

The Class Constrained Bin Packing Problem with applications to Video-on-Demand.*†

E. C. Xavier‡

F. K. Miyazawa§

September 20, 2007

Abstract

In this paper we present approximation results for the class constrained bin packing problem that has applications to Video-on-Demand Systems. In this problem we are given bins of capacity B with C compartments, and n items of Q different classes, each item $i \in \{1, \dots, n\}$ with class c_i and size s_i . The problem is to pack the items into bins, where each bin contains at most C different classes and has total items size at most B . We present several approximation algorithms for offline and online versions of the problem.

1 Introduction

In this paper we study a class constrained version of the well known bin packing problem, which we denote by CCBP (Class Constrained Bin Packing). In this problem we are given a tuple $I = (L, s, c, C, Q, B)$ where $L = (a_1, \dots, a_n)$ is a list of items, each item $a_i \in L$ with size $0 < s_{a_i} \leq B$ and class $c_{a_i} \in \{1, \dots, Q\}$, and a set of bins, each one with capacity B and C compartments. We also refer to B as the load capacity of the bin and C as its storage capacity. This notation will be made clear in Section 2, when we relate this problem with a video-on-demand problem. A packing \mathcal{P} of L is a partition of the items, where each part has total items size at most B and the number of different classes in each part is at most C . The problem is to find a packing of L into the minimum number of bins. In the online version of the CCBP problem the items must be packed in the order (a_1, \dots, a_n) , where each item a_i must be packed without knowledge of further items. We assume that $1 < C < Q$, otherwise the CCBP problem can be solved as the original bin packing, since if $C = 1$ then items of different classes must be packed in different bins and if $C \geq Q$ then the class constraints are irrelevant. We also consider the version of this problem with bins of different sizes. In this case we have T different bins size. The input instance is a tuple $I = (L, s, c, w, C, Q, B)$ where $w : \{1, \dots, T\} \rightarrow \mathbb{R}^+$ gives the bins size. We assume w.l.o.g that for each $i \in \{1, \dots, T\}$, $w(i) \leq B$. In this case, the problem is to pack all items into bins such that the total size of used bins is minimized. This problem is denoted by VCCBP (Variable Class Constrained Bin Packing). Packing problems with class constraints have

¹A preliminary version of this paper appeared as an extended abstract in COCOON 2006, LNCS 4112, pp. 439–448, 2006.

²This research was partially supported by CNPq (478470/06-1, 471460/04-4, 306526/04-2, and 490333/04-4) and ProNEX-FAPESP/CNPq(Proc. 03/09925-5).

³Corresponding author: Escola de Artes, Ciências e Humanidades – Universidade de São Paulo – USP Leste Av. Arlindo Bettio, 1000. Ermelino Matarazzo CEP: 03828-000, Brazil – ecx@usp.br

⁴Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084-971 — Campinas-SP — Brazil, fkm@ic.unicamp.br.

many applications in multimedia storage systems, resource allocation [23, 19, 8, 13, 22, 9, 21, 7] and in operations research like manufacturing systems [12, 17, 5, 26, 27].

1.1 Notation

In the online case, the bins used to pack the items are classified as *open* or *closed*. An empty bin is declared open when it receives its first item, and remains so until it is declared closed. Only open bins may receive items. Once a bin is closed, it cannot be declared open again. We consider the bounded and unbounded space versions of the online CCBP problem. In the l -bounded space problem, an algorithm must keep at any time during its execution at most l open bins. In the unbounded version, an algorithm may keep an unbounded number of open bins.

Given an algorithm \mathcal{A} for the CCBP problem and an instance I , we denote by $\mathcal{A}(I)$ the number of bins used by the algorithm to pack this instance. We denote by $\text{OPT}(I)$ the number of bins used by an optimum (offline) solution to pack the instance I . The algorithm \mathcal{A} has an absolute approximation factor α , for $\alpha \geq 1$, if for every I it satisfies $\mathcal{A}(I) \leq \alpha \text{OPT}(I)$. It has an asymptotic approximation factor α if for every I , the algorithm produces a solution such that $\mathcal{A}(I) \leq \alpha \text{OPT}(I) + \beta$ where β is a constant. Given an algorithm A_ε , for some $\varepsilon > 0$, and an instance I for some problem P we denote by $A_\varepsilon(I)$ the value of the solution returned by algorithm A_ε when executed on instance I . We say that A_ε , for $\varepsilon > 0$, is an asymptotic polynomial time approximation scheme (APTAS) for the problem CCBP if there exists a constant β such that $A_\varepsilon(I) \leq (1 + \varepsilon)\text{OPT}(I) + \beta$ for any instance I . An online algorithm \mathcal{A} for a minimization problem is said to have a competitive ratio α , for $\alpha \geq 1$, if there exists a constant β such that $\mathcal{A}(I) \leq \alpha \text{OPT}(I) + \beta$ for any instance I .

Let I be an instance of the CCBP problem and L be the list of items in I . We write that $a \in I$ with the same meaning of $a \in L$, and we denote $s(I) = s(L) = \sum_{a_i \in L} s_{a_i}$. Given an integer M , we denote by $[M]$ the set $\{1, \dots, M\}$.

Given two sequences $L_a = (a_1, \dots, a_n)$ and $L_b = (b_1, \dots, b_m)$, we denote the concatenation of these two lists by $L_a \parallel L_b$, i.e. $L_a \parallel L_b = (a_1, \dots, a_n, b_1, \dots, b_m)$. Given a packing \mathcal{P} we denote by $|\mathcal{P}|$ the number of bins in \mathcal{P} .

In the Appendix A we provide a table containing the most used symbols to help the reader follow the reading of this article.

Throughout this paper, we use the terms color and class with the same meaning. We say that a bin is *colored* if it contains items of C different classes. In this case, this bin cannot pack any other item of a different class. A bin is said to be *full* if the total size of the items packed inside it is equal to B .

1.2 Related Work

A special case of the CCBP problem is the Bin Packing problem, which is one of the most studied problems in the literature. Some of the most famous algorithms for the bin packing problem are the algorithms FF, BF, FFD and BFD, with asymptotic performance bounds $17/10$, $17/10$, $11/9$ and $11/9$, respectively. We refer the reader to Coffman *et al.* [2] for a survey on approximation algorithms for bin packing problems. Fernandez de la Vega and Lueker [6] presented an APTAS for the bin packing problem. The online bin packing is also a well studied problem. There are many online algorithms presented in the literature for the bin-packing problem. The algorithms FF, NF, and BF are online and were investigated by Ullman [24], Johnson [10] and Johnson *et al.* [11]. Subsequent papers proposed algorithms with better approximation ratios that pack items according to interval sizes. Yao [28], and Lee and Lee [15] presented the Harmonic and Refined Harmonic algorithms with competitive ratio 1.692 and 1.636

respectively. To our knowledge the best online algorithm, with a competitive ratio of 1.58889, was presented by Seiden [18]. The best lower bound for this problem is 1.54014 due to van Vliet [25]. Recently the class-constrained versions of packing problems have received attention. In [5, 4], Dawande *et al.* claimed an approximation scheme for the offline VCCBP problem when the number of different classes Q in the input instance is bounded by a constant. In [20], Shachnai and Tamir presented a dual polynomial time approximation scheme for the offline class constrained bin packing problem (CCBP). They also used the assumption that the number of different classes in the input instance is bounded by a constant. In this case, given an instance I , the problem is to find a packing of the items in at most $\text{OPT}(I)$ bins, each bin with size at most $(1 + O(\varepsilon))B$. In [19], Shachnai and Tamir presented theoretical results for a Multiple Knapsack problem with class constraints where all items have unit size. They introduced this problem with applications to video-on-demand servers. Subsequently to this work, Golubchik *et al.* [8] presented an approximation scheme to the problem. Later, Kashyap and Khuller [13], also presented approximation algorithms to the problem with variable item sizes. Shachnai and Tamir in [23], presented algorithms for the online CCBP problem when all items have unit size. In this case they provided a lower bound of 2 to the problem and also algorithms that have a competitive ratio of 2.

1.3 Results

In this paper we present practical approximation algorithms for the CCBP problem with applications to video-on-demand problems. We also present algorithms for the online CCBP problem generalizing the work presented by Shachnai and Tamir [23], since we assume that items can have different sizes. Finally we present an APTAS for the VCCBP problem for fixed Q .

We note that the VCCBP problem was first studied by Dawande *et al.* [5, 4] where a tentative of an APTAS was considered when Q is bounded by a constant. We observed that their algorithm does not lead to an APTAS as claimed. First of all, they do a linear rounding step of the list of items L and then obtain an optimal packing for the new list. Doing this they do not guarantee a packing for the original items because of the class constraints. To pack the small items they use a First Fit strategy, and claim that each bin, perhaps a constant number of bins, is filled by at least $(1 - O(\varepsilon))$, but this is also not true due to the class constraints. In our algorithm the linear rounding step is done with items separated by colors. It then generates all possible packings for the rounded items. To pack the small items we use another strategy.

Specifically our results are the following:

- For the offline CCBP problem when all items have unit size, we present an asymptotic $(1+1/C)$ -approximation algorithm. When items have size at most B/m , for some integer m , we show an algorithm with asymptotic approximation factor $(1 + 1/C + 1/\min\{C, m\})$.
- We implemented these practical algorithms and present some experimental results for them. The experiments show that the algorithms generate solutions of high quality and can be used in practice.
- We show that the bounded space online CCBP problem does not admit a constant competitive ratio algorithm. Moreover if any item of the instance has size in $(\varepsilon, B]$, where $\varepsilon < \min(B, 1/C)$, we show that any online algorithm has competitive ratio in $\Omega(1/(C\varepsilon))$.
- For the unbounded space online CCBP problem we present an online algorithm with competitive ratio in $[2.666, 2.75]$.

- We show the points where the algorithm presented in [5, 4] fails and present an APTAS for the offline VCCBP problem for fixed Q .

Organization: In Section 2 we present the application of the CCBP problem to the data placement of videos. In Section 3, motivated by the video-on-demand systems applications, we present practical approximation algorithms for the CCBP problem assuming that all items have unit size. In Section 4, we present lower bounds for the competitive ratio of any algorithm for the bounded space online CCBP problem. In this section, we also present an online algorithm with a competitive ratio in $[2.666, 2.75]$ for the unbounded problem. In Section 5 we present an APTAS for the VCCBP problem when Q is bounded by a constant. In Section 6 we show experimental results of the practical algorithms presented in Section 3.

2 Applications of the CCBP Problem to the Data Placement on Video-on-Demand Servers

The first work to consider packing problems with class constraints as a data placement problem was the one of Shachnai and Tamir [19]. They considered the multiple knapsack version of the CCBP problem. In this case N bins are given, and the objective is to pack the maximum number of items satisfying the class constraints in each bin. Suppose we have a server of videos with N disks, each disk $j \in \{1, \dots, N\}$ with storage capacity C_j and load capacity B_j . That is, each disk j can store C_j movies and can attend at most B_j simultaneously requests for videos. The problem is to construct a server such that, based on expected requests for movies (computed by movies popularity), the number of attended requests is maximized. This problem was shown to be NP -hard by Shachnai and Tamir [19], and Golubchik *et al.* [8] show that even if all disks are equal, i.e, have same load and storage capacities, the problem remains NP -hard.

The total load capacity of the server is $B_T = \sum_{j=1}^N B_j$. The movies considered to be stored in the server are F_1, F_2, \dots, F_f . Given popularity parameters we compute the number r_i of expected requests for each movie i at any time, such that $\sum r_i = B_T$. Consider for example that we have a server with two hard disks (Disk 1 and Disk 2) each one with $C = 2$ and $B = 6$. There are three movies F_1, F_2 and F_3 with expected requests $r_1 = 8, r_2 = 2$ and $r_3 = 2$. One optimal solution is given in Figure 1. One copy of movie F_1 is stored in disk 1 and disk 2, a copy of movie F_3 is stored in disk 1, and a copy movie F_2 is stored in disk 2. Notice that in this case all load capacity of the disks are used. We call a placement perfect when all load capacity is used, i.e, all requests are satisfied.

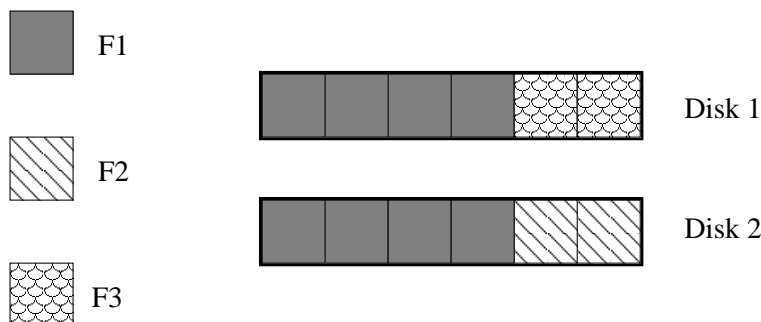


Figure 1: An optimal solution for the given video server.

We can also consider the following problem: given a set of requests for a set of movies, construct a server using

the minimum number of disks such that all requests are satisfied. This problem is NP -hard since, given an instance for the data placement with N disks, a perfect placement exists, if and only if we can find a packing for all requests using at most N disks. When all disks are equal, we can see this data placement problem as a special case of the CCBP problem. In this case we have an instance $I = (L, s, c, C, Q, B)$, where each item $i \in L$ is a request of class $c_i \in Q$ (the movie type). All items have the same size and C is the storage capacity of the disks, i.e., the number of different movies that the disk can store. In this case B is the maximum number of simultaneous requests that a disk can attend, i.e., the load capacity. That is, we want to construct a video server storing the videos and distributing all the requests minimizing the number of used disks.

3 Practical Approximation Algorithms

In this section we consider the problem where all items have unit size. As we saw, this problem is NP -hard and has applications in the data placement problem for video-on-demand. In this case, we can assume that items are given as a list of sets U_1, \dots, U_Q , where each set U_i has n_i items of unit size with class i . Each bin packs at most B items of at most C different sets. The problem is to pack all sets of items in the minimum number of bins.

We adapt here an algorithm known as Moving-Window (MW) first presented by Shachnai and Tamir [19] and also used later by Golubchik *et al.* [8] and Kashyap and Khuller [13]. In these previous works the algorithm was considered for the knapsack version of the problem, where one wishes to pack the maximum number of items in a given number of bins.

Moving-Window (MW): The algorithm keeps a vector $R = (R[1], R[2], \dots, R[Q])$ representing non-packed items in such a way that $R[i]$ is the number of remaining items to be packed of some set U_j . The vector is maintained in non-decreasing order of the values $R[i]$ during all the execution of the algorithm. If at any given moment, it packs part of the items represented by $R[i]$, then the vector must be reordered.

In any iteration of the algorithm, it tries to pack C different sets creating a new bin. For that, the algorithm keeps a window of C sets. At first, the window goes from $R[1]$ to $R[C]$. If $\sum_{i=1}^C R[i] \geq B$ then the algorithm packs the corresponding sets of $R[1], R[2], \dots, R[j]$, where $j \leq C$ is the first index such that $\sum_{i=1}^j R[i] \geq B$. Notice that $R[j]$ may be partially packed. The totally packed sets are removed from the vector. If $\sum_{i=1}^C R[i] < B$ then the algorithm moves the window to the right, until the first time that the window includes C sets such that their total size is greater than or equal to B . If this is the case, the C sets are packed and the vector R is reordered (if the last considered set was partially packed). Then the algorithm restarts. If in some iteration, the window reaches the end of the vector R , i.e., the C largest sets have total size smaller than B , then the algorithm generates bins by packing entirely C sets in each bin, with exception perhaps in the last bin that can pack less than C sets.

Let B_1, \dots, B_N be the bins created by the algorithm MW in the order they were created. Let N_F be the number of full bins and N_C be the number of bins that are not full which we call colored. Let $N = N_F + N_C$. Notice that bins B_1, \dots, B_{N_F} , are the full bins since when the algorithm creates the first non-full bin, when the window reaches the end of R and the C largest sets have total size smaller than B , then all other generated bins becomes non-full having C different sets each except perhaps the last.

The following two lemmas for the CCBP problem are a direct extension of the work of Golubchik *et al.* [8] done for the knapsack version of the problem.

Lemma 3.1 *If any of the first N_F bins produced by the algorithm MW packs less than C different sets (classes),*

then the algorithm produces an optimal solution.

Proof. Let B_i be the first bin, among the first N_F bins, that packs less than C different sets. In this case, the window must start from $R[1]$ and goes until $R[j']$ for some $j' \leq C - 1$. The vector R is ordered such that $R[j'] \leq R[j' + 1] \leq \dots \leq R[Q]$. Therefore, any $C - 1$ remaining sets have total size greater than B . That is, even if the set $R[j']$ was partially packed, all other created bins must be full (except perhaps the last), because the remaining items of a partially packed set with $C - 1$ sets have total size greater than B . \square

This way, we assume that for each one of the N_F first bins, the algorithm packs in each iteration, exactly C different sets and that at most one of these sets is partially packed. Clearly, for the remaining N_C bins, all of them packs totally C different sets except perhaps the last bin.

Let $\text{OPT}(I)$ be the number of bins used by an optimal solution to pack instance I . We assume that $N_F \leq \text{OPT}(I) - 1$, otherwise the algorithm generated an optimal solution. We have the following result.

Lemma 3.2 *After the MW algorithm has created the first $\text{OPT}(I)$ bins, there exists at most N_F sets to be packed.*

Proof. Notice that the number of different sets must satisfy $Q \leq \text{OPT}(I)C$. Since each one of the full bins packs C different sets, where one of these may be partially packed, then the algorithm partially packs at most N_F sets. These partially packed sets can be seen as new sets that are considered by the algorithm during its execution. That is, we can assume that the algorithm packs at most $Q + N_F$ different sets. Also remember that each one of the N_C colored bins packs entirely C different sets. Since each one of the first $\text{OPT}(I)$ bins packs C different sets and $Q \leq \text{OPT}(I)C$ we conclude that it remains at most N_F sets that are packed in extra colored bins. \square

With this result we can give the approximation factor of the MW algorithm.

Theorem 3.3 *The MW algorithm has an asymptotic approximation factor of $(1 + \frac{1}{C})$ for the CCBP problem when all items have unit size.*

Proof. Let I be an instance for the CCBP problem where all items have unit size. From Lemma 3.2, after the algorithm generates the first $\text{OPT}(I)$ bins, it remains at most N_F sets to be packed. Since each one of the generated bins packing these sets is colored, each bin entirely packs C different sets and then, the number of extra bins created can be bounded by

$$\left\lceil \frac{N_F}{C} \right\rceil \leq \frac{\text{OPT}(I) - 1}{C} + 1 = \frac{\text{OPT}(I)}{C} - 1/C + 1.$$

We can bound the number of generated bins by $\text{OPT}(I) + \text{OPT}(I)/C + 1$. \square

Proposition 3.4 *The bound of Theorem 3.3 is tight.*

Proof. Consider an input instance I having $N(C - 2)$ big sets with $2p + 2$ items each, and $2N$ small sets with p items each. The bin load capacity is $B = (C - 2)(2p + 2) + 2p + 2$ items. Notice that $(C - 2)$ big sets with two small sets does not fill the bin load capacity. When the MW algorithm is executed over this instance, the first generated bin packs one small set, $(C - 2)$ big sets entirely and another big set partially. The remaining items of the last packed big set becomes a small set with p items. Notice that the MW algorithm generates $N(C - 2)/(C - 1)$ bins by packing big sets and one small set that is a residual part of a big set. After that, there remain $2N$ small sets that are packed in $2N/C$ additional bins. When N and C increase enough, the number of bins tends to $N + N/C$.

An optimal packing of this instance uses N bins. In this packing, each bin packs $(C - 2)$ big sets and two small sets. \square

Notice that the MW algorithm is based in a heuristic that tries to pack C different sets in each bin. But the way the algorithm works, it tends to pack small and large sets in different bins. A good heuristic is to pack large and small sets together, in such a way that each generated bin has a good use of its load capacity, while trying to pack C different sets in each bin. For that, we propose a new algorithm that we call Modified-Moving-Window (MW').

Modified-Moving-Window (MW'): This algorithm is similar to the MW algorithm in such a way that it also keeps a window of size C over a vector $R = (R[1], R[2], \dots, R[Q])$ that is maintained ordered in non-decreasing order of the values $R[i]$. The algorithm also moves a window of size C until the total size of the sets in the window contains B or more items. In the MW' algorithm, the vector R is a circular list. At first, the window consists of the sets $R[1], \dots, R[C]$. If the total size of these sets is greater than or equal to B , then the algorithm packs the sets $R[1], \dots, R[j]$, where $j \leq C$ is the first index such that $\sum_{i=1}^j R[i] \geq B$, with the last set $R[j]$ probably partially packed. If the total size of these sets is smaller than B then instead of doing a move to the right, as in the original MW algorithm, the algorithm performs a move to the left and considers the sets $R[Q], R[1], \dots, R[C - 1]$. The algorithm performs moves to the left until the total size of the C sets are greater than or equal to B . In this case it packs the C sets and restarts. If the algorithm performs C moves to the left, and then considers the largest C sets, and this sets have total size less than B , then the algorithm generates a packing like the original MW algorithm, by packing entirely C sets in each bin.

It is not hard to prove similar results to Lemma 3.1 and Lemma 3.2 to the MW' algorithm. Using the same arguments of Theorem 3.3 we can prove the following result.

Theorem 3.5 *The MW' algorithm has an asymptotic approximation factor of $(1 + \frac{1}{C})$ for the CCBP problem when all items have unit size.*

Notice that this bound is tight since the algorithm MW' generates the same solution generated by the algorithm MW for the instance presented in Proposition 3.4. The advantage of the MW' algorithm is to try to pack small sets with large ones trying to guarantee a good filling of the bins, since it tries to pack the maximum number of small sets with large sets. To see this, consider for example an instance I that consists of $2n$ small sets, each one with one item, n large sets with 5 items each and n medium sets with 2 items each. Suppose $B = 7$ and $C = 3$. The MW algorithm first generates n bins by packing two medium sets and part of another large set. After that, it generates $2n/3$ new bins to pack the small sets. The MW' algorithm first generates n bins such that each one packs two small sets and a large set. The remaining medium sets are packed in $n/3$ bins.

Shachnai and Tamir [23] proved that the FF algorithm when applied to this problem have an approximation factor of 2. We can consider another simple approach to solve the problem using ideas similar to the ones used in algorithms FFD and BFD (see Coffman *et al.* [2]), but as we will see this approach does not gives better results.

Algorithm BFFD: The algorithm first sorts the sets U_1, \dots, U_Q in non-increasing order of their size and then apply the FF algorithm in the list obtained concatenating these sets.

Theorem 3.6 *The BFFD algorithm has an asymptotic approximation factor equal to 2 for the CCBP problem when all items have unit size.*

Proof. Let B_1, \dots, B_N be the bins created by the algorithm, N_F be the number of full bins and N_C be the number of colored bins. Clearly, $N_F \leq \text{OPT}$ and each bin that is not full must be colored except perhaps the last generated bin.

Also notice that two different bins that are colored cannot have items of a same color. Since $CN_C/C \leq \lceil Q/C \rceil \leq \text{OPT}$ we get that $N_C \leq \text{OPT}$. Then we can bound the number of generated bins by $N \leq 2\text{OPT} + 1$. \square

Since this algorithm does not try to optimize the class usage in the packing, it can generate poor quality packings. In fact, we show in the next proposition that the bound of Theorem 3.6 is tight.

Proposition 3.7 *The bound of Theorem 3.6 is tight.*

Proof. Let $I = (L, s, c, C, Q, B)$ be an instance to the CCBP problem where all items have unit size. Let the size of the bins be $B = C^2$. Suppose the input list of items consists of one big set with C^3 items and C^2 small sets with one item each. The BFFD algorithm first packs the big set in C^3/C^2 bins and the small sets in C^2/C bins giving a total of $2C$ bins. An optimal solution uses C bins packing in each bin $C^2 - (C - 1)$ items of the big set and $C - 1$ small sets. The remaining $C(C - 1)$ items of the big set, and C small sets can be packed in 2 extra bins. \square

Now we consider the case where items in each set may have different sizes. This case is also interesting for applications of the data-placement problem to video-on-demand servers. Suppose that users have different network access speeds. In this case, requests for load resources may have different sizes. This case can be mapped to the case in the CCBP problem where items have different sizes. Also notice that even if the items have different sizes, in practical instances it is expected that the size of the item is not too large. So, suppose that the maximum size of an item is an integer bounded by B/m for some $m \geq 1$. Problems with this restriction are also called parametric packing problems [16, 3]. Given an integer m , we denote this version of the problem as Parametric Class Constrained Bin Packing (CCBP_m) problem.

Let I be an instance of the CCBP_m problem where each item has size bounded by B/m . Assume that the input instance I consists of sets U_1, \dots, U_Q . We now present an algorithm to pack this instance. Although items may have different sizes, suppose that each item with size s greater than 1 is broken into s unit size pieces. Now apply the MW algorithm for this modified instance. Now consider this packing for the original items. For each full bin it may happen that the last item packed is fractionally packed. For each bin where this happens, remove the item from the bin. Notice that there are at most N_F items removed from the generated packing. For these remaining items, generate new bins packing at least $\min\{m, C\}$ items in each bin except perhaps in the last bin.

Theorem 3.8 *There exists an algorithm for the CCBP_m problem, for some $m \geq 1$, with asymptotic approximation factor equal to $(1 + 1/\min\{m, C\} + 1/C)$.*

Proof. From Theorem 3.3, the packing generated when items are fractionally packed, uses at most $(1+1/C)\text{OPT}(I)+1$ bins. Notice that the number of items fractionally packed in this packing is bounded by N_F , since the first N_F bins are the only ones that are full. These N_F extra items can be packed in at most $\lceil N_F/\min\{m, C\} \rceil$ extra bins. \square

4 The Online CCBP Problem

From now on, we assume that the load capacity of the bin is $B = 1$, and each item e has size $0 < s_e \leq 1$. In this section we consider the online class constrained bin packing problem. In this case each item in the list of items $L = (a_1, \dots, a_n)$, is packed without knowledge of subsequent items in the list. In subsection 4.1 we present lower bounds for any bounded space algorithm, in subsection 4.2 we present and analyze an algorithm based on the First-Fit strategy and finally in subsection 4.3 we present another online algorithm with a better competitive ratio.

4.1 Lower bounds for bounded space algorithms

In this section we present inapproximability results for the bounded space online CCBP problem. In this case, the basic strategy is to compare the result obtained by any algorithm with the optimum offline packing.

Theorem 4.1 *Let l be a constant, then the l -bounded space online CCBP problem does not admit an algorithm with constant competitive ratio. Moreover the competitive ratio of any online algorithm is $\Omega(\sqrt{|L|})$, where $|L|$ is the number of items in an input instance.*

Proof. Let \mathcal{A} be an algorithm for the l -bounded space online CCBP problem. Consider an instance I , such that $|L| = n^2l$, $Q = nl$, and n is divisible by C . The list L have nl different classes and all items have size $1/Cn$. Consider that $L = L_1 \parallel \dots \parallel L_n$, where each $L_i = (a_1, \dots, a_{nl})$ is a sequence of nl items where each a_j has class j .

Let t_i be the time immediately after the algorithm has packed the list L_i . At time t_1 the algorithm \mathcal{A} can have at most l open bins. Since each item of the first sequence is of a different class, the algorithm uses at least nl/C bins to pack L_1 , where at least $nl/C - l$ of these bins are closed. When the packing of the list L_2 starts, the algorithm has at most l open bins that can pack at most lC items of the sequence L_2 . To pack this sequence, the algorithm uses at least $(ln - lC)/C$ new bins. This is also valid for the other sequences L_3, \dots, L_n .

Therefore, to pack the list L , the algorithm \mathcal{A} uses at least

$$n(nl/C) - (n-1)l = n^2l/C - (n-1)l$$

bins.

Since all items have size $1/Cn$, an optimal offline solution can use at most ln/C bins, by packing Cn items in each bin. Therefore, the competitive ratio must be at least

$$\lim_{n \rightarrow \infty} \frac{n^2l/C - (n-1)l}{nl/C} = n - C,$$

which is $\Omega(\sqrt{|L|})$. Notice that this holds for any n . □

In Theorem 4.1 items may have arbitrary small sizes. If all items have size at least ε , for some constant ε , we may also obtain an inapproximability result using similar arguments. Notice that in this case, any simple algorithm has a competitive ratio of $1/\varepsilon$.

Theorem 4.2 *Let l and $\varepsilon < 1/C$ be constants, then any algorithm for the l -bounded space online CCBP problem has competitive ratio $\Omega(1/(C\varepsilon))$.*

Proof. Suppose that $1/\varepsilon$ divides n and we have the same instance presented in Theorem 4.1, modified such that all items have size equal to ε . In this case any algorithm uses at least $n^2l/C - (n-1)l$ bins. An optimal offline solution packs items of a given class in $n\varepsilon$ bins. To pack L an optimal offline algorithm uses at most $n^2l\varepsilon$ bins.

Therefore, the competitive ratio is at least

$$\lim_{n \rightarrow \infty} \frac{n^2l/C - (n-1)l}{n^2l\varepsilon} = \frac{1}{C\varepsilon}.$$

□

Given these negative results, for the remainder of this section we only consider the unbounded space online CCBP problem.

4.2 The First-Fit Algorithm

Given an online algorithm \mathcal{A} for the bin-packing problem, we can obtain an online algorithm \mathcal{A}^* for the online CCBP problem in a straightforward manner. To pack the next item e , the algorithm \mathcal{A}^* works as follows: Let c_e be the class of the item e , \mathcal{B} be the list of bins in the order they were opened. Let \mathcal{B}_e be the list of bins of \mathcal{B} , in the same order of \mathcal{B} , where each bin has at least one item of class c_e or has items of at most $C - 1$ different classes. The item e is packed with algorithm \mathcal{A} into the bins of \mathcal{B}_e .

One of the most famous algorithm for the bin-packing problem is the First-Fit (FF) algorithm. This algorithm packs the next item into the first bin, in the order they were opened, that has sufficient space for the item.

In this section we show that the competitive ratio of the algorithm FF* is in $[2.7, 3]$. We note that the upper bound was previously shown by Dawande *et al.* [4]. Notice that the algorithm FF* is online, since it only looks for the item it is packing and it is unbounded since it keeps all bins opened. In fact it closes a bin only if the bin is full. This algorithm is used in subsequent sections.

Now we show that the algorithm FF* cannot have a competitive ratio better than 2.7. We first give an intuitive lower bound of 2.666 and then we present the lower bound of 2.7.

Theorem 4.3 *There is an instance I_n with n items, $n \geq 1$, for the online CCBP problem such that $\text{FF}^*(I_n)/\text{OPT}(I_n) \rightarrow 2.666$ as $n \rightarrow \infty$.*

Proof. Let I be an instance with an input list of items $L = L_a || L_b || L_c || L_d$. Let C be the maximum number of classes allowable in each bin. The list $L_a = (a_1, \dots, a_{(C-1)6N})$ is such that each item a_i has class i , $i = 1, \dots, (C-1)6N$ and each item has size α , which is a very small value. This list is followed by a list $L_b = (b_1, \dots, b_{6N})$, where each item b_i has class $r = 6N(C-1) + 1$, and size $1/7 + \epsilon$. In the list $L_c = (c_1, \dots, c_{6N})$ each item c_i has size $1/3 + \epsilon$ and class r . Finally, in the list $L_d = (d_1, \dots, d_{6N})$ each item d_i has size $1/2 + \epsilon$ and class r .

Notice that α must satisfy

$$\alpha \leq \frac{1 - 126\epsilon}{42(C-1)}.$$

The FF* algorithm packs the list L_a in $\frac{6N(C-1)}{C}$ bins, the list L_b in N bins, the list L_c in $3N$ bins and the list L_d in $6N$ bins.

An optimal (offline) solution uses at most $6N$ bins. This packing is obtained by packing one item of L_d , one item of L_c , one item of L_b and $C - 1$ items of the list L_a in only one bin.

This gives a lower bound of

$$\lim_{N, C \rightarrow \infty} \frac{\frac{(C-1)6N}{C} + 10N}{6N} = 2.666.$$

□

The previous lower bound can be improved using an intricate instance presented by Johnson *et al.* [11] that provides a lower bound of 1.7 for the FF algorithm in the bin packing problem.

Theorem 4.4 *The competitive ratio of the algorithm FF* is at least 2.7.*

Proof. Consider an instance I such that each bin can pack at most C different classes. The input list L is the concatenation of four lists: $L = L_a || L_b || L_c || L_d$. In the list $L_a = (a_1, \dots, a_{5N(C-1)})$, each item a_i has class i , for $i = 1, \dots, 5N(C-1)$, and size α , which is a very small value. The list L_a is followed by an instance similar to

the one presented by Johnson *et al.* [11] that provides a lower bound of 1.7 for the FF algorithm in the bin packing problem. In the list $L_b = (b_1, \dots, b_{5N})$ each item b_i has size $1/7 + y_i$, where $y_i \in \mathcal{R}$, for $i = 1, \dots, 5N$. In the list $L_c = (c_1, \dots, c_{5N})$ each item c_i has size $1/3 + w_i$, where $w_i \in \mathcal{R}$, for $i = 1, \dots, 5N$. In the list $L_d = (d_1, \dots, d_{5N})$ each item d_i has size $1/2 + \varepsilon$. All items in the lists L_b , L_c and L_d have class $5N(C - 1) + 1$.

Notice that α must satisfy

$$\alpha \leq \frac{1 - 42(y_{max} + w_{max} + \varepsilon)}{42(C - 1)},$$

where y_{max} (resp. w_{max}) is the maximum value among all y_i (resp. w_i), $i = 1, \dots, 5N$.

The FF* algorithm packs the list L_a in $\frac{5N(C-1)}{C}$ bins, the list L_b in N bins, the list L_c in $2.5N$ bins and the list L_d in $5N$ bins.

That is,

$$\text{FF}^*(I) \geq \frac{5N(C-1)}{C} + N + 2.5N + 5N.$$

An optimal solution uses $5N + 2$ bins (see [11]), by packing one item of each list L_b , L_c and L_d and $C - 1$ items of the list L_a .

Therefore, the competitive ratio of the algorithm FF* is at least

$$\lim_{N, C \rightarrow \infty} \frac{5N(C-1)/C + 8.5N}{5N + 2} = 2.7.$$

□

4.3 A 2.75-competitive algorithm

In this section we present an algorithm, denoted by \mathcal{A}_C (Figure 2), with competitive ratio in the interval $(2.666, 2.75]$

ALGORITHM $\mathcal{A}_C(L, s, c, C, Q)$

1. Let $\mathcal{P}_i \leftarrow \emptyset$, for $i = 1, 2, 3$.
 2. For each $e \in L$ do
 3. if $s(e) \in (\frac{1}{2}, 1]$ then $k \leftarrow 1$.
 4. if $s(e) \in (\frac{1}{3}, \frac{1}{2}]$ then $k \leftarrow 2$.
 5. if $s(e) \in (0, \frac{1}{3}]$ then $k \leftarrow 3$.
 6. Let \mathcal{P}'_k be the sublist of bins in \mathcal{P}_k having items of class $c(e)$ or with at most $C - 1$ classes, preserving the order of the bins in \mathcal{P}_k .
 7. If possible pack the item e into the bins \mathcal{P}'_k using the algorithm FF*.
 Otherwise, pack e into a new empty bin in \mathcal{P}_k .
 8. Return $\mathcal{P}_1 \parallel \mathcal{P}_2 \parallel \mathcal{P}_3$.
-

Figure 2: Algorithm \mathcal{A}_C .

To prove the competitive ratio of the algorithm \mathcal{A}_C , we use the following lemma (The proof can be found in [16]).

Lemma 4.5 Suppose X, Y, x, y are real numbers such that $x > 0$ and $0 < X < Y < 1$. Then

$$\frac{x + y}{\max\{x, Xx + Yy\}} \leq 1 + \frac{1 - X}{Y}.$$

We also use the following result that is a straightforward extension of some results in [4].

Lemma 4.6 Let I be an instance of the online CCBP problem such that every item has size at most ε . Let \mathcal{P} be the set of bins generated by the algorithm FF^* , applied over the instance I , that are filled by less than $1 - \varepsilon$. Then: (i) Each bin in \mathcal{P} , which is not the last generated bin, is colored. (ii) There are no items of a same color in two different bins of \mathcal{P} .

Theorem 4.7 Algorithm \mathcal{A}_C has a competitive ratio of 2.75.

Proof. Let L_i be the list of items packed in \mathcal{P}_i , for $i = 1, 2, 3$.

Note that all bins of \mathcal{P}_1 have exactly one item with size greater than $\frac{1}{2}$. In fact we cannot pack more than one item of L_1 per bin. Therefore,

$$|\mathcal{P}_1| \leq \text{OPT}(I) \quad (1)$$

$$\frac{1}{2}|\mathcal{P}_1| \leq s(L_1). \quad (2)$$

The packing \mathcal{P}_2 has exactly two items per bin, except perhaps the last, each item with size at least $\frac{1}{3}$. Therefore,

$$(|\mathcal{P}_2| - 1)\frac{2}{3} \leq s(L_2). \quad (3)$$

Let \mathcal{P}'_3 be the set of bins in \mathcal{P}_3 that are filled by at least $\frac{2}{3}$ and \mathcal{P}''_3 the remaining bins (i.e., $\mathcal{P}''_3 = \mathcal{P}_3 \setminus \mathcal{P}'_3$). The following is valid

$$(|\mathcal{P}'_3|)\frac{2}{3} \leq s(L'_3). \quad (4)$$

where L'_3 is the set of items packed in \mathcal{P}'_3 . Let $N_A = |\mathcal{P}_1|$ and $N_B = |\mathcal{P}_2| + |\mathcal{P}'_3| - 1$. Since $\text{OPT}(I) \geq s(I) \geq s(L_1) + s(L_2 \| L'_3)$ from inequalities (2)–(4) we have

$$\begin{aligned} \text{OPT}(I) &\geq s(I) \geq s(L_1) + s(L_2 \| L'_3) \\ &\geq \frac{1}{2}N_A + \frac{2}{3}N_B. \end{aligned} \quad (5)$$

From inequalities (1) and (5) we have

$$\text{OPT}(I) \geq \max\{N_A, \frac{1}{2}N_A + \frac{2}{3}N_B\}. \quad (6)$$

From Lemma 4.5 we have that

$$|\mathcal{P}_1| + |\mathcal{P}_2| + |\mathcal{P}'_3| \leq \frac{N_A + N_B}{\max\{N_A, \frac{1}{2}N_A + \frac{2}{3}N_B\}} \text{OPT}(I) + 1 \quad (7)$$

$$\leq 1.75 \text{OPT}(I) + 1. \quad (8)$$

Now, consider the packing \mathcal{P}''_3 . From Lemma 4.6, we have

$$|\mathcal{P}''_3| - 1 \leq \frac{Q}{C} \leq \text{OPT}(I). \quad (9)$$

The proof can be completed summing the inequalities (8) and (9).

$$\begin{aligned}\mathcal{A}_C(I) &= |\mathcal{P}_1| + |\mathcal{P}_2| + |\mathcal{P}'_3| + |\mathcal{P}''_3| \\ &\leq 1.75 \text{OPT}(I) + \text{OPT}(I) + 2 = 2.75 \text{OPT}(I) + 2.\end{aligned}$$

□

Notice that the same instance used to prove a lower bound for the algorithm FF* in Theorem 4.3 can be used to prove a lower bound for the \mathcal{A}_C algorithm.

Theorem 4.8 *There is an instance I for the online CCBP problem such that $\mathcal{A}_C(I)/\text{OPT}(I) \geq 2.666$.*

5 An APTAS for Bounded Number of Classes

In this section we present an APTAS for the offline VCCBP problem. The input instance for this problem is a tuple $I = (L, s, c, w, C, Q, B)$ where $w : \{1, \dots, T\} \rightarrow \mathbb{R}^+$ is a function of bins size. The problem is to find a packing of all items minimizing the total size of used bins. In this section we assume that the maximum size of a bin is $B = 1$ and that the number of different classes Q in the input instance is bounded by a constant.

In subsection 5.1 we present the algorithm of Dawande, Kalagnanam and Sethuraman [5, 4] and show in what points their algorithm failed to be an APTAS. In subsection 5.2 we present an APTAS for the VCCBP problem. Given an ε , we will show an algorithm \mathcal{A} that runs in polynomial time and produces a packing for a given instance such that $\mathcal{A}(I) \leq (1 - O(\varepsilon))\text{OPT} + \beta$, where β is a constant.

As was noticed by Dawande *et al.* [5, 4], we only use bins of size at least ε , since this condition does not affect too much the cost of the solution, i.e, the algorithm remains an APTAS.

5.1 The Algorithm of Dawande, Kalagnanam and Sethuraman

In this section we give a brief description of the algorithm of Dawande *et al.* [5, 4] and present the points where their algorithm fails. The algorithm uses a shifting technique presented by Fernandez de la Vega and Lueker [6].

Let $I = (L, s, c, w, C, Q, 1)$ be an instance for the VCCBP problem and let L_b be the items in L with size at least ε^2 (big items) and let L_s be the remaining items in L (small items).

Let $n = |L_b|$. The algorithm sorts the list L_b in non-increasing order of size and partition this list into groups (lists) L_1, \dots, L_M , each one with $\lceil n\varepsilon^2 \rceil$ items except perhaps the last list that can have less than $\lceil n\varepsilon^2 \rceil$ items. Call the first item in each group as the group-leader. Let L'_i be the list having $|L'_i| = |L_i|$ items, where each item has size equal to the size of the group-leader of L_i . Let $L' = L'_1 \parallel \dots \parallel L'_M$.

For the list L' it is possible to generate all configurations of bins in constant time since the number of different items size is bounded by a constant M , the number of different item colors is also bounded by a constant Q and the maximum number of items that can be packed in a bin is $1/\varepsilon^2$. Let $t = MQ$. Given an item size and an item color, denote by d_i the number of items of this type $i \in [t]$.

Let N be the total number of bin configurations. Let x_j be a variable that represents the number of times a configuration $j \in [N]$ is used in a solution, a_{ij} be the coefficient that represents the number of times an item type $i \in [t]$ is used in configuration j and w_j the size of the bin used in configuration j . The next step of the algorithm is to solve the following linear program:

$$\begin{aligned}
& \min \sum_{j=1}^N w_j x_j \\
& \sum_{j=1}^N a_{ij} x_j \geq d_i \quad \forall i \in [t] \quad (1) \quad (\text{LP}) \\
& x_j \geq 0 \quad \forall j \in [N]. \quad (2)
\end{aligned}$$

The algorithm solves this linear program and generates an integer solution by rounding up the variables x . The solution is a packing for the list L' that is used to generate a packing for the list L_b .

The next step of the algorithm is to pack the small items in the solution provided by the linear program. To do this, it uses the FF* algorithm.

Dawande *et al.* [5, 4] claimed that this algorithm is an APTAS for the VCCBP problem.

The list L_b was partitioned into lists $L_1 \parallel \dots \parallel L_M$. Let L''_i be a list having $|L''_i| = |L_i|$ items, where each item has size equal to the group-leader of the list L_{i+1} , for $i = 1, \dots, M-1$, and L''_M be an empty list. Let $L'' = L''_1 \parallel \dots \parallel L''_M$. Clearly $\text{OPT}(L'') \leq \text{OPT}(L_b)$.

Dawande *et al.* claimed that the following relation is valid

$$\text{OPT}(L') \leq \text{OPT}(L'') + \lceil n\varepsilon^2 \rceil \leq \text{OPT}(L_b) + \lceil n\varepsilon^2 \rceil,$$

given the argument that L' and L'' differ only in their first and last groups. This way, given a packing for the list L'' it is easy to construct a packing for the list $L'_2 \parallel \dots \parallel L'_M$. Since $|L'_i| = |L''_{i-1}|$, for $i = 2, \dots, M$, and their items size are the same, this seems to be true, but notice that the color of items of L'_i and L''_{i-1} may be different. Then, it is not clear how to construct a packing for $L'_2 \parallel \dots \parallel L'_M$ given a packing for L'' .

Let B be the number of bins used by their algorithm. After packing the small items using the first-fit strategy, they claimed that at least $B - \lceil \frac{Q}{\varepsilon} \rceil$ bins have residual load capacity at most ε . This is also not true. Suppose all small items have different colors from the big items. It is easy to construct examples where optimal packings for the big items given by the linear program have all bins with C different colors and the residual space is larger than a given ε . This way no small item will be packed in the bins given as a solution of the linear program, and then all these bins will have residual load capacity greater than ε .

5.2 An APTAS for the VCCBP Problem

In this section we present an APTAS for the VCCBP problem. In the next subsection we show how to pack big items doing a linear rounding for each different color. The algorithm to pack the big items generates a polynomial number of packings for them, and also provides information of how to pack small items. In the following subsection, we present an algorithm to pack the small items that is based in the solution of a linear program. The algorithm generates a polynomial number of packings such that at least one is very close to the optimal.

5.2.1 Packing Big Items with Linear Rounding

Let L_b be the items in L with size at least ε^2 (big items) and let L_s be the remaining items in L (small items). In this section we show how to do the linear rounding for the big items and generate a packing for them.

The algorithm that packs the list L_b , denoted by A_{LR} , uses a shifting technique, presented by Fernandez de la Vega and Lueker [6], and considers only items with size at least ε^2 . The algorithm A_{LR} returns a pair $(\mathcal{P}_B, \mathbb{P})$, where \mathcal{P}_B is a packing for a list of very big items and \mathbb{P} is a set of packings for the remaining items of L_b .

For the use of the linear rounding technique, we use the following notation: Given two lists of items X and Y , let X_1, \dots, X_Q and Y_1, \dots, Y_Q be the partition of X and Y respectively in colors, where X_c and Y_c have only items of color c for each $c \in [Q]$. We write $X \preceq Y$ if there is an injection $f_c : X_c \rightarrow Y_c$ for each $c \in [Q]$ such that $s(e) \leq s(f(e))$ for all $e \in X_c$.

For any instance X , denote by \overline{X} the instance with precisely $|X|$ items with size equal to the size of the smallest item in X . Clearly, $\overline{X} \preceq X$.

The Algorithm also uses the variant of the First-Fit (FF*) that we presented in section 4.2.

The algorithm A_{LR} is presented in Figure 3. It proceeds as follows: Let L_1, \dots, L_Q be the partition of the input list L_b into colors $1, \dots, Q$ and let $n_c = |L_c|$ for each color c . The algorithm A_{LR} sorts each list L_c in non-increasing order of items size and then partition the list L_c into at most $M = \lceil 1/\varepsilon^3 \rceil$ groups $L_c^1, L_c^2, \dots, L_c^M$, where $L_c = L_c^1 \parallel \dots \parallel L_c^M$. Each group has $\lfloor n_c \varepsilon^3 \rfloor$ items except perhaps the last list (with the smallest items) that can have less than $\lfloor n_c \varepsilon^3 \rfloor$ items.

Let $L_B = \cup_{c=1}^Q L_c^1$. The algorithm generates a packing \mathcal{P}_B of L_B with cost at most $O(\varepsilon)\text{OPT}(I)$ and a set \mathbb{P} with a polynomial number of packings for the items in $L_b \setminus L_B$. The packing \mathcal{P}_B is generated by the algorithm FF* with bins of size 1. The following is valid for the packing \mathcal{P}_B of the list L_B .

Lemma 5.1 $w(\mathcal{P}_B) \leq Q\varepsilon\text{OPT}(I)$.

Proof. Notice that the algorithm FF* packs at least one item per bin and since $|L_B| \leq Qn\varepsilon^3$ and each item has size at least ε^2 , we have $|L_B| \leq Q\varepsilon\text{OPT}(I)$. □

The algorithm generates a set of packings \mathbb{Q} , of polynomial size, for the list $(\overline{L_1^1} \parallel \dots \parallel \overline{L_1^{M-1}} \parallel \dots \parallel \overline{L_Q^1} \parallel \dots \parallel \overline{L_Q^{M-1}})$. This can be done in polynomial time as the next lemma guarantees.

Lemma 5.2 *Given an instance $I = (L_b, s, c, w, C, Q, B)$, where the number of distinct items sizes of each color is at most a constant M , the number of different colors is bounded by a constant Q and each item $e \in L_b$ has size $s_e \geq \varepsilon^2$, then there exists a polynomial time algorithm that generates all possible packings of L_b . Moreover, each bin of each generated packing has an indication of the possible colors that may be used by further small items.*

Proof. The number of items in a bin is bounded by $y = 1/\varepsilon^2$. The number of distinct type of items is bounded by MQ . The number of different configurations of bins is bounded by $r' = \binom{y+MQ+1}{y}$. If we want to indicate the colors of small items that should be packed in each configuration, the number of different configurations will be $r = r'2^Q$, which is a constant. Notice that we only generate configurations that satisfy the color constraints.

For each given configuration, we pack it with the smallest bin that has enough space to pack the configuration. The number of all feasible packings is bounded by $\binom{n+r}{n}$, which is bounded by $(n+r)^r$, which in turn is polynomial in n . □

Since $\overline{L_c^i} \succeq L_c^{i+1}$, $i = 1, \dots, M-1$ for each color c , it is easy to construct a packing for the list $L_1^2 \parallel \dots \parallel L_1^M \parallel \dots \parallel L_Q^2 \parallel \dots \parallel L_Q^M$, given a packing for the list $(\overline{L_1^1} \parallel \dots \parallel \overline{L_1^{M-1}} \parallel \dots \parallel \overline{L_Q^1} \parallel \dots \parallel \overline{L_Q^{M-1}})$.

ALGORITHM $A_{LR}(L_b)$

Input: List L_b with n items, each item $e \in L_b$ with size $s_e \geq \varepsilon^2$.

Output: A pair $(\mathcal{P}_B, \mathbb{P})$, where \mathcal{P}_B is a packing and \mathbb{P} is a set of packings, where $\mathcal{P}_B \cup \mathcal{P}'$ is a packing of L_b for each $\mathcal{P}' \in \mathbb{P}$.

1. Partition L_b into lists L_c for each color $c = 1, \dots, Q$ and let $n_c = |L_c|$.
2. Sort each list L_c in non-increasing order of items size.
3. Partition each list L_c into $M \leq \lceil 1/\varepsilon^3 \rceil$ groups $L_c^1, L_c^2, \dots, L_c^M$, such that

$$\overline{L}_c^i \succeq L_c^{i+1}, \quad i = 1, \dots, M-1$$

$$\text{where } |L_c^i| = q_c = \lfloor n_c \varepsilon^3 \rfloor \quad \text{for all } i = 1, \dots, M-1,$$

$$\text{and } |L_c^M| \leq q_c.$$

4. Let $L_B = \cup_{c=1}^Q L_c^1$.
 5. Let \mathcal{P}_B be a packing of L_B obtained by the algorithm FF^* with bins of size 1.
 6. Let \mathbb{Q} be the set of all possible packings over the list $(\overline{L}_1^1 \parallel \dots \parallel \overline{L}_1^{M-1} \parallel \dots \parallel \overline{L}_Q^1 \parallel \dots \parallel \overline{L}_Q^{M-1})$, according to Lemma 5.2.
 7. Let \mathbb{P} be the set of packings for the items in $(L_1^2 \parallel \dots \parallel L_1^M \parallel \dots \parallel L_Q^2 \parallel \dots \parallel L_Q^M)$, using the packings $\mathcal{Q} \in \mathbb{Q}$.
 8. Return $(\mathcal{P}_B, \mathbb{P})$.
-

Figure 3: Algorithm to obtain packings for items with size at least ε^2 .

5.2.2 Packing the small items

Observe that algorithm A_{LR} generates a packing for very big items that costs at most $Q\varepsilon \text{OPT}(I)$, and a set \mathbb{P} of packings for the remaining big items. For a given packing $\mathcal{P} \in \mathbb{P}$, the algorithm marked colors of small items that should be packed in each bin of \mathcal{P} .

Let $\mathcal{P} = \{B_1, \dots, B_k\}$ be a packing of the list of items L_b and suppose we have to pack a list L_s of small items, with size at most ε^2 , into \mathcal{P} . The packing of the small items is obtained from a solution of a linear program. Let $N_i \subseteq [Q]$ be the set of possible colors that may be used to pack the small items in the bin B_i of the packing \mathcal{P} . For each color $c \in N_i$, define a non-negative variable x_c^i . The variable x_c^i indicates the total size of small items of color c to be packed in the bin B_i . Denote by $s(B_i)$ the total size of items already packed in the bin B_i and by $w(B_i)$ the load capacity of bin B_i . Consider the following linear program denoted by LPS:

$$\begin{aligned} & \max \sum_{i=1}^k \sum_{c \in N_i} x_c^i \\ s(B_i) + \sum_{c \in N_i} x_c^i & \leq w(B_i) \quad \forall i \in [k] \quad (1) \quad (\text{LPS}) \\ \sum_{i=1}^k x_c^i & \leq s(S_c) \quad \forall c \in [C], \quad (2) \end{aligned}$$

where S_c is the set of small items of color c in S .

The constraint (1) guarantees that the total size of items packed in each bin does not exceed its load capacity and constraint (2) guarantees that the sum of the values of variables x_c^i is not greater than the total size of small items.

Given a packing \mathcal{P} , and a list L_s of small items, the algorithm first solves the linear program LPS, and then packs small items in the following way: For each variable x_c^i it packs, while possible, the small items of color c into the bin B_i , so that the total size of the packed small items is at most x_c^i . The possible remaining small items are packed using the algorithm FF* into new bins of size 1. The algorithm to pack small items has polynomial time, since the linear program LPS can be solved in polynomial time.

The total size of small items that have to be packed in extra bins is at most

$$(s(L_s) - \sum_{i=1}^k \sum_{c \in N_i} x_c^i) + |\mathcal{P}| \varepsilon^2 Q$$

and then, these small items use at most

$$\left\lceil \frac{(s(L_s) - \sum_{i=1}^k \sum_{c \in N_i} x_c^i)}{(1 - \varepsilon^2)} + \frac{|\mathcal{P}| \varepsilon^2 Q}{(1 - \varepsilon^2)} \right\rceil + \lceil Q/C \rceil$$

new bins, since each bin is filled by at least $(1 - \varepsilon^2)$ except perhaps by at most $\lceil Q/C \rceil$ bins.

The algorithm packs the small items in each packing $\mathcal{P} \in \mathbb{P}$. In the end, the algorithm generates another set of packings \mathbb{P}' for all items. At least one of the generated packings has cost at most $(1 + O(\varepsilon))\text{OPT}(I) + \beta$, for a constant β . The algorithm returns the packing with smallest cost.

Now we prove that the presented algorithm is an APTAS for the VCCBP.

Theorem 5.3 *Let $I = (L, s, c, w, C, Q, B)$, be an instance for the VCCBP problem. The packing \mathcal{P} returned by the algorithm satisfies $w(\mathcal{P}) \leq (1 + O(\varepsilon))\text{OPT}(I) + \beta$, where $\beta = \lceil Q/C \rceil + 1$ is a constant.*

Proof. Let O be an optimal packing for the instance I . Let O' be the packing O without the small items and with the big items rounded according to the linear rounding of algorithm A_{LR} . Assume that each bin of O' has an indication of the colors of small items used in the corresponding bin of O . Clearly there exists a packing $O'' \in \mathbb{Q}$ with the same configurations of the bins of O' except that it can use smaller bins than the ones used in O' .

When the algorithm generates a packing \mathcal{P} for the list $L_1^2 \parallel \dots \parallel L_1^M \parallel \dots \parallel L_Q^2 \parallel \dots \parallel L_Q^M$ using the packing O'' with items $(\overline{L_1^1} \parallel \dots \parallel \overline{L_1^{M-1}} \parallel \dots \parallel \overline{L_Q^1} \parallel \dots \parallel \overline{L_Q^{M-1}})$, it is true that $w(\mathcal{P}) = w(O'') \leq w(O)$.

Let $\mathcal{P} = \{B_1, \dots, B_k\}$. Notice that we must have

$$w(O) \geq w(\mathcal{P}) + (s(L_s) - \sum_{i=1}^k \sum_{c \in N_i} x_c^i).$$

The total size of small items that are packed into new bins is at most

$$(s(L_s) - \sum_{i=1}^k \sum_{c \in N_i} x_c^i) + |\mathcal{P}| \varepsilon^2 Q.$$

The algorithm packs small items in bins of size 1 obtaining a new packing \mathcal{P}' . The total cost of the packing \mathcal{P}' is

$$w(\mathcal{P}') \leq w(\mathcal{P}) + \left\lceil \frac{(s(L_s) - \sum_{i=1}^k \sum_{c \in N_i} x_c^i)}{(1 - \varepsilon^2)} + \frac{|\mathcal{P}| \varepsilon^2 Q}{(1 - \varepsilon^2)} \right\rceil + \lceil Q/C \rceil \quad (10)$$

$$\leq \frac{w(O)}{(1 - \varepsilon^2)} + \frac{|\mathcal{P}| \varepsilon^2 Q}{(1 - \varepsilon^2)} + \lceil Q/C \rceil + 1 \quad (11)$$

$$\leq \frac{w(O)}{(1 - \varepsilon^2)} + \frac{\varepsilon Q w(O)}{(1 - \varepsilon^2)} + \lceil Q/C \rceil + 1. \quad (12)$$

The last inequality follows from the fact that $|\mathcal{P}| \leq |O|$ and the smallest size of a bin is ε . Using this result, Lemma 5.1 and the fact that Q is bounded by a constant we conclude the proof. \square

6 Experimental Results of the Practical Algorithms

In this section we provide experimental results for the algorithms MW, MW' and BFFD presented in Section 3. As we mentioned, these algorithms were developed motivated by the data placement problem in video servers. This problem is a special case of the CCBP problem. All these algorithms were implemented in C and we made a series of practical tests with them.

The instance set is constructed in some way to represent the real problem. A movie in MPEG format uses about 2Gbytes of space, and requires a transference rate of 3Mbits/sec (384Kbytes/sec) [1]. Suppose that the server uses disks of 100Gbytes of storage capacity with transference rate of 60Mbytes/sec. In this case, each disk have storage capacity $C = 50$ and load capacity $B = 160$.

We call *single-disk* server, the systems that are constructed in such a way that a entire copy of a movie is done in one disk. But most video servers uses *striped-disks* [1]. In this case, a video is broken into several pieces and each one of these pieces is stored in a different disk. This is done to increase the number of requests that can be attended by the system and to balance the load capacity of the disks. Suppose for example that each disk have transference rate of 60Mbytes/sec and storage capacity of 100Gbytes. Theoretically a disk can support 160 users simultaneously. If we strip the movie along 3 disks, and assume that users requests over the time are distributed uniformly among the three parts of the movie, then the striped-disk can support 480 simultaneously users requests to this movie. For our purposes, we can view each striped-disk as one disk with storage capacity equal to 300Gbytes and load capacity equal to 480. In practice it is better to use striped-disks to balance requests. Consider for example, a single-disk server where a copy of a movie A is in disk 1 and a copy of a movie B is in another disk 2, and there are 320 requests for the movie A and none to the movie B . The system becomes unable to attend 160 requests to the movie A . In a striped-disk system, where the first half part of movie A is stored in disk 1 while the last half part is stored in disk 2, it can attend more users if their requests are distributed along the movie in such a way that requests are divided through the two disks.

We have generated classes of instances represented by a tuple (Q, N, T) . The value Q corresponds to the number of different movies (different classes) and we consider that $Q \in \{250, 500, 1000\}$. The value N is the number of requests (number of items) and we assume that $N \in \{5000, 10000, 20000\}$. Finally the value T corresponds to the system type, where T is equal to SC for single-disk system or ST for striped-disk system. In the single-disk system, we have $C = 50$ and $B = 160$, and in the striped-disk system, we have $C = 150$ and $B = 480$.

The requests for movies are generated using the Zipf distribution [14]. This distribution was used previously to generate data for video-on-demand systems [1]. This distribution have the property that the generated data have

locality properties. In movies servers it is expected that recent movies are the most requested ones. It is expected that most of the requests goes to a small subset of the movies in the server. The Zipf distribution have this property. Let δ be a small positive number. The probability that the n -th movie among Q movies will be requested is p_n given as

$$p_n = \frac{c}{n^{(1+\delta)}}$$

where

$$c = \frac{1}{\sum_{i=1}^Q (1/i^{(1+\delta)})}$$

As δ increases, the distribution becomes more localized and as δ decreases the distribution becomes more uniformly. Considering $Q = 1000$, if $\delta = 0.0$, then 80% of the requests are to approximately 20% of the movies. If $\delta = 1.0$, then 80% of the requests are to approximately 0.3% of the movies. When $\delta = -1.0$ we get the uniform distribution where each movie have the same probability $1/Q$ to be requested.

We present some experimental results in Tables 1 and 2. All results were obtained in a few seconds. In the tests of these tables, we generate data using $\delta \in \{0.0, 0.5, 1.0\}$, $N \in \{5000, 20000\}$ and $Q \in \{250, 500, 1000\}$. The lower bound is given by $\max\{\lceil Q/C \rceil, \lceil N/B \rceil\}$. In Table 1 we consider single-disk system, and in Table 2 we consider the striped-disk system. We also performed tests with $N = 10000$ but we do not present the results here since we get similar results to the tests with $N = 5000$ and $N = 10000$. We observed that the BFFD algorithm generate good results and it becomes better for the striped-disk system. But in comparison with the MW and MW' algorithms it performs worst, since these algorithms generated optimal solutions for all tests. The MW and MW' shows to be very effective algorithms to be used in practical instances to construct video-on-demand servers.

Single-Disk 250 Movies 5000 Requests				500 Movies 5000 Requests			1000 Movies 5000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	BFFD	32	32	BFFD	34	32	BFFD	42	33
	MW	32		MW	32		MW	33	
	MW'	32		MW'	32		MW'	33	
$\delta = 0.5$	BFFD	34	32	BFFD	39	33	BFFD	48	36
	MW	32		MW	33		MW	36	
	MW'	32		MW'	33		MW'	36	
$\delta = 1.0$	BFFD	35.8	33	BFFD	40.6	34	BFFD	50	37.2
	MW	33		MW	34		MW	37.2	
	MW'	33		MW'	34		MW'	37.2	
Single-Disk 250 Movies 20000 Requests				500 Movies 20000 Requests			1000 Movies 20000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	BFFD	125	125	BFFD	125	125	BFFD	127	126
	MW	125		MW	125		MW	126	
	MW'	125		MW'	125		MW'	126	
$\delta = 0.5$	BFFD	126	126	BFFD	129.4	126	BFFD	138	128
	MW	126		MW	126		MW	128	
	MW'	126		MW'	126		MW'	128	
$\delta = 1.0$	BFFD	128	126	BFFD	133	128	BFFD	143	131
	MW	126		MW	128		MW	131	
	MW'	126		MW'	128		MW'	131	

Table 1: Performance of the algorithms for Single-Disk.

Striped-Disk 250 Movies 5000 Requests				500 Movies 5000 Requests			1000 Movies 5000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	BFFD	11	11	BFFD	12	11	BFFD	14	11
	MW	11		MW	11		MW	11	
	MW'	11		MW'	11		MW'	11	
$\delta = 0.5$	BFFD	12	11	BFFD	13	11	BFFD	16	12
	MW	11		MW	11		MW	12	
	MW'	11		MW'	11		MW'	12	
$\delta = 1.0$	BFFD	12	11	BFFD	14	12	BFFD	17	13
	MW	11		MW	12		MW	13	
	MW'	11		MW'	12		MW'	13	
Striped-Disk 250 Movies 20000 Requests				500 Movies 20000 Requests			1000 Movies 20000 Requests		
Delta	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound	Algorithm	Result	Lower Bound
$\delta = 0.0$	BFFD	42	42	BFFD	42	42	BFFD	43	42
	MW	42		MW	42		MW	42	
	MW'	42		MW'	42		MW'	42	
$\delta = 0.5$	BFFD	42	42	BFFD	43	42	BFFD	46	43
	MW	42		MW	42		MW	43	
	MW'	42		MW'	42		MW'	43	
$\delta = 1.0$	BFFD	43	42	BFFD	45	43	BFFD	48	44
	MW	42		MW	43		MW	44	
	MW'	42		MW'	43		MW'	44	

Table 2: Performance of the algorithms for Striped-Disk.

In Figures 4 to 8 we present graphics of the results of the algorithms varying the disk storage capacity. The results are given in the y -axis and the storage capacity of the bin is given in the x -axis. In all these tests we assume the load capacity $B = 160$, the number of different movies $Q = 250$ and the number of requests equal to 5000. In Figure 4 (resp. 5, 6, 7, and 8) we use δ equal to 1.0 (resp. 0.5, 0.0, -0.5 and -1). In the graphics the MW' algorithm is denoted by MW2. The lower bound is given by $\max\{\lceil Q/C \rceil, \lceil N/B \rceil\}$. Notice that the problem becomes easier as the distribution of requests becomes uniformly, i.e, the value of δ decreases. When $\delta = -1.0$ all algorithms generated solutions almost equal to the lower bound. Another point is that the problem is harder when the storage capacity is small, as one could expect. When the storage capacity becomes equal to approximately 10 the algorithms MW and MW' produces optimal solutions. When was considered storage capacity greater than 100, the algorithm BFFD generated optimal solutions (for δ equal to 1 and 0.5). The MW' algorithm generated better solutions than the MW algorithm in several instances for δ equal to 1.0, 0.5, 0.0 and -0.5 . Generally the solutions generated by the algorithm MW' uses 2 or 1 fewer disks than MW. Most of these better solutions were obtained with capacities between 2 and 8. It is also interesting to notice that the MW algorithm generated a better solution than the MW' algorithm in one test, the one with $\delta = -1$ and storage capacity equal to 8. In this case the solution found by the MW' algorithm uses 34 disks while the solution generated by the MW algorithm uses 33 disks.

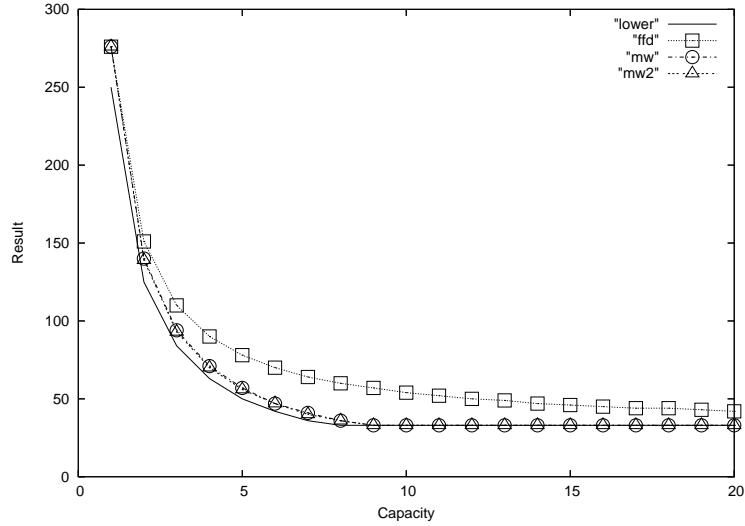


Figure 4: Results with $\delta = 1$.

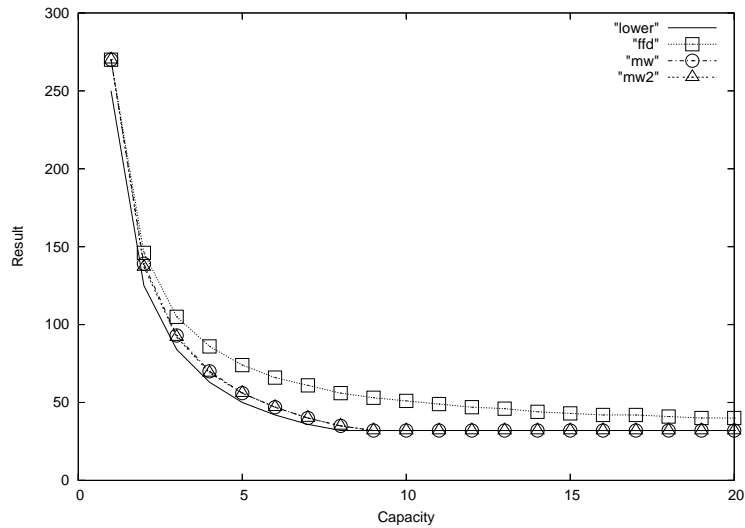


Figure 5: Results with $\delta = 0.5$.

7 Conclusions and Future Work

In this paper we present approximation algorithms for the online and offline class-constrained bin packing problem. The problem is motivated by applications in the data-placement problem to video-on-demand servers and applications in the cutting and packing area. For the online problem we provide lower bounds for any bounded space algorithm and we also present an algorithm for the unbounded version with approximation factor 2.75. For the offline problem we present practical approximation algorithms for two special cases of the problem, with conditions already considered in the literature: when all items have the same size and the parameterized version of the problem. We also perform several tests with these practical algorithms. For the instances we considered representing practical ones, the algorithms MW and MW' obtained optimal solutions. Finally we present an APTAS for the special case where the number of different classes of the input instance is bounded by a constant.

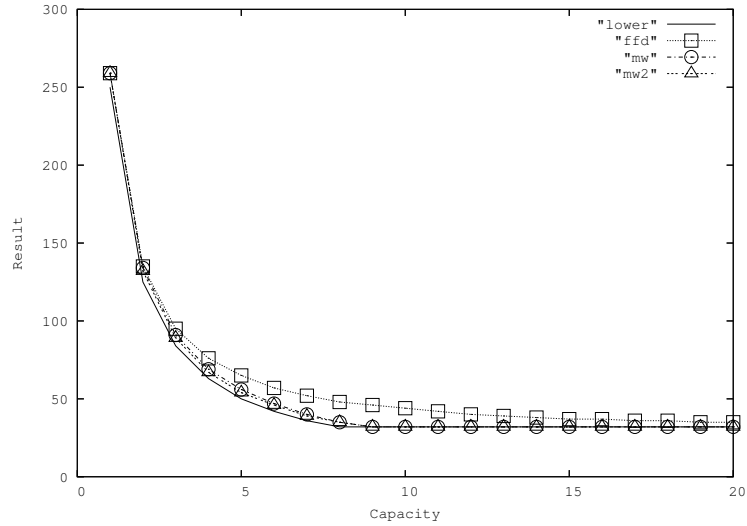


Figure 6: Results with $\delta = 0$.

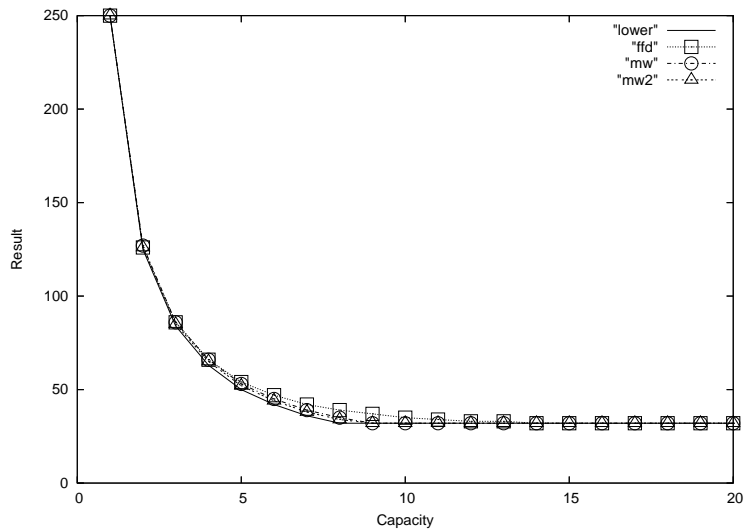


Figure 7: Results with $\delta = -0.5$.

References

- [1] A. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD thesis, University of California at Berkeley, Computer Science Division, 1994.
- [2] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2, pages 46–93. PWS, 1997.
- [3] J. Csirik. The parametric behavior of the first-fit decreasing bin packing algorithm. *Journal of Algorithms*, 15:1–28, 1993.
- [4] M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. Technical report, IBM, T.J. Watson Research Center, NY, 1998.

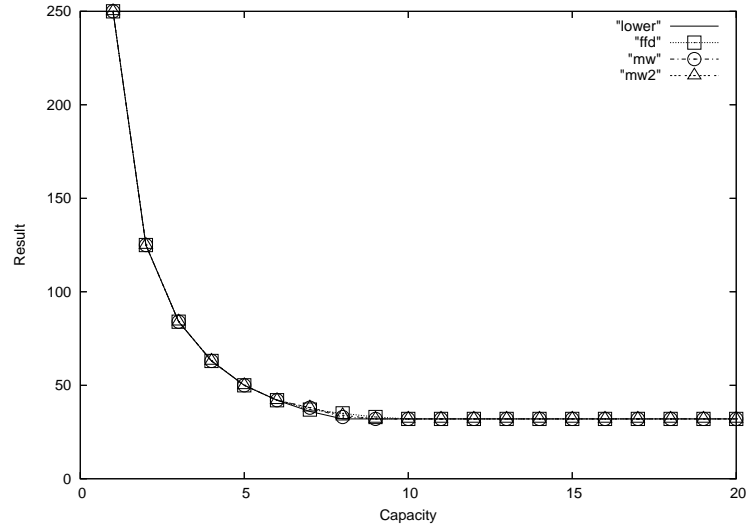


Figure 8: Results with $\delta = -1$.

- [5] M. Dawande, J. Kalagnanam, and J. Sethuranam. Variable sized bin packing with color constraints. In *Proceedings of the 1th Brazilian Symposium on Graph Algorithms and Combinatorics*, volume 7 of *Electronic Notes in Discrete Mathematics*, 2001.
- [6] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
- [7] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing Journal*, 24(1):91–122, 1998.
- [8] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proceedings of SODA*, pages 223–232, 2000.
- [9] P. S. Yu J. L. Wolf and H. Shachnai. Disk load balancing for video-on-demand-systems. *ACM Multimedia Systems Journal*, 5:358–370, 1997.
- [10] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.
- [11] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.
- [12] J. R. Kalagnanam, M. W. Dawande, M. Trumbo, and H. S. Lee. The surplus inventory matching problem in the process industry. *Operations Research*, 48(4):505–516, 2000.
- [13] S. R. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. *Journal of Algorithms*, 60(2):144–167, 2006.
- [14] D. E. Knuth. *The Art of Computer Programming. Volume 3*. Addison-Wesley, 1973.
- [15] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. Association Comput. Mach.*, 32(3):562–572, July 1985.

- [16] F. K. Miyazawa and Y. Wakabayashi. Parametric on-line algorithms for packing rectangles and boxes. *European Journal on Operational Research*, 150:281–292, 2003.
- [17] M. Peeters and Z. Degraeve. The co-printing problem: A packing problem with a color constraint. *Operations Research*, 52(4):623–638, 2004.
- [18] S. S. Seiden. On the online bin packing problem. *Journal of ACM*, 49(5):640–671, 2002.
- [19] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.
- [20] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338, 2001.
- [21] H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.
- [22] H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *Proceedings of 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, RANDOM-APPROX*, volume 2764 of *Lecture Notes in Computer Science*, pages 165–177, 2003.
- [23] H. Shachnai and T. Tamir. Tight bounds for online class-constrained packing. *Theoretical Computer Science*, 321(1):103–123, 2004.
- [24] J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, 1971.
- [25] A. van Vliet. An improved lower bound for online bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.
- [26] E. C. Xavier and F. K. Miyazawa. A one-dimensional bin packing problem with shelf divisions. In *2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics*, volume 19 of *Electronic Notes in Discrete Mathematics*, 2005.
- [27] E. C. Xavier and F. K. Miyazawa. Approximation schemes for knapsack problems with shelf divisions. *Theoretical Computer Science*, 352(1-3):71–84, 2006.
- [28] A. C. Yao. New algorithms for bin packing. *J. Association Comput. Mach.*, 27:207–227, 1980.

A Table with Notation Symbols

Symbol	Meaning
I	Instance of the problem.
L	List of items.
s_e	Size of an item e .
c_e	Class of an item e .
C	Maximum number of class that a bin can have / Storage capacity.
Q	Number of different classes in the input instance.
B	Size of the bin.
\mathcal{P}	Packing of the items in bins.
$ \mathcal{P} $	Number of bins used by the packing \mathcal{P} .
w	Function that gives the size of bins in the variable bins size problem.
$\text{OPT}(I)$	Number of bins used by an optimal solution for an instance I .
$[M]$	For a positive integer M , corresponds to the set $\{1, \dots, M\}$.
$L_a L_b$	The concatenation of two lists L_a and L_b of items.