Matrix Columns Allocation Problems

Amos Beimel[1]

Department of Computer Science, Ben-Gurion University, Be'er Sheva, Israel

Boaz Ben-Moshe

College of Judea & Samaria, Ariel, 44837, Israel

Yehuda Ben-Shimol

Department of Communication Systems Engineering, Ben-Gurion University, Be'er Sheva, Israel

Paz Carmi

School of Computer Science, Carleton University, Ottawa, Canada

Eldad Chai

Department of Communication Systems Engineering, Ben-Gurion University, Be'er Sheva, Israel

Itzik Kitroser

Department of Communication Systems Engineering, Ben-Gurion University, Be'er Sheva, Israel

Eran Omri[2]

Department of Computer Science, Ben-Gurion University, Be'er Sheva, Israel

---

Abstract

Orthogonal Frequency Division Multiple Access (OFDMA) transmission technique is gaining popularity as a preferred technique in the emerging broadband wireless access standards. Motivated by the OFDMA transmission technique we define the following problem: Let M be a matrix (over $\mathbb{R}$) of size $a \times b$. Given a vector of non-negative integers $\vec{C} = \langle c_1, c_2, \ldots, c_b \rangle$ such that $\sum c_j = a$, we would like to allocate a cells in M such that (i) in each row of M there is a single allocation, and (ii) for each element $c_i \in \vec{C}$ there is a unique column in M which contains exactly c_i allocations. Our goal is to find an allocation with minimal value, that is, the sum of all the a cells of M which were allocated is minimal. The nature of the suggested new problem is investigated in this paper. Efficient algorithms are suggested for some interesting cases. For other cases of the problem, NP-hardness proofs are given followed by inapproximability results.

Key words: Allocation problems, NP-completeness, Inapproximability

1 Introduction

The growing use of communication networks and the emerging techniques constantly developed for them require solving many new algorithmic problems. Such problems are often raised by mapping and allocating optimization tasks [4, 7, 8, 10]. In particular, the Orthogonal Frequency Division Multiple Access (OFDMA) transmission technique introduces such new allocation problems. In this work we introduce the Matrix Column Allocation Problem ($\mathcal{MCAP}$) which models one such problem and present theoretical results concerning it: we construct an efficient algorithm for one version of the problem and prove NP-hardness results for the other variants of this problem.

1.1 Background and Motivation

The OFDMA modulation technique is gaining popularity as a preferred technique in the emerging broadband wireless access standards. The IEEE802.16 standard [1] with its mobility extension IEEE802.16e-2005 [2] (also known as *WiMax*) is based on OFDMA and is considered as a candidate for the next generation broad wireless access systems.

In a general digital communication systems, the information is in the form of bits, or collections of bits called symbols, that are modulated onto the carrier. The time duration of a symbol depends on the used bandwidth. As higher bandwidths (data rates) are used, the duration of one bit or symbol of information becomes smaller. In an OFDMA modulation, the channel bandwidth is broken into a large number of closely and equally spaced orthogonal low rate sub-carriers, hence an OFDMA symbol duration depends directly on the

Email addresses: `beimel@cs.bgu.ac.il` (Amos Beimel), `benmo@yosh.ac.il` (Boaz Ben-Moshe), `benshimo@bgu.ac.il` (Yehuda Ben-Shimol), `carmip@cs.bgu.ac.il` (Paz Carmi), `kitroser@bgu.ac.il` (Itzik Kitroser), `omrier@cs.bgu.ac.il` (Eran Omri).

[1] Part of this work was done while the author was on sabbatical at the University of California, Davis, partially supported by the David and Lucile Packard Foundation.

[2] Research is partially supported by the Frankel Center for Computer Science.

number of used sub-carriers. In addition, the sub-carriers are divided into subsets of sub-carriers where each subset represents a *sub-channel* which is the minimum transmission unit in an OFDMA symbol. The allocation unit in such systems is a combination of OFDMA time symbol and a sub-channel and is called a *slot*.

We use the model for uplink resource allocations given in [1, 2], which is compliant with the OFDMA slot definition. In this model slots are indexed starting at the lowest numbered sub-channel and the first OFDMA symbol up to the last. When the last OFDMA symbol is reached, the indexing continues from the next sub-channel and the first OFDMA symbol. At each allocation period (time frame), the clients are allocated sets of slots which are usually successive, but may also be scattered. The specific number of required slots per client depends on its transmission capability and the transmission rate on those slots is set to meet the client's needs and capabilities (i.e., packet size and available rate). In addition, scattered allocations may imply additional cost (in term of system resources) to describe such allocations. The allocation slots and numbering scheme in OFDMA systems are illustrated in Figure 1.

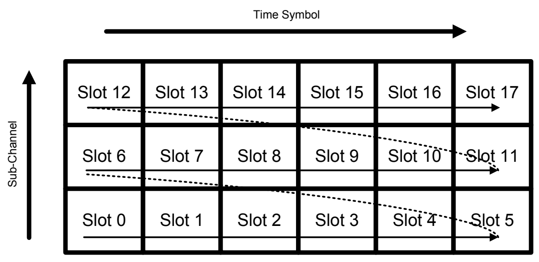


Figure 1. The slot numbering process.

In OFDMA based wireless systems, mapping is the process of allocating resources (slots) to clients according to their demands and is performed after the scheduling process.[3] The slots are represented by an indexed list from which clients are allocated sets of resources. Most mapping models do not allow mapping optimizations, thus the mapping process generates a large amount of mapping overhead (since the mapping description must be sent to the clients); this overhead reduces the system performance. In this paper we consider a mapping model that can be used to reduce the mapping overhead. In this model each resource is assigned a weight for every client. Each weight represents both description cost and mapping overhead that may be generated if the resource is allocated to a certain client. For example, in [3] the authors presented a method of persistent allocation of slots to clients. In their system, the clients are allocated specific slots in a certain time frame without a need for description in subsequent time frames if no change is required. Each change in allocated slots or reallocation due to other clients' needs would result in system overhead for description. The reallocation of slots may create high overhead since it may require reallocation to other clients as well. Hence, weights may vary for a client even for consecutive slots.

[3] Further discussion on the mapping process can be found in [3].

We define an allocation problem on a matrix to represent the suggested mapping model and the resource allocation process in IEEE802.16. First, we define a matrix M interpreting the OFDMA resource list where there is a row for each slot available for allocation and there is a column for each active client. The cell $M_{i,j}$ is the overhead if slot i is allocated to client j. We also define a vector $\vec{C}$ describing the slot requirements of the clients, where c_j – the jth coordinate of $\vec{C}$ – denotes the number of slots required by client j for transmission. Given the matrix M and the vector $\vec{C}$, we want an allocation of slots to the clients that minimizes the total overhead. There are two requirements: (1) we should allocate each slot to exactly one client, and (2) we should allocate c_j slots for the jth client. We consider two versions of the problem; in one version the slots allocated to a client should be consecutive and in the other version the slots need not be consecutive.

1.2 Problem Formulation

Motivated by the above discussion, we next formally define various versions of the matrix columns allocation problem. In addition to distinguishing between allocations where allocated cells are consecutive or not, we also define versions where the solution can apply a permutation to the vector $\vec{C}$. The reason for allowing the permutation is two-fold. First, we believe that this is a natural generalization of the original problem and it is interesting to see if this generalization makes the problem harder or easier. Second, we use the permutation version to present the ideas of the more complicated reduction needed for the version without the permutation.

Definition 1.1 (The Matrix Columns Allocation Problem – $\mathcal{MCAP}$) *Let M be a matrix (over $\mathbb{R}$) of size $a \times b$ and $\vec{C} = \langle c_1, c_2, \ldots, c_b\rangle$ be a vector of non-negative integers, called the demand vector, such that $\sum c_j = a$. A solution to the $\mathcal{MCAP}$ is an allocation $S = \{\langle \alpha_i, \beta_i\rangle : 1 \le i \le a\}$ of a cells in M (i.e., $\alpha_i \in \{1, \ldots, a\}$ and $\beta_i \in \{1, \ldots, b\}$) such that:*

Row constraint. *There is exactly one cell in S from each row, and*
Column constraint. *There is a permutation $P : \{1, \ldots, b\} \to \{1, \ldots, b\}$ of the columns of M such that in S there are exactly $c_{P(i)}$ cells from the ith column of M for every column $i \in \{1, \ldots, b\}$.*

The value of an allocation S is the sum of the values of all the cells of M which were allocated. Our goal is to find an allocation whose value is minimal.

The generic $\mathcal{MCAP}$ is considered with the following possible requirements:

Consecutive allocation (blocks). *Here we require that, for each column, the $c_{P(i)}$ cells are consecutive (block constraint).*
Fixed permutation. *Here we require that P is the identity permutation,*

(i)

No Block Any Permutation			
	c_1	c_2	c_3
r_1	**0**	5	1
r_2	4	**0**	1
r_3	10	**0**	1
r_4	50	25	**0**
r_5	1	**0**	3
r_6	1	50	**0**
r_7	8	**0**	50
r_8	**0**	10	1
r_9	10	**0**	50
r_{10}	1	4	**0**

Solution value: 0

Permutation: $< 2, 5, 3 >$

(ii)

No Block Fixed Permutation			
	c_1	c_2	c_3
r_1	**0**	5	1
r_2	4	**0**	1
r_3	10	0	**1**
r_4	50	25	**0**
r_5	**1**	0	3
r_6	**1**	50	0
r_7	8	**0**	50
r_8	**0**	10	1
r_9	10	**0**	50
r_{10}	**1**	4	0

Solution value: 4

(iii)

Block Any Permutation			
	c_1	c_2	c_3
r_1	0	5	**1**
r_2	4	0	**1**
r_3	10	0	**1**
r_4	50	25	**0**
r_5	1	0	**3**
r_6	**1**	50	0
r_7	**8**	0	50
r_8	**0**	10	1
r_9	10	**0**	50
r_{10}	1	**4**	0

Solution value: 19

Permutation: $< 3, 2, 5 >$

(iv)

Block Fixed Permutation			
	c_1	c_2	c_3
r_1	0	**5**	1
r_2	4	**0**	1
r_3	10	**0**	1
r_4	50	25	**0**
r_5	1	0	**3**
r_6	**1**	50	0
r_7	**8**	0	50
r_8	**0**	10	1
r_9	**10**	0	50
r_{10}	**1**	4	0

Solution value: 28

Figure 2. Example instances of $\mathcal{MCAP}$. The above tables represent the appropriate solutions of all four $\mathcal{MCAP}$ cases for the same 3×10 matrix with the same demand vector $\vec{C} =< 5, 3, 2 >$.

that is, S should contain exactly c_j cells from the jth column. In this case, we say that there is a fixed permutation.

If we remove the first requirement, then we simply allow any $c_{P(i)}$ cells from the ith column. Similarly, If we remove the second requirement, we simply allow any permutation P. Considering the problem with or without these two requirements, four cases of the problem are defined.

Example 1.1 In Figure 2 we describe an $\mathcal{MCAP}$ instance (a matrix and a demand vector), and show optimal allocations for it for the four variants of $\mathcal{MCAP}$. The reader can easily verify that cells allocated (appearing in **Bold-Face**) in each table indeed form a legal solution for the appropriate case.

We next briefly explain why each solution is also optimal. For case (i) (*no block, any permutation*), this is clear as we cannot expect better than a zero valued solution. For case (ii) (*no block, fixed permutation*), it can be easily verified that our solution is optimal (although not a unique optimal solution). For cases (iii) (*block, any permutation*) and (iv) (*block, fixed permutation*), note that an allocation of the block of size 5 in rows $r_5 - r_9$ (respectively, $r_2 - r_6$) is not possible, as there will be no way to allocate cells in row r_{10}

(respectively, row r_1). Next observe that any possible allocation of the block of size 5, anywhere other than the choices of (iii) and (iv), will result in a solution of value at least 30. Using this observation, verifying the optimality of our solutions is easy.

It is quite straightforward that the value of an optimal allocation for the *no block, any permutation* version for a given instance of $\mathcal{MCAP}$ is at most the value of an optimal allocation for any of the other three $\mathcal{MCAP}$ versions. Conversely, the value of an optimal allocation for the *block, fixed permutation* version for a given instance of $\mathcal{MCAP}$ is at least the value of an optimal allocation for any of the other three $\mathcal{MCAP}$ versions. However, for a given instance the value of an optimal allocation for the *no block, fixed permutation* instance can be less, equal, or more than the value of an optimal allocation for the *block, any permutation* version.

Our main result in this paper is a proof that three versions of $\mathcal{MCAP}$ – *block, fixed permutation* $\mathcal{MCAP}$, *no block, any permutation* $\mathcal{MCAP}$, and *block, any permutation* $\mathcal{MCAP}$ – are NP-hard, and, in fact, they cannot be efficiently approximated within any factor unless P=NP. To prove the NP-completeness we construct two reductions from the vertex cover problem, which was proved to be NP-complete by Karp [5]. The generalization of the hardness results to inapproximability results is quite simple and does not use the PCP theorem. The proofs are similar to the proof that the traveling sales person (TSP) problem (without the triangular inequality) cannot be approximated [9].

To complement our results, we present efficient exact algorithms for one version and a few restrictions of the $\mathcal{MCAP}$. First, we construct a polynomial-time algorithm for the *no block, fixed permutation* $\mathcal{MCAP}$ using the algorithm for minimum perfect matching [6]. We then show, using dynamic programming, that the *block, fixed permutation* $\mathcal{MCAP}$ and the *block, any permutation* $\mathcal{MCAP}$ have polynomial-time algorithms if the number of columns in the matrix is at most logarithmic in the number of rows.

The positive results presented in this paper (i.e., the polynomial-time algorithms) can be used for improving performance in OFDMA systems implementing the suggested mapping model where clients can be allocated any set of resources. Furthermore, the negative results presented in this paper suggest that optimizing the mapping process in OFDMA systems, in particular minimizing the mapping overhead, is infeasible. Therefore, if such optimization is desired, then a heuristic approach should be considered.

Related Work. The problems that are considered in this work are a generalization of simple assignment problems [6], where we want an allocation with exactly one cell from each row and exactly one cell from each column (that is, find a minimum weighted perfect matching in a bipartite graph). Indeed, our polynomial-time solution for the *no block, fixed permutation* $\mathcal{MCAP}$ is by a reduction to this problem. Generalizations of the simple assignment problem (different than the ones considered in this paper) appear in the literature (e.g., [11]).

Organization. Section 2 shows that the *no block, fixed permutation* $\mathcal{MCAP}$ has a polynomial-time algorithm, followed with a set of special cases for which the other versions of $\mathcal{MCAP}$ also have polynomial-time algorithm. Section 3 contains the main part of this paper; NP hardness proofs for versions (#2−#4) of the problem are given, followed with inapproximability results.

2 Positive results

We next prove that *no block, fixed permutation* $\mathcal{MCAP}$ has a polynomial-time algorithm. We then show that the *block, fixed permutation* $\mathcal{MCAP}$ and the *block, any permutation* $\mathcal{MCAP}$ have a polynomial-time algorithm if the number of columns in the matrix is logarithmic in the number of rows.

Lemma 2.1 *The* no block, fixed permutation $\mathcal{MCAP}$ *has a polynomial-time algorithm.*

PROOF. The proof is by reduction to the minimum weight perfect matching problem. Given an instance $(M, \vec{C})$ of $\mathcal{MCAP}$ (where M has a rows and b columns), we first construct a new matrix M' of size $a \times a$ where every column i in M is duplicated c_i times in M', and a new demand vector $\vec{C'} := \vec{1}$ (i.e., $c'_i = 1$ for all $1 \leq i \leq a$). Notice that the number of columns in M' is $\sum_{i=1}^{b} c_i$, which is the number of rows in M. Thus, constructing $(M', \vec{C'})$ can be done in polynomial time. Now, a solution S' for the instance $(M', \vec{C'})$ straightforwardly implies a solution S for the original instance $(M, \vec{C})$ with the same value, where for every column i in M, we select all cells which are taken from the c_i copies of that column in M'. Similarly, a solution S for $(M, \vec{C})$ implies a solution S' for $(M, \vec{C'})$ with the same value.

Observe that such a solution for $(M', \vec{C'})$ takes exactly one cell in each row and one cell in each column, that is, it is a matching between the rows and the columns. Thus, to find a minimum solution to $(M, \vec{C})$, we need to solve the minimum weighted perfect matching problem on the bipartite graph described by the matrix M'. The latter task can be solved in polynomial-time [6]. □

We next show that when the number of columns is small we can solve the two versions of the block constraint $\mathcal{MCAP}$ in polynomial time. In contrast, we will prove that without this restriction the problem is NP-hard.

Lemma 2.2 *The* block, fixed permutation $\mathcal{MCAP}$ *and the* block, any permutation $\mathcal{MCAP}$ *can be solved in time $a \cdot 2^{O(b)}$, where a is the number of rows in the input matrix, and b is the number of columns in the input matrix.*

PROOF. We first present an algorithm for *block, fixed permutation $\mathcal{MCAP}$*. Later, we show how this algorithm may be modified to solve the *block, any permutation $\mathcal{MCAP}$*. The idea of the algorithm is to apply a divide and conquer technique. Assume we have an instance $(M, \vec{C})$ and a "guess" $\vec{C_1}, \vec{C_2}$, where $\vec{C_1}$ contains half of the coordinates of $\vec{C}$ and $\vec{C_2}$ contains the other half. This guess states that all the coordinates of $\vec{C_1}$ should be allocated higher in M than all coordinates of $\vec{C_2}$. We then obtain two independent instances of the problem. We solve each instance recursively, and combine the minimum solutions of these instances into a minimum solution for the original instance. As we do not know how to make such guesses of $\vec{C_1}, \vec{C_2}$, we check all possible partitions of $\vec{C}$ and take the minimum solution over all partitions.

Formally, given an instance $(M, \vec{C})$, the algorithm goes over all possible partition sets $I \subset \{1, \ldots, b\}$ of size $b/2$. For each such I, the algorithm constructs two independent instances $(M_1^I, \vec{C_1^I})$ and $(M_2^I, \vec{C_2^I})$ as follows: First, denote $a_1 := \sum_{i \in I} c_i$. The matrix M_1^I contains the a_1 top rows of the columns indexed by I in M, and the matrix M_2^I contains the $a - a_1$ bottom rows of the columns not indexed by I in M. Finally, define $\vec{C_1^I} := \langle c_i \rangle_{i \in I}$ and $\vec{C_2^I} := \langle c_i \rangle_{i \notin I}$. Notice that every pair of solutions S_1^I and S_2^I, for $(M_1^I, \vec{C_1^I})$ and $(M_2^I, \vec{C_2^I})$ respectively, implies a solution S^I to $(M, \vec{C})$ whose value is the sum of the values of the solutions S_1^I and S_2^I. The algorithm, therefore, sets the solution S for $(M, \vec{C})$ to be $\min(S^I)$. Furthermore, any solution S' to $(M, \vec{C})$, defines a partition set I: Take I to be the set of indices such that the consecutive blocks in the columns in I are the $b/2$ top blocks S' allocates in M. Thus S is optimal.

There are at most $\binom{b}{b/2} < 2^b$ ways of choosing a set $I \subset \{1, \ldots, b\}$ of size $b/2$. For each such I, the algorithm computes its minimal value recursively, and the total value is the sum of this two values. Denote the running time of this recursive algorithm on an instance with a rows and b columns by $T(a, b)$. Thus, $T(a, b) \leq 2^b \cdot 2 \cdot T(a, b/2)$ (since the algorithm constructs two instances, each with $b/2$ columns and less than a rows), and $T(a, 1) = O(a)$. Thus, solving this recursion yields $T(a, b) \leq O(a \prod_{i=1}^{\log b} 2^{1+b/2^i}) = O(a \cdot 2^{2b})$.

To solve the *block, any permutation $\mathcal{MCAP}$*, we can apply the above algorithm for every permutation, and taking the solution whose value is minimum over all permutations. We get an algorithm whose complexity is $O(a \cdot 2^{2b} \cdot b!)$. A more efficient algorithm with complexity $O(a \cdot 2^{2b})$ is obtained by modifying the above recursive algorithm. In the recursive step, in addition to the choice of the partition of the columns to the top and bottom matrices, we also need to partition the demand vector. Formally, to define a partition of an instance

$(M, \vec{C})$ into two instances, we choose *two* sets $I, J \subset \{1, \ldots, b\}$ of size $b/2$ each. We define the two instances $(M_1^I, \vec{C_1^J})$ and $(M_2^I, \vec{C_2^J})$, where the matrices M_1^I and M_2^I are as above and $\vec{C_1^J} := \langle c_i \rangle_{i \in J}$ and $\vec{C_2^J} := \langle c_i \rangle_{i \notin J}$. A similar analysis shows that the resulting algorithm is correct and that its running time is $a \cdot 2^{O(b)}$. □

3 Hardness and Inapproximability Results

To prove the hardness (and inapproximability) of the three remaining versions of $\mathcal{MCAP}$, we present two related reductions from the Vertex-Cover problem, one of the original problems proved to be NP-complete by Karp [5]. The first reduction proves that the *no block, any permutation* $\mathcal{MCAP}$ is NP-Hard. The second reduction uses similar ideas, however with a more complicated construction, and proves that the *block, fixed permutation* $\mathcal{MCAP}$ and the *block, any permutation* $\mathcal{MCAP}$ are NP-hard.

3.1 Hardness of the no block, any permutation $\mathcal{MCAP}$

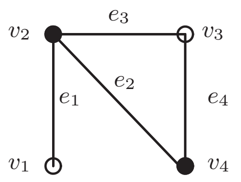
Lemma 3.1 *The* no block, any permutation $\mathcal{MCAP}$ *is NP-Hard.*

PROOF. Given an instance G, k of the vertex cover problem, we construct an instance $(M, \vec{C})$ of the *no block, any permutation* $\mathcal{MCAP}$ such that G has a vertex cover of size k iff $(M, \vec{C})$ has an allocation whose value is zero.

We start by describing a construction that does not have this property and then we "fix" it. Given a graph $G = \langle V, E \rangle$ with n vertices, we construct a matrix M over $\{0, 1\}$, in which each column represents a vertex in V and each row stands for an edge in E. For the row representing $e = (v, u)$, we assign the value 0 to the cells of the columns representing v and u, and 1 to all other cells. We observe that, by the row constraints,

Observation 3.2 *Every allocation whose value is 0 must choose in the row representing $e = (v, u)$ either the cell in the column representing v or the cell in the column representing u.*

Now, given an allocation whose value is 0, consider the set of vertices corresponding to columns having at least one cell in an allocation. By Observation 3.2, this set is a vertex cover of the graph G. Thus, the graph G has a vertex cover of size k iff there exists a demand vector $\vec{C}$ with k non-zero entries such that $(M, \vec{C})$ has an allocation whose value is 0.



	v_1	v_2	v_3	v_4
e_1	0	**0**	1	1
e_2	1	**0**	1	0
e_3	1	**0**	0	1
e_4	1	1	0	**0**
r_5	0	**0**	0	0
r_6	0	0	0	**0**
r_7	0	0	0	**0**
r_8	0	0	0	**0**

Figure 3. A graph with a vertex cover of size 2 and the corresponding instance for the $\mathcal{MCAP}$, where the demand vector is $\langle 4, 4, 0, 0 \rangle$. The vertex cover is composed of the two black vertices and the allocation contains all **bold-face** cells.

Trying to apply this approach we face two hurdles in constructing the vector $\vec{C}$. The first is that we must allow two adjacent vertices to be chosen in the vertex cover, and therefore, allow two cells to be taken from the same row. The second hurdle is that we only know that $\vec{C}$ should contain k non-zero entries, but we do not know their values as each coordinate in $\vec{C}$ should correspond to the number of edges covered by each vertex in the cover. Hence, we have no good way to define the number of edges that are covered by a given vertex, without knowing the solution.

We use a rather simple approach to overcome both hurdles. We add padding rows, in which all cells have zero values, and then set the demand vector $\vec{C}$ to be $n^k \cdot 0^{n-k}$, where n is the number of vertices in the graph G. As the number of rows in M should be $\sum c_i = nk$, the number of padding rows that we add is $nk - |E| \geq 0$.[4]

Before proving that a solution whose value is zero exists iff there is a vertex cover for G of size k, we first describe the reduction formally. Given a graph $G = \langle V, E \rangle$ with n vertices and a natural number k, we construct a matrix M of size $nk \times n$. We tag each column $1 \leq i \leq n$ by the vertex v_i. Let $\left\{e_1, e_2, \ldots, e_{|E|}\right\}$ be some ordering on the edges of G. We tag each row $1 \leq i \leq |E|$ by e_i and we tag each row $|E| + 1 \leq i \leq kn$ by r_i. In a row tagged by the edge $e = (v_i, v_j)$, the cells in the columns tagged by v_i and v_j are set to 0, and all other cells in this row are set to 1. In a row tagged by r_i, all cells are set to 0. Finally, we set the demand vector $\vec{C} := \langle c_1, c_2, \ldots, c_n \rangle$, where $c_i = n$ for $1 \leq i \leq k$ and $c_i = 0$ otherwise. For example, a graph and the corresponding instance of the $\mathcal{MCAP}$ are described in Figure 3. Clearly, M and $\vec{C}$ can be

[4] If $nk < |E|$, then no vertex cover of size k exists and the reduction outputs a fixed instance of the $\mathcal{MCAP}$ whose minimal allocation has value greater than 0.

constructed in polynomial time from G and k. We next argue the validity of the reduction, that is, $(M, \vec{C})$ has an allocation whose value is zero iff G has a vertex cover of size k.

First, assume there exists a zero valued solution S for the allocation problem $(M, \vec{C})$. Let P be the permutation implied by S and let $H := \{v_i \in V : 1 \leq P(i) \leq k\}$ (that is, H is the set of the k vertices tagging columns which have n cells in the allocation S). We next argue that H is a vertex cover of size k. Clearly, $|H| = k$ since P is a permutation. Now, for every edge $e_j = (u, v)$ in E, by the row constraints, one cell in the row tagged e_j is in S. Since the value assigned to all cells in S is 0 and since the only two 0 valued cells in the row tagged e_j appear in the column tagged u and in the column tagged v, one of these two cells is in S. Since cells in S are only taken from columns v_i such that $1 \leq P(i) \leq k$, it holds that at least one of u and v is in H.

Second, assume $H \subseteq V$ is a vertex cover for G, where $|H| = k$. We claim that $(M, \vec{C})$ has a solution S whose value is 0. This solution takes cells only in the k columns which are tagged by vertices from H. For each edge $e = (u, v)$ such that exactly one of u and v is in H, the cell in the row tagged by e and in the column tagged by this vertex is in the allocation S. For each edge $e = (u, v)$ such that both vertices u and v are in H, we arbitrarily decide to select the zero valued cell in the column tagged by the smallest vertex between u and v. To guarantee that exactly n cells are selected from each column tagged by a vertex from H, we augment the above cells by cells from padding rows such that exactly one cell is selected from each padding row. Since there are $kn - |E|$ padding rows, such augmentation is possible. □

3.2 Hardness of Block Constraint $\mathcal{MCAP}$

In this section we prove that the two versions of the block constraint $\mathcal{MCAP}$ (with or without the permutation constraint) are NP-hard. We prove the NP-hardness of the two $\mathcal{MCAP}$ version by presenting a reduction from vertex cover. For clarity, we first describe a first attempt of constructing the reduction, and explain why it fails. We then show a second attempt fixing the shortcoming of the first attempt.

First Attempt. In the first attempt, we slightly change our approach in overcoming the first hurdle described earlier (that is, the problem of choosing two end-points of an edge). This new approach would enable us to kill two birds with one stone – overcoming the first hurdle and enabling the selection of cells from each column to be in one block. Instead of adding padding rows (as in the proof of Lemma 3.1), we represent each edge $e_i = (v, u)$ by two directed edges $e_{v,u} = (v, u)$ and $e_{u,v} = (u, v)$ and assign a row to each of them. Furthermore, we add a column representing e_i. The row representing $e_{v,u}$ has

		v_1	v_2	v_3	v_4	e_1	e_2	e_3	e_4	Padd$_1$	Padd$_2$	Padd$_3$	Padd$_4$	Padd$_5$	Padd$_6$
	e_{v_1,v_2}	0	1	1	1	**0**	1	1	1	1	1	1	1	1	1
v_1		0	1	1	1	1	1	1	1	**0**	0	0	0	0	0
		0	1	1	1	1	1	1	1	0	**0**	0	0	0	0
	e_{v_2,v_1}	1	**0**	1	1	0	1	1	1	1	1	1	1	1	1
v_2	e_{v_2,v_3}	1	**0**	1	1	1	1	0	1	1	1	1	1	1	1
	e_{v_2,v_4}	1	**0**	1	1	1	0	1	1	1	1	1	1	1	1
	e_{v_3,v_2}	1	1	0	1	1	1	**0**	1	1	1	1	1	1	1
v_3	e_{v_3,v_4}	1	1	0	1	1	1	1	**0**	1	1	1	1	1	1
		1	1	0	1	1	1	1	1	0	0	**0**	0	0	0
	e_{v_4,v_2}	1	1	1	**0**	1	0	1	1	1	1	1	1	1	0
v_4	e_{v_4,v_3}	1	1	1	**0**	1	1	1	0	1	1	1	1	1	1
		1	1	1	**0**	1	1	1	1	0	0	0	0	0	0

Figure 4. The matrix M' that the reduction of the first attempt outputs for the graph described in Figure 3 with $k = 2$. The demand vector $\vec{C}$ is $\left\langle 3, 3, 1^6, 0^6 \right\rangle$. The cells in **bold-face** are an allocation corresponding to the vertex cover $\{v_2, v_4\}$.

a 0 value only in the column representing v and in the column representing e_i. The row representing $e_{u,v}$ has a 0 value only in the column representing u and in the column representing e_i. Observe that any selection of vertices which cover at least one of $e_{v,u}$ and $e_{u,v}$, for every such e_i, is a vertex cover for G.

Now, any two columns representing two adjacent vertices may both be chosen with all their zero cells as they do not collide. Moreover, we may reorder the rows of the matrix so that all rows corresponding to the same column form a single consequent segment in M'. While this helps us overcome the first hurdle, it presumably sets the bar for the second one a little higher. It seems we must be able to predict the number of vertices in the cover that touch each edge as well as the degree of each vertex. In fact, this is not true, as we may apply the same method as we did in the first reduction, namely add padding rows to the matrix and demand d cells from each column representing a vertex, where $d > 1$ is at least the maximum degree of any vertex in G. More specifically, we add $d - \text{degree}(v)$ rows in the bottom of the block of rows corresponding to the vertex v, each such row having zero value cells only in the column representing v. We finally add "padding" columns to enable $d(n-k)$ more cells to be allocated, that is, to ensure that for every padding row we can choose one cell. These "padding" columns have zero cells only in the padding rows. The demand vector $\vec{C'}$ is chosen to be $d^k \cdot 1^{d\cdot(n-k)} \cdot 0^{|V|-k}$. An example of the reduction is given in Figure 4.

Thus far, we described the first attempt of the reduction. The resulting matrix and vector $(M', \vec{C'})$ of this first attempt almost form a reduction to the *block, any permutation* $\mathcal{MCAP}$. First, it can be checked that if G has a cover of size k, then $(M', \vec{C'})$ has a solution whose value is 0. We would like to claim that if the input $(M', \vec{C'})$ has a solution whose value is 0, then G has a vertex cover of size k. In an allocation whose value is zero, there are exactly k columns with a block of d cells in the assignment; these columns must be labeled by vertices. We would like to argue that the set of vertices labeling columns with a block of d cells in the assignment is a cover. This is true if no column labeled by a vertex is assigned a single cell (as opposed to d or 0 cells) in the solution. However, if this is not the case, there might not be a cover of size k in G and the reduction is not valid. To fix this reduction we need to ensure that in each column labeled by a vertex, either 0 or d cells are allocated. Furthermore, we want the reduction to also work for the $\mathcal{MCAP}$ version where the permutation is fixed.

3.2.1 The Correct Reduction

Lemma 3.3 *The* block, any permutation $\mathcal{MCAP}$ *and the* block, fixed permutation $\mathcal{MCAP}$ *are NP-Hard.*

PROOF. We present a reduction from vertex cover that outputs the pair $(M, \vec{C})$. The matrix M is a column-wise compression of the matrix M' constructed in the first attempt. In M', every vertex $v \in V$ is represented by a column in M' and by a segment of rows. All cells within the column of v which do not appear within the segment of rows representing v have a non-zero value and, thus, can never be allocated in a 0 valued solution. We compress these n columns into a single column that no longer represents a single vertex, but rather has a segment of cells (rows) representing each vertex in V. Thus, every vertex is now represented only by a segment of rows and no longer by a column. We also add a "separation" row between any 2 segments of different vertices; this separation row has non-zero entries (except for 1 column we add). That is, a compressed column has n segments; each segment represents a different vertex and it contains d zero entries followed by an entry whose value is 1.

In any zero-value allocation exactly n cells from one segment may be allocated in each compressed column. In this case we will take the vertex labeling this segment to the vertex cover. Since every such compressed column allows an allocation of any segment of rows representing any vertex in V, it suffices to have exactly k such columns. The padding columns of M' are compressed in the following manner. Each of the (first) $|E|$ columns which represents an edge $e = (v, u)$, namely, had only two zero valued cells, will now have zero valued cells in all the $d - \text{degree}(w)$ complementary rows of the segment of every vertex w different from v and u. The rest of the padding columns will stay the same as in M'. Finally, we set $\vec{C} := d^k \cdot 1^{d(n-k)+n}$.

We now describe the reduction formally and show that there is a vertex cover of size k for G iff there is an optimal solution of value 0 for $(M, \vec{C})$. The illustration of the reduction in Figure 5 can help understanding the reduction. Given a graph $G = \langle V, E \rangle$ with $|V| = n$ and $|E| = m$, and a number k, let $d_{\max}$ be the maximum degree of any vertex in V and set $d = \max\{d_{\max}, \lceil m/(n-k) \rceil\}$. Assume, w.l.o.g., that the degree of every vertex in V is at least 1. We define a matrix M with $(d+1)n$ rows and $(d+1)n - (d-1)k$ columns. Let $\{e_1, e_1, \ldots, e_m\}$ be some ordering on the edges of G.

For convenience, we tag rows and columns of M by names (instead of indices). Let us begin with the rows. We divide the rows of the matrix into n segments of $d+1$ rows each, where the ℓth segment contains rows corresponding to v_ℓ which are tagged as follows. Let d_ℓ be the degree of $v_\ell \in V$ and let $e_{i_1}, \ldots, e_{i_{d_\ell}}$ be the edges touching v_ℓ. Given the tth row of the ℓth segment, we tag it $r_{\ell, e_{i_t}}$ if $1 \le t \le d_\ell$, we tag it $r_{\ell,t}$ if $d_\ell + 1 \le t \le d$, and we tag it Sep_ℓ if $t = d+1$.

We next explain how we tag the columns. The first k columns are the "compressed" vertex columns. We tag each column $1 \le j \le k$ by h_j. The next m columns are the "edge" columns. We tag each column $k+1 \le j \le k+m$ by e_{j-k}. The following $d(n-k) - m$ columns are the "padding" columns. We tag each column $k+m+1 \le j \le k + d(n-k)$ by $\text{Padd}_{j-(k+m)}$. Finally, the last n columns are the "extra" columns which are used for allocation of cells in the separating rows. We tag each column $k + d(n-k) + 1 \le j \le k + d(n-k) + n$ by $\text{Extra}_{j-(k+d(n-k))}$. From here on we use the tags instead of indices, e.g., we write $M(\text{Sep}_i, e_j)$ instead of $M_{di,j+k}$.

We next describe the entries of M. All row segments are of the same form, thus we confine our description to the ith row segment (which represents $v_i \in V$). In the separation row Sep_i we set the cell of the column tagged Extra_i to 0 and we set all other cells to zero. We next describe the first d rows. The reader is encouraged to look at the example in Figure 5 while reading the construction.

- For $1 \le j \le k$ (the compressed vertex columns), we set all cells to 0 (namely, $M(r_{i,x}, h_j) := 0$ for all x).
- For an edge column tagged by $e = (v_i, v')$ or $e = (v', v_i)$, we set $M(r_{i,e}, e) := 0$. We set all other cells within this column to 1.
- For an edge column tagged by $e = (v', v'')$ where $v', v'' \ne v_i$, we set all padding rows (that is, rows tagged $r_{i,t}$) within the segment of v_i to 0 and all other (edge) rows to 1.
- We set cells of columns tagged by Padd_j for some j to 1 if the row they are in represents an edge and 0 otherwise.
- Finally, we set all entries within the last segment of columns (tagged Extra_j for some j) to 1.

If we examine the patterns of the columns of M, we find that the compressed columns are made of n blocks of d zeros, each followed by a single one value. On the other hand, there are no blocks of d consecutive zeros in any other type of column in M. Thus, the following observation can be made.

		h_1	h_2	e_1	e_2	e_3	e_4	Padd$_1$	Padd$_2$	Extra$_1$	Extra$_2$	Extra$_3$	Extra$_4$
	r_{1,e_1}	0	0	**0**	1	1	1	1	1	1	1	1	1
v_1	$r_{1,2}$	0	0	1	**0**	0	0	0	0	1	1	1	1
	$r_{1,3}$	0	0	1	0	0	0	**0**	0	1	1	1	1
	Sep$_1$	1	1	1	1	1	1	1	1	**0**	1	1	1
	r_{2,e_1}	**0**	0	0	1	1	1	1	1	1	1	1	1
v_2	r_{2,e_2}	**0**	0	1	0	1	1	1	1	1	1	1	1
	r_{2,e_3}	**0**	0	1	1	0	1	1	1	1	1	1	1
	Sep$_2$	1	1	1	1	1	1	1	1	1	**0**	1	1
	r_{3,e_3}	0	0	1	1	**0**	1	1	1	1	1	1	1
v_3	r_{3,e_4}	0	0	1	0	1	**0**	1	1	1	1	1	1
	$r_{3,3}$	0	0	0	0	1	1	0	**0**	1	1	1	1
	Sep$_3$	1	1	1	1	1	1	1	1	1	1	**0**	1
	r_{4,e_2}	0	**0**	1	0	1	1	1	1	1	1	1	1
v_4	r_{4,e_4}	0	**0**	1	1	1	0	1	1	1	1	1	1
	$r_{4,3}$	0	**0**	0	1	0	1	0	0	1	1	1	1
	Sep$_4$	1	1	1	1	1	1	1	1	1	1	1	**0**

Figure 5. The matrix that the reduction outputs for the graph described in Figure 3 with $k = 2$. The demand vector is $\vec{C} = \left\langle 3, 3, 1^{10} \right\rangle$. The cells in **bold-face** are an allocation corresponding to the vertex cover $\{v_2, v_4\}$.

Observation 3.4 *blocks of d consecutive cells of value 0 appear only within the first k columns and only within a segment of rows of some vertex, namely rows tagged $r_{i,x}$ for some $1 \leq i \leq n$.*

We set the demand vector to be $\vec{C} = \left\langle c_1, c_2, \dots, c_{(d+1)n-(d-1)k} \right\rangle$, where $c_i = d$ if $1 \leq i \leq k$ and $c_i = 1$ otherwise.

We now show that there is a vertex cover of size k for G iff there is an optimal solution of value 0 for $(M, \vec{C})$. By Observation 3.4 every zero valued solution for the any permutation $\mathcal{MCAP}$ is also a valid zero valued solution for the fixed permutation $\mathcal{MCAP}$, thus it suffices to only consider fixed permutation $\mathcal{MCAP}$ in our proof. Assume there exists a zero valued solution S for $(M, \vec{C})$. Let

$$H := \{v_i \in V : M(r_{i,x}, h_j) \in S \text{ for some } x \text{ and some } 1 \leq j \leq k\}.$$

We claim that H is a vertex cover of size k for G. Clearly, $|H| = k$. Note that by Observation 3.4, for every $1 \leq j \leq k$, all cells allocated in the column tagged

h_j are within a single segment of rows. Namely, there exists some $1 \leq i \leq n$ such that all d allocated cells are of the form $M(r_{i,x}, h_j)$. Moreover, no cell $M(r_{i',x}, h_j)$ for $i' \neq i$ is in S. Now, given an edge $e = (v_s, v_t)$, assume $v_s \notin H$. Hence no cell is allocated in the row tagged $r_{s,e}$ within the first k columns. Since the only other zero valued cell of this row is $M(r_{s,e}, e)$, it has to be allocated. Therefore, the cell $M(r_{t,e}, e)$ which appears in the same column is not in S. Thus, since S must contain a zero valued cell from the row tagged $r_{s,e}$ and since all other zero cells in this row (other than the on in column e) appear within the first k columns, there is one which is allocated within the columns tagged by h_j. In other words, there exists a $0 \leq j \leq k$ such that $M(r_{t,e}, h_j) \in S$ and, hence, $v_t \in H$, that is, the edge e is covered.

For the other direction, assume $H = \{v_{i_1}, v_{i_1}, \ldots v_{i_k}\} \subseteq V$ is a vertex cover for G. We construct a legal zero value allocation for $(M, \vec{C})$. First, we set $S_{\text{Extra}} := \{M(\text{Sep}_i, \text{Extra}_i) : 1 \leq i \leq n\}$, covering all separating rows. Next, we set $S_{h_t} := \{M(r_{i_t,x}, h_t)\}$ for $1 \leq t \leq k$, that is, S_{h_t} is the zero-segment representing v_{i_t} in h_t, and we set $S_V := \bigcup S_{h_t}$. We also set $S_E := \{M(r_{i,e}, e) : v_i \notin H \text{ and } e = (v_i, v) \text{ for some } v \in V\}$. Note that if $e = (v_i, v)$, where $v_i \notin H$, then $v \in H$ and S_E contains at most one cell from each column tagged by an edge.

We have covered all rows except for padding rows of vertices not in the cover H, that is, rows tagged by $r_{i,j}$ for $v_i \notin H$ and $d_i+1 \leq j \leq d$. Furthermore, we have covered all columns except for columns tagged by Padd_j and columns tagged by edges $e = (u, v)$ where both u and v are in H (otherwise one of the cells in the column is in S_E). However, for every row tagged $r_{i,t}$ where $v_i \notin H$ and $d_i \leq t \leq d$ and for every column tagged by $e = (u, v)$ where $v_i \neq u, v$ we have $M(r_{i,t}, e) = M(r_{i,t}, \text{Padd}_j) = 0$ for any j. That is, the remaining submatrix of M is a square matrix whose entries are all 0. Thus, we can choose an allocation S_{Padd} containing exactly one cell from each row of this submatrix and one cell from each column. Clearly, all cells in $S := S_V \cup S_E \cup S_{\text{Padd}} \cup S_{\text{Extra}}$ have 0 value and, therefore, the total value of this solution is 0.

Finally, we show that S is a legal solution. One can easily observe that no row in M has more than one cell allocated in it and, since the number of allocated cells is exactly the number of rows, in each row there is exactly one cell. From the definition of S it follows that in each of the first k columns of M there are d consecutive cells allocated and in any other column exactly one cell is allocated. $\square$

3.3 Hardness of Approximation

The zero-one matrices constructed in the reductions of Lemmas 3.1, 3.3 yield a somewhat trivial multiplicative inapproximability result. Naturally, any multiplicative approximation algorithm must always return a zero valued solution whenever such solutions exist. Thus, our reductions imply that any such algorithm would solve the Vertex-Cover problem. In the following lemma we show that the inapproximability of $\mathcal{MCAP}$ is not only a trivial result of the zero valued solutions defined in the aforementioned reductions. The proof of the lemma is similar to the proof that the traveling sales person (TSP) problem (without the triangular inequality) cannot be approximated [9].

Lemma 3.5 *Let $\rho(a) \leq 2^{\text{poly}(a)}$. If $P \neq NP$, then for the* no block, any permutation $\mathcal{MCAP}$*, the* block, any permutation $\mathcal{MCAP}$*, and the* block, fixed permutation $\mathcal{MCAP}$*, there is no polynomial-time algorithm that $\rho(a)$-approximates the problem on instances $(M, \vec{C})$ where M has a rows.*

PROOF. Take the instance $(M, \vec{C})$ generated by the reduction in Lemma 3.1 or Lemma 3.3 and replace any 0 entry by 1, and replace any non-zero entry by $2n^2\rho(n)$. If we started with a graph G which has a vertex cover of size k, there is an allocation whose value is at most n^2. If we started with a graph G which does not have a vertex cover of size k, in every legal allocation in $(M, \vec{C})$ there is a non-zero cell and the value of the minimum allocation in $(M, \vec{C})$ is at least $2n^2\rho(n)$. If there is an approximation algorithm for one of the three versions of $\mathcal{MCAP}$, it would distinguish between the case that G has a vertex cover of size k and the case that G does not have a vertex cover of size k. $\square$

If in an instance with a rows, the matrix contains values only in the range $1, \ldots, H$ (for some value H), then the value of an optimal allocation is at least a while the value of any allocation is at most aH, thus a multiplicative approximation with ratio H is trivial. This is the reason that in our reduction we replace 1 by a big value (namely, $2n^2\rho(n)$). One may also consider additive approximability of the aforementioned cases of $\mathcal{MCAP}$. In instances where the matrix contains values only in the range $1, \ldots, H$ (for some value H), any solution is an $a \cdot H$ additive approximation. On the other hand, our reductions show that no additive approximation better than H may be achieved.

References

[1] 802.16-2004: IEEE standard for local and metropolitan area networks part 16: Air interface for fixed broadband wireless access systems (2004).

[2] 802.16e-2005: IEEE standard for local and metropolitan area networks part 16: Air interface for fixed and mobile broadband wireless access systems, amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1 (2006).

[3] Y. Ben-Shimol, E. Chai, I. Kitroser, Efficient mapping of voice calls in wireless OFDMA systems, IEEE Communication Letters 10 (9) (2006) 641 – 643.

[4] Y. Ben-Shimol, I. Kitroser, Y. Dinitz, Two dimensional mapping for wireless OFDMA systems, IEEE Transactions on Broadcasting 52 (3) (2006) 388– 396.

[5] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[6] H. W. Kuhn, The Hungarian method for assignment problem, Naval Res. Logist. Quart. 2 (1955) 83–98.

[7] N. Menakerman, R. Rom, Bin packing problems with item fragmentation, Tech. Rep. CCIT.342, Faculty of Electrical Engineering, Technion (2001).

[8] N. Naaman, R. Rom, Analysis of packet scheduling with fragmentation, Proceedings of Infocom'02, New York (2002) 824 – 831.

[9] S. Sahni, T. Gonzalez, P-complete approximation problems, J. ACM 23 (3) (1976) 555–565.

[10] H. Shachnai, T. Tamir, O. Yehezkely, Approximation schemes for packing with item fragmentation, in: Proceeding of the 3rd WAOA, vol. 3879 of Lecture Notes in Computer Science, 2006, pp. 334–347.

[11] D. B. Shmoys, E. Tardos, An approximation algorithm for the generalized assignment problem, Math. Program. 62 (3) (1993) 461–474.