Multi-level bottleneck assignment problems: complexity and sparsity-exploiting formulations

Trivikram Dokka[1] and Marc Goerigk*[2]

[1]Advanced Analytics Group, Air Products Plc, United Kingdom
(trivikram.dokka@yahoo.co.uk)
[2]Network and Data Science Management, University of Siegen, Germany
(marc.goerigk@uni-siegen.de)

Abstract

We study the multi-level bottleneck assignment problem: given a weight matrix, the task is to rearrange entries in each column such that the maximum sum of values in each row is as small as possible. We analyze the complexity of this problem in a generalized setting, where a graph models restrictions how values in columns can be permuted. We present a lower bound on its approximability by giving a non-trivial gap reduction from three-dimensional matching to the multi-level bottleneck assignment problem. We present new integer programming formulations and consider the impact of graph density on problem hardness in numerical experiments.

Keywords: combinatorial optimization; bottleneck assignment; approximation; computational complexity

1 Introduction

1.1 Problem definition

The following axial assignment problem arises to scheduling, rostering and finance applications: Given are m pairwise disjoint sets $S_1, S_2, \ldots, S_m$ each of cardinality n, and a weight $w(s) \in \mathbb{N}$ for each $s \in S$ where $S = \cup_{i\in[m]} S_i$ and $[m] := \{1, 2, \ldots, m\}$. The set S can be seen as the node-set of an m-partite graph that has a given set of arcs $E = \bigcup_{i\in[m-1]} E_i$, where $E_i = \{(u, s)|\ u \in S_i, s \in S_{i+1}\}$ connects nodes from S_i with nodes from S_{i+1}. An m-tuple $D = (s_1, s_2, \ldots, s_m)$ is feasible if $s_i \in S_i$ for $i \in [m]$ and $(s_i, s_{i+1}) \in E$. The weight of an m-tuple D equals $w(D) = \sum_{s\in D} w(s)$. The problem is

*Corresponding author.

to find a partition of S into n feasible m-tuples $D_1, D_2, \ldots, D_n$ such that $\max_{j\in[n]} w(D_j)$ is as small as possible. We refer to this partition of S into $\{D_1, D_2, \ldots, D_n\}$ as a solution M, and the weight $w(M)$ of a solution $M = \{D_1, D_2, \ldots, D_n\}$ equals $\max_{j\in[n]} w(D_j)$. This problem is known as the multi-level bottleneck assignment problem (MBAP). The MBAP with $m = 2$ is a special case of the classical bottleneck assignment problem and belongs to a class of high dimensional (axial) generalization of the well-known assignment problem, see Burkard et al. (2009). Therefore, MBAP is a hypergraph version of bottleneck assignment where the bottleneck weight is taken over all hyperedges. It is often seen through the lens of permutations of entries within the columns of a matrix (under constraints), such that the maximum row sum is minimized.

An example instance of the MBAP is illustrated in Figure 1. There are $m = 3$ layers, containing $n = 3$ nodes each. Next to each node, we indicate its weight.

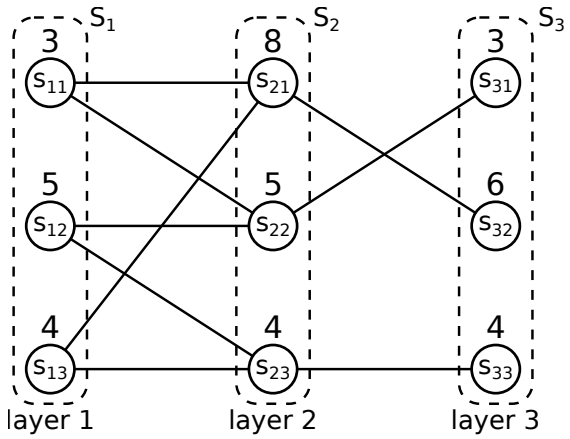


Figure 1: Example MBAP instance.

Note that there are only two feasible solutions in this instance. The first solution consists of the partition $\{(s_{11}, s_{22}, s_{31}), (s_{12}, s_{23}, s_{33}), (s_{13}, s_{21}, s_{32})\}$ with weight

$$\max\{3 + 5 + 3,\ 5 + 4 + 4,\ 4 + 8 + 6\} = \max\{11, 13, 18\} = 18.$$

The second solution consists of the partition $\{(s_{11}, s_{21}, s_{32}), (s_{12}, s_{22}, s_{31}), (s_{13}, s_{23}, s_{33})\}$ with weight $\max\{17, 13, 12\} = 17$, and is hence optimal.

1.2 Related literature

The MBAP has connections to many important problems in scheduling and finance. The MBAP was first introduced and studied by Carraresi and Gallo (1984), motivated by an application in bus driver scheduling. Special cases of the problem have been studied even before Carraresi and Gallo (1984). A particularly important special case which we call the *complete-MBAP*, as referred to in Dokka et al. (2012), is when each E_i is complete. The approximability of this special case has been studied by Hsu (1984) and by

Coffman Jr and Yannakakis (1984). For the complete-MBAP, Hsu (1984) gave a $(2-\frac{1}{n})$-approximation algorithm that runs in $O(mn\log n)$, while Coffman Jr and Yannakakis (1984) gave a $(\frac{3}{2}-\frac{1}{2n})$-approximation algorithm that runs in $O(n^2m)$. For the case where $m=3$, Hsu (1984) gave a $\frac{3}{2}$-approximation algorithm that runs in $O(n\log n)$, and a $\frac{4}{3}$-approximation algorithm that runs in $O(n^3\log n)$.

Another important problem that the MBAP contains as a special case is the bi-criteria scheduling problem (BCSP) in which one tries to find a schedule of jobs on parallel machines which has minimum makespan over all flow time optimal schedules on identical parallel machines. Here makespan indicates completion time of last job and flowtime indicates total time that jobs spend in the system. To see the connection, a characteristic of BCSP is that jobs (after ordering according to non-increasing processing times) can be grouped, referred to as *rank groups* of size at most equal to number of machines, where jobs in a lower ranked group have higher processing time than any job in any higher ranked group, see Ravi et al. (2016). A schedule is flow time optimal only if on every machine for any two jobs s_1 and s_2, s_1 precedes s_2 if and only if group of s_1 is lower than s_2. This implies each rank group can be seen as equivalent to a set in S in MBAP, with cardinality of largest rank group being equal to the number of machines. Note that this leads to a special case of MBAP because this leads to an instance where $w(s_1) \geq w(s_2)$ when $s_1 \in S_i$ and $s_2 \in S_j$ with $j > i$. This problem was first studied in 1976 by Coffman Jr and Sethi (1976), where a $\frac{5}{4}$-approximation algorithm is given. Eck and Pinedo (1993) give a $\frac{28}{27}$-approximation for the two machine case. More recently Ravi et al. (2016) proved the Coffman and Sethi conjecture on the performance of (a natural extension) of the longest processing time algorithm to this bi-criteria scheduling problem. We note that the greedy algorithm often used for the MBAP when applied to the above special case of bi-criteria scheduling can be interpreted as an extension of the longest processing time list scheduling algorithm.

The MBAP is an important special case of the parallel machine scheduling with conflicts (PMC) which has recently received attention due to its practical importance, see for example the ROADEF challenge furnished by Google (ROADEF, 2011). We note that the conflict graphs used in this stream of literature are the complements to the layered graphs in our setting. In other words, conflict graphs capture the notion of incompatible jobs whereas in our setting the graph captures the notion of compatible jobs. Indeed, the associated conflict graph of layered graph in MBAP also will have layered structure. To draw the parallel between MBAP and PMC, similar to BCSP, all jobs that are together in a m-tuple should be seen as being scheduled on same machine. Bodlaender et al. (1994) studied hardness and approximation for specific conflict graphs. Integer programming formulations and exact algorithms are still to receive good attention, see Kowalczyk and Leus (2017). Our work both complements and extends the research on PMC. The MBAP is also connected to yet another variant of PMC identified as parallel machine scheduling with bags, see Das and Wiese (2017) and Page and Solis-Oba (2018).

The MBAP also finds application in estimating variance and VaR bounds by inferring stochastic dependence between many random variables in quantitative finance, see Haus (2015). Here, each set S_i in MBAP is a sample from the known marginal distribution

of the underlying random variable and a solution to MBAP expresses the dependence structure between random variables. The problem is inherent to the concepts of joint and completely mixability of matrices, see Wang and Wang (2011, 2016). A number of studies explored complete-MBAP to bound risk measures, see Puccetti and Ruschendorf (2012), Boudt et al. (2017), Bernard et al. (2017) and Bernard et al. (2018a). More recently problems closely related to MBAP have been explored within this literature stream, see Bernard et al. (2021) and Bernard et al. (2018b).

More generally, bottleneck assignment problems have also been studied within other applications, see Klinz and Woeginger (1996) and Goossens et al. (2010). A classification of bottleneck problems into two classes is based on graph theoretical interpretation. In the first class, problems are graph-based, while in the second class, the underlying constraint is defined over a hypergraph. In graph-based problems a single edge plays the role of the bottleneck, whereas in hypergraph-based problems it is a hyperedge. There are many graph-based bottleneck problems studied including the bottleneck assignment problem and p-center problems, see Çalık et al. (2019). Hypergraph-based problem generalize the graph-based problems. This generalization makes these problems harder to extend radius-type approaches that work well for graph-based bottleneck problems.

1.3 Contributions

While considerable work is dedicated to understanding the complexity of the MBAP when all edge sets are complete, not much is known about the case when the edge sets are arbitrary. The only known result is from Dokka et al. (2012), which gives a lower bound of $2 \cdot OPT$ on approximability in the case when $m = 3$ and shows a simple greedy approach to achieve a matching upper bound. The complete-MBAP is shown to admit a PTAS, first shown in Dokka et al. (2012), while the simple greedy approach is already known to yield a 2-approximation in Hsu (1984). Given that the greedy approach gives a constant factor approximation in the complete case, it is tempting to believe a similar approach may be used to construct a constant factor approximation for the general case. As our first contribution we show that there is no polynomial-time algorithm which approximates the MBAP to within a constant factor of the optimum objective value, by giving a non-trivial gap reduction from three-dimensional matching (3DM). More specifically, we prove that the existence of a polynomial-time $((u+1)-\epsilon)$-approximation algorithm for the MBAP with $m = 3u$ implies $P = NP$.

The hardness of MBAP is tightly linked with the sparsity of the underlying graph. For example the complexity of the problem changes going from complete-MBAP to partially complete case when $m = 3$. The instance constructed in Section 2 to prove the inapproximability result will further establish that sparsity increases the hardness of the problem. Graph sparsity is a natural aspect in problems with conflicts, as an example, machine scheduling with conflicts. Naturally, as the sparsity increases the feasibility of the instance may not be guaranteed. On the other hand very sparse instances may be expected to be easy given that very few feasible solutions need to be explored, provided they exist. This naturally gives rise to the idea of integer programming (IP) models which explicitly make use of graph sparsity. It is conceivable that for different ranges

of graph sparsity different models are useful, with tighter relaxations and may even be solvable using commercial solvers such as CPLEX. Although it is not uncommon for combinatorial problems to have multiple IP formulations, however, we are not aware of any studies which explore the connections to the underlying graph density.

As our second contribution we give a new integer programming formulation and show that this formulation has inherent advantages manifested by graph sparsity. We illustrate using computational experiments that dense instances result in smaller IP gaps than sparse instances. Furthermore, the new formulation offers significant benefits compared to a standard formulation, which uses variables for nodes instead of edges. In graph based bottleneck problems, an attractive feature is that the candidates for bottleneck value are at most equal to number of edges of graph. Therefore, the set of candidate values can be pre-computed and can be used within a search framework or to construct formulations such as radius-based formulations (see Çalık et al. (2019)). In hypergraph bottleneck problems, like MBAP, computing such a candidate set is hard even for slightly larger $m > 2$ as it amounts to calculating the weight of each hyperedge. Hence, radius-based formulations are hard to extend to MBAP. On the other hand, both graph-based and hypergraph-based problems can be modelled as set covering/partitioning type problems. A set partitioning formulation for MBAP can be seen as aggregate formulation, as it explicitly models variables on hyperedges (m-tuples) as against dis-aggregate or decomposed type formulations which are more compact owing to variables defined over edges instead of hyperedges. However, aggregate formulations grow very large as the dimension of the problem (m) increases from graphs to hypergraphs. Our new proposed formulation can be seen as a semi-aggregate formulation which combines advantages from both dis-aggregate and aggregate formulations.

The rest of the paper is structured as follows: We give the inapproximability result for the general case in Section 2. In Section 3, we present integer programming formulations. These methods are compared experimentally using random MBAP instances in Section 4. Section 5 summarizes our findings and points out further research questions.

2 Inapproximability of the arbitrary case

In Hsu (1984) it is shown that for the complete-MBAP the natural sequential heuristic achieves a 2-approximation. It is tempting to believe that this may be true even in the arbitrary case. We show that the MBAP for a fixed $m > 3$ cannot be approximated within a factor of $\lfloor \frac{m}{3} \rfloor + 1$ unless $P = NP$. To do so, we show that a YES-instance of 3-dimensional matching (3DM) corresponds to an instance of the MBAP with weight 1, whereas a NO-instance corresponds to an instance of our problem with weight $\frac{m}{3} + 1$. Then, a polynomial-time approximation algorithm with a worst case ratio strictly less than $\frac{m}{3} + 1$ would be able to distinguish the YES-instances of 3DM from the NO-instances, and this would imply $P = NP$.

Let us first recall the 3-dimensional matching problem:

Instance: three sets $X = \{x_1, ..., x_q\}$, $Y = \{y_1, ..., y_q\}$, and $Z = \{z_1, ..., z_q\}$, and a subset of tuples $T \subseteq X \times Y \times Z$.

Question: Does there exist a subset T' of T such that each element of $X \cup Y \cup Z$ is in exactly one triple of T'?

Let the number of triples be denoted by $|T| = p$. Further, let the number of triples in which element y_ℓ occurs be denoted by $\#occ(y_\ell)$, $\ell = 1, 2, \ldots, q$.

Starting from an arbitrary instance of 3DM, we now build a corresponding instance of the MBAP by specifying sets S_i, edges E, and the weights w. Before we explain the construction we first explain the basic building blocks and gadgets which are pieced together to form an MBAP instance.

Example 2.1. *To illustrate our construction, we use the following example instance of 3DM:*

$$X = \{x_1, x_2\},\ Y = \{y_1, y_2\},\ Z = \{z_1, z_2\}$$
$$T = \{t_1, t_2, t_3\}$$
$$t_1 = (x_1, y_1, z_1),\ t_2 = (x_2, y_2, z_2),\ t_3 = (x_1, y_2, z_1)$$
$$q = 2,\ p = 3,\ d = p - q = 1$$
$$\#occ(y_1) = 1,\ \#occ(y_2) = 2$$

2.1 Building sub-blocks

There are two types of nodes in the resulting MBAP instance, which we call *main* and *dummy* nodes.

The main nodes in each set in the MBAP instance are partitioned into sub-blocks of nodes. Each sub-block is of cardinality q, p, or $d := p - q$. We use the following types:

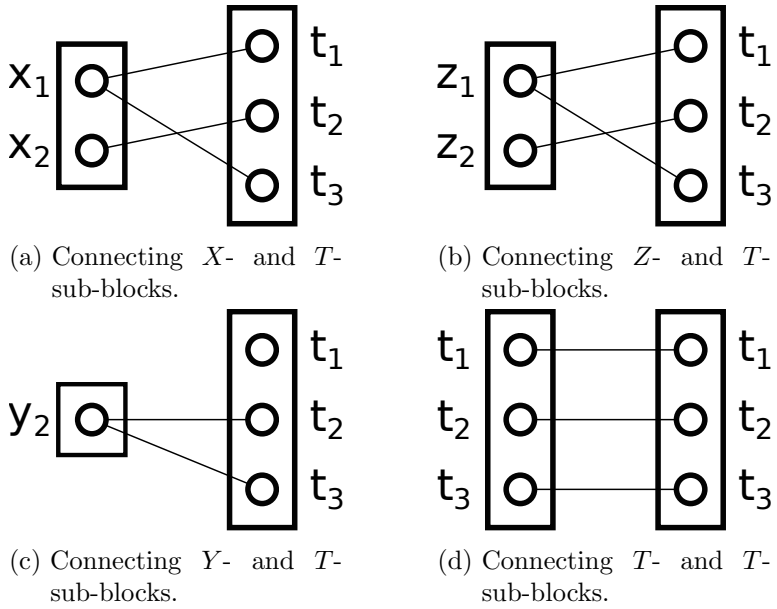
- X-sub-blocks, where each node corresponds to one element in X (cardinality q)
- Z-sub-blocks, where each node corresponds to one element in Z (cardinality q)
- Y-sub-blocks, where $\#occ(y_i) - 1$ many nodes correspond to one of each element $y_i \in Y$ (cardinality d)
- T-sub-blocks, where each node corresponds to one triple in T (cardinality p)

In our construction, we may refer to two sub-blocks as being *connected*. When this is the case, the corresponding edge set depends on the type of sub-blocks:

- X-sub-blocks are connected to T-sub-blocks by connecting a node in the X-sub-block corresponding to an element $x_i \in X$ with those nodes in the T-sub-block corresponding to tuples that contain x_i
- Y- and Z-sub-blocks are connected to T-sub-blocks in the same way
- two T-sub-blocks are connected by connecting each two nodes corresponding to the same tuple

The role of the dummy nodes and their exact number will be apparent when we explain the connections between gadgets in the later sections.

Example 2.2 (continued)**.** *In our 3DM example, X- and Z- sub-blocks have* 2 *nodes each. A Y-sub-block has only one node, corresponding to y_2. T-sub-blocks have three nodes. Figure 2 shows the different possibilities how these sub-blocks can be connected.*



(a) Connecting X- and T-sub-blocks.

(b) Connecting Z- and T-sub-blocks.

(c) Connecting Y- and T-sub-blocks.

(d) Connecting T- and T-sub-blocks.

Figure 2: Connecting sub-blocks in the example.

2.2 Gadget construction

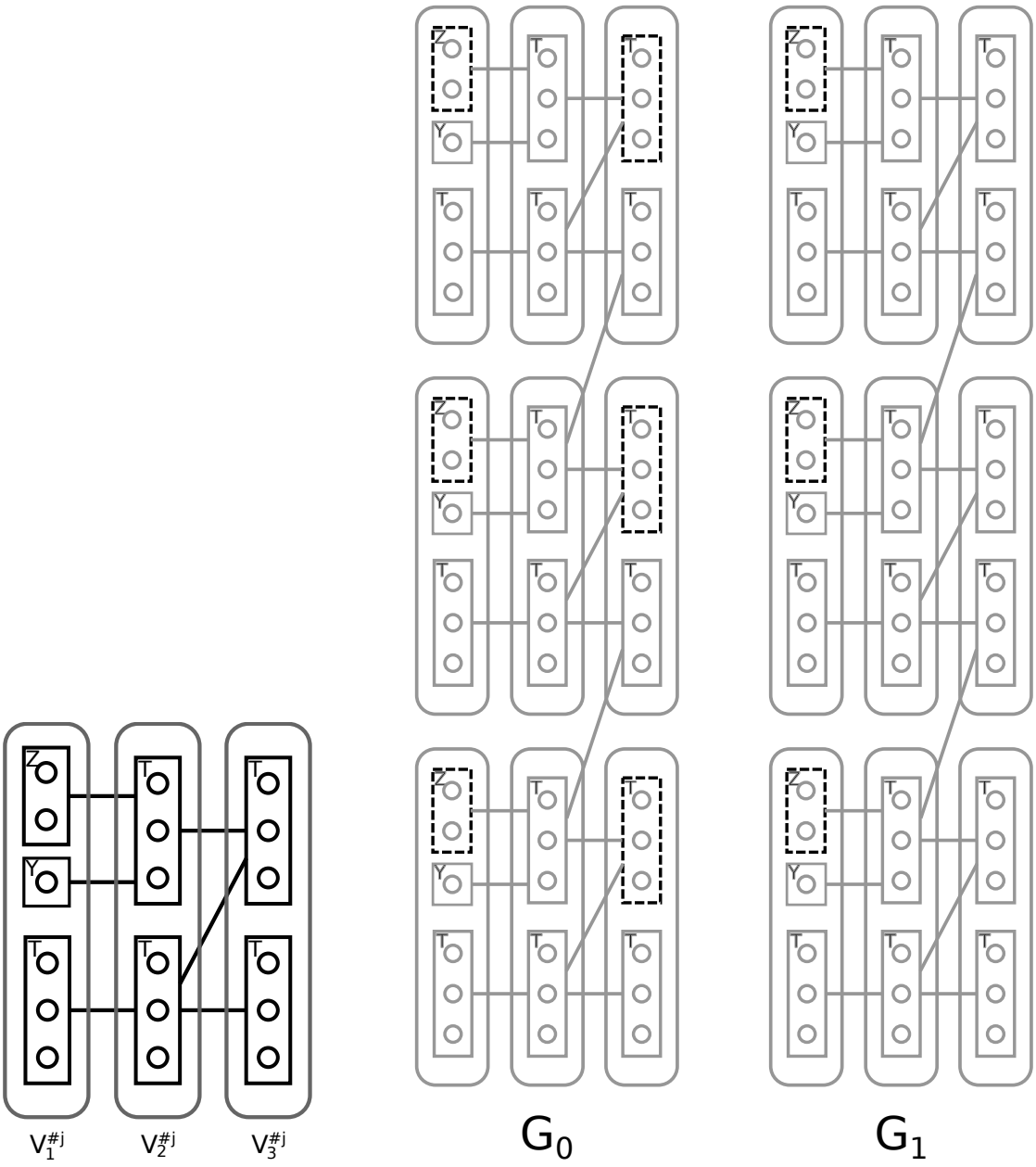
The construction is mainly based on two gadgets G_0 and G_1, which are both MBAP instances with $m = 3$. Each gadget consists of multiple blocks j, where each block is made up of three sets $V_1^{\#,j}$, $V_2^{\#,j}$, and $V_3^{\#,j}$ for $G_\#$, $\# \in \{0, 1\}$.

Blocks and connections within blocks: Each block j of $G_\#$, $\# \in \{0, 1\}$, has three times $2p$ nodes grouped as follows:

- in $V_1^{\#,j}$, there is a Z-sub-block, a Y-sub-block, and a T-sub-block
- in both $V_2^{\#,j}$ and $V_3^{\#,j}$, there are two T-sub-blocks

The head T-sub-block in $V_2^{\#,j}$ is connected to the Z-sub-block and Y-sub-block in $V_1^{\#,j}$, and to the head T-sub-block in $V_3^{\#,j}$. The tail T-sub-block in $V_2^{\#,j}$ is connected to the tail T-sub-block in $V_1^{\#,j}$, and to the head and tail T-sub-blocks in $V_3^{\#,j}$.

Example 2.3 (continued)**.** *Figure 3a illustrates how blocks $V_1^{\#,j}$, $V_2^{\#,j}$, and $V_3^{\#,j}$ are constructed.*



(a) Illustration of j^{th} block in G_0 and G_1. A single connection represents multiple edges as indicated in Figure 2.

(b) Connections between blocks. The highlighted sub-blocks have nodes with node weights equal to 1.

Figure 3: Illustration of block construction and connection.

Connections between blocks: We denote by *height* the number of blocks in a gadget. For $1 \leq j < height$, the tail T-sub-block in $V_3^{\#,j}$ is connected to head T-sub-block of $V_2^{\#,j+1}$. Note that this way, only adjacent blocks are connected by edges.

Weights: Note that in the construction so far, gadgets G_0 and G_1 are identical. They differ with respect to their weights. The head T-sub-block in $V_3^{0,j}$ and the Z-sub-block in $V_1^{0,j}$ and $V_1^{1,j}$ have all nodes weight equal to 1 for all j. All other nodes are weighted 0.

Example 2.4 (continued)**.** *Figure 3b shows how blocks are connected. Sub-blocks with weight 1 are highlighted with dashed lines.*

2.3 Main construction

2.3.1 Overview

We intend to show that it is not possible to approximate the MBAP within $\lfloor \frac{m}{3} \rfloor + 1$ of optimum, for every fixed m. The exact structure of the MBAP instance constructed from 3DM depends on m. We assume that $m = 3u$ for an integer u. We put together $(u-1)$ many G_1 gadgets and one G_0 gadget, each of different height, in that order to create an MBAP instance. Since each gadget is itself an MBAP instance with three columns, the resulting instance is an MBAP instance with $m = 3u$. For the ease of explanation we refer to each triple (S_i, S_{i+1}, S_{i+2}) as a layer, $i = 1, 4, 7, \ldots, 3u-2$. The number of layers is equal to u. While we count the sets starting from left to right, we count layers from right to left. That is, layer $k+1$ is placed to the left of layer k. The height of the gadget in layer $k \in [u]$ is equal to $q^{k-1}p^{u-k}$ blocks. Before we explain how these gadgets are connected in sequence we need some additional blocks of nodes as follows:

Example 2.5 (continued)**.** *In our example, we would like to use the 3DM instance to construct an MBAP instance with $m = 6$. This means that $u = 2$. Hence, we have one G_1 gadget and one G_0 gadget. The height of the first (G_0) gadget is $q^0p^1 = 3$, and the height of the second (G_1) gadget is $q^1p^0 = 2$.*

2.3.2 Non-gadget nodes

X-sub-blocks: We have one X-sub-block in every S_i of each layer k. These are connected as follows:

- the X-sub-block in S_{3k} is connected to the head T-sub-block of $V_2^{\#,1}$ (first block of $G_\#$ in the k^{th} layer) in S_{3k-1},
- the X-sub-block in S_{3k-1} is connected to the tail T-sub-block of $V_3^{\#,height}$ (in the last block of $G_\#$ in the k^{th} layer) in S_{3k},
- the X-sub-block in S_{3k-2} is connected to the X-sub-block of S_{3k-1} element-wise,

where # takes a value 0 in layer u and 1 for all other layers.

Dummy blocks: Each S_i has an additional set of nodes apart from gadget and non-gadget sub-blocks. We refer to these nodes as *dummy* nodes. The number of dummy nodes in layer k is equal to sum of non-dummy nodes in all other layers. The dummy nodes in layer k are evenly distributed over the three columns S_{3k}, S_{3k-1} and S_{3k-2}. All dummy nodes in S_{3k} are connected to all dummy nodes in S_{3k-1}, and all dummy nodes in S_{3k-1} are connected to all dummy nodes in S_{3k-2}. All dummy nodes have a weight equal to 0.

2.4 Connecting gadgets

There are two types of edges connecting gadgets: edges to dummy nodes, and edges between Z- and T-sub-blocks. Edges to dummy nodes in layer k are constructed as follows:

- Every node from every Y- and T-sub-block in S_{3k-2} is connected to every dummy node in $S_{3(k+1)}$ of the neighboring layer $k+1$.
- Every node from every bottom T-sub-block in S_{3k} is connected to every dummy node in $S_{3(k-1)-2}$ of the neighboring layer $k-1$.

Nodes in Z-sub-blocks of layer k are connected to T-sub-blocks in $S_{3(k+1)}$ of next layer $k+1$ in the following way. Blocks in layer k are grouped into p^{u-k} groups of q^{k-1} blocks each. In each group, there are thus q^{k-1} Z-sub-blocks with q nodes each.

Figure 4 provides an example with $q = 2$, $p = 3$, and $u = 3$. Layer $k = 1$ has 9 blocks that are partitioned into 9 groups of one block each. Layer $k = 2$ has 6 blocks that are partitioned into 3 groups of 2 blocks each. Finally, layer $k = 3$ has 4 blocks that form a single group.

Using p groups at a time, we connect the ℓ^{th} z-node to all nodes of the ℓ^{th} top T-sub-block in the neighboring layer. In Figure 4 we see that for the top p blocks of the first layer, always the first node of each Z-block is connected to the top T-sub-block of the first block. The second node of each Z-block is connected to the top T-sub-block of the second block. In the first group of the middle layer, there are two Z-sub-blocks with two nodes each. These four nodes are connected to the four blocks of the third layer. In the same way, the four nodes of the second and the four nodes of the third group are connected to the four blocks of the third layer.

Example 2.6 (continued)**.** *We show the fully constructed MBAP instance resulting from the example 3DM instance in Figure 5. As there are two layers, Y- and T-sub-blocks on the left of the right layer are connected to dummy nodes on the right of the left layer. Bottom T-sub-blocks on the right of the left layer are connected to dummy nodes on the left of the right layer. Z-sub-blocks of the right layer are connected to T-sub-blocks on the right of the left layer as indicated.*

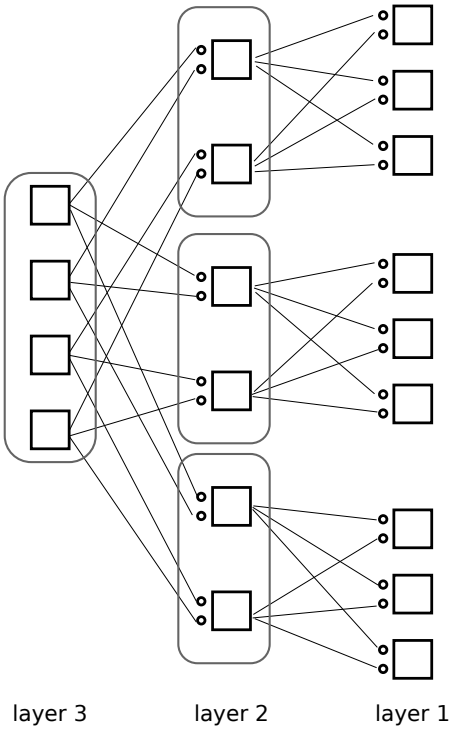


Figure 4: Connecting gadgets by edges between Z- and T-sub-blocks.

2.5 Instance analysis

We first note that the size of instance grows with m, but is bounded by a polynomial when m is fixed.

Lemma 2.1. *The number of nodes in the constructed instance is polynomial in p, q for each fixed m.*

Proof. The total number of nodes in layer k is

$$3q + 6pq^{k-1}p^{u-k} + \sum_{\bar{k}\in[u];\bar{k}\neq k} (3q + 6pq^{\bar{k}}p^{u-\bar{k}}) < u(3q + 6p^{u+1}),$$

which is polynomial for constant $u = m/3$. □

Lemma 2.2. *Consider only the following construction: The first column contains a Z- and Y-sub-block. These are connected to a T-sub-block in the second column. This T-sub-block is connected to an X-sub-block in the third column, and also to a dummy sub-block with the same size as the Y-sub-block. Then it holds that the corresponding 3DM instance is a YES instance if and only if it is possible to match all nodes of the Z-sub-block through the T-sub-block with the X-sub-block.*

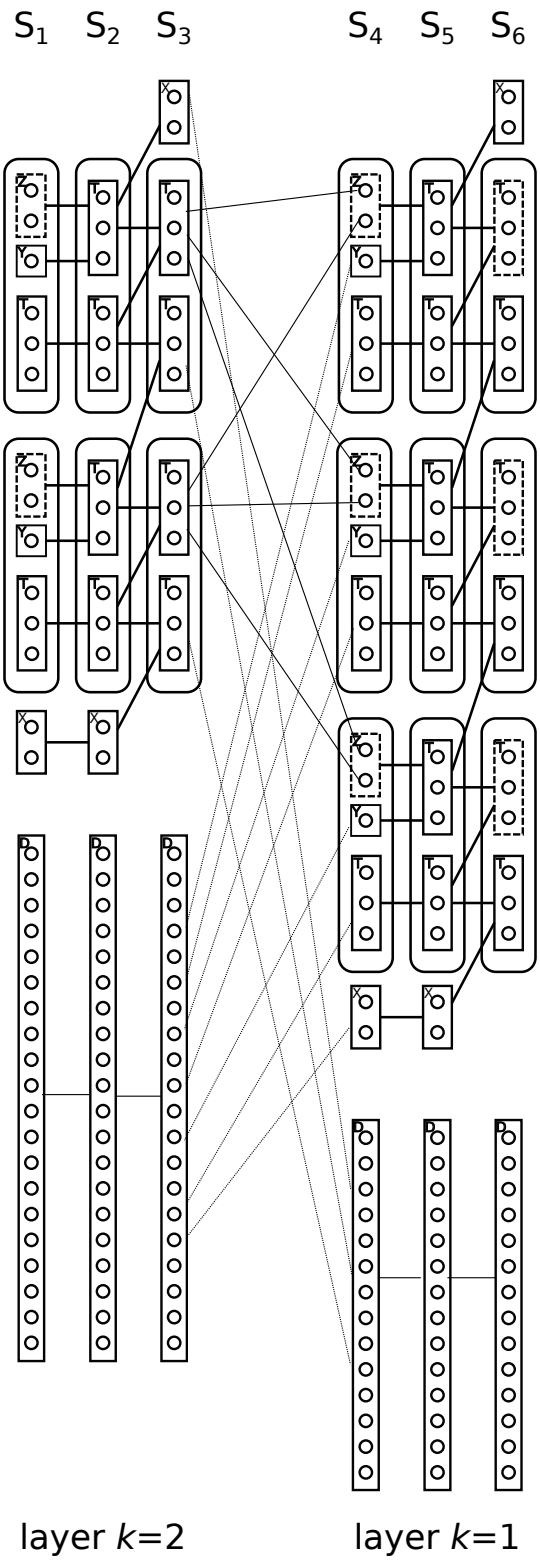


Figure 5: Fully constructed MBAP instance. Dashed edges represent connections to dummy blocks.

Proof. Let 3DM be a YES instance, and let $T' \subseteq T$ be the choice of tuples in an optimal solution. We can build a feasible solution to the corresponding MBAP by matching the pairs of X and Z elements contained in T' through the respective T-sub-block nodes. Remaining nodes in the Y-sub-block are matched with the remaining nodes in the T-sub-block and the dummy-sub-block.

On the other hand, if the MBAP instance allows a feasible matching of all nodes of the Z- and X-sub-block, then one can create a feasible solution to the 3DM instance by only choosing the corresponding nodes of the T-sub-block that are traversed. □

Example 2.7 (continued)**.** *Figure 6 shows the construction described in Lemma 2.2 for our example 3DM instance.*

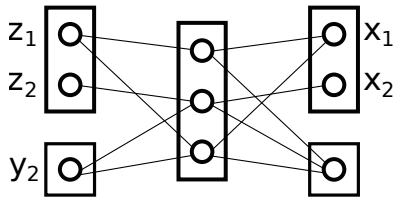


Figure 6: Construction in Lemma 2.2.

Note that the 3DM instance is indeed a YES instance, and it is possible to match the Z- and X-sub-block using t_1 and t_2 in this example.

Lemma 2.3. *In every layer the following is true for any feasible solution of the MBAP instance: The matching between head T-sub-blocks in $V_2^{\#,j}$ and $V_3^{\#,j}$; and tail T-sub-blocks in $V_2^{\#,j}$ and $V_3^{\#,j}$ is exactly identical in every block j of $G_\#$, where # is 0 in the first layer, and 1 in every other layer.*

Proof. First we observe that the degree of each node in $V_2^{\#,j}$ and $V_3^{\#,j}$ in the bipartite graph with node-sets $\bigcup_j V_2^{\#,j}$ and $\bigcup_j V_3^{\#,j}$ is equal to 2. Since the X-sub-block in S_{3k} is only connected to the head T-sub-block of $V_2^{\#,1}$ and two T-sub-blocks are connected node-wise, this implies that

- the head T-sub-block in $V_3^{\#,j}$ is connected to the tail T-sub-block of $V_2^{\#,j}$ in the same way in every j,
- the tail T-sub-block in $V_3^{\#,j}$ is connected to the head T-sub-block of $V_2^{\#,j+1}$ in the same way in every $j < height(G_\#)$.

□

Proposition 2.1. *If 3DM is a NO instance then in any solution of the resulting MBAP instance the following is true: In every layer, there is an $\ell \in [q]$ such that the ℓ^{th} node in the Z-sub-block in $V_1^{\#,j}$ in $G_\#$ is matched with the head T-sub-block in $V_3^{\#,j}$ in $G_\#$, for all $j = 1, 2, \ldots, height(G_\#)$.*

Proof. Let Γ be the nodes in head T-sub-block in $V_3^{\#,j}$ which are matched with the head T-sub-block in $V_2^{\#,j}$. Lemma 2.3 shows that the sets Γ are the same in each j. Since it is a NO instance there must exist a $z \in Z$ such that all triple nodes intersecting z are contained in Γ. If not, this implies that all nodes in the Z-sub-block in $V_1^{\#,1}$ can be fully matched with the X-sub-block, which is contradiction by Lemma 2.2. Since the node corresponding to z in the Z-sub-block of $V_1^{\#,j}$ is only connected to the head T-sub-block in $V_2^{\#,j}$ and only to nodes of Γ, the statement follows. □

We denote by $\zeta_k^{j,\ell}$ the set of z-nodes in layer k over all groups, which are within block $j \in [q^{k-1}]$ in position $\ell \in [q]$. For example in Figure 4, the set $\zeta_2^{2,1}$ denotes those nodes in the middle layer that are in the first position of the second block in each group.

Proposition 2.2. *If 3DM is a NO instance then the following is true for the resulting MBAP instance: In any partial solution in sets combining layers* $1, 2, \ldots, k$*, there is an* $\ell \in [q]$ *and* $j \in [q^{k-1}]$*, such that the weight of the* $3k$*-tuples containing nodes of set* $\zeta_k^{j,\ell}$ *is equal to* $k+1$*.*

Proof. Following Proposition 2.1, there is an $\ell \in [q]$ such that in every $j \leq height(G_{\#})$, the ℓ^{th} node is connected to head T-sub-block of $V_3^{\#,j}$. The proof is by induction. Suppose that the statement is true up to layer $k-1$, that is, there exists a block in the lowest section of layer $k-1$ with at least one element in Z-sub-block in a $3(k-1)$-tuple with weight k. Since head T-sub-block of $V_3^{\#,j}$ is only connected to ζ sets constructed from Z-sub-blocks in layer $k-1$, one of which by construction has all nodes in $3(k-1)$ tuples with weight k; and weight of each node in Z-sub-blocks of $V_1^{\#,j}$ is 1, it follows that at least one node in some block of lowest section of layer k will be in a $3k$-tuple with weight $k+1$.

It remains to show that statement is true in the base case, that is, for layer 2. Since the all nodes in Z- and head T-sub-blocks in each block of G_0 have weight 1, from Proposition 2.1, the ℓ^{th} node in Z-sub-block in each block of G_0 is in a triple with weight 2 and now the statement for base case follows by construction of ζ sets. □

Proposition 2.3. *If the instance of 3DM is a YES instance, then there exists a solution in the corresponding MBAP instance with weight equal to* 1*.*

If the instance of 3DM is a NO instance, then any solution in the corresponding MBAP instance has weight equal to $u+1$*.*

Proof. YES case: To prove the statement we need to show that each 1 is not matched with another 1 in the same $3k$-tuple. By Lemma 2.2, in every layer the Z-sub-block in $V_1^{\#,1}$ is fully matched with the X-sub-block. Recall that matching within gadgets is completely defined by the matching between the Z-sub-block in $V_1^{\#,1}$ and the X-sub-block by construction. Using Lemma 2.2, we match

- the X-sub-block in S_{3k} of layer k, for all $k > 1$, to the dummy block in $S_{3(k-1)-2}$,
- all nodes in the tail T-sub-block in S_{3k} of layer k, $k > 1$, to the dummy block in $S_{3(k-1)-2}$,

- all nodes except the Z-sub-block in S_{3k-2}, $k < u$, to the dummy block in $S_{3(k+1)}$.

To complete the proof it is enough observe that each sub-block in $G_{\#}$ is matched with a matching sub-block in the adjacent layer and the X-sub-blocks are matched only with dummy blocks in this matching.

NO case: From Proposition 2.2, it follows that there exists a $3u$-tuple with weight $u+1$. □

Using the above (gap) reduction and Proposition 2.3, we can now state the following result

Theorem 2.1. *The MBAP with $m = 3u$ cannot be approximated to within a ratio $u+1$ unless P=NP.*

Remark 2.1. *Note that, similar to Dokka et al. (2012), the MBAP instance constructed here has weights only from $\{0, 1\}$. Clearly, this is the simplest possible weight set which implies the hardness of the problem originates mainly from the edge-set of the underlying graph. This observation was already made in Dokka et al. (2012) in the case when $m = 3$. In fact it is easy to see that when weights are taken from $\{0, 1\}$ then the PTAS for the case when edge sets are complete can be used to construct a PTAS for the case when every second edge set is complete. We conjecture that a similar approach can be used with a general weight set. Since our interest in the paper is in the complexity of the general case we leave out the study of these special cases for future study.*

3 Integer programming models

3.1 Motivation

The construction in the proof of Theorem 2.1 is delicate and results in a sparse graph structure. Recalling that the MBAP can be seen as a special case of parallel machine scheduling with conflicts (PMC), our observation connecting the hardness to the sparsity of the underlying graph contrasts with similar observations in the PMC literature. For example, Kowalczyk and Leus (2017) observes that the PMC problem can be hard with a dense conflict graph (recall that the conflict graph is the complement of the layered graph, where an edge between nodes corresponding to jobs implies they are incompatible). However, this increasing hardness mainly comes from the feasibility itself, whereas in our case feasibility of a given instance is easy to check given the layered structure of the graph. A second advantage of the layered structure is that symmetry of solutions, which makes natural integer programming models impractical for the PMC problem, is easier to handle in the case of the MBAP. This suggests that the special structure may be used to design integer programming models which can make better use of sparsity. This motivates the study in this section. More specifically, we present two IP models in this section and illustrate using experiments that there exists a partition of the range of graph sparsity where one formulation dominates the other.

Let $G = (V, A)$ denote the layered graph underlying the MBAP. Without loss of generality, we assume G to be directed from the first layer towards the last layer. We denote by V^j the nodes in layer $j \in [m]$. Every node $v = (i, j) \in V$ can be represented by its node number $i \in [n]$ and layer number $j \in [m]$. Let $\delta^+(v) \subseteq V^{j+1}$ contain the successor nodes of $v \in V^j$ for $j \in [m-1]$. Analogously, we denote by $\delta^-(v) \subseteq V^{j-1}$ the predecessor nodes of $v \in V^j$ for $j \in \{2, 3, \ldots, m\}$. In the following, we refer to the weights in sets as a weight matrix and denote it as w_{ij} for $i \in [n]$, $j \in [m]$. That is, w_{ij} refers to the weight of item number i in layer j. Note that while we use two indices, this notation still refers to a single node.

3.2 Model 1

Our first model uses a binary variable x_{ijk} for each $i, k \in [n]$ and $j \in [m]$ that is equal to one if and only if tuple k contains element i in level j. Let D be a variable modeling the largest weight over all tuples. Then, the MBAP can be written as the following integer programming model:

$$\min D \tag{3.1}$$

$$\text{s.t.} \sum_{i \in [n]} x_{ijk} = 1 \qquad \forall j \in [m], k \in [n] \tag{3.2}$$

$$\sum_{k \in [n]} x_{ijk} = 1 \qquad \forall i \in [n], j \in [m] \tag{3.3}$$

$$\sum_{i \in [n]} \sum_{j \in [m]} w_{ij} x_{ijk} \leq D \qquad \forall k \in [n] \tag{3.4}$$

$$\sum_{(i', j+1) \in \delta^+(i,j)} x_{i', j+1, k} \geq x_{ijk} \qquad \forall i, k \in [n], j \in [m-1] \tag{3.5}$$

$$x_{i1i} = 1 \qquad \forall i \in [n] \tag{3.6}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, k \in [n], j \in [m] \tag{3.7}$$

Objective (3.1) is to minimize the largest weight. Constraints (3.2) and (3.3) ensure that each tuple uses exactly one element from each set, and each element in each set is used exactly once, respectively. By Constraint (3.4), we enforce D to become equal to the largest weight over all tuples in an optimal solution. Constraint (3.5) models the connectivity of the graph: If (i, j) is used, then it is only possible to use elements $(i', j+1)$ that are in $\delta^+(i, j)$. Finally, the purpose of Constraint (3.6) is to break the symmetry in the solution variables. Notice that this is very similar to the formulation of the PMC problem. However, there are key differences due to the special layered structure, due to which the constraints usually referred to as symmetry breaking constraints often used to strengthen the formulation for the PMC problem, see Kowalczyk and Leus (2017), are not applicable to the MBAP.

Note that this model has $O(n^2 m)$ many variables and constraints. In particular, its size does not depend on the density of the graph G.

3.3 Model 2

The main focus of this paper are MBA problems where the edge sets of G are not complete. As the theoretical analysis indicates, such problems are harder than their complete counterparts. The model we present here takes advantages of the potential sparsity of an instance.

For each $e \in E$, we define c_e as the weight of the node that this edge points to, i.e., $c_e = w_{t(e)}$ for $e = (s(e), t(e))$. We introduce binary variables x_{ek} modeling whether the path number $k \in [n]$ uses edge $e \in E$. The following mixed-integer programming model can then be used to formulate the MBAP:

$$\min D \tag{3.8}$$

$$\text{s.t } w_{k1} + \sum_{e \in E} c_e x_{ek} \leq D \qquad \forall k \in [n] \tag{3.9}$$

$$\sum_{e \in \delta^+(v)} x_{ei} = 1 \qquad \forall v = (i, 1) \in V^1 \tag{3.10}$$

$$\sum_{e \in \delta^+(v)} x_{ek} = \sum_{e \in \delta^-(v)} x_{ek} \qquad \forall j \in \{2, 3, \ldots, m-1\}, v \in V^j, k \in [n] \tag{3.11}$$

$$\sum_{k \in [n]} \sum_{e \in \delta^-(v)} x_{ek} = 1 \qquad \forall j \in \{2, 3, \ldots, m\}, v \in V^j \tag{3.12}$$

$$\sum_{k \in [n]} \sum_{e \in \delta^+(v)} x_{ek} = 1 \qquad \forall j \in \{1, 2, \ldots, m-1\}, v \in V^j \tag{3.13}$$

$$x_{ek} \in \{0, 1\} \qquad \forall e \in E, k \in [n] \tag{3.14}$$

As before, our objective is to minimize the largest path weight, denoted by the variable D. By Constraint (3.9), we ensure that D equals the largest such weight in an optimal solution (recall that w_{k1} refers to the weight of the kth item in the first layer). We enforce that path k starts at node $(k, 1)$ using Constraint (3.10). This is for the purpose of symmetry breaking, but also allows us to formulate Constraint (3.9) using costs c_e that only depend on the node edge e points to. Constraint (3.11) is a classic network flow constraint to ensure that the x-variables model paths. Finally, Constraints (3.12) and (3.13) ensure that each node is visited by exactly one path. Note that only one of the two constraints would be required; we use both constraints to strengthen the formulation.

The advantage of this model is that it uses $O(|E|n)$ many variables instead of $O(n^2m)$ many variables as in the previous model. The number of constraints is $O(n^2m)$ as well. In general, we have $|E| \in O(n^2m)$, i.e., this model requires more variables on dense instances, but has the same asymptotic size for sparse instances, where $|E| \in O(nm)$. To the best of our knowledge such a formulation is also not investigated in the context of the PMC problem. The variables in this model can be seen as inspired from the flow formulation in Berghman et al. (2014), however, the constraint sets are different due to different problem characteristics.

Remark 3.1. *Another natural way to model MBAP is using variables built on paths under the paradigm of set covering and partitioning problems, see our earlier working paper Dokka and Goerigk (2020). Such models often suffer from too many variables even with modest graph densities and form the basis of Branch and Price exact algorithms and Price and Branch Heuristics. The formulation (3.8-3.14) can be seen as in-between the extremes of node based and path (hyper-edge) based formulations. In the same vein, it is possible to construct formulations, for example, with variables modeled on hyper-edges smaller than paths, such as triples of nodes. However, we found that such formulations become impractical and quickly lose the advantage gained by sparsity and hence omit their presentation.*

4 Computational experiments

4.1 Setup

We present two experiments to answer the question how graph density affects the hardness of the MBAP. In the first experiment, we generate a set of sparse and a set of dense random instances using different instance sizes n and m. In the second experiment, we fix the instance size and only vary the instance density. In the following, we refer to the triple-index formulation from Section 3.2 as M1, and to the edge-based formulation from Section 3.3 as M2.

We generate random instances in the following way. Given n and m, a weight matrix is generated by sampling each w_{ij} for $i \in [n]$, $j \in [m]$ uniformly randomly from $\{1, 2, \ldots, 100\}$. To create the graph structure, we first generate all horizontal arcs, i.e., arcs connecting (i, j) with $(i, j + 1)$ to ensure that a feasible solution exists. We then create tuples from the first to the last layer by choosing random nodes in each layer. Arcs along these tuples are added to the graph, if they don't already exist. For a density parameter $d \geq 0$, we create d such tuples.

Each instance is first solved heuristically using a greedy method that works analogously to the complete case considered, e.g., in Hsu (1984). The resulting solution is then used as a warmstart for M1 and M2. The greedy method has proved highly effective for the complete-MBAP, and can be directly extended to the arbitrary case. We construct a solution layer by layer, balancing the weights of the tuples in each iteration. The problem solved in each iteration corresponds to a bottleneck assignment problem, which can be solved in polynomial time.

For each model and instance, we allow a time limit of 10 minutes. We solve IPs using CPLEX version 12.8. All experiments are conducted on a virtual Ubuntu server with ten Xeon CPU E7-2850 processors at 2.00 GHz speed and 23.5 GB RAM. All processes are restricted to one thread. All our code, including our instance generator as well as further solution methods that were excluded in preliminary testing are available online on GitHub[1].

[1] https://github.com/goerigk/MLBA-code

4.2 Experiment 1

We consider all combinations of $n \in \{10, 15, 20, 25, 30\}$ and $m \in \{5, 10, 15\}$ with two density parameters. For sparse instances, we set $d = 2n$ (i.e., $|E| \in O(nm)$). For dense instances, this is increased to $d = 2n^2$ (i.e., $|E| \in O(n^2 m)$). For each combination of parameters, we generate 50 random instances and solve each using models M1 and M2. Contrasting our problem sizes with those used in the literature, our problem sizes are similar or bigger than those reported in experiments with IP formulations, see Kowalczyk and Leus (2017).

We first discuss the model size in Table 4.1 for sparse and dense instances, respectively. Column $|V|$ shows the number of nodes in the graph (being equal to nm), while column $|E|$ shows the average number of edges. In column Density we show the percentage of possible edges that are present, i.e. the value $|E|/(n^2(m-1))$. Columns Vars1 and Vars2 show the average number of variables for M1 and M2, respectively, while Cons1 and Cons2 show the number of constraints. Note that the number of variables in M1 and the number of constraints in both models do not depend on $|E|$. Recall that for sparse instances, both models have the same asymptotic size. While M2 has slightly less constraints, the number of variables is higher (roughly by factor 2 to 3 based on our choice of edge density).

			sparse		dense			sparse	dense		
n	m	$\|V\|$	$\|E\|$	Density	$\|E\|$	Density	Vars1	Vars2	Vars2	Cons1	Cons2
10	5	50	105.3	26.3%	351.2	87.8%	501	1053.6	3512.6	520	390
15	5	75	164.8	18.3%	787.4	87.5%	1126	2472.7	11812.0	1080	810
20	5	100	224.4	14.0%	1391.1	86.9%	2001	4489.8	27823.4	1840	1380
25	5	125	284.0	11.4%	2175.7	87.0%	3126	7101.5	54393.0	2800	2100
30	5	150	345.3	9.6%	3127.7	86.9%	4501	10360.6	93830.8	3960	2970
10	10	100	238.0	26.4%	791.7	88.0%	1001	2381.2	7917.6	1120	990
15	10	150	371.4	18.3%	1770.7	87.4%	2251	5572.0	26561.2	2355	2085
20	10	200	504.4	14.0%	3134.7	87.1%	4001	10088.6	62694.2	4040	3580
25	10	250	639.2	11.4%	4897.3	87.1%	6251	15982.0	122434.5	6175	5475
30	10	300	774.7	9.6%	7036.9	86.9%	9001	23242.6	211106.8	8760	7770
10	15	150	368.9	26.4%	1231.9	88.0%	1501	3690.2	12319.8	1720	1590
15	15	225	576.7	18.3%	2754.0	87.4%	3376	8651.8	41311.3	3630	3360
20	15	300	785.4	14.0%	4880.2	87.1%	6001	15709.0	97604.2	6240	5780
25	15	375	994.8	11.4%	7613.6	87.0%	9376	24870.5	190342.0	9550	8850
30	15	450	1204.4	9.6%	10961.1	87.0%	13501	36133.6	328832.8	13560	12570

Table 4.1: Average instance and model size.

This is very different for dense instances, where the number of constraints remains the same for both models, but the number of variables for M2 increases considerably.

We now consider the efficiency of both models in Tables 4.2 and 4.3. In columns Time1 and Time2 we show the average computation times of each model in seconds (bounded by 600 seconds per instance), where the average is only taken over instances that were solved

to optimality. Columns Opt1 and Opt2 show the percentage of instances out of 50 that could be solved to optimality within the time limit. Gap1 and Gap2 show the average remaining MIP gap over all instances that were not solved to optimalty (i.e., the average relative difference between best lower and upper bound). Finally, columns Greedy, Obj1 and Obj2 show the average objective value achieved by the Greedy warmstart solution, and the average upper bounds found by M1 and M2, respectively.

		sparse		dense		sparse		dense	
n	m	Time1	Time2	Time1	Time2	Opt1	Opt2	Opt1	Opt2
10	5	0.1	0.1	176.9	224.8	100%	100%	80%	48%
15	5	0.2	0.2	116.1	246.6	100%	100%	18%	12%
20	5	0.4	0.4	244.5	281.7	100%	100%	12%	12%
25	5	0.5	0.6	129.1	-	100%	100%	2%	0%
30	5	1.5	1.3	-	-	100%	100%	0%	0%
10	10	31.3	18.2	40.7	282.0	100%	100%	94%	30%
15	10	140.4	52.7	168.1	-	68%	92%	50%	0%
20	10	175.2	107.7	307.9	-	22%	64%	32%	0%
25	10	147.2	149.1	502.7	-	6%	36%	2%	0%
30	10	250.2	161.3	512.9	-	12%	38%	2%	0%
10	15	179.6	186.8	27.8	242.2	18%	40%	100%	6%
15	15	297.7	47.2	144.9	-	2%	2%	76%	0%
20	15	-	7.3	305.9	-	0%	2%	32%	0%
25	15	-	-	-	-	0%	0%	0%	0%
30	15	-	-	-	-	0%	0%	0%	0%

Table 4.2: Average solution times and instances solved to optimality

The comparison shows that M2 clearly outperforms M1 on sparse instances. While both models solve all instances to optimality for $m = 5$, these numbers drop for $m = 10$ and $m = 15$. While M2 can solve 64% of problems for $n = 20$ and $m = 10$, M1 only achieves 22%. For the largest instances, where both models can solve only up to 2% of instances to optimality, the remaining MIP gap is smaller for M2. While both models can improve the solution provided by the warmstart, the solutions found by M2 perform best, which shows that the improvement in MIP gap is at least in part because of a better upper bound.

In Figure 7 shows additional details on the distribution of the percentage of instances solved to optimality and the gap distribution for sparse instances. That is, the value in Figure 7a at a specific time on the horizontal axis indicates the amount of instances solved to optimality out of the $15 \cdot 50$ sparse instances considered for this experiment. The value for a specific gap on the horizontal axis in Figure 7c shows how many instances have reached an MIP gap that is at most this value. For a gap of 0%, this is equal to the proportion of instances solved to optimality. These plots complement the results from Tables 4.2 and 4.3, indicating the strong performance of M1 over M2.

		sparse		dense		sparse			dense		
n	m	Gap1	Gap2	Gap1	Gap2	Greedy	Obj1	Obj2	Greedy	Obj1	Obj2
10	5	-	-	0.41%	0.42%	328.3	308.3	308.3	269.5	247.3	247.6
15	5	-	-	0.51%	0.60%	336.4	317.1	317.1	275.2	253.2	253.5
20	5	-	-	0.73%	0.69%	344.2	319.4	319.4	275.7	253.2	253.1
25	5	-	-	0.95%	2.14%	355.6	328.4	328.4	276.7	253.5	256.7
30	5	-	-	1.10%	3.43%	362.7	333.1	333.1	276.3	254.6	260.7
10	10	-	-	0.20%	0.23%	599.5	530.1	530.1	525.4	502.7	503.4
15	10	3.88%	3.07%	0.21%	1.06%	619.7	545.1	544.6	525.2	503.6	508.4
20	10	7.26%	4.19%	0.25%	3.64%	632.4	554.1	546.7	527.3	506.8	525.1
25	10	9.47%	5.96%	0.37%	3.67%	642.7	571.3	557.4	525.9	508.1	525.6
30	10	12.12%	7.57%	0.52%	6.95%	645.6	587.4	567.9	525.1	507.5	525.1
10	15	1.58%	1.28%	-	0.21%	867.0	769.4	767.7	774.9	754.3	755.8
15	15	5.18%	3.09%	0.13%	1.70%	891.1	799.0	781.5	778.4	757.2	770.1
20	15	9.08%	4.50%	0.19%	2.70%	897.0	833.7	794.9	778.6	758.3	778.4
25	15	14.14%	7.32%	0.26%	11.24%	910.1	884.9	820.4	778.6	760.1	778.6
30	15	17.16%	15.26%	0.38%	87.54%	915.4	915.4	896.1	778.1	760.5	778.1

Table 4.3: Average gaps and objective values.

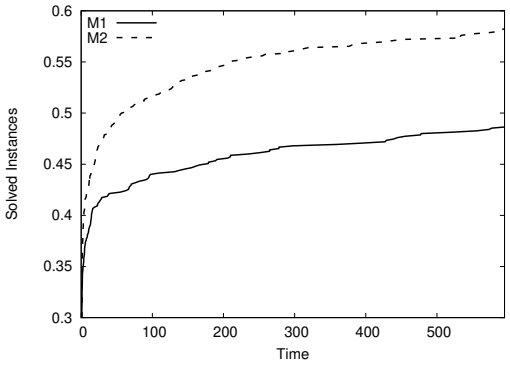
While M2 performs best for sparse instances, it is the other way around for dense instances, as can be seen in the results tables and Figures 7b and 7d. For every instance size except $m = 5$, $n = 20$, M1 solves more instances to optimality and has a lower remaining MIP gap. For the largest instances, the MIP gap of M2 is at 87.54% on average, which is due to weak lower bounds but also because the solution provided by the Greedy warm start cannot be further improved.

4.3 Experiment 2

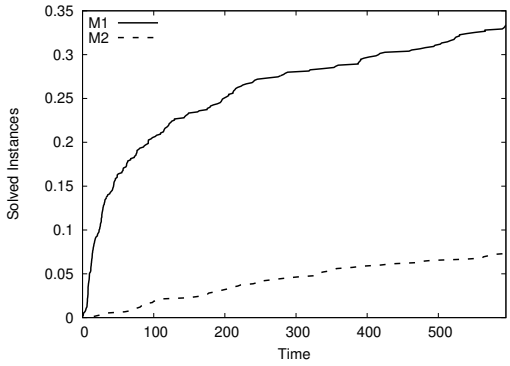
While the previous experiment already indicated that M1 performs best for dense instances and M2 performs best for sparse instances, we now consider in more detail the effect of instance density. To this end, we fix the instance size (in this case, to $n = 15$ and $m = 10$, being the smallest size for which not all sparse instances were solved to optimality in the previous experiment) and vary the density parameter from $1.0n$ to $10.0n$ using a step size of $0.5n$ and from $11n$ to $30n$ using a step size of $1n$. The average number of edges thus increases from 257.4 for $1.0n$ to 1769.3 for $30.0n$. We generate 100 instances for every density parameter.

In Figure 8, we present our results using the density on the horizontal axis. In Figure 8a, we present the average gap over all instances (including those that were solved to optimality), while Figure 8b shows the proportion of instances that have been solved to optimality. On the horizontal axis, we show the density $|E|/n^2(m-1)$, which is averaged for each density parameter d.

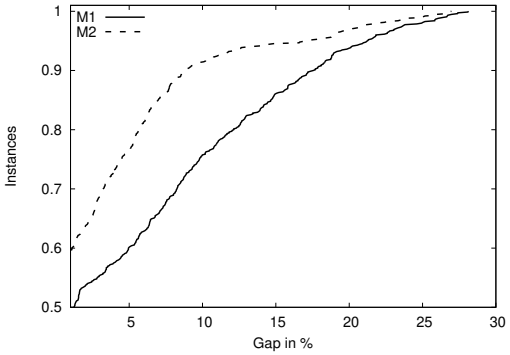
Interestingly, instance hardness does not correspond linearly to instance density. For



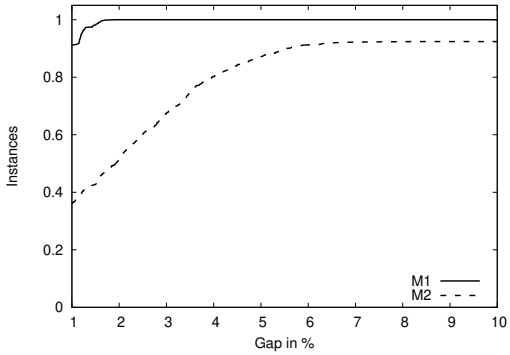
(a) Distribution of solution times for sparse instances.



(b) Distribution solution times for dense instances.



(c) Distribution of gaps for sparse instances.



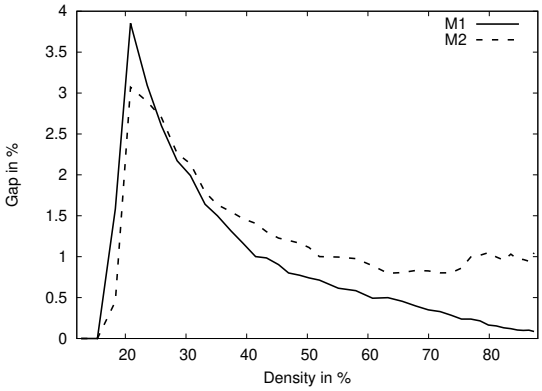
(d) Distribution gaps for dense instances.

Figure 7: Performance plots for Experiment 1.

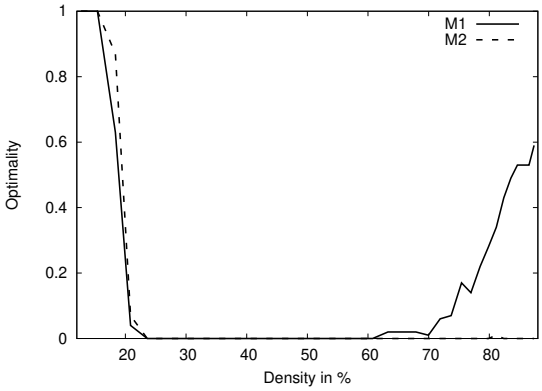
small density values, there are only few feasible solutions, and both models can solve all instances to optimality. As the density increases, instances allow more feasible solutions and cannot be solved to optimality anymore; at the same time, the reduction in MIP gap indicates that instances become simpler with higher density. From a density of around 70% upwards, M1 can solve some instances to optimality again, while M2 fails to do so.

There is a peak in MIP gap for density values around 20%. The results show that for those hardest instances, M2 performs best (with a smaller MIP gap and a larger proportion of instances solved to optimality). For higher density values (around 30% and upwards), M1 becomes the better model choice.

Recall that our theoretical analysis indicates that sparse instances are harder to solve than dense instances. Overall, our experiments show differences in complexity depending on the density of instances. For larger instances with $m = 15$, more sparse instances remain unsolved within the time limit than dense instances. For such hardest instances, model M2 is the best performing approach, while it does not perform well for dense instances.



(a) Average gap for different density values.



(b) Instances solved to optimality for different density values.

Figure 8: Performance plots for Experiment 2.

5 Conclusions

We considered the multi-level bottleneck assignment problem, which has attracted considerable attention in application areas such as finance and scheduling. While previous models allowed that any element in one set can be paired with any other element in the next set, we generalized this setting such that any bipartite graph can represent the feasible pairings between two consecutive columns. We analyzed the complexity of this problem, showing that it is not approximable better than $(\lfloor \frac{m}{3} \rfloor + 1)$ in polynomial time.

We presented two mixed-integer programming models for the MBAP, whose sizes scale differently in the number of edges of the graph underlying the instance. In computational experiments we applied these models to better understand the impact on graph density on problem hardness. We found that dense instances result in smaller IP gaps overall.

Many opportunities for future research arise. With regards to the approximability results, we have presented a lower bound, i.e., no algorithm can achieve an approximation guarantee better than $\lfloor m/3 \rfloor + 1$ unless $P = NP$ for the MBAP. This means that in particular the greedy heuristic is not a 2-approximation in this case. It remains open whether there exists an approximation algorithm that matches our lower bound.

Within the scheduling context our work indicates more can be done algorithmically in solving MBAP and hence also the PMC problem. However, this should specifically use the underlying (sparsity) structure. Our results show that alternative tailored formulations can lead to more promising results. As a further study it would be interesting to explore similar ideas in the PMC context by considering special structures on conflict graphs.

We believe our approach on different classes of IPs for different graph sparsity ranges is an exciting idea and opens up similar questions for many other problems, such as classical problems with conflicts, studied only recently such as maximum flow and linear assignment with conflicts (see Şuvak et al. (2020)).

References

Berghman, L., Leus, R., and Spieksma, F. (2014). Optimal solutions for a dock assignment problem with trailer transportation. *Annals of Operations Research*, 213:3–25.

Bernard, C., Bondarenko, O., and Vanduffel, S. (2018a). Rearrangement algorithm and maximum entropy. *Annals of Operations Research*, 261:107–134.

Bernard, C., Bondarenko, O., and Vanduffel, S. (2021). A model-free approach to multivariate option pricing. *Review of Derivatives Research*, 24(2):135–155.

Bernard, C., Denuit, M., and Vanduffel, S. (2018b). Measuring portfolio risk under partial dependence information. *The Journal of Risk and Insurance*, 85(3):843–863.

Bernard, C., Ruschendorf, L., and Vanduffel, S. (2017). Value-at-risk bounds with variance constraints. *Journal of Risk and Insurance*, 84(3):923–959.

Bodlaender, H., Jansen, K., and Woeginger, G. (1994). Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55:219–232.

Boudt, K., Vanduffel, S., and Verbeken, K. (2017). Block rearranging elements within matrix columns to minimize the variability of the row sums. *4OR*, 16:31–50.

Burkard, R., Dell'Amico, M., and Martello, S. (2009). *Assignment Problems*. SIAM, Philadelphia.

Çalık, H., Labbé, M., and Yaman, H. (2019). p-center problems. In *Location science*, pages 51–65. Springer.

Carraresi, P. and Gallo, G. (1984). A multi-level bottleneck assignment approach to the bus drivers rostering problem. *European Journal of Operational Research*, 16:163–173.

Coffman Jr, E. and Sethi, R. (1976). Algorithms minimizing mean flow time: schedule-length properties. *Acta Informatica*, 6:1–14.

Coffman Jr, E. and Yannakakis, M. (1984). Permuting elements within columns of a matrix in order to minimize maximum row sum. *Mathematics of Operations Research*, 9:384–390.

Das, S. and Wiese, A. (2017). On minimizing the makespan when some jobs cannot be assigned on the same machine. *In Proceedings of the 25th Annual European Symposium on Algorithms*, pages 31:1–31:14.

Dokka, T. and Goerigk, M. (2020). The multi-level bottleneck assignment problem: complexity and solution methods. *ArXiv:1910.12504*.

Dokka, T., Kouvela, A., and Spieksma, F. (2012). Approximating the multi-level bottleneck assignment problem. *Operations Research Letters*, 40:282–286.

Eck, B. and Pinedo, M. (1993). On the minimization of the makespan subject to flowtime optimality. *Operations Research*, 41(4):797–801.

Goossens, D., Polyakovskiy, S., Spieksma, F., and Woeginger, G. (2010). The approximability of three-dimensional assignment problems with bottleneck objective. *Optimization Letters*, 4:4–17.

Haus, U. (2015). Bounding stochastic dependence, joint mixability of matrices,and multidimensional bottleneck assignment problems. *Operations Research Letters*, 43:74–79.

Hsu, W. (1984). Approximation algorithms for the assembly line balancing crew scheduling problem. *Mathematics of Operations Research*, 9:376–383.

Klinz, B. and Woeginger, G. (1996). A new efficiently solvable case of the three-dimensional axial bottleneck assignment problem. *Lecture Notes in Computer Science*, 1120:150–162.

Kowalczyk, D. and Leus, R. (2017). An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20:355–372.

Page, D. and Solis-Oba, R. (2018). Makespan minimization on unrelated parallel machines with a few bags. *Lecture Notes in Computer Science, International Conference on Algorithmic Applications in Management*, 11343:24–35.

Puccetti, G. and Ruschendorf, L. (2012). Computation of sharp bounds on the distribution of a function of dependent risks. *Journal of Computational and Applied Mathematics*, 236(7):1833–1840.

Ravi, P., Tuncel, L., and Huang, M. (2016). Worst case performance analysis of some approximation algorithms for minimizing makespan and flow-time. *Journal of Scheduling*, 19(5):547–561.

ROADEF (2011). Google ROADEF/EURO challenge 2012: Machine reassignment. *Available online at: http://challenge.roadef.org/2012/files/ problem_definition_v1.pdf.*

Şuvak, Z., Altınel, İ. K., and Aras, N. (2020). Exact solution algorithms for the maximum flow problem with additional conflict constraints. *European Journal of Operational Research*, 287(2):410–437.

Wang, B. and Wang, R. (2011). The complete mixability and convex minimization problems with monotone marginal densities. *Journal of Multivariate Analysis*, 102(10):1344–1360.

Wang, B. and Wang, R. (2016). Joint mixability. *Mathematics of Operations Research*, 41:808–826.