Approximation Algorithms for Min-Max Generalization Problems

Piotr Berman and Sofya Raskhodnikova*

Pennsylvania State University
{berman, sofya}@cse.psu.edu

Abstract. We provide improved approximation algorithms for the *min-max generalization problems* considered by Du, Eppstein, Goodrich, and Lueker [1]. In min-max generalization problems, the input consists of data items with weights and a lower bound w_{lb}, and the goal is to partition individual items into groups of weight at least w_{lb}, while minimizing the maximum weight of a group. The rules of legal partitioning are specific to a problem. Du *et al.* consider several problems in this vein: (1) partitioning a graph into connected subgraphs, (2) partitioning unstructured data into arbitrary classes and (3) partitioning a 2-dimensional array into non-overlapping contiguous rectangles (subarrays) that satisfy the above size requirements.

We significantly improve approximation ratios for all the problems considered by Du *et al.*, and provide additional motivation for these problems. Moreover, for the first problem, while Du *et al.* give approximation algorithms for specific graph families, namely, 3-connected and 4-connected planar graphs, no approximation algorithm that works for all graphs was known prior to this work.

1 Introduction

We provide improved approximation algorithms for the *min-max generalization problems* considered by Du, Eppstein, Goodrich, and Lueker [1]. In min-max generalization problems, the input consists of data items with weights and a lower bound w_{lb}, and the goal is to partition individual items into groups of weight at least w_{lb}, while minimizing the maximum weight of a group. The rules of legal partitioning are specific to a problem. Du *et al.* consider several problems in this vein: (1) partitioning a graph into connected subgraphs, (2) partitioning unstructured data into arbitrary classes and (3) partitioning a 2-dimensional array into non-overlapping contiguous rectangles (subarrays) that satisfy the above size requirements. We call these problems (1) Min-Max Graph Partition, (2) Min-Max Bin Covering and (3) Min-Max Rectangle Tiling.

Du *et al.* motivate the min-max generalization problems by applications to privacy-preserving data mining. Generalization is widely used in the data mining

* S.R. was supported by National Science Foundation (NSF/CCF award 0729171 and NSF/CCF CAREER award 0845701).

community as means for achieving k-anonymity (see [2] for a survey). Generalization involves replacing a value with a less specific value. To achieve k-anonymity each record should be generalized to the same value as at least $k - 1$ other records. For example, if the records contain geographic information (e.g., GPS coordinates), and the plane is partitioned into axis-parallel rectangles each containing locations of at least k records, to achieve k-anonymity, the coordinates of each record can be replaced with the corresponding rectangle. Generalization can also be viewed as a natural way of compressing a dataset.

We briefly discuss several other applications of generalization. Geographic Information Systems contain very large data sets that are organized either according to the (almost) planar graph of the road network, or according to geographic coordinates (see, e.g., [3]). These sets have to be partitioned into *pages* that can be transmitted to a mobile device or retrieved from secondary storage. Because of the high overhead of a single transmission/retrieval operation, we want to assure a minimum size of a single part (page), while controlling the maximum size. When the process that is exploring a graph needs to investigate a node whose information it has not retrieved yet, it has to request a new page. Therefore, pages are more useful if they contain information about connected subgraphs. Min-Max Graph Partition captures the problem of distributing information about the graph among pages.

Min-Max Bin Covering is a variant of the classical Bin Covering problem. In the classical version, the input is a set of items with positive weights and the goal is to pack items into bins, so that the number of bins that receive items of total weight at least 1 is maximized (see [4–6] and references therein). Both variants are natural. For example, when Grandfather Frost[1] partitions presents into bundles for kids, he clearly wants to ensure that each bundle has items of at least a certain value to make kids happy. Grandfather Frost could try to minimize the value of the maximum bundle, to avoid jealousy (Min-Max Bin Covering), or to maximize the number of kids who get presents (classical Bin Covering). Min-Max Bin Covering can also be viewed as a variant of scheduling on parallel identical machines where, given n jobs and their processing times, the goal is to schedule them on m identical parallel machines while minimizing *makespan*, that is, the maximum time used by any machine [8]. In our variant, the number of machines is not given in advance, but instead, there is a lower bound on the processing time. This requirement is natural, e.g., when "machines" represent workers that must be hired for at least a certain number of hours.

Rectangle tiling problems with various optimization criteria arise in applications ranging from databases and data mining to video compression and manufacturing, and have been extensively studied [9–16]. The min-max version can be used to design a Geographic Information System, described above. If the data is a set of coordinates specifying object positions, as opposed to a road network, we would like to partition it into pages that correspond to rectangles on the

[1] Grandfather Frost is a secular character that played the role of Santa Claus for Soviet children. The Santa Claus problem [7] is not directly related to our problem.

plane. As before, we would like to ensure that pages have at least the minimum size while controlling the maximum size.

1.1 Problems

In each of the problems we consider, the input is an item set $\mathcal{I}$, non-negative weights w_i for all $i \in \mathcal{I}$ and a non-negative bound w_{lb}. For $\mathcal{I}' \subseteq \mathcal{I}$, we use $w(\mathcal{I}')$ to denote $\sum_{i \in \mathcal{I}'} w_i$. Each problem below specifies a class of *allowed* subsets of $\mathcal{I}$. A valid solution is a partition P of $\mathcal{I}$ into allowed subsets such that $w(\mathcal{I}') \geq w_{\text{lb}}$ for each $\mathcal{I}' \in P$. The goal is to minimize the cost of P, defined as $\max_{\mathcal{I}' \in P} w(\mathcal{I}')$.

In *Min-Max Graph Partition*, $\mathcal{I}$ is the vertex set V of an (undirected) graph (V, E), and a subset of V is allowed if it induces a connected subgraph. In *Min-Max Bin Covering*, every subset of $\mathcal{I}$ is allowed. A partition of $\mathcal{I}$ is called a *packing*, and the parts of a partition are called *bins*. In *Min-Max Rectangle Tiling*, $\mathcal{I} = \{1, \ldots, m\} \times \{1, \ldots, n\}$, and the allowed sets are rectangles, i.e., sets of the form $\{a, \ldots, b\} \times \{c, \ldots, d\}$. A partition of $\mathcal{I}$ is called a *tiling*, and the parts of a partition are called *tiles*.

All three min-max problems above are NP-complete. Moreover, if P≠NP no polynomial time algorithm can achieve an approximation ratio better than 2 for Bin Covering (and hence for Graph Partition) or better than 1.33 for Graph Partition on 3-connected planar graphs and Rectangle Tiling [1].

1.2 Our Results and Techniques

Our main technical contribution is a 3-approximation algorithm for Min-Max Graph Partition. The remaining algorithms are very simple, even though the analysis is non-trivial.

Min-Max Graph Partition. We present the first polynomial time approximation algorithm for Min-Max Graph Partition. Du *et al.* gave approximation algorithms for specific graph families, *i.e.*, a 4-approximation for 3-connected and

Min-Max Problem	Hardness [1]	Ratio in [1]	Our ratio
Graph Partition	2	—	3
on 3-connected planar graphs	1.33	4	
on 4-connected planar graphs	—	3	2.5
Bin Covering	2	$2 + \varepsilon$ in time exp in ε^{-1}	2
Rectangle Tiling	1.33	5	4
with 0-1 entries	—	—	3

Table 1. Approximation Ratios for Min-Max Generalization Problems. (Note: Graph Partition generalizes Bin Covering, and hence inherits its inapproximability.)

a 3-approximation for 4-connected planar graphs. We give a 3-approximation algorithm for the general case, simultaneously improving the approximation ratio and applicability of the algorithm. We also improve the approximation ratio for 4-connected planar graphs from 3 to 2.5.

Our 3-approximation algorithm for Min-Max Graph Partition constructs a 2-tier partition where nodes are partitioned into *groups*, and groups are partitioned into *supergroups*. Intuitively, supergroups represent parts in a legal partition, while groups represent (nearly) indivisible subparts. The initial 2-tier partition is obtained greedily and then transformed using 4 carefully designed transformations until all supergroups of large weight have well-defined central nodes, and almost all non-central nodes in those supergroups are only connected to central nodes (possibly of multiple supergroups). Supergroups of small weight are used as parts in the final solution. The remaining supergroups are more tricky to deal with. We create one part in the final solution for each supergroup or, more precisely, for each group with a central node. We redistribute other groups among supergroups using a scheduling algorithm of Lenstra, Shmoys and Tardos [17], while leaving all central nodes in separate parts. Roughly, central nodes play a role of the machines and the groups that we need to redistribute play a role of jobs to be scheduled on these machines. The final part of the algorithm repairs parts of insufficient weight to obtain the final partition.

Our use of the scheduling algorithm of Lenstra *et al.* is *gray-box* in the following sense: our algorithm runs the scheduling algorithm in a black-box manner. However, in the analysis, we look inside the black box. Namely, we apply the Rounding Theorem of Lenstra *et al.* to show that the LP used by their algorithm yields a good solution for our problem.

For partitioning 4-connected planar graphs, following Du *et al.*, we use the fact that such graphs have Hamiltonian cycles [18] which can be found in linear time [19]. Our algorithm is simple and efficient: It goes around the Hamiltonian cycle and greedily partitions the nodes, starting from the lightest contiguous part of the cycle that satisfies the weight lower bound. If the last part is too light, it is combined with the first part. Thus, the algorithm runs in linear time. Our algorithm and analysis apply to any graph that contains a Hamiltonian cycle which can be computed efficiently or is given as part of the input.

Min-Max Bin Covering. We present a simple 2-approximation algorithm that runs in linear time. Du *et al.* gave a schema with approximation ratio $2+\varepsilon$, and time complexity exponential in ε^{-1}. They also showed that approximation ratio better than 2 cannot be achieved in polynomial time unless P=NP. Thus, we completely resolve the approximability of this problem.

Our algorithm greedily places items in the bins in the order of decreasing weights, and then redistributes items in the first and the last three bins.

Min-Max Rectangle Tiling. We improve the approximation ratio for this problem from 5 to 4. We can get a better ratio of 3 when the entries in the matrix are restricted to be 0 or 1. This case covers the scenarios where each entry indicates the presence or absence of some object, as in applications with geographic data, such as GPS coordinate data originally considered by Du *et al.*

Our algorithm builds on the *slicing and dicing* method introduced by Berman *et al.* [15]. The idea is to first partition the rectangle horizontally into *slices*, and then partition slices vertically. The straightforward application of *slicing and dicing* gives ratio 5. We improve it by doing simple preprocessing. For the case of 0-1 entries, the preprocessing step is more sophisticated.

Summary and Organization. We summarize our results in Table. 1. The results on Graph Partition are stated in Theorems 2.1 and 2.2 in Sect. 2, on Bin Covering, in Theorem 3.1 in Sect. 3, and on Rectangle Tiling, in Theorems 4 and 4.2 in Sect. 4. All omitted proofs are deferred to the full version.

Terminology and Notation. Here we describe terminology and notation common to all technical sections. We use *opt* as the cost of an optimal solution.

Definition 1.1. *An item (or a set of items) is* fat *if it has weight at least* w_{lb}*, and* lean *otherwise. We apply this terminology to nodes and sets of nodes in an instance of Graph Partition, and to elements and rectangles in Rectangle Tiling.*

A solution is *legal* if it obeys the *minimum weight constraint*, i.e., all parts are fat.

2 Min-Max Graph Partition

We present two approximation algorithms for Min-Max Graph Partition whose performance is summarized in Theorems 2.1 and 2.2.

Theorem 2.1. *Min-Max Graph Partition can be approximated with ratio 3 in polynomial time.*

Theorem 2.2. *Min-Max Graph Partition on 4-connected planar graphs can be approximated with ratio* 2.5 *in linear time.* (The proof is omitted.)

The rest of this section is devoted to the proof of Theorem 2.1.

Recall that an input to Min-Max Graph Partition is a graph (V, E) with node weights $w : V \to \mathbb{R}^+$ and a weight lower bound w_{lb}. W.l.o.g. assume that $w_{\text{lb}} = 1$. (All weights can be divided by w_{lb} to obtain an equivalent instance with $w_{\text{lb}} = 1$.) For now, we will also assume that all nodes in the graph are *lean.* (Recall Definition 1.1 of *fat* and *lean.*) We remove this assumption in Sect. 2.4.

As described in Sect. 1.2, our algorithm first constructs a 2-tier partition into groups and supergroups (Sect. 2.1), then transforms it until all supergroups of large weight have well-defined central nodes, and nearly all non-central nodes in those supergroups are only connected to central nodes (Sect. 2.2) and finally solves an instance of Scheduling on Unrelated Parallel Machines (Sect. 2.5), interprets it as a partition and adjusts it to get the final solution.

2.1 A Preliminary 2-Tier Partition

We start by defining a 2-tier partition. (See illustration in Fig. 1.)

Definition 2.1 (2-tier partition). *A 2-tier partition of a graph (V, E, w) containing only lean nodes is a partition of V into lean sets, called* groups*, together with a partition of the groups into fat sets, called* supergroups*. The set of nodes in a group, or in a supergroup, should induce a connected graph. The set of groups contained in a supergroup S is denoted by $\mathcal{G}(S)$.*

Since groups are lean and supergroups are fat, each supergroup contains at least two groups. We assign names to some types of groups and supergroups.

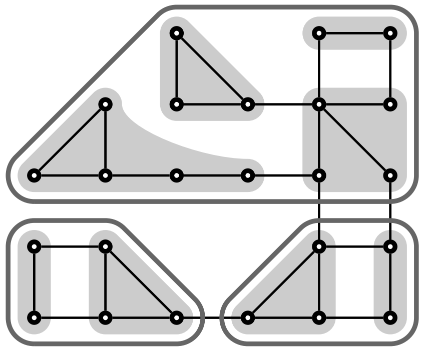


Fig. 1. An example of a 2-tier partition. Shaded background indicates groups; curved lines indicate supergroups. The top supergroup has a 4-node central group and 3 mobile groups. The two bottom supergroups are group-pairs.

Definition 2.2. Group-pair, triangle, star supergroups; central group

- *A supergroup is a* group-pair *if it consists of two groups.*
- *A supergroup is a* triangle *if it consists of three groups, pairwise connected by an edge.*
- *A supergroup S with 3 or more groups is a* star *if it forms a star graph on groups, i.e., it contains a group G, called* central*, such that groups in $\mathcal{G}(S) - \{G\}$ form connected components of $S - G$.*

Lemma 2.1 (Initial partition). *Given a connected graph on lean nodes, in polynomial time we can compute a 2-tier partition where*
(a) *each supergroup is a group-pair, a triangle or a star and*
(b) $w(G) + w(H) \geq 1$ *for all adjacent groups G and H.*

Proof. First, form the groups greedily: Make each node a group. While there are two groups G, H such that $G \cup H$ is lean and connected, merge G and H.

Second, form group-pairs greedily: While there are two adjacent groups G, H that are not included in a supergroup, form a supergroup $G \cup H$.

Next, insert remaining groups into supergroups: For each group G still not included in a supergroup, pick an adjacent group H. Since the second step halted, H is in some group-pair created in that step. Insert G into H's supergroup.

Finally, break down supergroups that are not stars: Consider a group-pair P created in the second step from groups G and H, and let S be the supergroup that was formed from P. Suppose S has 4 or more groups, but is not a star. Since groups in $S - P$ are not connected, and neither G nor H can become the center of S, there are two different groups G' and H' in S that are adjacent to G and H, respectively. Let S_1 be the the union of G, G' and all other groups in S that are not adjacent to H. Replace S with S_1 and $S - S_1$. In the resulting 2-tier partition, all supergroups with 4 or more groups are stars, so item (a) of the lemma holds. Item (b) is guaranteed by the first step of the construction. $\square$

2.2 Improving the Initial 2-Tier Partition

In this section, we modify the initial 2-tier partition, while maintaining property (a) and a weaker version of property (b) of Lemma 2.1. As we are working on our 2-tier partition, we will rearrange groups and supergroups. A group G is called *mobile* if it can be removed from its supergroup S while keeping property (a) of Lemma 2.1. Namely, the modified S has to be a group-pair or a star.

Definition 2.3 (Mobile group). *A group is* mobile *if it is not in a group-pair and it is not a central group.*

The goal of this phase of the algorithm is to separate supergroups into two *types*: (i) the ones that will be repartitioned by the scheduling algorithm and (ii) the ones that will be used in the final partition as they are. Supergroups of *type* (i) will be well structured: in such a supergroup, the central group will have a unique *central* node, and mobile groups will be connected only to central nodes (possibly in multiple central groups). Supergroups of *type* (ii) will have at most 3 groups, and thus weight at most 3— sufficiently light to form parts in a 3-approximate solution. Central groups of supergroups of *type* (i) will be allocated their own parts in the final partition. Mobile groups will be distributed among these parts by the scheduling algorithm. To guarantee that the optimal distribution of central nodes and mobile groups into parts provides a sufficiently good solution, we require that mobile groups are connected only to central nodes of the supergroups of *type* (i). (Non-central nodes of central groups will join the parts of their central nodes after the scheduling algorithm produces a 2-approximate solution. Since, by definition, each group is lean, even after adding central groups, we will still be able to guarantee a 3-approximation.)

We explain this phase of the algorithm by specifying several transformations of a 2-tier partition (see Figs. 2 and 3). The algorithm applies these transformations to the initial 2-tier partition from Lemma 2.1. Each transformation is defined by the *trigger* and the *action*. The algorithm performs the action for the first transformation for which the trigger condition is satisfied for some group(s) in the current 2-tier partition. This phase terminates when no transformation can be applied.

The purpose of the first transformation, **CombG**, is to ensure that $w(G) + w(H) \geq 1$ for all adjacent groups G and H, where one of the groups is mobile. Even though an even stronger condition, property (b) of Lemma 2.1, holds for the initial 2-tier partition, it might be violated by other transformations. The second transformation, **ConP**, is getting rid of edges between mobile group. The third transformation, **SplitC**, is ensuring that each central group has a unique central node to which mobile groups connect. To accomplish this, while there is a central group G that violates this condition, **SplitC** splits G into two parts, each containing a node to which mobile groups connect. Later, it rearranges resulting groups and supergroups to ensure that all previously achieved properties of our 2-tier partition are preserved (in some cases, relying on **CombG** and **ConP** to reinstate these properties).

Fig. 2. Transformations. (Perform the first one that applies.)

- **CombG = Combine groups.**

Trigger: Groups G and H are connected by an edge, $G \cup H$ is lean and H is mobile.

Action: Remove H from its supergroup and merge the two groups.

- **ConP = Connect group-pairs.**

Trigger: Two mobile groups are connected with an edge, and they belong either to two different supergroups or to a supergroup with more than three groups.

Action: Remove them from their supergroup(s) and combine them into a group-pair.

- **SplitC = Split the center.**

Trigger: G is the central group of a supergroup S, u and v are two different nodes in G, and two mobile groups H_u, H_v (not necessarily from $\mathcal{G}(S)$) have edges to u and v, respectively.

Action: Split G into two connected sets, G_u and G_v, containing u and v, respectively. Split S into S_u and S_v, by attaching each non-central group to G_u or G_v. If $H_u \in \mathcal{G}(S)$ attach H_u to G_u. Similarly, if $H_v \in \mathcal{G}(S)$ attach H_v to G_v.

[LeanLean case]: If both S_u and S_v are lean, we make them groups, and S becomes a group-pair.

[FatFat case]: If both S_u and S_v are fat, they become new supergroups.

Now assume that S_u is fat and S_v is lean.

[FatLean-IN case]: If $H_v \in \mathcal{G}(S)$ then change the partition of S by replacing G and H_v with G_u and S_v. If S is not a star, but has 4 or more groups, apply **CombG** or **ConP**.

[FatLean-OUT case]: If $H_v \notin \mathcal{G}(S)$ then remove S_v from G and S and treat it like a mobile group in contact with H_v, which triggers **CombG** or **ConP**.

- **ChainR = Chain Reconnect.**

Trigger: An unstructured supergroup S has 4 or more groups.

Action: Since **CombG**, **ConP** cannot be applied, we have a chain of supergroups $S = S_1, \ldots, S_k$ where S_k is a group-pair, a mobile group of S_{k-1} is adjacent to S_k and for $i = 1, \ldots, k-2$ a mobile group of S_i is adjacent to the central group of S_{i+1}. Then for $i = 1, \ldots, k-1$, move a mobile group from S_i to S_{i+1}.

If the previously described transformations cannot be applied, star supergroups in the current 2-tier partition are well structured: they have unique central nodes, and all mobile groups connect only to these central nodes, with one exception—they could still connect to group-pairs. Group-pairs are not guaran-

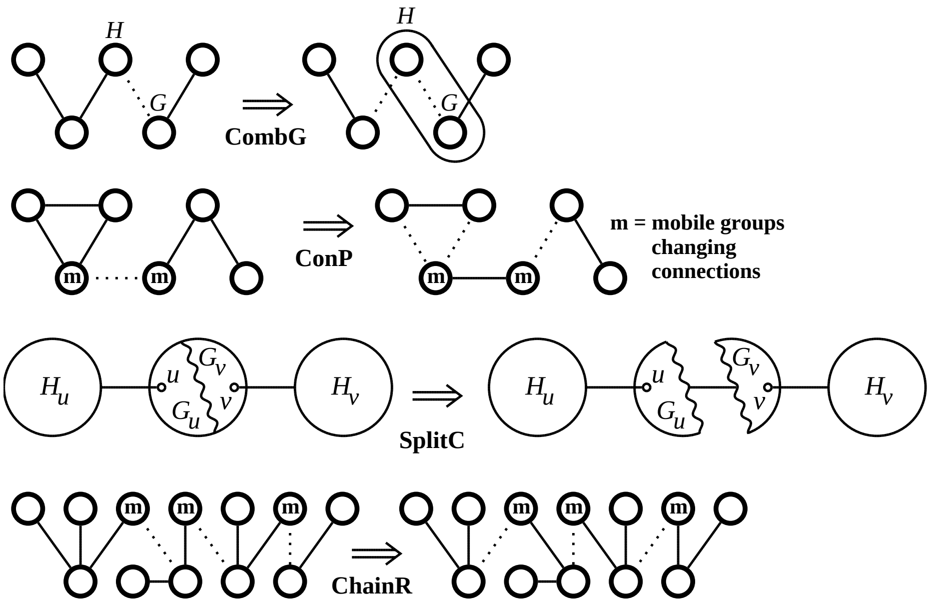


Fig. 3. Transformations that improve the initial partition. Solid lines connect groups of a supergroup, circles indicate groups, unless they are within ovals—then ovals are groups and circles are fragments of groups. **SplitC** transformation has four cases: all split a central group G into two parts and combine them with groups H_u and H_v to form new groups or supergroups (depending on the weight of the resulting pieces).

teed to have any structure. But they are light enough to be used as parts in the final partition. The same applies to triangles and stars with 3 groups. All group-pairs and triangles will be used as parts in the final partition, and thus will be of *type* (ii), according to the description after Definition 2.3. Stars with 3 groups could be of *type* (i) or (ii). As we already explained, it is important for the success of the next (scheduling) phase of the algorithm that mobile groups of supergroups of *type* (i) are adjacent only to the central nodes of supergroups of *type* (i). Next, we define *structured* and *unstructured* supergroups. After this phase completes, *structured* supergroups of the resulting 2-tier partition will be assigned *type* (i) and *unstructured* supergroups will be assigned *type* (ii). We call all group-pairs *unstructured*. Each star whose mobile group is adjacent to an unstructured supergroup is not ready to become a group of *type* (i) and is also called *unstructured*.

Definition 2.4 (Structured and unstructured supergroups). *An* unstructured *supergroup is either a group-pair or (recursively) a star that has a mobile group adjacent to an unstructured supergroup. A* structured *supergroup is a star that is not* unstructured.

The purpose of **ChainR** is to ensure that each remaining unstructured supergroup has at most 3 groups. **ChainR** is triggered if there is an unstructured supergroup S with 4 or more groups. This can happen only if S is connected by a chain of unstructured supergroups to a group-pair. The mobile nodes along this chain are reconnected, as explained in Fig. 2 and illustrated in Fig. 3. This completes the description of transformations and this phase of the algorithm.

2.3 Analysis of Transformations

We analyze the properties of a 2-tier partition to which our transformations cannot be applied in Lemma 2.2 and bound the running time of this stage of the algorithm in Lemma 2.3. (The proofs of these lemmas are omitted.)

Lemma 2.2. *When transformations* **CombG**, **ConP**, **SplitC** *and* **ChainR** *cannot be applied, the resulting 2-tier partition satisfies the following:*

a. *If G is a center group and H is a mobile group of the same supergroup then $w(G) + w(H) \geq 1$.*
b. *No edges exist between mobile groups except for groups in the same triangle.*
c. *Each supergroup S with a central group G also has a central node $c(S)$ such that all edges between G and mobile groups include node $c(S)$.*
d. *Each supergroup with 4 or more groups is structured.*

Lemma 2.3. *An algorithm performing transformations defined in Fig. 2 on a 2-tier partition until none of them are applicable runs in polynomial time.*

2.4 A 2-Tier Partition on Graphs with Arbitrary Weights

In this section we remove the assumption that all nodes in our input graph are lean. To obtain a 2-tier partition of a graph with arbitrary node weights, first allocate a separate supergroup for each fat node. Let V_{lean} be the set of lean nodes. Form *isolated groups* from lean connected components of V_{lean}. For fat connected components of V_{lean}, compute the 2-tier partition using the method from Sections 2.1 and 2.2.

The next lemma summarizes the main outcome of improving the 2-tier partition using transformations in Fig. 2. It follows directly from Lemma 2.2.

Lemma 2.4 (Main). *Consider a 2-tier partition of a graph $G = (V, E, w)$ obtained by our method. Let C be the set consisting of fat nodes and central nodes of structured supergroups in that 2-tier partition. Then mobile groups of structured supergroups are connected components of $V - C$.*

Proof. By definition, each group is connected. It remains to show that a node in a mobile group cannot be adjacent to nodes of $V - C$ which are in different groups. Recall that all groups are either central, mobile or in a group-pair. A node in a mobile group cannot be adjacent to a node in a different mobile group by Lemma 2.2(b). It cannot be adjacent to a non-central node in a central group by Lemma 2.2(c). Finally, it cannot be adjacent to a node in a group-pair by Definition 2.4 and Lemma 2.2(d). □

2.5 Reduction to Scheduling and the Final Partition

We reduce Min-Max Graph Partition to Scheduling Unrelated Parallel Machines (SUPM), and use a 2-approximation algorithm of Lenstra *et al.* for SUPM to get a 3-approximation for graph partition.

The number of parts in the final partition will be equal to the number of supergroups in the 2-tier partition of Sect. 2.4. We use all unstructured supergroups and triangles as parts in the final partition. By Lemma 2.2(d), the weight of these supergroups is below 3. We use central groups of structured supergroups and fat nodes as *seeds* of the remaining parts, that is, in the final partition, we create a part for each central group and each fat node, and partition the remaining groups among these parts using a reduction to SUMP.

Now we explain our reduction. In SUPM, the input is m parallel machines, n jobs and processing times p_{ji} of job j on machine i. For each job j, we can also specify a set $M(j)$ of machines on which it can be scheduled. (This is equivalent to setting p_{ji} to infinity for $i \notin M(j)$). The starting point of the reduction is the 2-tier partition from Sect. 2.4. We create a machine for every node in C, where C is the set consisting of fat nodes and central nodes of structured supergroups, as defined in Lemma 2.4. We create a job for every node in C, and for every mobile and isolated group. To simplify the notation, we identify the names of the machines and jobs with the names of the corresponding nodes and groups. A job corresponding to a node i in C can be scheduled only on machine i, that is, $M(i) = \{i\}$, and we set $p_{ii} = w(i)$. A job corresponding to a mobile or isolated group j can be scheduled on machine m_i iff group j is connected to C-node i. This defines $M(j)$. We set $p_{ji} = w(j)$.

We run the algorithm of [17] for SUPM on the instance defined above. The solution returned by the algorithm is interpreted as a partition of the nodes of the original graph as follows. If job j is scheduled on machine i then node (group) j is assigned to part i of the partition. Each central group is assigned to the same part as the central node of the group.

The final part of the algorithm repairs lean parts in the resulting partition. While there is a lean part P in the partition, reassign a group as follows. Let S be the supergroup in the 2-tier partition whose center was a seed for P. (A lean part cannot have a fat node as a seed.) Let C be the central group of S. Then, by construction, P contains C. Remove a mobile group of S, say H, from its current part and insert it into P. Now, by Lemma 2.2(a), $w(P) \geq w(C) + w(H) \geq 1$ because P contains C and H.

This repair process will terminate because each part is repaired at most once. Since we repair P using a mobile group from the supergroup corresponding to P (that is, the supergroup from the 2-tier partition whose center is C), the future repairs of other parts will not remove H from part P. Later, even if P looses a mobile group when we repair some other part P', the weight of P will still satisfy: $w(P) \geq w(C) + w(H) \geq 1$. Thus, after a number of steps which is at most the number of parts, all parts will be fat.

Theorem 2.1 follows from the following lemma whose proof is omitted.

Lemma 2.5. *The final partition returned by the algorithm above has parts of weight at most* $opt + 2$.

3 Min-Max Bin Covering

In this section, we present our algorithm for Min-Max Bin Covering.

Theorem 3.1. *Min-Max Bin Covering can be approximated with ratio* 2 *in time* $O(n)$.

Proof. W.l.o.g. assume that $w_{\mathrm{lb}} = 1$, $\mathcal{I} = \{1, \ldots, n\}$ and $w_1 \geq w_2 \geq \ldots \geq w_n$. We also assume that $w_i < 1$ for all items i, since items of larger weight can be placed in their own bins without affecting the quality of the solution. (Each such bin has weight at least 1 and at most *opt*.)

If $w(\mathcal{I}) < 3$, a legal packing consists of ≤ 2 bins. Therefore, $opt \geq w(\mathcal{I})/2$. Thus, $w(\mathcal{I}) \leq 2opt$, and we get a 2-approximation by returning one bin $B_1 = \mathcal{I}$. Theorem 3.1 follows from Lemma 3.1, dealing with instances with $w(\mathcal{I}) \geq 3$. □

Lemma 3.1. *Given a Min-Max Bin Covering instance* $\mathcal{I}$ *with* n *items and* $w(\mathcal{I}) \geq 3$, *a solution with cost at most* $opt + 1$ *can be found in time* $O(n)$.

Proof. We compute a preliminary packing greedily, filling successive bins with items in order (of decreasing weights), and moving to a new bin when the weight of the current bin reaches or exceeds 1. Let $B_1, \ldots, B_k$ be the resulting bins.

Definition 3.1. *A bin* B *is* good *if* $w(B) \in [1, 2]$. *A packing where all bins are good is called* good.

All bins in the preliminary packing, excluding B_k, are good. If $w(B_k) \geq 1$, the preliminary packing is good. However, B_k can have weight less than 1. If $w(B_{k-1}) + w(B_k) \leq 2$, we obtain a good packing by combining B_{k-1} and B_k. In the remainder of the proof, we show how to rearrange items in B_k when

$$w(B_k) < 1; \tag{1}$$

$$w(B_{k-1}) + w(B_k) > 2 \tag{2}$$

to obtain a legal packing with cost at most $opt + 1$.

Observation 3.2 *If* $i \in B_j$ *then* $w(B_j) < 1 + w_i$. *Thus,* $w(B_j) - w_i < 1$.

Definition 3.2. *An item* i *is called* small *if* $w_i \leq 1/2$, *and* large *otherwise.*

Since $w(B_k) < 1$, $w(B_{k-1}) < 2$ and $w(\mathcal{I}) \geq 3$, the number of bins $k \geq 3$. We repack bins B_1, B_{k-2}, B_{k-1} and B_k to ensure that the last bin satisfies the weight lower bound. The remaining proof (omitted) is broken down into cases, depending on how many bins contain small items. □

4 Min-Max Rectangle Tiling

We present two approximation algorithms for Min-Max Rectangle Tiling whose performance is summarized in Theorems 4.1 and 4.2.

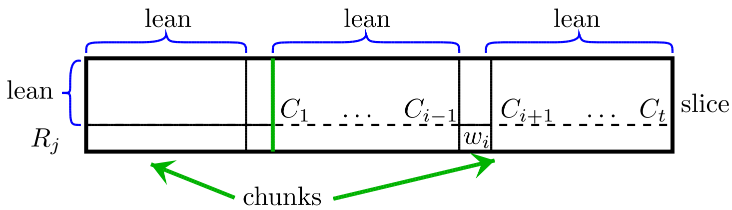
Theorem 4.1. *Min-Max Rectangle Tiling can be approximated with ratio 4 in time $O(mn)$.*

Proof. Our algorithm first preprocesses the array to ensure that the last row is fat. (Recall that *fat* and *lean* were defined in Definition 1.1.) Then it greedily *slices* the array, that is, partitions it using horizontal lines. The resulting groups of consecutive rows are called *slices*. Finally, each slice is greedily *diced* using vertical lines into sub-rectangles, called *chunks*.

Let R_i denote the ith row of A. While R_m is thin, we perform a step of preprocessing that replaces the last two rows, R_{m-1} and R_m, with row $R_{m-1} + R_m$ (and decrements m by 1). When R_m is thin, every subset of R_m is thin, and cannot be a valid tile. Thus, every element of R_m has to be in the same tile as the element directly above it. Therefore, a preprocessing step does not change the set of valid tilings of A.

In a step of slicing, we start at the top (that is, go through the rows in the increasing order of indices). Let j be the smallest index such that remaining (not yet sliced) top rows up to row R_j form a fat rectangle. Then we cut horizontally between rows R_j and R_{j+1}, and call the top set of rows a slice. Continue on the matrix formed by the bottom rows. Since the preprocessing ensured that the last row is fat, all resulting slices are fat.

In a step of dicing, analogously to the slicing step, we cut up a slice vertically, dicing away *chunks*, minimal fat sets of leftmost columns, unless the remaining columns form a lean rectangle.



Consider a tile/chunk produced by our algorithm. The rectangle formed by all rows of the tile, excluding the bottom row, is lean because it is obtained by partitioning a valid slice. Thus, the weight of this rectangle is less than w_{lb}, and consequently, less than *opt*. Let $C_1, \ldots, C_t$ be the columns of the tile (partial columns of the original matrix), and $w_1, \ldots, w_t$ be the entries in the bottom row of the slice. Let i be the smallest index such that $C_1, \cdots, C_i$ form a fat rectangle. (If this tile is the last chunk in its slice, then i might be less than t.) By the choice of i, the rectangle formed by $C_1, \ldots, C_{i-1}$ is lean, and so is the rectangle formed by $C_{i+1}, \ldots, C_t$. C_i without w_i is also lean, because it is a subset of the lean part of the slice. Finally, since w_i has to participate in a tile, $w_i \geq opt$. Consequently, the weight of the tile is smaller than $opt + 3w_{\mathrm{lb}} \leq 4opt$.

It is easy to implement the algorithm so that each step performs a constant number of operations per matrix entry, and the algorithm takes time $O(mn)$. □

We can get a better approximation ratio when the entries in the matrix are restricted to be 0 or 1. This case covers the scenarios where each entry indicates the presence or absence of some object.

Theorem 4.2. *Min-Max Rectangle Tiling with 0-1 entries can be approximated with ratio 3 in time* $O(mn)$*.* (The proof is omitted.)

References

1. Du, W., Eppstein, D., Goodrich, M.T., Lueker, G.S.: On the approximability of geometric and geographic generalization and the min-max bin covering problem. In: WADS. (2009) 242–253
2. Ciriani, V., di Vimercati, S.D.C., Foresti, S., Samarati, P.: k-anonymous data mining: A survey. In Aggarwal, C.C., Yu, P.S., eds.: Privacy-Preserving Data Mining: Models and Algorithms. Springer (2008)
3. Garcia, Y.J., Lopez, M.A., Leutenegger, S.T.: A greedy algorithm for bulk loading r-trees. In: GIS '98: Proceedings of the 1998 ACM Int. Symp. on Advances in Geographic Information Systems, ACM (1998) 163–164
4. Assmann, S.F., Johnson, D.S., Kleitman, D.J., Leung, J.Y.T.: On a dual version of the one-dimensional bin packing problem. J. Algorithms **5** (1984) 502–525
5. Csirik, J., Johnson, D.S., Kenyon, C.: Better approximation algorithms for bin covering. In: SODA. (2001) 557–566
6. Jansen, K., Solis-Oba, R.: An asymptotic fully polynomial time approximation scheme for bin covering. Theor. Comput. Sci. **306** (2003) 543–551
7. Bansal, N., Sviridenko, M.: The santa claus problem. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (2006) 31–40
8. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics **5** (1979) 287–326
9. Manne, F.: Load Balancing in Parallel Sparse Matrix Computation. PhD thesis, University of Bergen, Norway (1993)
10. Khanna, S., Muthukrishnan, S., Paterson, M.: On approximating rectangle tiling and packing. In: SODA. (1998) 384–393
11. Sharp, J.P.: Tiling multi-dimensional arrays. In: FCT. (1999) 500–511
12. Smith, A., Suri, S.: Rectangular tiling in multi-dimensional arrays. In: SODA. (1999) 786–794
13. Muthukrishnan, S., Poosala, V., Suel, T.: On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In: ICDT. (1999) 236–256
14. Berman, P., DasGupta, B., Muthukrishnan, S., Ramaswami, S.: Improved approximation algorithms for rectangle tiling and packing. In: SODA. (2001) 427–436
15. Berman, P., DasGupta, B., Muthukrishnan, S.: Slice and dice: A simple, improved approximate tiling recipe. In: SODA. (2002) 455–464
16. Berman, P., DasGupta, B., Muthukrishnan, S.: Approximation algorithms for max-min tiling. J. Algorithms **47** (2003) 122–134
17. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46** (1990) 259–271
18. Tutte, W.T.: A theorem on planar graphs. Trans. Amer. Math. Soc. **82** (1956) 99–116
19. Chiba, N., Nishizeki, T.: The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. J. Algorithms **10** (1989) 187–211