An APTAS for Bin Packing with Clique-Graph Conflicts

Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai(✉)

Computer Science Department, Technion, 3200003 Haifa, Israel
{idoron-arad,kulik,hadas}@cs.technion.ac.il

Abstract. We study the following variant of the classic *bin packing* problem. Given a set of items of various sizes, partitioned into groups, find a packing of the items in a minimum number of identical (unit-size) bins, such that no two items of the same group are assigned to the same bin. This problem, known as *bin packing with clique-graph conflicts*, has natural applications in storing file replicas, security in cloud computing and signal distribution.

Our main result is an *asymptotic polynomial time approximation scheme (APTAS)* for the problem, improving upon the best known ratio of 2. As a key tool, we apply a novel *Shift & Swap* technique which generalizes the classic linear shifting technique to scenarios allowing conflicts between items. The major challenge of packing *small* items using only a small number of extra bins is tackled through an intricate combination of enumeration and a greedy-based approach that utilizes the rounded solution of a *linear program*.

1 Introduction

In the classic *bin packing (BP)* problem, we seek a packing of items of various sizes into a minimum number of unit-size bins. This fundamental problem arises in a wide variety of contexts and has been studied extensively since the early 1970's. In some common scenarios, the input is partitioned into *disjoint groups*, such that items in the same group are *conflicting* and therefore cannot be packed together. For example, television and radio stations often assign a set of programs to their channels. Each program falls into a genre such as comedy, documentary or sports on TV, or various musical genres on radio. To maintain a diverse daily schedule of programs, the station would like to avoid broadcasting two programs of the same genre in one channel. Thus, we have a set of items (programs) partitioned into groups (genres) that need to be packed into a set of bins (channels), such that items belonging to the same group cannot be packed together.

We consider this natural variant of the classic bin packing problem that we call *group bin packing (GBP)*. Formally, the input is a set of N items $I = \{1, \ldots, N\}$ with corresponding sizes $s_1, ..., s_N \in (0, 1]$, partitioned into n disjoint groups $G_1, ..., G_n$, i.e., $I = G_1 \cup G_2 \cup \ldots \cup G_n$. The items need to be packed in

A. Lubiw et al. (Eds.): WADS 2021, LNCS 12808, pp. 286–299, 2021.
https://doi.org/10.1007/978-3-030-83508-8_21

unit-size bins. A packing is *feasible* if the total size of items in each bin does not exceed the bin capacity, and no two items from the same group are packed in the same bin. We seek a feasible packing of all items in a minimum number of unit-size bins. We give in [6] some natural applications of GBP.

Group bin packing can be viewed as a special case of *bin packing with conflicts (BPC)*, in which the input is a set of items I, each having size in $(0, 1]$, along with a conflict graph $G = (V, E)$. An item $i \in I$ is represented by a vertex $i \in V$, and there is an edge $(i, j) \in E$ if items i and j cannot be packed in the same bin. The goal is to pack the items in a minimum number of unit-size bins such that items assigned to each bin form an *independent set* in G.

Indeed, GBP is the special case where the conflict graph is a union of cliques. Thus, GBP is also known as *bin packing with clique-graph conflicts* (see Sect. 1.2).

1.1 Contribution and Techniques

Our main result (in Sect. 3) is an APTAS for the group bin packing problem, improving upon the best known ratio of 2 [1].[1]

Existing algorithms for BPC often rely on initial *coloring* of the instance. This enables to apply in later steps known techniques for bin packing, considering each color class (i.e., a subset of non-conflicting items) separately. In contrast, our approach uses a refined packing of the original instance while eliminating conflicts, thus generalizing techniques for classic BP.

Our first technical contribution is an enhancement of the *linear shifting* technique of [8]. This enables our scheme to enumerate in polynomial time over packings of relatively large items, while guaranteeing that these packings respect the group constraints. Our *Shift & Swap* technique considers the set of large items that are associated with different groups (satisfying certain properties) as a classic BP instance, i.e., the group constraints are initially relaxed. Then the scheme applies to these items the linear shifting technique of [8]. In the process, items of the same group may be packed in the same bin. Our *Swapping* algorithm resolves all conflicts, with no increase in the total number of bins used (see Sects. 3.1 and 3.2).

A common approach used for deriving APTASs for BP is to pack in a bounded number of extra bins a set of discarded small items of total size $O(\varepsilon)OPT$, where $OPT = OPT(I)$ is the minimum number of bins required for packing the given instance I, and $\varepsilon \in (0, 1)$ is the accuracy parameter of the scheme. As shown in [6], this approach may fail for GBP, e.g., when the discarded items belong to the same group. Our second contribution is an algorithm that overcomes this hurdle. The crux is to find a set of small items of total size $O(\varepsilon)OPT$ containing $O(\varepsilon)OPT$ items from each group. This would enable to pack these items in a small number of extra bins. Furthermore, the remaining small items should be

[1] We note that 2 is the best known absolute as well as *asymptotic* approximation ratio for the problem (see Sect. 1.2). We give formal definitions of absolute/asymptotic ratios in Sect. 2.

feasibly assigned to partially packed OPT bins. Our algorithm identifies such sets of small items through an intricate combination of enumeration and a greedy-based approach that utilizes the rounded solution of a *linear program*.

1.2 Related Work

The classic bin packing problem is known to be NP-hard. Furthermore, it cannot be approximated within a ratio better than $\frac{3}{2}$, unless P = NP. This ratio is achieved by the simple First-Fit Decreasing algorithm [21]. The paper [8] presents an APTAS for bin packing, which uses at most $(1+\varepsilon)OPT+1$ bins, for any fixed $\varepsilon \in (0, 1/2)$. The paper [16] gives an approximation algorithm that uses at most $OPT + O(\log^2(OPT))$ bins. The additive factor was improved in [20] to $O(\log OPT \cdot \log\log OPT)$. For comprehensive surveys of known results for BP see, e.g., [3,4].

The problem of *bin packing with conflicts (BPC)* was introduced in [15]. As BPC includes as a special case the classic *graph coloring* problem, it cannot be approximated within factor $N^{1-\varepsilon}$ for an input of N items, for all $\varepsilon > 0$, unless $P = NP$ [22]. Thus, most of the research work focused on obtaining approximation algorithms for BPC on classes of conflict graphs that can be optimally colored in polynomial time. Epstein and Levin [7] presented sophisticated algorithms for two such classes, namely, a $\frac{5}{2}$-approximation for BPC with a *perfect* conflict graph,[2] and $\frac{7}{4}$-approximation for a *bipartite* conflict graph.

The hardness of approximation of GBP (with respect to absolute approximation ratio) follows from the hardness of BP, which is the special case of GBP where the conflict graph is an independent set. A 2.7-approximation algorithm for general instances follows from a result of [15]. Oh and Son [19] showed that a simple algorithm based on First-Fit outputs a packing of any GBP instance I in $1.7OPT + 2.19v_{max}$ bins, where $v_{max} = \max_{1 \leq j \leq n} |G_j|$. The paper [18] shows that some special cases of the problem are solvable in polynomial time. The best known ratio for GBP is 2 due to [1].

Jansen [12] presented an *asymptotic fully polynomial time approximation scheme (AFPTAS)* for BPC on d-inductive conflict graphs,[3] where $d \geq 1$ is some constant. The scheme of [12] uses for packing a given instance I at most $(1 + \varepsilon)OPT + O(d/\varepsilon^2)$ bins. This implies that GBP admits an AFPTAS on instances where the maximum clique size is some constant d. Thus, the existence of an asymptotic approximation scheme for general instances remained open.

Das and Wiese [5] introduced the problem of makespan minimization with bag constraints. In this generalization of the classic makespan minimization problem, each job belongs to a *bag*. The goal is to schedule the jobs on a set of m identical machines, for some $m \geq 1$, such that no two jobs in the same bag are assigned to the same machine, and the makespan is minimized. For the classic

[2] For the subclass of interval graphs the paper [7] gives a $\frac{7}{3}$-approximation algorithm.
[3] A graph G is *d-inductive* if the vertices of G can be numbered such that each vertex is connected by an edge to at most d lower numbered vertices.

problem of makespan minimization with no bag constraints, there are known PTAS [11,17] as well as EPTAS [2,10,13,14]. Das and Wiese [5] developed a PTAS for the problem with bag constraints. Later, Grage et al. [9] obtained an EPTAS.

Due to space constraints, some of our results and formal proofs are given in the full version of the paper [6].

2 Preliminaries: Scheduling with Bag Constraints

Our scheme is inspired by the elaborate framework of Das and Wiese [5] for makespan minimization with bag constraints. For completeness, we give below an overview of the scheme of [5]. Given a set of jobs I partitioned into bags and m identical machines, let $p_\ell > 0$ be the processing time of job $\ell \in I$. The instance is scaled such that the optimal makespan is 1. The jobs and bags are then classified using the next lemma.

Lemma 2.1. *For any instance I and $\varepsilon \in (0,1)$, there is an integer $k \in \{1, ..., \lceil \frac{1}{\varepsilon^2} \rceil\}$ such that $\sum_{\ell \in I:\, p_\ell \in [\varepsilon^{k+1}, \varepsilon^k)} p_\ell \leq \varepsilon^2 m$.*

A job ℓ is *small* if $p_\ell < \varepsilon^{k+1}$, *medium* if $p_\ell \in [\varepsilon^{k+1}, \varepsilon^k)$ and *large* if $p_\ell \geq \varepsilon^k$, where k is the value found in Lemma 2.1. A bag is *large* if the number of large and medium jobs it contains is at least εm, and *small* otherwise.

The scheme of [5] initially enumerates over *slot patterns* for packing large and medium jobs from large bags optimally in polynomial time. The enumeration is enhanced by using dynamic programming and a flow network to schedule also the large jobs from small bags. The medium jobs in each small bag are scheduled across the m machines almost evenly, causing only small increase to the makespan. The small jobs are partitioned among *machine groups* with the same processing time and containing jobs from the same subset of large bags. Then, a greedy approach is used with respect to the bags to schedule the jobs within each machine group, such that the overall makespan is at most $1 + O(\varepsilon)$.

Our scheme classifies the items and groups similar to the classification of jobs and bags in [5]. We then apply enumeration over patterns to pack the large and medium items. Thus, Lemmas 3.2, 3.6 and 3.8 in this paper are adaptations of results obtained in [5]. However, the remaining components of our scheme are different. One crucial difference is our use of a Shift & Swap technique to round the sizes of large and medium items. Indeed, rounding the item sizes using the approach of [5] may cause overflow in the bins, requiring a large number of extra bins to accommodate the excess items. Furthermore, packing the *small* items using $O(\varepsilon)OPT$ extra bins requires new ideas (see Sect. 3).

We use standard definitions of approximation ratio and *asymptotic* approximation ratio. Given a minimization problem Π, let $\mathcal{A}$ be a polynomial-time algorithm for Π. For an instance I of Π, denote by $OPT(I)$ and $\mathcal{A}(I)$ the values of an optimal solution and the solution returned by $\mathcal{A}$ for I, respectively. We say that $\mathcal{A}$ is a ρ-approximation algorithm for Π, for some $\rho \geq 1$, if $\mathcal{A}(I) \leq \rho \cdot OPT(I)$ for any instance I of Π. $\mathcal{A}$ is an *asymptotic* ρ-approximation for Π if there is a constant $c \in \mathbb{R}$ such that $\mathcal{A}(I) \leq \rho \cdot OPT(I) + c$ for any instance I of Π. An APTAS for Π is a family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ such that A_ε is a polynomial-time asymptotic $(1+\varepsilon)$-approximation for each $\varepsilon > 0$. When clear from the context, we use $OPT = OPT(I)$.

3 An APTAS for GBP

In this section we present an APTAS for GBP. Let OPT be the optimal number of bins for an instance I. Our scheme uses as a subroutine a BalancedColoring algorithm proposed in [1] for the *group packing* problem (see the details in [6]). Let $S(I)$ be the total size of items in I, i.e., $S(I) = \sum_{\ell \in [N]} s_\ell$. Recall that v_{max} is the maximum cardinality of any group. The next lemma follows from a result of [1].

Lemma 3.1. *Let I be an instance of GBP. Then BalancedColoring packs I in at most* $\max\{2S(I), S(I) + v_{max}\}$ *bins.*

By the above, given an instance I of GBP, we can guess OPT in polynomial time, by iterating over all integer values in $[1, \max\{2S(I), S(I)+v_{max}]$ and taking the minimal number of bins for which a feasible solution exists.

Similar to Lemma 2.1, we can find a value of k, $1 \leq k \leq \lceil \frac{1}{\varepsilon^2} \rceil$, satisfying $\sum_{\ell \in I:\, s_\ell \in [\varepsilon^{k+1}, \varepsilon^k)} s_\ell \leq \varepsilon^2 \cdot OPT$. Now, we classify item ℓ as *small* if $s_\ell < \varepsilon^{k+1}$, *medium* if $s_\ell \in [\varepsilon^{k+1}, \varepsilon^k)$ and *large* otherwise. A group is *large* if the number of large and medium items of that group is at least $\varepsilon^{k+2} \cdot OPT$, and *small* otherwise. Given an instance I of GBP and a constant $\varepsilon \in (0,1)$, we also assume that $OPT > \frac{3}{\varepsilon^{k+2}}$ (otherwise, the conflict graph is d-inductive, where d is a constant, and the problem admits an AFPTAS [12]).

Lemma 3.2. *There are at most $\frac{1}{\varepsilon^{2k+3}}$ large groups.*

3.1 Rounding of Large and Medium Items

We start by reducing the number of distinct sizes for the large and medium items. Recall that in the linear shifting technique we are given a BP instance of N items and a parameter $Q \in (0, N]$. The items are sorted in non-increasing order by sizes and then partitioned into classes. Each class (except maybe the last one) contains $\max\{Q, 1\}$ items. The items in class 1 (i.e., largest items) are discarded (the discarded items are handled in a later stage of the algorithm). The sizes of items in each class are then rounded up to the maximum size of an item in this class. For more details see, e.g., [8].

We apply linear shifting to the large and medium items in each large group with parameter $Q = \lfloor \varepsilon^{2k+4} \cdot OPT \rfloor$. Let I, I' be the instance before and after the shifting over large groups, respectively.

Lemma 3.3. $OPT(I') \leq OPT(I)$.

Lemma 3.4. *Given a feasible packing of I' in OPT bins, we can find a feasible packing of I in $(1+O(\varepsilon))OPT$ bins.*

Next, we round the sizes of large items in small groups. As the number of these groups may be large, we use the following *Shift & Swap* technique. We merge all of the large items in small groups into a single group, to which we apply linear shifting with parameter $Q = \lfloor 2\varepsilon \cdot OPT \rfloor$. In addition to items in class 1, which are discarded due to linear shifting, we also discard the items in the last size class; these items are packed in a new set of bins (see the proof of Lemma 3.15 in [6]).

Lemma 3.5. *After rounding, there are at most $O(1)$ distinct sizes of large and medium items from large groups, and large items from small groups.*

Relaxing the *feasibility* requirement for the packing of rounded large items from small groups, the statements of Lemma 3.3 and Lemma 3.4 hold for these items as well. To obtain a feasible packing of these items, we apply a Swapping subroutine which resolves the possible conflicts caused while packing the items.

Our scheme packs in each step a subset of items, using OPT bins, while discarding some items. The discarded items are packed later in a set of $O(\varepsilon)\cdot OPT+1$ extra bins. In Sect. 3.2 we pack the large and medium items using enumeration over patterns followed by our Swapping algorithm to resolve conflicts. Section 3.3 presents an algorithm for packing the small items by combining recursive enumeration (for relatively "large" items) with a greedy-based algorithm that utilizes the rounded solution of a linear program (for relatively "small" items). In Sect. 3.4 we show that the components of our scheme combine together to an APTAS for GBP.

3.2 Large and Medium Items

The large items and medium items from large groups are packed in the bins using *slot patterns.* Let $G_{i_1},\dots,G_{i_L}$ be the large groups, and let 'u' be a label representing all the small groups. Given the modified instance I', a slot is a pair (s_ℓ, j), where s_ℓ is the rounded size of a large or medium item $\ell \in I'$ and $j \in \{i_1,\dots,i_L\} \cup \{u\}$. A *pattern* is a multiset $\{t_1,\dots,t_\beta\}$ for some $1 \le \beta \le \lfloor \frac{1}{\varepsilon^{k+1}} \rfloor$, where t_i is a slot for each $i \in [\beta]$.[4]

Lemma 3.6. *By using enumeration over patterns, we find a pattern for each bin for the large and medium items, such that these patterns correspond to an optimal solution. The running time is $O(N^{O(1)})$.*

Given slot patterns corresponding to an optimal solution, large and medium items from large groups can be packed optimally, since they are identified both by a label and a size. On the other hand, large items from small groups are

[4] Recall that the number of medium/large items that fit in a single bin is at most $\lfloor \frac{1}{\varepsilon^{k+1}} \rfloor$.

identified solely by their sizes. A greedy packing of these items, relating only to their corresponding patterns, may result in conflicts (i.e., two large items of the same small group are packed in the same bin). Therefore, we incorporate a process of swapping items of the same (rounded) size between their hosting bins, until there are no conflicts.

Given an item ℓ that conflicts with another item in bin b, for an item y in bin c such that $s_\ell = s_y$, $swap(\ell, y)$ is *bad* if it causes a conflict (either because y conflicts with an item in bin b, ℓ conflicts with an item in bin c, or $c = b$); otherwise, $swap(\ell, y)$ is *good*. We now describe our algorithm for packing the large items from small groups.

Let ζ be the given slot patterns for OPT bins. Initially, the items are packed by these patterns, where items from small groups are packed ignoring the group constraints. This can be done simply by placing an arbitrary item of size s from some small group in each slot (s, u). If ζ corresponds to an optimal solution, we meet the capacity constraint of each bin. However, this may result with conflicting items in some bins. Suppose there is a conflict in bin b. Then for one of the conflicting items, ℓ, we find a good $swap(\ell, y)$ with item y in a different bin, such that $s_y = s_\ell$. We repeat this process until there are no conflicts. We give the pseudocode of Swapping in Algorithm 1.

Algorithm 1. $Swapping(\zeta, G_1, \ldots, G_n)$

```
1: Pack the large and medium items from large groups in slots corresponding to their
   sizes and by labels.
2: Pack large items from small groups in slots corresponding to their sizes.
3: while there is an item $\ell$ involved in a conflict do
4:     Find a good $swap(\ell, y)$ and resolve the conflict.
```

Theorem 3.7. *Given a packing of large and medium items by slot patterns corresponding to an optimal solution, Algorithm 1 resolves all conflicts in polynomial time.*

We use the Swapping algorithm for each possible guess of patterns to obtain a feasible packing of the large items and medium items from large groups in OPT bins.

Now, we discard the medium items from small groups and pack them later in a new set of bins with other discarded items. This requires only a small number of extra bins (see the proof of Lemma 3.15 in [6]).

3.3 Small Items

Up to this point, all large items and the medium items from large groups are feasibly packed in OPT bins. We proceed to pack the small items. Let I_0, B be the set of unpacked items and the set of OPT partially packed bins, respectively.

The packing of the small items is done in four phases: an *optimal phase*, an *eviction phase*, a *partition phase* and a *greedy phase.*

The optimal phase is an iterative process consisting of a constant number of iterations. In each iteration, a subset of bins is packed with a subset of items whose (rounded) sizes are large relative to the free space in each of these bins. As these items belong to a *small* collection of groups among $G_1, \ldots, G_n$, they can be selected using enumeration. Thus, we obtain a packing of these items which corresponds to an optimal solution. For packing the remaining items, we want each item to be small relative to the free space in its assigned bin. To this end, in the eviction phase we discard from some bins items of non-negligible size (a single item from each bin). Then, in the partition phase, the unpacked items are partitioned into a constant number of sets satisfying certain properties, which guarantee that these items can be feasibly packed in the available free space in the bins. Finally, in the greedy phase, the items in each set are packed in their allotted subset of bins greedily, achieving a feasible packing of all items, except for a small number of items from each group, of small total size. The pseudocode of our algorithm for packing the small items is given in Algorithm 4.

The Optimal Phase: For any $b \in B$, denote by f_b^0 the free capacity in bin b, i.e., $f_b^0 = 1 - \sum_{\ell \in b} s_\ell$. We say that item ℓ is *b-negligible* if $s_\ell \leq \varepsilon^2 f_b^0$, and ℓ is *b-non-negligible* otherwise. We start by classifying the bins into two disjoint sets. Let $E_0 = \{b \in B|\ 0 < f_b^0 < \varepsilon\}$ and $D_0 = B \setminus E_0$.

We now partition B into *types.* Each type contains bins having the same total size of packed large/medium items; also, the items packed in each bin type belong to the same set of large groups, and the same number of slots is allocated in these bins to items from small groups. Formally, for each pattern p we denote by t_p the subset of bins packed with p.[5] Let T denote the set of bin types. Then $|T| = |P|$, where P is the set of all patterns. The *cardinality* of type $t \in T$ is the number of bins of this type. We use for the optimal phase algorithm *RecursiveEnum* (see the pseudocode in Algorithm 2).

Lemma 3.8. *There are $O(1)$ types before Step 1 of Algorithm 2.*

Once we have the classification of bins, each type t of cardinality smaller than $1/\varepsilon^4$ is padded with empty bins so that $|t| \geq 1/\varepsilon^4$. An item ℓ is *t-negligible* if ℓ is b-negligible for all bins b of type t (all bins in the same type have the same free capacity), and *t-non-negligible* otherwise. Denote by I'_t the large/medium items that are packed in the bins of type t, and let $I_t(g)$ be the set of small items that are packed in t in some solution g (in addition to I'_t). For any $1 \leq i \leq n$, a group G_i is *$t(g)$-significant* if $I_t(g)$ contains at least $\varepsilon^4|t|$ t-non-negligible items from G_i, and G_i is *$t(g)$-insignificant* otherwise.

RecursiveEnum proceeds in iterations. In the first iteration, it *guesses* for each type $t \subseteq E_0$ a subset of the items $I_t(g_{opt}) \subseteq I_0$, where g_{opt} corresponds to an optimal solution for completing the packing of t. Specifically, *RecursiveEnum* initially guesses $L(t, g_{opt})$ groups that are $t(g_{opt})$-significant: $G_{i_1}, \ldots, G_{i_{L(t,g_{opt})}}$.

[5] For the definition of patterns see Sect. 3.2.

For each $G_{i_j}, j \in \{1, \ldots, L(t, g_{opt})\}$, the algorithm guesses which items of G_{i_j} are added to $I_{t(g_{opt})}$. Since the number of guesses might be exponential, we apply to G_{i_j} linear shifting as follows. Guess $\lceil \frac{1}{\varepsilon^3} \rceil$ *representatives* in G_{i_j}, of sizes $s_{\ell_1} \leq s_{\ell_2} \leq \ldots s_{\ell_{\lceil 1/\varepsilon^3 \rceil}}$. The kth representative is the largest item in size class k, $1 \leq k \leq \lceil \frac{1}{\varepsilon^3} \rceil$ for the linear shifting of G_{i_j} in type t. Using the parameter $Q^t_{i_j} = \varepsilon^3 |t|$, the item sizes in class k are rounded up to s_{ℓ_k}, for $1 \leq k \leq \lceil \frac{1}{\varepsilon^3} \rceil$. Given a correct guess of the representatives, the actual items in size class k are selected at the end of algorithm *RecursiveEnum* (in Step 16). Denote the chosen items from G_{i_j} to bins of type t by $G^t_{i_j}$.

We now extend the definition of patterns for each type t. A slot is a pair (s_ℓ, j), where s_ℓ is the (rounded) size of a t-non-negligible item $\ell \in I_t(g_{opt})$, and there is a label for each $t(g_{opt})$-significant group G_{i_j}, $j \in \{1, \ldots, L(t, g_{opt})\}$.[6] A *t-pattern* is a multiset $\{q_1, \ldots, q_{\beta_t}\}$ containing at most $\lfloor \frac{1}{\varepsilon^2} \rfloor$ elements, where q_i is a slot for each $i \in \{1, \ldots, \beta_t\}$. Now, for each type $t \in T$ we use enumeration over patterns for assigning $G^t_{i_1}, \ldots, G^t_{i_{L(t,g_{opt})}}$ to bins in t. This completes the first iteration, and the algorithm proceeds recursively.

We now update D_0, E_0 for the next iteration by removing from E_0 bins b that have a considerably large free capacity with respect to f^0_b. For each $b \in B$, let f^1_b be the capacity available in b after iteration 1. Then $E_1 = \{b \in E_0 | \; 0 < f^1_b < \varepsilon f^0_b\}$ and $D_1 = B \setminus E_1$.

Now, each type $t \in T$ is partitioned into *sub-types* that differ by the packing of $I_t(g_{opt})$ in the first iteration. The set of types T is updated to contain these sub-types. At this point, a recursive call to *RecursiveEnum* computes for each bin type $t \subseteq E_1$ a guessing and a packing of its t-non-negligible items.[7] We repeat this recursive process $\alpha = \frac{1}{\varepsilon} + 5 = O(1)$ times.

Let G^t_i be the subset of items (of rounded sizes) assigned from G_i to bins of type t at the end of *RecursiveEnum*, for $1 \leq i \leq n$ and $t \in T$. Recall that the algorithm did not select specific items in G^t_i; that is, we only have their rounded sizes and the number of items in each size class. The algorithm proceeds to pack items from G_i in all types t for which G_i was $t(g_{opt})$-significant in some iteration. Let T_{G_i} be the set of these types. The algorithm considers first the type $t \in T_{G_i}$ for which the class C of largest size items contains the item of maximal size, where the maximum is taken over all types $t \in T_{G_i}$. The algorithm packs in bins of type t the Q^t_i largest remaining items in G_i in the slots allocated to items in C; it then proceeds similarly to the remaining size classes in types $t \in T_{G_i}$ and the remaining items in G^t_i.

Lemma 3.9. *The following hold for RecursiveEnum: (i) the running time is polynomial; (ii) the increase in the number of bins in Step 7 is at most εOPT; (iii) In Step 11 we discard at most εOPT items from each group of total size at most εOPT. (iv) One of the guesses in Steps 8, 11 corresponds to an optimal solution.*

[6] Note that we do not need a label for the $t(g_{opt})$-insignificant groups, because their items are packed separately.

[7] An item is b-non-negligible w.r.t f^1_b in this iteration, or w.r.t f^h_b in iteration $h+1, h \in \{0, \ldots, \alpha - 1\}$.

Algorithm 2. *RecursiveEnum*(I_0, B)

```
1: Let $f_b^0$ be the remaining free capacity in bin $b \in B$.
2: Let $E_0 = \{b \in B | 0 < f_b^0 < \varepsilon\}$ and $D_0 = B \setminus E_0$.
3: Denote by $T$ the collection of bin types.
4: for $h = 0, \dots, \alpha$ do
5:     for all types $t \subseteq E_h$ do
6:         if $|t| < \frac{1}{\varepsilon^4}$ then
7:             increase the cardinality of $t$ to $\frac{1}{\varepsilon^4}$.
8:         Guess $t(g_{opt})$-significant groups: $G_{i_1}, \dots, G_{i_{L(t,g_{opt})}}$
9:         for $j = 1, \dots, L(t, g_{opt})$ do
10:            Guess the number of items from $G_{i_j}$ to be added to bins of type $t$.
11:            Guess a representative for each size class of $t$-non-negligible items of $G_{i_j}$
                 for linear shifting.
12:        Guess $|t|$ $t$-patterns for bins in $t$ using the sizes after linear shifting of
                 $G_{i_1}, \dots, G_{i_{L(t,g_{opt})}}$.
13:        Replace type $t$ in $T$ by all of the sub-types of $t$.
14:    Let $f_b^{h+1}$ be the remaining free capacity in bin $b \in B$.
15:    Let $E_{h+1} = \{b \in E_h | 0 < f_b^{h+1} < \varepsilon f_b^h\}$ and $D_{h+1} = B \setminus E_{h+1}$.
16: Complete the packing of all size classes by assigning items greedily.
```

The Eviction Phase: One of the guesses in the optimal phase corresponds to an optimal solution. For simplicity, henceforth assume that we have this guess. Recall that E_α is the set of all bins b for which $0 < f_b^\alpha < \varepsilon f_b^{\alpha-1}$. In Step 3 of *PackSmallItems* (Algorithm 4) we evict an item from each $b \in E_\alpha$ such that the available capacity of b increases to at least $\frac{f_b^\alpha}{\varepsilon}$. This is done greedily: consider the bins in E_α one by one in arbitrary order. From each bin discard a small item $\ell \in G_i$, for some G_i, $1 \leq i \leq n$, such that the following hold: (i) $s_\ell \geq \frac{f_b^\alpha}{\varepsilon}$, and (ii) less than εOPT items were discarded from G_i in this phase. Since α is large enough, this phase can be completed successfully, as shown below. Let $T = \{t_1, \dots, t_\mu, t'\}$ be the types after the optimal phase, where t' is a new type such that $|t'| = \varepsilon OPT$. Bins of type t' are empty, i.e., each bin b of type t' has free space 1. Denote by $f(t)$ the free space in each bin b of type t after the eviction phase, and let I_L be the large items from small groups (already packed in the bins).

Lemma 3.10. *After Step 4 of PackSmallItems there exists a partition of I_α into types $I_{t_1}, \dots, I_{t_\mu}, I_{t'}$, for which the following hold. For each $t \in T$, (i) $|G_j^t| = |G_j \cap I_t| \leq |t| - |(I_t' \setminus I_L) \cap G_j|$, for all $1 \leq j \leq n$. (ii) for any $\ell \in I_t : s_\ell \leq \varepsilon f(t)$, and (iii) $S(I_t) \leq f(t)|t|$.*

We explain the conditions of the lemma below.

The Partition Phase: Let T be the set of types after Step 4 of Algorithm 4, and I_α the remaining unpacked items.[8] We seek a partition of I_α into subsets

[8] Recall that we consider only items that were not discarded in previous steps, as discarded items are packed in a separate set of bins.

associated with bin types such that the items assigned to each type t are relatively tiny; also, the total size and the cardinality of the set of items assigned to t allow to feasibly pack these items in bins of this type. This is done by proving that a polytope representing the conditions in Lemma 3.10 has vertices at points which are integral up to a constant number of coordinates. Each such coordinate, $x_{\ell,t}$, corresponds to a fractional selection of some item $\ell \in I_\alpha$ to type $t \in T$. We use G_j to denote the subset of remaining items in G_j, $1 \leq j \leq n$.

Formally, we define a polytope P as the set of all points $x \in [0,1]^{I_\alpha \times T}$ which satisfy the following constraints.

$$\begin{aligned}
&\forall \ell \in I_\alpha, t \in T \text{ s.t. } s_\ell > \varepsilon f(t) &&: \quad x_{\ell,t} = 0 \\
&\forall t \in T &&: \quad \sum_{\ell \in I_\alpha} x_{\ell,t} s_\ell \leq f(t)|t| \\
&\forall \ell \in I_\alpha &&: \quad \sum_{t \in T} x_{\ell,t} = 1 \\
&\forall 1 \leq j \leq n, t \in T &&: \quad \sum_{\ell \in G_j} x_{\ell,t} \leq |t| - |(I'_t \setminus I_L) \cap G_j|
\end{aligned}$$

The first constraint refers to condition (ii) in Lemma 3.10, which implies that items assigned to type t need to be tiny w.r.t the free space in the bins of this type. The second constraint reflects condition (iii) in the lemma, which guarantees that the items in I_t can be feasibly packed in the bins of type t. The third constraint ensures that overall each item $\ell \in I_\alpha$ is (fractionally) assigned exactly once.

The last constraint reflects condition (i) in Lemma 3.10. Overall, we want to have at most $|t|$ items of G_j assigned to bins of type t. Recall that these bins may already contain large/medium items from G_j packed in previous steps. While large/medium items from *large* groups are packed optimally, the packing of large items from *small* groups, i.e., I_L, is not necessarily optimal. In particular, the items in I_L packed by our scheme in bins of type t may not appear in these bins in the optimal solution g_{opt} to which our packing corresponds. Thus, we exclude these items and only require that the number of items assigned from G_j to bins of type t is bounded by $|t| - |(I'_t \setminus I_L) \cap G_j|$.

Theorem 3.11. *Let $x \in P$ be a vertex of P. Then,*

$$|\{\ell \in I_\alpha \mid \exists t \in T : \ x_{\ell,t} \in (0,1)\}| = O(1).$$

By Theorem 3.11, we can find a feasible partition (with respect to the constraints of the polytope) by finding a vertex of the polytope, and then discarding the $O(1)$ fractional items. These items can be packed in $O(1)$ extra bins. By Lemma 3.10 we have that $P \neq \emptyset$; thus, a vertex of P exists and the partition can be found in polynomial time.

The Greedy Phase: In this phase we pack the remaining items using algorithm *GreedyPack* (see the pseudocode in Algorithm 3). Let $G_1^t, \ldots, G_n^t$ be the items in I_t from each group, and let $S(I_t)$ be the total size of these items, i.e., $S(I_t) = \sum_{j=1}^{n} \sum_{\ell \in G_j^t} s_\ell$.

Algorithm 3. *GreedyPack*$(I_t = \{G_{i_1}^t, \ldots, G_{i_H}^t\}, t = \{b_1, \ldots, b_{|t|}\}$

```
1:  for $j = 1, \ldots, H$ do
2:      Sort $G_{i_j}^t$ in a non-increasing order by sizes.
3:  Let $y_{i_j}$ be the largest remaining item in $G_{i_j}^t$, $j = 1, \ldots, H$.
4:  for each bin $b \in t$ do
5:      Add to bin $b$ the items $y_{i_1}, \ldots, y_{i_H}$.
6:      while total size of items packed in bin $b > 1$ do
7:          Select a group $G_{i_j}^t \in \{G_{i_1}^t, \ldots, G_{i_H}^t\}$ such that $y_{i_j}$ is not last in $G_{i_j}^t$.
8:          if cannot complete last step then
9:              return $failure$
10:         Return $y_{i_j}$ to $G_{i_j}^t$.
11:         Let $y'_{i_j}$ be the next largest item in $G_{i_j}^t$.
12:         Add $y'_{i_j}$ to bin $b$.
13:     for $j = 1, \ldots, H$ do
14:         if $G_{i_j}^t$ has a large item in bin $b$ then
15:             discard the small item.
```

We now describe the packing of the remaining items in I_t in bins of type t. First, we add $2\varepsilon|t|$ extra bins to t. The extra bins are empty and thus have capacity 1; however, we assume that they have capacity $f(t) \leq 1$. This increases the overall number of bins in the solution by $2\varepsilon OPT$. Consider the items in each group in non-increasing order by sizes. For each bin $b \in t$ in an arbitrary order, GreedyPack assigns to b the largest remaining item in each group $G_1^t, \ldots, G_n^t$. If an overflow occurs, replace an item from some group G_j^t by the next item in G_j^t. This is repeated until there is no overflow in b. W.l.o.g., we may assume that $|G_j| = OPT$ for all $1 \leq j \leq n$; thus, b contains one item from each group (otherwise, we can add to G_j dummy items of size 0, with no increase to the number of bins in an optimal solution).

Recall that the large items from small groups are packed using the Swapping algorithm, that yields a feasible packing. Yet, it does not guarantee that the small items can be added without causing conflicts. Hence, GreedyPack may output a packing in which a small and large item from the same small group are packed in the same bin. Such conflicts are resolved by discarding the small item in each.

Lemma 3.12. *The total size of items discarded in GreedyPack in Step 15 due to conflicts is at most εOPT, and at most $\varepsilon^{k+2} \cdot OPT$ items are discarded from each group.*

Proof. The number of items discarded from each group is at most $\varepsilon^{k+2} \cdot OPT$, since all groups are small. Assume that the total size of these items is strictly larger than εOPT. Since each discarded item is *coupled* with a large conflicting item from the same group, whose size is at least $1/\varepsilon$ times larger (recall that the medium items are discarded), this implies that the total size of large conflicting items is greater than OPT. Contradiction. □

Algorithm 4. $PackSmallItems(I_0, B)$

```
1: for each guess of $RecursiveEnum(I_0, B)$ do
2:     for $b \in E_\alpha$ do
3:         evict from $b$ the largest item $\ell$ satisfying: $\ell$ is small, and less than $\varepsilon OPT$
             items where evicted from $G_i$, where $\ell \in G_i$.
4:     Add to $T$ a new type $t'$ consisting of $\varepsilon OPT$ empty bins.
5:     Compute a feasible partition of $I_\alpha$ into the types in $T$.
6:     for $t \in T$ do
7:         Add $2\varepsilon|t|$ extra bins to $t$.
8:         Assign $I_t$ to bins of type $t$ using $GreedyPack(I_t, t)$.
```

Lemma 3.13. *For any $t \in T$, given a parameter $0 < \delta < \frac{1}{2}$ and a set of items I_t such that (i) $|G_j^t| \leq |t| - |(I'_t \setminus I_L) \cap G_j|$; (ii) for all $\ell \in I_t : s_\ell \leq \delta f(t)$, and (iii) $S(I_t) \leq (1-\delta) f(t)|t|$, GreedyPack finds a feasible packing of I_t in bins of type t.*

Lemma 3.14. *Algorithm 4 assigns in Step 8 to OPT bins all items except for $O(\varepsilon)OPT$ items from each group, of total size $O(\varepsilon)OPT$.*

3.4 Putting It All Together

It remains to show that the items discarded throughout the execution of the scheme can be packed in a small number of extra bins.

Lemma 3.15. *The medium items from small groups and all discarded items can be packed in $O(\varepsilon) \cdot OPT$ extra bins.*

We summarize in the next result.

Theorem 3.16. *There is an APTAS for the group bin packing problem.*

References

1. Adany, R., et al.: All-or-nothing generalized assignment with application to scheduling advertising campaigns. ACM Trans. Algorithms (TALG) **12**(3), 1–25 (2016)

2. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. J. Sched. **1**(1), 55–66 (1998)
3. Christensen, H.I., Khan, A., Pokutta, S., Tetali, P.: Approximation and online algorithms for multidimensional bin packing: a survey. Comput. Sci. Rev. **24**, 63–79 (2017)
4. Coffman, E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: survey and classification. In: Handbook of Combinatorial Optimization, pp. 455–531 (2013)
5. Das, S., Wiese, A.: On minimizing the makespan when some jobs cannot be assigned on the same machine. In: 25th Annual European Symposium on Algorithms, ESA, pp. 31:1–31:14 (2017)
6. Doron-Arad, I., Kulik, A., Shachnai, H.: An APTAS for bin packing with clique-graph conflicts. arXiv preprint arXiv:2011.04273 (2020)
7. Epstein, L., Levin, A.: On bin packing with conflicts. SIAM J. Optim. **19**(3), 1270–1298 (2008)
8. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1**, 349–355 (1981)
9. Grage, K., Jansen, K., Klein, K.M.: An EPTAS for machine scheduling with bag-constraints. In: The 31st ACM Symposium on Parallelism in Algorithms and Architectures, pp. 135–144 (2019)
10. Hochbaum, D.S. (ed.): Approximation Algorithms for NP-Hard Problems. PWS Publishing Co., USA (1996)
11. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM **34**(1), 144–162 (1987)
12. Jansen, K.: An approximation scheme for bin packing with conflicts. J. Comb. Optim. **3**(4), 363–377 (1999)
13. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. SIAM J. Discret. Math. **24**(2), 457–485 (2010)
14. Jansen, K., Klein, K., Verschae, J.: Closing the gap for makespan scheduling via sparsification techniques. In: 43rd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 72:1–72:13 (2016)
15. Jansen, K., Öhring, S.R.: Approximation algorithms for time constrained scheduling. Inf. Comput. **132**(2), 85–108 (1997)
16. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd Annual Symposium on Foundations of Computer Science, pp. 312–320. IEEE (1982)
17. Leung, J.Y.: Bin packing with restricted piece sizes. Inf. Process. Lett. **31**(3), 145–149 (1989)
18. McCloskey, B., Shankar, A.: Approaches to bin packing with clique-graph conflicts. Computer Science Division, University of California (2005)
19. Oh, Y., Son, S.: On a constrained bin-packing problem. Technical report CS-95-14 (1995)
20. Rothvoß, T.: Approximating bin packing within O(log OPT * log log OPT) bins. In: 54th Annual IEEE Symposium on Foundations of Computer Science, pp. 20–29. IEEE Computer Society (2013)
21. Simchi-Levi, D.: New worst-case results for the bin-packing problem. Naval Res. Logist. (NRL) **41**(4), 579–585 (1994)
22. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(1), 103–128 (2007)