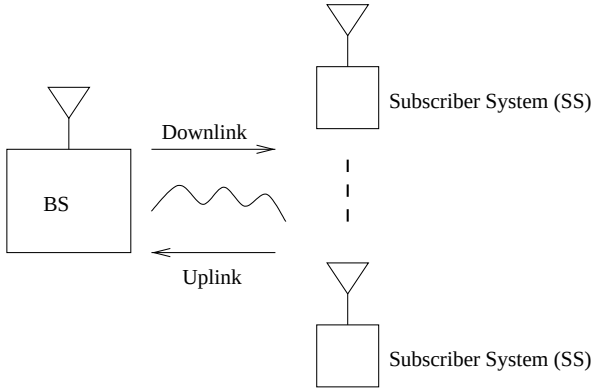

Fig. 1. The physical architecture of an 802.16 system.

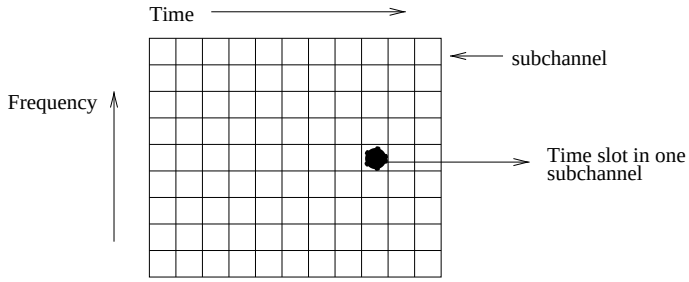

Fig. 2. The downlink time/frequency matrix in OFDMA.

In each time slot the BS may transmit some fixed number of bits in some subchannel. We refer to such a time slot (in some subchannel) as a *cell*. Each individual transmission of the BS to some SS is called a *packet*. In order to transmit packets on the downlink, each packet should be assigned a set of matrix cells on which the packet will be transmitted, and all sets must be disjoint. Every SS listens for some predefined number of time slots on predefined subchannels in order to receive packets destined to itself. These time slots and subchannels can change from one downlink subframe to another.

The assignment of packets to sets of matrix cells should follow some requirements: for each packet transmitted on the downlink, the IEEE 802.16 standard mandates that the set of scheduling matrix cells allocated for the packet transmission must be rectangular. Further, an IEEE 802.16 system is intended to support various "high level" protocols, such as ATM and IP. Thus, the system is supposed to have QoS capabilities for, e.g., ATM VCs. This means that the BS should be able to transmit packets according to some scheduling disciplines that guarantee delay bounds, such as Delay-Earliest-Due-Date [3] in which every packet is assigned a deadline for its transmission, and packets are transmitted according to these deadlines: those with earlier deadlines are transmitted first. In addition, it is also important to preserve FIFO order among transmitted packets in some data streams, e.g., in TCP connections. Keeping the relative order among packets in such connections is necessary in order to avoid a false activation of Fast Retransmit/Fast Recovery algorithms, which may cause lower transmission rates and reduced throughput [1].

To summarize, the BS needs to schedule its packets for transmission on the downlink matrix according to the following requirements: (i) packets must be transmitted in a given rigid order, e.g., FIFO, and (ii) every packet transmission requires a rectangular set of matrix cells.

Allocation of rectangular cell sets for each packet is a resource management problem for which the 802.16 standard specifies no algorithm. Any vendor that implements an 802.16 system is expected to implement its own allocating algorithm. In this paper, we investigate the complexity of this problem and develop an algorithm adhering to the aforementioned two requirements.

We note that the relative order among packets within a given matrix is not of concern to the problem we investigate in this paper. The given order of packets refers to the order in which the BS allocates transmission resources to the packets it transmits. It is up to the receiver to decide on the order in which it processes packets that it receives in a given matrix. This can be done by means that are out of the scope of this paper, e.g., by sequence numbers as in TCP [1].

1.2. Problem definition

In this paper, we investigate the *sequential rectangle placement problem* (SRP, for short). The input for this problem consists of:

Matrix : The (scheduling) matrix is given by its dimensions: $L \times H$. Without loss of generality we assume that $L \geqslant H$. We also define $S \triangleq L \cdot H$.

Jobs : The input sequence of jobs $N = J_1 J_2, \ldots J_n$. Each job J_i is associated with a *size* $r_i \in \mathcal{N}$ and a non-negative *weight* w_i.

A solution is a *placement* which is an assignment of a rectangular set of matrix cells for each job in some *prefix* of N, where the number of matrix cells assigned for J_i is not smaller than r_i, and all rectangles are *non-intersecting*. Throughout the paper, we assume, without loss of generality, that the sum of all the job sizes is at most $L \times H$. The goal of the sequential rectangle placement problem is to find a placement of the longest possible prefix of jobs of the input sequence. In other words, our goal is to find a placement for the jobs $J_1, \ldots J_t$, where t is as high as possible.

A *bounding rectangle* of job J_i is a rectangular set of cells whose area is at least r_i, and that does not contain any proper sub-rectangle whose area is at least r_i. That is, a bounding rectangle is a rectangle of length L_i and height H_i such that (1) $r_i \leqslant L_i \cdot H_i$, (2) $r_i > L_i \cdot (H_i - 1)$, and (3) $r_i > (L_i - 1) \cdot H_i$. Henceforth, we assume without loss of generality that a rectangle that is reserved for a job is a bounding rectangle. Notice that a bounding rectangle can be larger than the job size. For instance, a bounding rectangle for a job of size 5 can be of size 6, with length 3 and height 2. In this example, the bounding rectangle occupies one additional cell above the minimal number of matrix cells needed for placing this job. Thus, it wastes resources because free cells in a bounding rectangle cannot be used by other jobs [6]. On the other hand, the somewhat weakened requirement to use bounding rectangles yields some additional freedom in rectangle shaping, e.g., in the above example we are not restricted to a rectangle of dimensions 5×1 and can also use a rectangle of dimensions 3×2.

Given a placement of the prefix $J_1, \ldots J_t$, the sum $\sum_{i=1}^{t} w_j$ is referred to as the weight of the placement. We evaluate the placement by its weight. In the sequel, we prove that SRP is NP-hard, and in general is also hard to approximate. Therefore, we turn to look for approximation algorithms for some special cases. We consider two weight functions: the *unit* weight function, $w_i = 1$ for every job J_i, and the *proportional* weight function, $w_i = r_i$ for every job J_i. In the unit weight function the weight of the placement is simply the number of jobs that were placed on the matrix, while in the second the weight of the placement is the total area that is occupied by the placed jobs. The motivation for these two weight functions is as follows. In the unit weight function we count the number of served jobs, which is important in order to serve as many clients in the system as possible. In the proportional weight function we evaluate the amount of transmission resources we use (cells in the transmission matrix). This weight function is important because the wireless transmission resource is a very scarce resource that must be used as efficiently as possible [12].

1.3. Related work

As far as we know this is the first paper to consider SRP. However, the following *rectangle packing problem* was considered by several studies. The input consists of a set of rectangles $R_i = (a_i, b_i)$, for $i = 1, \ldots, n$, where a_i and b_i are the length and height of R_i, and a larger rectangle $R = (a, b)$. Each rectangle has a profit p_i. The goal is to pack a subset of the rectangles into R such that the total profit of packed rectangles is maximized. The packed rectangles may not overlap. This problem in NP-hard since it contains *knapsack* as the special case in which $a_i = a$ for every i. Constant factor approximation algorithms for this problem were given in [10,9], and a PTAS that packs the rectangles into a rectangle that is slightly bigger than R was presented in [4].

Note that SRP is very different from this rectangle packing problem. First, in SRP the requirement that the output placement is computed for a *prefix* of the input job sequence is crucial, while in the latter problem the input is an unordered set and the only optimized parameter is the total weight of the successfully placed rectangles. Furthermore, although in some studies of the rectangle packing problem, the given rectangles may be rotated (see, e.g., [10]) the dimensions of the rectangles are predetermined, while in SRP the input consists of requests for areas only.

The problem of *scheduling malleable parallel tasks* can also be viewed as a sort of rectangle packing problem. In this problem one is given a set of n jobs to be scheduled on m identical processors with linear topology. Similarly to SRP, each job has a work requirement, and it must be scheduled on processors contiguously for an amount of time so that the product of number of processors and time is at least the work requirement. In other words, each job is alloted a rectangle that is induced by a consecutive set of processors and a time interval whose area is at least the work requirement. The goal in this problem is to minimize the makespan. Jansen and Porkolab [8] considered that case where the scheduling is done on a fixed number of parallel processors and propose a PTAS for this case. Turek et al. [14] showed that any r-approximation algorithm for the problem of scheduling non-malleable tasks can be used to obtain an r-approximation algorithm for malleable tasks. Mounié et al. [11] improved the result of [14] and obtained a $\sqrt{3}$-approximation algorithm for the problem of scheduling malleable tasks. For more details about this problem the reader is referred to [14].

Recently, Cohen and Katzir [2] studied the OFDMA scheduling problem. They considered a version of the problem where the FIFO requirement is relaxed and obtained hardness results and constant factor approximation algorithms.

1.4. Our results

The results of our paper are both negative and positive. On the negative side we prove that SRP is very hard to approximate. Specifically, we show that it is NP-hard to approximate SRP within a factor of $O(n^{1-\varepsilon})$ even when restricted to the case of unit weights. We also show that it is NP-hard to approximate SRP within a factor of $\frac{1}{2}(\sqrt{S}-1)$ or within a factor of cn^2 for some constant c, even when restricted to the case of proportional weights. Our third hardness result is that SRP is NP-hard even when $r_i \leqslant \frac{1+\varepsilon}{n} \cdot S$ for every i, for any constant $\varepsilon > 0$. Moreover, we show that for this special case of SRP (with arbitrary weights) there cannot exist an r-approximation algorithm for any ratio r, unless P=NP. All our hardness results are obtained using reductions from the well known NP-hard *partition problem* (PARTITION) [5].

On the positive side we present an approximation algorithm for SRP with proportional weights for the special case where $r_i \leqslant \beta S$, for some $\beta \in (0,1)$. The approximation ratio of the algorithm is $1-\beta-\varepsilon$ and its running time is $O(n)$, for every constant $\varepsilon > 0$ (Albeit, the running time is not polynomial in $\frac{1}{\varepsilon}$). Our algorithm works as follows. First, it divides the range of job sizes into $O(\log H)$ subranges. For each such size subrange, the jobs in this subrange are further divided into job sets such that each set holds $O(\sqrt{L})$ jobs. The jobs in each such set are placed on a separate set of matrix rows. Furthermore, they are placed such that the unused area for each set is relatively small. In the analysis of our algorithm we compare the solution obtained by the algorithm to the area of the matrix, S, which is an upper bound on the weight of an optimal solution in the case of proportional weights.

1.5. Organization

The remainder of the paper is organized as follows. Section 2 contains our hardness results. Our approximation algorithm is presented and analyzed in Section 3. Finally, we conclude in Section 4 with some open problems.

2. Hardness results

In this section, we present our hardness results. In the first part of the section we show that SRP is NP-hard to approximate within a factor of $O(n^{1-\varepsilon})$ even for the case of unit weights and within a factor of $\frac{1}{2}(\sqrt{S}-1)$ or within a factor of cn^2 for some constant c, even in the case of proportional weights. In the second part of the section we show that SRP is NP-hard even when $r_i \leqslant \frac{1+\varepsilon}{n} \cdot S$ for every i, for any constant $\varepsilon > 0$. Moreover, we show that for this special case of SRP there cannot exist an r-approximation algorithm for any r, unless P=NP.

2.1. Proportional and unit weights

We present a reduction from PARTITION to SRP. Intuitively, given a PARTITION instance the reduction works as follows. First, it creates m jobs whose sizes are inflated versions of the PARTITION numbers. Second, it produces a square matrix whose length (or height) is a bit more than half of the sum of the inflated numbers. Then, it adds a huge job whose dimensions are $(L-1) \times (H-1)$. The role of this job is to make sure that the other jobs use only one row and one column. The last set of jobs contains jobs of size one.

Reduction 1. *Let $(x_1,\ldots,x_m)$ be a* PARTITION *instance, and denote $B \triangleq \frac{1}{2}\sum_{j=1}^{m} x_i$ (henceforth, we assume that B is integral). We construct an* SRP *instance $(r_1,\ldots,r_n,L,H)$ as follows. First, let m' be an even integer. (The exact value of m' will be determined later.) We define $b = m'(B+\frac{1}{2})$ and $L = H = b+1$. Also, let $n = 2+m+m'$, and let $r_j = m' \cdot x_j$ for every $1 \leqslant j \leqslant m$, $r_{m+1} = b^2$, and $r_j = 1$ for every $m+2 \leqslant j \leqslant n$. We refer to jobs $J_1,\ldots J_m$ as medium jobs and to jobs $J_{m+2},\ldots J_n$ as small jobs.*

Note that the reduction is polynomial in case $m' = O(m^k)$ for some constant k.

Observation 1. $\sum_{j=1}^{n} r_j = L \cdot H$.

Proof. $\sum_{j=1}^{n} r_j = b^2 + \sum_{j=1}^{m} m' \cdot x_j + (n-m-1) = b^2 + 2B \cdot m' + (m'+1) = b^2 + 2b + 1 = L \cdot H$. □

We show that if $(x_1,\ldots,x_m)$ belongs to PARTITION then all jobs can be placed. Otherwise, we will not be able to place more than m jobs.

Lemma 1. $(x_1,\ldots,x_m) \in$ PARTITION *if and only if the jobs $J_1,\ldots J_n$ can be placed on an $L \times H$ matrix. Furthermore, if $(x_1,\ldots,x_m) \notin$* PARTITION *then jobs $J_1,\ldots J_{m+1}$ cannot be placed on the matrix.*

Proof.

First, observe that since $r_{m+1} = b^2$, job J_{m+1} must be placed in such a way that leaves one empty column and one empty row, if it is placed on an $L \times H$ matrix. Hence, every other job must be placed as a rectangle of either length 1 or height 1. It follows that, if job J_{m+1} is placed on the matrix then the only freedom we have in deciding how to place a job $J_j \neq J_{m+1}$ is whether to place it in the empty row or in the empty column (see example in Fig. 3).

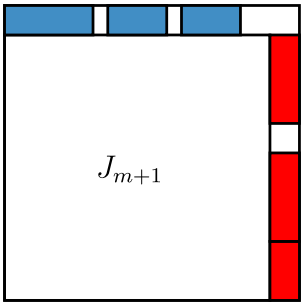

Fig. 3. Example of placement of J_{m+1}.

If $(x_1,\ldots,x_m) \in$ PARTITION then there exists a subset $I \in \{1,\ldots,m\}$ such that $\sum_{i\in I} x_i = B$. Hence, we can place the medium jobs on the matrix as follows. If $j \in I$ we place the job in a rectangle of length r_j and height 1 in the empty row of the matrix, and otherwise we place the job as a rectangle of length 1 and height r_j in the empty column of the matrix. We can place all the medium jobs that are contained in I (not contained in I) in a single row (column) since $m'B < b$. The small jobs can be placed in the remaining space due to Observation 1. It follows that if $(x_1,\ldots,x_m) \in$ PARTITION, then all jobs can be placed on the matrix.

Now, for the other direction, let $(x_1,\ldots,x_m) \notin$ PARTITION and assume that jobs $J_1,\ldots J_{m+1}$ are placed on the matrix. Denote by S_1 the set of medium jobs whose rectangle is placed in the empty row (the row that is left empty after the placement of job J_{m+1}), and denote by S_2 the set of medium jobs whose rectangle is placed in the empty column. We now claim that it follows that $\sum_{j\in S_1} x_j = \sum_{j\in S_2} x_j = B$. Notice that if this is not the case then $|\sum_{j\in S_1} x_j - \sum_{j\in S_2} x_j| \geqslant 2$, and therefore $|\sum_{j\in S_1} r_j - \sum_{j\in S_2} r_j| \geqslant 2m'$. Without loss of generality let $\sum_{j\in S_1} r_j > \sum_{j\in S_2} r_j$. Hence, since $\sum_{j=1}^{m} r_j = m' \cdot 2B$ we get that $\sum_{j\in S_1} r_j \geqslant m' \cdot B + m' = m'(B+1)$ and $\sum_{j\in S_2} r_j \leqslant m' \cdot B - m' = m'(B-1)$. It follows that $\sum_{j\in S_1} r_j > m'(B+\frac{1}{2}) + 1 = b + 1 = L$, a contradiction. (Note that here we implicitly assumed that $m' > m'/2 + 1$, and this holds for $m' \geqslant 3$.) □

The lemma directly implies that SRP is NP-hard. Next, we show that this problem is also very hard to approximate. In the sequel we denote the optimum of an SRP instance by OPT.

Theorem 1. *It is NP-hard to approximate* SRP *within a factor of* $\frac{\sqrt{5}-1}{2}$ *or within a factor of* cn^2 *for some constant* c, *even for the restricted case of proportional weights.*

Proof. We use Reduction 1 by letting m' be the first even number that is greater than or equal to m. By Lemma 1 if $(x_1,\ldots,x_m) \in$ PARTITION then all jobs can be placed, and the weight of the placement is $(b+1)^2$. On the other hand, if $(x_1,\ldots,x_m) \notin$ PARTITION, then the weight of any placement is less than $(b+1)^2 - b^2 = 2b+1$. Hence, for instances that are generated by Reduction 1 with proportional weights, distinguishing between instances for which OPT $= (b+1)^2$ and instances for which OPT $\leqslant 2b$ is NP-hard.

Suppose now that there exists a polynomial algorithm that computes $\frac{\sqrt{5}-1}{2}$-approximate solutions for SRP. Then, we can use it on instances generated by Reduction 1 to determine whether OPT $= (b+1)^2$ and thus solve PARTITION. Since $\frac{\sqrt{5}-1}{2} = \frac{b}{2}$ for such instances, it follows that if OPT $= (b+1)^2$ then the algorithm computes a placement whose weight is at least $2b+4$. Otherwise, the algorithm computes a placement of weight at most $2b$. We therefore conclude that there cannot exist a $\frac{\sqrt{5}-1}{2}$-approximation algorithm for SRP, unless P=NP.

Using similar arguments we can show that it is NP-hard to approximate SRP within a factor of cn^2 for some constant c. By definition $b = m'(B+\frac{1}{2})$, hence $\frac{b}{2} \geqslant \frac{m}{2}(B+\frac{1}{2})$. Since $B \geqslant m$ and $n = \Theta(m)$, it follows that there exists a constant c such that $\frac{b}{2} \geqslant cn^2$. It follows that a cn^2-approximation algorithm can distinguish between Reduction 1 instances for which OPT $= (b+1)^2$ and instances for which OPT $\leqslant 2b$. □

Theorem 2. *It is NP-hard to approximate* SRP *within a factor of* $O(n^{1-\varepsilon})$ *for every* $\varepsilon > 0$, *even for the restricted case of unit weights.*

Proof. For a given ε, we use Reduction 1 by setting $k = \lceil \frac{2}{\varepsilon} \rceil$ and letting m' be the first even number that is greater than or equal to m^k. Note that k is a constant, and therefore the reduction is polynomial. In Lemma 1 we showed that if $(x_1,\ldots,x_m) \in$ PARTITION then all $n = m + m' + 2$ jobs can be placed on the matrix, and that otherwise at most m jobs can be placed on the matrix. Thus, for instances that are generated by Reduction 1 with unit weights, distinguishing between instances for which OPT $= m + m' + 2$ and instances for which OPT $\leqslant m$ is NP-hard.

Suppose now that there exists a polynomial time algorithm that computes $O(n^{1-\varepsilon})$-approximate solutions for SRP. If this is the case, we can use this algorithm on instances generated by Reduction 1 to determine whether OPT $= \Theta(m^k)$ and thus solve PARTITION.

Notice that since $n = m + m' + 2$ and $m' = \Theta(m^k)$, we get that

$$O(n^{1-\varepsilon}) = O(n^{1-\frac{2}{\lceil 2/\varepsilon \rceil}}) = O(n^{1-\frac{2}{k}}) = \Theta(n^{\frac{k-2}{k}}) = \Theta(m^{k-2}).$$

Therefore, it follows that if OPT $= \Theta(m^k)$ then the algorithm computes a placement that places $\Omega(m^2)$ jobs. Otherwise, the algorithm outputs a placement containing at most m jobs. We conclude that there cannot exist an $O(n^{1-\varepsilon})$-approximation algorithm for SRP, unless P=NP. □

2.2. General weights and small jobs

In this section, we show that SRP is NP-hard even when $r_i \leqslant \frac{1+\varepsilon}{n} \cdot S$ for every i, for any constant $\varepsilon > 0$. Moreover, we show that for this special case of SRP there cannot exist an r-approximation algorithm for any r.

As a first step we show that PARTITION remains hard even if all the numbers in the instance are odd and not very large with respect to the sum of the numbers. For an odd integer q we define PARTITION$_q$ to be the language that contains PARTITION instances $(x_1,\ldots,x_m)$ that satisfy the following additional two conditions: (i) x_i is odd for every i, and (ii) $x_i \leqslant \frac{1}{q}\sum_j x_j$ for every i.

Reduction 2. *Let $(x_1,\ldots,x_m)$ be a* PARTITION *instance, and denote $B \triangleq \frac{1}{2}\sum_{j=1}^{m} x_i$. We construct an instance $(y_1,\ldots,y_M)$ of* PARTITION$_q$*, where $M = 2m + q - 1$. We define $y_i = 2m \cdot x_i + 1$ for every $i \in \{1,\ldots,m\}$, $y_i = 1$ for every $i \in \{m+1,\ldots,2m\}$, and $y_i = 4mB + 2m - 1$ for every $i \in \{2m+1,\ldots,M\}$.*

We first prove that PARTITION$_q$ is NP-hard if q is polynomial in m.

Lemma 2. PARTITION$_q$ *is NP-hard for $q = O(m^c)$ for some constant c.*

Proof. First, Reduction 2 is polynomial if $q = O(m^c)$ for some constant c.

$y_1,\ldots,y_M$ are odd by the construction. We show that $y_i \leqslant \frac{1}{q}\sum_i y_i$ for every i. Clearly, $y_i \leqslant 4mB + 2m - 1$ for every i. On the other hand,

$$\sum_{i=1}^{M} y_i = \sum_{i=1}^{m}(2mx_i + 1) + m + (q-1)\cdot(4mB + 2m - 1) = q\cdot(4mB + 2m) - (q-1).$$

Hence, $y_i \leqslant \frac{1}{q}\sum_i y_i$.

Next, we show that $(x_1,\ldots,x_m) \in$ PARTITION if and only if $(y_1,\ldots,y_M) \in$ PARTITION$_q$. If $(x_1,\ldots,x_m) \in$ PARTITION then there exists an index set I such that $\sum_{i\in I} x_i = \sum_{i\notin I} x_i$. In this case let $I' = I \cup \{m+1,\ldots,2m-|I|\} \cup \{2m+1,\ldots,2m+(q-1)/2\}$ (recall that q is odd). It is not hard to verify that $\sum_{i\in I'} y_i = \sum_{i\notin I'} y_i$.

For the other direction, if $(y_1,\ldots,y_M) \in$ PARTITION$_q$ then there exists an index set I' such that $\sum_{i\in I'} y_i = \sum_{i\notin I'} y_i$. It must be the case that $|I' \cap \{2m+1,\ldots,2m+q-1\}| = \frac{q-1}{2}$, since otherwise the total sum of one of the partitions is at least

$$(4mB + 2m - 1)\left(\frac{q-1}{2} + 1\right) = \frac{q+1}{2}\cdot(4mB + 2m) - \frac{q+1}{2} > \frac{1}{2}\sum_i y_i.$$

Hence, $\sum_{i\in I', i\leqslant 2m} y_i = \sum_{i\notin I', i\leqslant 2m} y_i$. Since $\sum_{i=m+1}^{2m} r_i = m$, it follows that $\sum_{i\in I} x_i = \sum_{i\notin I} x_i$, where $I = I' \cap \{1,\ldots,m\}$. Otherwise, $|\sum_{i\in I} x_i - \sum_{i\notin I} x_i| \geqslant 2$, which means that $|\sum_{i\in I'} y_i - \sum_{i\notin I'} y_i| > 2\cdot 2m - m - m > 0$. A contradiction. □

Now we move to the hardness result.

Reduction 3. *Let $(x_1,\ldots,x_m)$ be a* PARTITION$_q$ *instance. We construct an* SRP *instance $(r_1,\ldots,r_n,L,H)$ as follows. Define $n = m$, $r_i = x_i$ for every i, $L = \frac{1}{2}\sum_i x_i$ and $H = 2$ (the weights of the jobs are defined later).*

Clearly, the reduction is polynomial. Moreover, notice that $\sum_i r_i = L \cdot H$ and that $r_i \leqslant \frac{1}{q}\cdot S$, for every i.

Lemma 3. $(x_1,\ldots,x_m) \in$ PARTITION$_q$ *if and only if all the jobs can be placed on the matrix.*

Proof. If $(x_1,\ldots,x_m) \in$ PARTITION$_q$ then there exists an index set I such that $\sum_{i\in I} x_i = \sum_{i\notin I} x_i$. In this case it is not hard to verify that we can place the jobs that correspond to I in the top row, and the rest in the bottom row.

Since $\sum_i r_i = L \cdot H$ and $r_1,\ldots,r_n$ are odd, it follows that if we are able to place all the jobs on the matrix, then J_i must be placed as a rectangle of height 1 for every i. Such a placement induces a partition of $(x_1,\ldots,x_m)$. □

The previous lemma directly implies that it is NP-hard to distinguish between OPT $= \sum_{i=1}^{n} w_i$ and OPT $= \sum_{i=1}^{n-1} w_i$. Hence, SRP is hard in the case of relatively small jobs even for unit or proportional weights.

Corollary 3. *Let $\beta \in (0,1)$.* SRP *is NP-hard even when $r_i \leqslant \beta \cdot S$ for every i. Moreover, for this special case of* SRP *there cannot exist an r-approximation algorithm for any r, unless P=NP.*

Proof. We use Reduction 2 with an odd number $q \geqslant \lceil 1/\beta \rceil$. Now, define $w_i = 0$ for every $i < n$, and $w_n = 1$. Hence, by Lemma 3 it is NP-hard to distinguish between OPT $= 0$ and OPT > 0. This implies that there cannot exist an r-approximation algorithm for any r, unless P=NP. □

However, we can obtain an even stronger hardness result.

Corollary 4. SRP *is NP-hard even when $r_i \leqslant \frac{1+\varepsilon}{n} \cdot S$ for every i, for any constant $\varepsilon > 0$. Moreover, for this special case of* SRP *there cannot exist an r-approximation algorithm for any r, unless P=NP.*

Proof. We use Reduction 2 and set $q = 2m^2 + 1$. The resulting instance contains $M = 2m + 2m^2$ integers. Observe that

$$\frac{M}{q} = \frac{2m^2 + 2m}{2m^2 + 1} = 1 + \frac{2m - 1}{2m^2 + 1} = 1 + \frac{1}{\Theta(m)}.$$

Hence, for every constant $\varepsilon > 0$ there exists m_0 such that $\frac{M}{q} \leqslant 1 + \varepsilon$ for every $m \geqslant m_0$. By Lemma 3 it follows that SRP is NP-hard even when $r_i \leqslant \frac{1+\varepsilon}{n} \cdot S$ for every i, for any constant $\varepsilon > 0$.

The rest of the proof is similar to the proof of Corollary 3. □

3. An algorithm for SRP with proportional weights

In this section, we develop a linear time approximation algorithm for SRP with proportional weights. For the special case in which $r_i \leqslant \beta \cdot S$ for every job J_i where $\beta \in (0,1)$, the algorithm achieves an approximation ratio of $1 - \beta - \varepsilon$, for any constant $\varepsilon > 0$.

3.1. Definitions and notation

Before presenting the algorithm, we introduce some notation. The placement computed by our algorithm is *row oriented*.

Definition 1. *A job placement of a job set JS is called row oriented if the following conditions hold:*

(1) *The set JS is divided into some disjoint subsets $\{JS_i\}_{i=1}^{k}$.*
(2) *The jobs in each job set JS_i are placed on a set of consecutive matrix rows dedicated to JS_i. The rows occupied by distinct job sets are distinct.*
(3) *For any job set JS_i, and for any job $J_k \in JS_i$, the base of the bounding rectangle of J_k is placed on the first row dedicated to JS_i and the height of the bounding rectangle is equal to the number of rows dedicated to JS_i.*

Let P be a row oriented placement and let JS_i be a job set of P. The number of rows *required by* JS_i, denoted by ROWS(JS_i), is the minimal number of rows on which all jobs of JS_i can be placed adhering to Definition 1. The number of rows *required by* P, denoted by ROWS(P), is the total number of rows required by the job sets of P.

Our algorithm maintains a collection of disjoint job sets, where the sizes of all jobs in each set are within a certain range. The ranges for the job sets depend on L and H, the matrix dimensions, and are defined as follows.

Definition 2. *We divide the jobs into three types* [2]*:*

- ***Small jobs:*** *A job J_k is called small if $r_k \leqslant 2\sqrt{L}$.*
- ***Medium jobs:*** *A job J_k is called medium if $2 \cdot \sqrt{L} < r_k \leqslant \frac{H}{2} \cdot \sqrt{L}$.*
  A medium job is called i-medium if $2^{i-1}\sqrt{L} < r_k \leqslant 2^i\sqrt{L}$ for $i \in \{2, \ldots, \log H - 1\}$. [3]
- ***Large jobs:*** *A job J_k is called large if $r_k > \frac{H}{2} \cdot \sqrt{L}$.*

We consider row oriented placements in which for each type of a job set, the number of rows required by the set is limited.

Definition 3. *A row oriented placement P is called bounded if the job sets of P follow Definition 2 and the number of rows required by each job set is bounded as follows:*

- ***Small jobs:*** *A job set of small jobs can have at most one row.*

[2] The distinction between job sizes is different from the one that was made in Section 2.
[3] For now, we assume that $\log H$ is integral for reasons of clarity.

Algorithm 1 - Placement$(L,H,N = J_1,\ldots J_n)$

1: $P \leftarrow \emptyset$; $i \leftarrow 0$
2: **repeat**
3: $P' \leftarrow P$; $i \leftarrow i+1$
4: Let M be the open set of P that corresponds to J_i (according to Definition 3)
If there is no such job set, then open such a job set
5: **if** $\sum_{J_j \in M \cup \{J_i\}} \left\lceil \frac{r_j}{\text{MAX}(M)} \right\rceil \leqslant L$ **then**
6: $M \leftarrow M \cup \{J_i\}$ ▷ *J_i fits in open job set M*
7: **if** $r_i > 2\sqrt{L}$ **then**
8: $\text{ROWS}(M) \leftarrow \min\left\{ \left\lceil \frac{\sum_{J_j \in M} r_j}{L-|M|} \right\rceil, \text{MAX}(M) \right\}$ ▷ *J_i is medium or large*
9: **end if**
10: **else**
11: Close M and open a new job set for J_i with $\left\lceil \frac{r_i}{L} \right\rceil$ rows ▷ *No room for J_i; M is closed*
12: **end if**
13: **until** $\text{ROWS}(P) > H$ or $i = n$
14: **if** $\text{ROWS}(P) > H$ **then** $P \leftarrow P'$ ▷ *Revert to previous placement*
15: **return** P

- ***Medium jobs:*** *A job set of i-medium jobs, where $i \in \{2,\ldots,\log H - 1\}$, can have at most 2^i rows.*
- ***Large jobs:*** *A job set of large jobs can have at most $2H$ rows (note that the value $2H$ is used for intermediate computations. The output placement P satisfies: $\text{ROWS}(P) \leq H$).*

Henceforth, the maximal number of matrix rows allowed for JS_i is denoted by $\text{MAX}(JS_i)$.

3.2. The algorithm

In this section, we present an algorithm for SRP with proportional weights called Algorithm **Placement**. The algorithm gets as input the matrix dimensions, L and H, and the sequence of jobs, $N = J_1,\ldots J_n$, and computes a row oriented placement P. A job set of P can be either *open* or *closed*, where jobs can be added only to open job sets. At any given time during execution of the algorithm there exists a single open job set for every range of job sizes specified by Definition 2. Therefore, at any given time, the number of non-empty open sets is at most $\log H$.

Algorithm **Placement** (see Algorithm 1) works as follows. After placing the jobs $J_1,\ldots J_{i-1}$, it tries to place J_i. The algorithm tries to add J_i to its corresponding open set (according to Definition 2), and if it fails, i.e., if the placement of $J_1,\ldots J_i$ requires more than H matrix rows, then the algorithm terminates with the placement of $J_1,\ldots J_{i-1}$.

Let M be the open set that corresponds to J_i. If J_i is large than it is simply put in M which is the single open set of large jobs. If J_i is either small or medium the algorithm places it in M, as long as the number of rows required for M does not exceed $\text{MAX}(M)$ (Line 6). If there is no room in M, then M is closed and a new set containing J_i is opened (Line 11).

After adding J_i to M the number of rows required for M is updated. If M is small than $\text{ROWS}(M) = 1$, otherwise we set

$$\text{ROWS}(M) = \min\left\{ \left\lceil \frac{\sum_{J_j \in M} r_j}{L - |M|} \right\rceil, \text{MAX}(M) \right\}. \tag{1}$$

As we shall see in the sequel this number ensures that (i) there is enough room for the jobs of M, and (ii) there is not a lot of wasted space after placing the jobs of M on $\text{ROWS}(M)$ matrix rows.

We note that the number of rows in (1) can be reduced by substituting the sum $\sum_{J_j \in M} r_j$ with $\sum_{J_j \in M} r_j - |M|$. However, this does not improve the approximation ratio. Furthermore, in the conference version of this paper [7], we used binary search to compute the minimum number of rows required for the job set M, but this increased the running time without improving the approximation ratio.

In the next lemma, we show that the computed placement P is indeed feasible.

Lemma 4. *Let M be an open job set of P. Then the jobs of M can be placed on $\text{ROWS}(M)$ matrix rows.*

Proof. First, if M contains small jobs, then $\text{ROWS}(M) = 1$ and we are done. If M contains either large or medium jobs we need to show that the jobs of M fit on $\text{ROWS}(M)$ matrix rows. Observe that due to Line 5 the jobs of M fit on $\text{MAX}(M)$ rows. Hence, for the rest of the proof we assume that

$$\text{ROWS}(M) = \left\lceil \frac{\sum_{J_j \in M} r_j}{L - |M|} \right\rceil < \text{MAX}(M).$$

We now have to show that the sum of bases of all bounding rectangles of jobs in M is not greater than L. The length of the base of the bounding rectangle of J_i is

$$B_i = \left\lceil \frac{r_i}{\text{ROWS}(M)} \right\rceil \leqslant \frac{r_i}{\text{ROWS}(M)} + 1 \leqslant \frac{r_i(L - |M|)}{\sum_{J_i \in M} r_i} + 1.$$

Hence,

$$\sum_{J_i \in M} B_i \leqslant \sum_{J_i \in M} \left(\frac{r_i(L - |M|)}{\sum_{J_i \in M} r_i} + 1 \right) = \frac{(L - |M|) \sum_{J_i \in M} r_i}{\sum_{J_i \in M} r_i} + |M| = L$$

as required. □

It is not hard to verify that Algorithm **Placement** invests $O(1)$ time in any job. We note that P' was added to the description of the algorithm for reasons of readability. There is no need to 'copy' the whole placement. When $\text{ROWS}(P) > H$ we only need to revert to the previous placement, and this takes $O(1)$ time. Hence, the running time of Algorithm **Placement** is $O(n)$.

3.3. Approximation ratio

If Algorithm **Placement** managed to place all jobs, then it obtained an optimal solution. Hence, in the sequel we assume that Algorithms **Placement** succeeded in placing jobs $J_1, \ldots J_t$, but failed to place jobs $J_1, \ldots J_{t+1}$. We compute the wasted matrix space of the placement P of jobs $J_1, \ldots J_t$.

Observe that the space wasted by P is comprised of two parts. First, there is the space that is wasted within the rows that P uses. The second part is the space that P does not use because this part is not large enough to accommodate J_{t+1}. We first bound the wasted space due to open job sets and due to closed job sets. Then, we calculate the overall wasted space within the rows that P uses. Afterwards, we bound the space that is not used by P. We show that the total wasted space is at most $\left(\frac{\log H}{H} + \frac{3}{\sqrt{L}} + \beta\right) \cdot S$.

We start the analysis by bounding the number of jobs in a job set.

Lemma 5. *Let M be a job set of medium or large jobs. Then, $|M| \leqslant 2\sqrt{L}$.*

Proof. First, assume that M is a set of i-medium jobs. By Definition 3, the number of rows in M is at most 2^i. Since the size of the smallest job in M is at least $2^{i-1}\sqrt{L}$, the length of the base of each job in M is at least $\frac{2^{i-1} \cdot \sqrt{L}}{2^i} \geqslant \frac{\sqrt{L}}{2}$. Since the base of the bounding rectangle of each job resides on the first matrix row dedicated to M, the number of jobs in M is not greater than $\lfloor \frac{L}{\sqrt{L}/2} \rfloor \leqslant 2\sqrt{L}$.

Now, assume that M is the (single) set of large jobs. Since the sum of the sizes of the jobs in the input is at most $L \cdot H$, and the size of a large job is at least $\frac{H}{2}\sqrt{L}$, there can be at most $2\sqrt{L}$ large jobs. □

Denote the wasted space due to job set M by $A_w(M)$. We first bound the waste in closed sets.

Lemma 6. *Let M be a closed set. Then, $A_w(M) \leqslant \text{ROWS}(M) \cdot 3\sqrt{L}$.*

Proof. The lemma is immediate for small jobs, since $A_w(M) < 2\sqrt{L}$ if M contains small jobs. Also, notice that there is no closed set of large jobs. Observe that if $L < 9$, then the lemma is trivial. Hence, for the rest of the proof we assume that $L \geqslant 9$ and that M is a job set consisting of i-medium jobs.

Now, by Line 5 we know that there was a job, denoted by J_q, that was rejected from M just before M was closed since $M \cup \{J_q\}$ did not fit on $\text{MAX}(M)$ rows, namely because

$$\sum_{J_j \in M \cup \{J_q\}} \left\lceil \frac{r_j}{\text{MAX}(M)} \right\rceil > L.$$

Since $\text{MAX}(M) - 1$ is the maximum wasted space per job, we get that

$$\text{MAX}(M) \left\lceil \frac{r_j}{\text{MAX}(M)} \right\rceil \leqslant r_j + \text{MAX}(M) - 1$$

Hence,

$$\sum_{J_j \in M} [r_j + (\text{MAX}(M) - 1)] + r_q > \text{MAX}(M) \cdot L.$$

(The addition of J_q requires more space than $\text{MAX}(M)$ rows.)

Denote by $A'_w(M)$ the wasted space when placing M on $\text{MAX}(M)$ rows. It follows that

$$A'_w(M) = \text{MAX}(M) \cdot L - \sum_{J_j \in M} r_j < (\text{MAX}(M) - 1) \cdot |M| + r_q$$

Since $|M| \leqslant 2\sqrt{L}$ by Lemma 5, and since $r_q \leqslant 2^i\sqrt{L} = \text{MAX}(M) \cdot \sqrt{L}$, we get that

$$A'_w(M) < \text{MAX}(M) \cdot 2\sqrt{L} + \text{MAX}(M) \cdot \sqrt{L} < \text{MAX}(M) \cdot 3\sqrt{L}.$$

Now, due to Line 8 $\text{ROWS}(M) \leqslant \text{MAX}(M)$. Hence,

$$A_w(M) \leqslant \text{ROWS}(M) \cdot L - (\text{MAX}(M) \cdot L - A'_w(M)) < \text{MAX}(M) \cdot 3\sqrt{L} - (\text{MAX}(M) - \text{ROWS}(M)) \cdot L \leqslant \text{ROWS}(M) \cdot 3\sqrt{L}$$

where the last inequality follows from $L \geqslant 9$. □

Next, we bound the waste in open sets.

Lemma 7. *Let M be an open set of P. Then, $A_w(M) < L + 2\sqrt{L} \cdot (\text{ROWS}(M) - 1)$.*

Proof. First, notice that if $\text{ROWS}(M) = 1$ then we are done. Hence, the lemma is immediate for the set of small jobs. Assume that M is either i-medium or large and that $\text{ROWS}(M) > 1$.

Since

$$\text{ROWS}(M) < \frac{\sum_{J_j \in M} r_j}{L - |M|} + 1$$

it follows that

$$\sum_{J_i \in M} r_i > (\text{ROWS}(M) - 1)(L - |M|) = \text{ROWS}(M) \cdot L - L - (\text{ROWS}(M) - 1)|M|.$$

Hence,

$$A_w(M) \leqslant L \cdot \text{ROWS}(M) - \sum_{J_i \in M} r_i < L + (\text{ROWS}(M) - 1)|M| \leqslant L + 2\sqrt{L}(\text{ROWS}(M) - 1)$$

where the last inequality is due to Lemma 5. □

Next, we bound the total waste within the matrix rows used by P.

Lemma 8. *The space wasted within the rows that P uses is less than $L \cdot \log H + 3\sqrt{L} \cdot \text{ROWS}(P)$.*

Proof. By Lemma 6, the wasted space incurred by each closed job set M of placement P is at most $\text{ROWS}(M) \cdot 3\sqrt{L}$. Thus, the total wasted space incurred by all closed job sets is at most:

$$\sum_M \text{ROWS}(M) \cdot 3\sqrt{L} \leqslant \text{CLOSED}(P) \cdot 3\sqrt{L}$$

where the summation is taken over all closed sets and $\text{CLOSED}(P)$ denotes the number of rows dedicated to closed sets.

By Lemma 7, the wasted space incurred by the open set M is less than $L + 2\sqrt{L} \cdot (\text{ROWS}(M) - 1)$. Hence, the total wasted space incurred by open job sets is less than:

$$\sum_{j=1}^{\log H} [L + 2 \cdot \sqrt{L} \cdot (\text{ROWS}(M^j) - 1)] < L \cdot \log H + \text{OPEN}(P) \cdot 2\sqrt{L}$$

where M^j is the open set of type j according to Definition 2 and $\text{OPEN}(P)$ denotes that number of rows dedicated to open sets.

Since $\text{ROWS}(P) = \text{OPEN}(P) + \text{CLOSED}(P)$, we get that the wasted space in P is

$$A_w(P) < L \cdot \log H + \text{OPEN}(P) \cdot 3\sqrt{L} + \text{CLOSED}(P) \cdot 3\sqrt{L} < L \cdot \log H + 3\sqrt{L} \cdot \text{ROWS}(P)$$

and the lemma follows. □

We now turn to bound the total wasted area.

Theorem 5. *Let $L \geqslant 9$. Then, the wasted space of the solution computed by Algorithm **Placement** is less than $L \log H + 3H\sqrt{L} + \beta \cdot S$.*

Proof. Consider the infeasible placement Q on $J_1, \ldots J_{t+1}$. Clearly, $\text{ROWS}(Q) > H$. By lemma 8, the space wasted by Q satisfies:

$$A_w(Q) = L \cdot \text{ROWS}(Q) - \sum_{j=1}^{t+1} r_j < L \cdot \log H + 3\sqrt{L} \cdot \text{ROWS}(Q)$$

Hence, the total matrix space wasted by P satisfies:

$$\begin{aligned}
A_w(P) &= L \cdot H - \textstyle\sum_{j=1}^{t} r_j \\
&= L \cdot \textsc{rows}(Q) - L \cdot (\textsc{rows}(Q) - H) - \textstyle\sum_{j=1}^{t+1} r_j + r_{t+1} \\
&< L \log H + 3\sqrt{L} \cdot \textsc{rows}(Q) - L \cdot (\textsc{rows}(Q) - H) + r_{t+1} \\
&= L \log H + 3H\sqrt{L} + 3\sqrt{L} \cdot (\textsc{rows}(Q) - H) - L \cdot (\textsc{rows}(Q) - H) + r_{t+1} \\
&= L \log H + 3H\sqrt{L} + (\textsc{rows}(Q) - H) \cdot (3\sqrt{L} - L) + r_{t+1}
\end{aligned}$$

If $L \geqslant 9$ it follows that $A_w(P) < L \log H + 3H\sqrt{L} + \beta \cdot S$. □

Corollary 6. *Let $L \geqslant 9$. Then, the approximation ratio of Algorithm **Placement** is*

$$1 - \frac{\log H}{H} - \frac{3}{\sqrt{L}} - \beta.$$

Proof. Let OPT be the area of the matrix that is covered by an optimal placement. Clearly, $\textsc{opt} \leqslant S$. Let ALG be the area covered by the jobs placed by the algorithm. Then, by Theorem 5 it follows that

$$\frac{\textsc{alg}}{\textsc{opt}} \geqslant \frac{S - A_w(P)}{S} > \frac{S - L\log H - 3H\sqrt{L} - \beta \cdot S}{S} \geqslant 1 - \frac{\log H}{H} - \frac{3}{\sqrt{L}} - \beta$$

as required. □

Notice that since $L \geqslant H$ we can solve the problem exactly for small L's using exhaustive search. Hence, the approximation ratio is in fact $1 - \frac{\log H}{H} - \beta - \varepsilon$, for any $\varepsilon > 0$.

In order to further improve the approximation ratio we may use the powers of some number b instead of powers of 2 in Definition 2. In this case, if M is job set of medium or large jobs, then $|M| \leqslant b\sqrt{L}$. It follows that, for $L \geqslant (b+1)^2$, the approximation ratio of Algorithm **Placement** is $1 - \frac{\log_b H}{H} - \frac{b+1}{\sqrt{L}} - \beta$. Hence, for every $\varepsilon > 0$ there exist b and L_0 such that $1 - \frac{\log_b H}{H} - \frac{b+1}{\sqrt{L}} - \beta \geqslant 1 - \varepsilon - \beta$ for every $L \geqslant L_0$. This brings us to the following result.

Theorem 7. *For every constant $\varepsilon > 0$, there exists a linear time $(1 - \varepsilon - \beta)$-approximation algorithm for* SRP.

Finally, throughout this section we assumed that $\log H$ is integral. If this is not the case, we modify Definition 2. We define $\lfloor \log H \rfloor - 2$ medium jobs types that correspond to $i \in \{2, \ldots, \lfloor \log H \rfloor - 1\}$. There are now jobs that are not covered by any job range — the jobs whose sizes are between $b^{\lfloor \log H \rfloor - 1}\sqrt{L}$ and $\frac{H}{2}\sqrt{L}$. We add these jobs to the largest medium job type. This ensures that the number of ranges is $\log H$, but increases the waste within closed sets. It is not hard to verify that the resulting approximation ratio still remains $1 - \varepsilon - \beta$, for any $\varepsilon > 0$.

4. Conclusion

In this paper, we studied the sequential rectangle placement problem which is the problem of scheduling transmissions on the downlink of IEEE 802.16/WiMAX systems that use the OFDMA technology.

We showed that the sequential rectangle placement problem is very hard to approximate in the case of general weights, even when the transmissions are relatively small, and in the case of proportional weights. Moreover, it was shown that the problem is NP-hard in the special case of proportional weights and relatively small transmissions. We presented a constant factor approximation algorithm for this special case. We also showed that the problem is very hard to approximate in the case of unit weights, and that the problem remains NP-hard in the special case of unit weights and relatively small transmissions. Obtaining an approximation algorithm for this special case is an open problem.

Finally, all our negative results rely on the hardness of the weakly NP-hard PARTITION. It follows that the complexity of the special case in which the dimensions of the matrix are polynomial in the number of jobs remains unresolved.

Acknowledgment

We thank Yaron Alpert from Alvarion Ltd. for introducing the problem to us.

References

[1] M. Allman, V. Paxson, W. Stevens, TCP congestion control, RFC, 2581 (1999.)
[2] R. Cohen, L. Katzir, Computational analysis and efficient algorithms for micro and macro ofdma scheduling, in: 27th IEEE International Conference on Computer Communications, 2008.

[3] D. Ferrari, D.C. Verma, A scheme for real-time channel establishment in wide-area networks, IEEE Journal on Selected Areas in Communications 8 (3) (1990) 368–379.
[4] A.V. Fishkin, O. Gerber, K. Jansen, R. Solis-Oba, Packing weighted rectangles into a square, in: 30th International Symposium on Mathematical Foundations of Computer Science 2005, vol. 3618 of LNCS, 2005.
[5] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[6] IEEE Standard for Local and Metropolitan Area Networks, IEEE 802.16e, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Sytems, 2005.
[7] A. Israeli, D. Rawitz, O. Sharon, On the complexity of sequential rectangle placement in IEEE 802.16/WiMAX systems, in: 15th Annual European Symposium on Algorithms, vol. 4698 of LNCS, 2007.
[8] K. Jansen, L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, in: 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999.
[9] K. Jansen, G. Zhang, Maximizing the number of packed rectangles, in: 9th Scandinavian Workshop on Algorithm Theory, vol. 3111 of LNCS, 2004.
[10] K. Jansen, G. Zhang, On rectangle packing: Maximizing benefits, in: 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004.
[11] G. Mounié, C. Rapine, D. Trystram, Efficient approximation algorithms for scheduling malleable tasks, in: 11th Annual ACM Symposium on Parallel Algorithms and Architectures, 1999.
[12] P. Nicopolitidis, M.S. Obaidat, G.I. Papadimitriou, A.S. Pomportsis, Wireless Networks, John Wiley & Sons Ltd., 2003.
[13] A. Nordboten, LMDS systems and their application, IEEE Communications Magazine 38 (6) (2000) 150–154.
[14] J. Turek, J.L. Wolf, P.S. Yu, Approximate algorithms scheduling parallelizable tasks, in: 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992.