Variable Sized Bin Packing With Color Constraints

Milind Dawande [1] Jayant Kalagnanam[1] Jay Sethuraman[2]

Abstract

Motivated by a real problem from the steel industry, we consider the variable sized bin packing problem with color constraints (VBPC). In an instance of this problem, we are given a set of n items with each item having a *color* $c \in \{c_1, c_2, ..., c_k\}$, and a *size*. We are also given m distinct bin sizes. Our objective is to pack all the items into bins such that (i) the total size of the items assigned to a bin is not more than its capacity, (ii) no bin contains items with more than two distinct colors, and (iii) the total capacity of the bins used is minimized. Problem (VBPC) generalizes the variable sized bin-packing problem (VBP) (and hence, the classical bin-packing problem (BP)). In this paper we provide several approximation algorithms for (VBPC). We first describe an asymptotic polynomial time approximation scheme (APTAS) for (VBPC) for fixed k using ideas similar to those of Karmarkar and Karp [5]. In fact, we show that this result holds for a generalization of (VBPC) where the number of colors allowed in any bin is p where $2 \leq p \leq k$. In particular, this yields an APTAS for (VBP), improving upon earlier results of [3]. Our APTAS, however, is only of theoretical interest and serves to classify this problem from a complexity-theoretic viewpoint; it cannot be used to solve even moderately sized practical problems. Motivated by this shortcoming of the APTAS, we provide several practical algorithms, including two 3-approximation algorithms for (VBPC).

1 Introduction

In this paper, we introduce the *variable sized bin packing problem* with novel packing constraints, called the *color constraints*. In an instance of this problem, we have n items to be packed into bins of m distinct sizes. Items come with two attributes: *color* and *size*. In addition to the usual capacity constraints, we require that each bin contain items with at most p distinct colors, where p is a pre-specified positive integer. Our objective is to minimize the total capacity of the bins used in the packing. This problem arises as a model for the *slab design problem* in the production planning process of a steel plant.

Production planning in a steel plant starts with an *order book*. Associated with each order is (i) the weight to be supplied and (ii) the quality requirement, which translates into a routing through the processing plant. The objective of the slab design problem is to design slabs (the basic units of production in a steel mill) of minimum total weight which can fulfill the order book. This minimization is constrained by the size of the slabs that can be designed, and by some rules which forbid certain subsets of orders from being assigned to the same slab.

[1]IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York 10598, {milind,jayant}@watson.ibm.com

[2]OR Center, MIT , Cambridge, Massachusetts, MA 02139, jayc@mit.edu

Steel is produced by casting molten iron into slabs. Typically, a plant can produce only a finite number of slab sizes (equivalently weights), depending on the type and vintage of the caster. The optimization problem, then, is to pack orders into slabs so as to minimize the total capacity of the slabs used to fulfill the order book. Two classes of packing constraints arise in practice. First, we have the usual (physical) packing constraints: the total weight of the orders assigned to the slab cannot exceed the slab weight. Second, a packing constraint arises from the routing specification associated with each order. If we associate with each route a distinct color, then packing multiple colors on a slab would necessitate cutting the slab into several pieces *before* routing. In general, cutting slabs into pieces is expensive and time consuming and often results in scheduling bottlenecks. As a result, very often strict constraints are imposed on the *number of cuts* or equivalently the number of colors allowed on a slab. We call this constraint the *color* constraint. In our specific application, the number of cuts allowed for a slab was one. Equivalently, a slab can contain items with at most two colors. Note, however, that the slabs do have to be cut eventually in the shipping area (because, for instance, the orders may belong to different customers). Forcing the cutting of the slabs at an earlier stage (e.g. rolling or finishing) causes scheduling bottlenecks at the earlier stages, and so we require that any slab contain few (two, in our application) different-route orders.

More formally, we associate with each order O_i an item I_i of weight $s(I_i)$ and color $c(I_i)$. The various bin sizes that can be used for packing correspond to the various slab sizes that can be produced by the caster; thus, we have bins of size w_j if and only if we can cast slabs of size w_j. The slab design problem can then be modeled as a variable sized bin packing problem with color constraints.

The rest of the paper is organized as follows: In section 2 we define the notation used and introduce the many variants of (VBPC) we study in this paper. We present an APTAS for the most general variant of (VBPC) in section 3. We provide several practical approximation algorithms for (VBPC) in section 4.

2 VBPC and its variants

Let $\{w_1, w_2, ..., w_m\}$ be the distinct bin sizes available, $S = \{I_1, I_2, ..., I_n\}$ be the set of items to be packed, and $C = \{c_1, c_2, ..., c_k\}$ be the set of distinct colors. For $i = 1, ..., k$, let $S^i = \{I^i_1, ..., I^i_{n_i}\}$ be the set of items of color c_i sorted by descending item weight. (Clearly, $S = \cup_{i=1}^{k} S^i$.) The weight of an item I_j is $s(I_j)$. Also, the *contents* of a bin B is the total weight of the items packed into B and is denoted by $c(B)$. We study the following problems in this paper: (In all of these problems, by an *optimal packing* we mean a packing that minimizes the total capacity of the bins used in the packing.)

- $(VBPC)$: Find an optimal packing of all the items into the bins (note that there are different bin sizes available) such that the number of distinct colors in each bin is no more than two.

- $(GVBPC)$: Find an optimal packing of all the items into the bins such that the number of distinct colors in each bin is at most a given integer p, where $2 \leq p \leq k$.

- (BPC): Find an optimal packing of all the items using bins of size 1 such that the number of distinct colors in each bin is no more than two.

- $(GBPC)$: Find an optimal packing of all the items using bins of size 1 such that the number of distinct colors in each bin is at most a given integer p, where $2 \leq p \leq k$.

Note: The case $p = 1$ is not very intersting in studying $(GVBPC)$; if $p = 1$, the problem decomposes and can be solved as k variable sized bin packing problems. For the same reason, we exclude the case $p = 1$ in studying $(GBPC)$.

3 An asymptotic PTAS for GVBPC

3.1 Preliminaries

Although our main problem (VBPC) requires that the number of colors in a bin be at most *two*, we present the approximation scheme for the generalization (GVBPC) where the number of colors allowed per bin is at most p $(2 \leq p \leq k)$.
Consider an instance I of the variable sized bin packing problem with color constraints. There are two key ideas in the development of an APTAS: (i) *discretize* the sizes of the items so that there are at most M distinct sizes, and (ii) ignore bin-sizes that are too small. In discretizing the item sizes, we need to round *up* the sizes of the given items so as to reduce the number of different item sizes, while still being able to derive a packing for the given list of items. This rounding idea, coupled with the fact that there are k colors means that every item is on one of $k \cdot M$ types—all items belonging to a particular type have the same color and size.

Without loss of generality, we assume that items have size less than one, and that the largest bin has size one. As we will see later the amount of "wasted" space is directly related to the *number of bins* used in the packing; as a result it will be necessary that we do not use *too many* bins, and so we will have to throw away small bin-sizes. So it will be useful to understand the extra costs incurred if we place additional restrictions on the bin-sizes. In particular, we first address the following question.

Problem 3.1 *Consider an instance I of the variable sized bin packing problem. Let $\gamma < 1/2$. Suppose $OPT(I;\gamma)$ denotes the optimal objective value for the instance I when we are only allowed to use bins of size at least γ (from among the given bin sizes). How does $OPT(I;\gamma)$ compare to $OPT(I;0)$?*

Consider an optimal packing for I with cost $OPT(I;0)$. If this packing does not use any bin of size $< \gamma$, $OPT(I;\gamma) = OPT(I;0)$. If not, consider the set L of all the items packed in bins of size less than γ. We pack L using a simple first-fit type heuristic on bins of size one: We pack an item into the first bin into which it *can be* packed; we open a new bin of size one only when an item cannot be packed into any of the bins that are already open. Observe that we may not be able to pack an item i into an already opened bin B_j for two reasons:

- The size of the item i is more than the amount of space available in the bin B_j.
- Packing item i into bin B_j will violate the color constraint for B_j.

We now show that $OPT(I;\gamma)$ is reasonably close to $OPT(I;0)$.

Lemma 3.2
$$OPT(I;\gamma) \leq \frac{1}{(1-\gamma)} OPT(I;0) + \left\lceil \frac{k}{2} \right\rceil.$$

Proof: Let J be the number of new unit-sized bins introduced for packing the items in L; for convenience, call the new bins $B_1, B_2, \ldots, B_J$. Lemma 3.2 is true if $J = 0$, and so we assume $J > 0$. Let $s(L)$ denote the total size of the items in L. Since all of the items in L have size less than γ, we know that for each B_j $(1 \leq j \leq J-1)$, either the available space is less than γ or it already has two colors. Consider the set of bins with residual capacity more than γ. Clearly, each color can appear in at most one of these bins. Thus, the number of new unit-sized bins with residual capacity more than γ is at most $\lceil \frac{k}{2} \rceil$. This argument also shows that $s(L) \geq (1-\gamma)(J - \lceil \frac{k}{2} \rceil)$. Let R be the set of bins of size at least γ used by $OPT(I;0)$, and let $s(R)$ denote the total size of the bins in R. Clearly,

$$\begin{aligned}
OPT(I;\gamma) &\leq s(R) + J \\
&\leq s(R) + \frac{s(L)}{1-\gamma} + \left\lceil \frac{k}{2} \right\rceil \\
&\leq \frac{1}{(1-\gamma)} OPT(I;0) + \left\lceil \frac{k}{2} \right\rceil.
\end{aligned}$$

The last inequality follows from the fact that the total capacity of the bins with size less than γ used in $OPT(I;0)$ is at least $s(L)$. ■

Lemma 3.2 establishes that we can exclude arbitrarily small bin sizes with only a slight loss in the performance. In essence Lemma 3.2 enables us to do without arbitrarily small bin sizes in our algorithm; this ensures that we do not use *too many* bins. Unless otherwise stated, we assume that we do not use bins of size smaller than γ. We will choose γ later. To avoid cumbersome notation, we will drop the "γ" term hereafter: thus we write $OPT(I)$ when we actually mean $OPT(I;\gamma)$.

In the next section we show how we can round the sizes of the items to derive a new instance in which the number of distinct sizes is *small*.

3.2 Rounding

In the terminology of Karmarkar and Karp [5], we use linear rounding. To start with, we assume that all items with size smaller than β are discarded, where $\beta \leq \gamma$; we will present a simple post-processing step that packs these items without significantly affecting the asymptotic performance guarantee.

Let L be the list of items to be packed. We split L into two parts—the list L_s consists of all items with size smaller than β, and the list L_b consists of the remaining items. In the rest of this section we will ignore L_s completely. Alternatively, we assume that L_s is empty and that $L_b = [I_1, I_2, \ldots, I_n]$. We will later discuss how we can pack items in L_s without destroying the asymptotic performance guarantee.

We renumber the items in L_b, if necessary, so that $s(I_1) \geq s(I_2) \geq \ldots \geq s(I_n)$. We first divide the items into equal-sized groups, where the number of items in each group is $\lceil n\beta^2 \rceil$. Call the first item in a group as the *group-leader*—recall that the group-leader has the maximum size among all items in its group. (The group-leader of the first group is assumed to have size one; this can always be achieved by adding an item of size one to L_b.) Let $\hat{L_b}$ be the list obtained from L_b, in which the size of an item is rounded up to the size of its group-leader. Similarly, let $\tilde{L_b}$ be the list obtained from L_b, in which the size of an item is rounded down to the size of the group-leader of the next group. (The sizes of the items in the last group are rounded down to zero.) If $OPT(L_b)$ is the value of the optimal solution to the bin-packing problem for the list L_b, then,

$$OPT(\tilde{L_b}) \leq OPT(L_b) \leq OPT(\hat{L_b}).$$

Notice, however, that $\hat{L_b}$ and $\tilde{L_b}$ differ only in their first and last groups. The list $\hat{L_b}$ has $\lceil n\beta^2 \rceil$ additional items of size one, whereas the list $\tilde{L_b}$ has some additional items at the end. In any case, we can pack $\hat{L_b}$ using the optimal packing for $\tilde{L_b}$ and at most $\lceil n\beta^2 \rceil$ extra bins, and so

$$\begin{aligned} OPT(\hat{L_b}) &\leq OPT(\tilde{L_b}) + \lceil n\beta^2 \rceil \\ &\leq OPT(L_b) + \lceil n\beta^2 \rceil. \end{aligned}$$

Since each item is of size at least β, we have

$$OPT(L_b) \geq n\beta.$$

From this discussion and our assumption that each item is of size at least β, we see that

$$\begin{aligned} OPT(\hat{L_b}) &\leq OPT(L_b)\Big(1 + \frac{\lceil n\beta^2 \rceil}{OPT(L_b)}\Big) \\ &\leq OPT(L_b)\left(1 + \frac{\lceil n\beta^2 \rceil}{n\beta}\right) \\ &< (1 + \beta + \frac{1}{\beta n})\, OPT(L_b). \end{aligned}$$

3.3 The Linear Program

As in [5], we now proceed to define the linear program to be used in obtaining a $(1+\epsilon)$ approximation scheme. We assume that the item sizes have been discretized, resulting in a list $\hat{L_b}$; observe that the size of each item in $\hat{L_b}$ is one of M distinct values. (Note that M is $O(\beta^{-2})$.)

To define the linear program, we need to enumerate all possible configurations—ways in which a bin can be packed in a feasible solution. In our case, each item has a size and a color attribute, and so the total number of *types* of items we have is $t = kM$. Let d_i be the total number of items of type i, for $1 \leq i \leq t$.

Since every item we pack has is of size at least β, items could be one of k distinct colors, and each feasible bin can contain at most p different colors. Thus, the total number of configurations is at most N, which is independent of n, but depends exponentially on k, p and M.

Let a_{ij} be the number of items of type i used in configuration j, for $1 \leq i \leq t, 1 \leq j \leq N$. Let x_j be the number of times configuration j is used in a feasible solution to $(GVBPC)$ and let S_j be the size of the bin used in configuration j. Consider the following linear program:

$$\text{LP-GVBPC:} \quad \text{Minimize} \quad \sum_{j=1}^{N} S_j x_j \tag{1}$$

$$
\begin{aligned}
&\text{Subject to} && && (2)\\
\sum_{j=1}^{N} a_{ij}x_j &\geq d_i, \quad i = 1, 2, \ldots t, && && (3)\\
x_j &\geq 0, \quad j = 1, 2, \ldots, N. && && (4)
\end{aligned}
$$

The linear programming problem $(LP - GVBPC)$ has N variables and t constraints, and so has an optimal basic feasible solution with at most t non-zero entries; rounding up the fractional solution thus obtained will result in the addition of at most t bins. Calling this solution (RLP), we see that

$$RLP(\hat{L}_b) \leq OPT(\hat{L}_b) + t < (1 + \beta + \frac{1}{\beta n})OPT(L_b) + t.$$

3.4 Post-Processing

Our analysis upto this point has relied on the crucial assumption that L_s is empty. In this section, we explain how this assumption can be relaxed by a simple post-processing procedure. Suppose that L_s is non-empty; we assume that the items in L_b have been packed using the algorithms described in sections 3.2 and 3.3. We now describe a simple algorithm to pack the list L_s of discarded items; recall that, in L_s each item has size at most β. We use the following simple strategy, similar to first-fit. Assume the current packing obtained from rounding the optimal linear programming solution uses r bins, labeled $B_1, B_2, \ldots, B_r$. Sort the small items in decreasing order of weight. Pack each item in the first bin to which this item can be assigned. If an item cannot be packed into any nonempty bin (either because of its size or because of its color), open a new unit-sized bin. Let $A(L)$ be the objective value after packing all of the discarded items. We now have the following two lemma.

Lemma 3.3 *If additional bins are opened during the post-processing phase, then*

$$A(L) \leq \frac{1}{1 - \frac{\beta}{\gamma}} OPT(L; 0) + \frac{1 - \beta}{1 - \frac{\beta}{\gamma}} \left\lceil \frac{k}{2} \right\rceil.$$

Proof: Let the total number of bins used by our algorithm be B. Then, since we use a first-fit post-processing step, it follows that at least $B - \lceil \frac{k}{2} \rceil$ bins have residual capacity at most β. Thus, the total size of the items $s(L)$ is such that

$$A(L) \leq s(L) + \beta\left(B - \left\lceil \frac{k}{2} \right\rceil\right) + \left\lceil \frac{k}{2} \right\rceil.$$

Since each bin has size at least γ, we know that $B \leq \frac{A(L)}{\gamma}$. Thus,

$$A(L) \leq s(L) + \beta\left(\frac{A(L)}{\gamma} - \left\lceil \frac{k}{2} \right\rceil\right) + \left\lceil \frac{k}{2} \right\rceil. \qquad (5)$$

Noting that $s(L) \leq OPT(L; 0)$, and re-writing equation (5) in a slightly different manner, we have

$$
\begin{aligned}
A(L) &\leq \frac{1}{1 - \frac{\beta}{\gamma}} \left\{ s(L) - \beta \left\lceil \frac{k}{2} \right\rceil + \left\lceil \frac{k}{2} \right\rceil \right\}\\
&\leq \frac{1}{1 - \frac{\beta}{\gamma}} \left\{ OPT(L; 0) - \beta \left\lceil \frac{k}{2} \right\rceil + \left\lceil \frac{k}{2} \right\rceil \right\}\\
&= \frac{1}{1 - \frac{\beta}{\gamma}} OPT(L; 0) + \frac{1 - \beta}{1 - \frac{\beta}{\gamma}} \left\lceil \frac{k}{2} \right\rceil.
\end{aligned}
$$

$\blacksquare$

Lemma 3.4 *If no additional bins are opened during the post-processing phase, then*

$$A(L) \leq \frac{1+\beta}{1-\gamma} OPT(L;0) + \left\lceil \frac{k}{2} \right\rceil + O(\beta^{-2}).$$

Proof: Recall that instead of packing L, we pack a discretized version of L called $\hat{L}$. By assumption, we know that all of the small items fit into the packing obtained by rounding the linear programming solution. Thus, we know that $A(L)$, the total space consumed by our heuristic is exactly $RLP(\hat{L}_b;\gamma)$. We know from previous results that $t = O(\beta^{-2})$,

$$RLP(\hat{L}_b;\gamma) \leq OPT(\hat{L}_b;\gamma) + t$$

and

$$OPT(\hat{L}_b;\gamma) \leq \frac{1}{(1-\gamma)} OPT(\hat{L}_b;0) + \left\lceil \frac{k}{2} \right\rceil.$$

Using these two equations and the fact that $OPT(\hat{L}_b;0) < (1+\beta)OPT(L_b;0) \leq OPT(L;0)$, we have

$$A(L) < \frac{1+\beta}{1-\gamma} OPT(L;0) + \left\lceil \frac{k}{2} \right\rceil + t,$$

which proves the lemma. $\blacksquare$

To derive a $(1+\epsilon)$ asymptotic performance guarantee, we need to pick β and γ appropriately. We set $\beta = \epsilon^4$ and $\gamma = \epsilon^2$. For this choice of β and γ, and for ϵ sufficiently small, we can see that $\frac{1}{1-\beta} \leq 1+\epsilon$, and $\frac{1}{1-\frac{\beta}{\gamma}} \leq 1+\epsilon$, which immediately yield the required performance guarantee.

3.5 Solving the LP in polynomial time

Finally we need to argue that the LP can actually be solved efficiently. To do this we use a technique similar to Karmarkar and Karp [5]. The key idea is to try to solve the dual LP. The dual LP will have few variables but exponentially many constraints. However, using the well-known idea due to Grotschel, Lovasz and Schrijver [6], we know that if we can solve the separation problem in polynomial time, we can solve the dual LP in polynomial time. It is easy to see that the separation problem is actually a knapsack problem, which is NP-hard. Observe, however, that we do not need an *exact* solution to the separation problem; an approximate solution will suffice. This observation coupled with the fact that the knapsack problem can be approximated to within any δ efficiently helps us solve the LP efficiently. We omit the details in this extended abstract.

4 More practical approximation algorithms

The APTAS presented in Section 2 uses the ellipsoid method to solve the linear program and as such is not suited well for solving VBPC in practice. In this section, we present simple and efficient approximation algorithms for VBPC and its variants.

Our algorithms can be broadly classified into two categories.

1. Algorithms which pack items of each color seperately. Here, each bin will contain items of a single color. For example, Procedure PSB presented below is an algorithm of this type.

2. Algorithms which 'mix' items of different colors and then collectively pack the items. Here, each bin may contain upto two different colors. An example is algorithm $FFCL$ presented in Section 4.2.

4.1 A 3-approximation algorithm for $VBPC$

Procedure $PSB(S^i)$

1. (Initialization): $i = 1$. $S^i = \{I_1^i, ..., I_{n_i}^i\}$ is the set of items of color c_i sorted by descending item weight.

2. Try to pack item I_j^i into bins already open. That is, if any of the bins already open have residual capacity greater than $s(I_j^i)$, we pack item I_j^i in that bin. If all items in S^i have been packed, STOP.

3. If none of the currently open bins can accomodate I_j^i, open a new bin of *smallest* size that can accomodate I_j^i and pack I_j^i in this new bin. If all items in S^i have been packed, STOP.

4. $i = i + 1$. Return to Step 2.

Algorithm PSB

1. $i = 1$.

2. while $i \leq k$
   - Pack items of color c_i using procedure $PSB(S^i)$.
   - Increment i; return to Step 2.

Theorem 4.1 *Algorithm PSB is 3-approximation for VBPC.*

Proof: We denote as "big" colors, those colors for which algorithm PSB uses more than one bin (to pack the items of that color). Colors for which PSB uses a single bin are the "small" colors. Let W be the set of all items from all the big colors and let D be the space used by the heuristic to pack the items in W. WLOG, let the small colors be c_1, c_2, ..., c_l and let L_1, L_2, ..., L_l be the bin sizes used by the heuristic to pack each of the small colors respectively. For color c_i, let a_i denote the item of largest size. Let $A = \{a_1, a_2, ..., a_l\}$. Thus, the total space used by the heuristic is $H = D + \sum_{i=1}^{l} L_i$. Consider an optimal solution. We partition the bins used by the optimal solution into the following sets:

1. (B_1) Bins which contain no items from the set A.

2. (B_2) Bins which contain exactly one item from the set A. WLOG, let $A_1 = \{a_1, ..., a_q\}$ be the set of items (from A) in these bins.

3. (B_3) Bins which contain two items from the set A. Note that such a bin can contain no items from W for otherwise it would have three colors in it. Let $A \backslash A_1 = A_2 = \{a_{q+1}, ..., a_l\}$ be the set of items (from A) in these bins.

Now we note the following:

1. Bins in sets B_1 and B_2 include all items in W. Also, a bin in set B_3 (having,say, items a_i and a_j) has size atleast $\max(L_i, L_j) \geq \frac{L_i+L_j}{2}$. Using these two observations, it is easy to see that $OPT \geq \frac{D}{2} + \frac{1}{2}\sum_{i=q+1}^{l} L_i$.

2. A bin in set B_2 (having, say, item a_m) has size atleast L_m. Thus, $OPT \geq \sum_{i=1}^{q} L_i + \frac{1}{2}\sum_{i=q+1}^{l} L_i$.

Adding (i) and (ii) above and the inequality $OPT \geq \frac{D}{2}$, we get $3OPT \geq D + \sum_{i=1}^{l} L_i = H$. ∎

It is easy to construct examples for which the ratio $\frac{H}{OPT}$ is arbitrarily close to 3. For example, consider an instance with 3 items I_1, I_2 and I_3 of weight $\frac{2}{3}$, $\frac{1}{3}$ - ϵ and ϵ respectively. Items I_1 and I_2 are of color c_1 while item I_3 is of color c_2. There are two bin sizes available: 1 - 2ϵ and 1. The total space required by the heuristic is 3 - 6ϵ while optimum solution requires a space of 1.

4.2 First-Fit with color constraints

In this section, we consider variations of the well-known First-Fit algorithm for the classical bin-packing problem and adapt them for $VBPC$.

4.2.1 First-Fit decreasing with color constraints

Algorithm $FFDC$, given below, is an adaptation (for color constraints and variable bin-sizes) of the classical first-fit decreasing (FFD) algorithm for the bin-packing problem. Since we have variable sized bins, algorithm $FFDC$ opens the *smallest* bin which can accomodate an item. Also, while packing an item into a bin, the residual capacity as well as the color constraint is checked.

Algorithm $FFDC$

1. Sort the items in decreasing order of size. Thus, $s(I_1) \geq s(I_2)... \geq s(I_n)$.

2. For packing item i, traverse the currently open bins in the order in which they were opened. Pack item i in the first bin which can accomodate it. Note that we have to consider the *residual capacity* of the currently open bins as well as the *number of distinct colors* of the items currently packed into the bin. If no bin can accomodate item i, open a new bin of smallest size which can accomodate item i and assign item i to that bin.

3. Repeat Step 1 until all the items have been packed.

Theorem 4.2 *Algorithm FFDC is 3-approximation for VBPC.*

Proof: Similar to the proof of Theorem 4.1. ∎

Note that algorithm $FFDC$ sorts all the items before packing them and hence is not an online algorithm. To get an online algorithm, we consider another variation of a first-fit type algorithm where we always open a bin of size 1. Also, we do not sort the items. We call this variation *First-Fit with color constraints using largest bin (FFCL).*

4.2.2 First-Fit with colors using largest bin: An online algorithm

Algorithm $FFCL$

1. For packing item i, traverse the currently open bins in the order in which they were opened. Pack item i in the first bin which can accomodate it. If no bin can accomodate item i, open a new bin of size 1 and assign item i to that bin.
2. Repeat Step 1 until all the items have been packed.
3. At the end of Step 2, suppose there are l bins with residual capacity at least $\frac{1}{2}$. If there exists a pair of such bins such that both the bins in the pair have only one color in them, we repack the contents of one bin in the pair into the other bin.

Note that, for the l bins which have residual capacity at least $\frac{1}{2}$, each color can appear in at most one of these bins. This simple observation gives the following result.

Lemma 4.3 *In Step 3 of Algorithm $FFCL$,* $l \leq k$.

Theorem 4.4 *The total space used by Algorithm $FFCL$ is* $\leq 2 * OPT + \lceil \frac{k}{2} \rceil$.

Proof: By Lemma 4.3, at the end of Step 2 of Algorithm $FFCL$, the number of bins with residual capacity greater than or equal to $\frac{1}{2}$ is at most k. Since, each color appears in at most one of these bins, Step 3 of Algorithm $FFCL$ repacks the contents of these bins into at most $\lceil \frac{k}{2} \rceil$ bins. Thus, at the end of Step 3 of Algorithm $FFCL$, we have that the number of bins with residual capacity greater than or equal to $\frac{1}{2}$ is at most $\lceil \frac{k}{2} \rceil$. It then follows that the total space used by the Algorithm $FFCL$ is $\leq 2 * OPT + \lceil \frac{k}{2} \rceil$. ∎

Many other well-known algorithms for the classical bin-packing problem provide variants which give similar approximation algorithms for $VBPC$. For example *Next-Fit with color constraints* and *Best-Fit with color constraints* also provide a $2 * OPT + \lceil \frac{k}{2} \rceil$ approximation algorithm for the generalized problem $GVBPC$. The proofs are similar, and so omitted.

Both VBPC and its generalization GVBPC allow for variable sized bins. Consider the variants of these problems (defined as BPC and GBPC, respectively, in Section 2) in which the bin-size is fixed, i.e., all the bins are of size 1. It is easy to see that Algorithm $FFCL$ provides a 3-approximation for BPC. This is immediate, since, any feasible packing for BPC uses atleast $\lceil \frac{k}{2} \rceil$ bins. This observation combined with Theorem 4.3 given us

$$\begin{aligned} \text{Total space used by Algorithm } FFCL &\leq 2 * OPT + \left\lceil \frac{k}{2} \right\rceil \\ &\leq 2 * OPT + OPT \\ &= 3 * OPT \end{aligned}$$

For the generalization $GBPC$ where the number of colors allowed per bin is a given integer $p \geq 2$, algorithm PSB provides a 4-approximation. The proof is similar to that of Theorem 4.1. We omit the details in this extended abstract.

The 3-approximation algorithm for $VBPC$, given in Section 4.2.1 has been used to solve real-world instances of the application described in Section 1. For problems upto 3000 orders (items) and a maximum of 3000 different bin sizes, the heuristic provides good quality solutions in about 1-2 seconds of computing time on an IBM RS/6000 machine. We omit detailed computational results and comparisons in this extended abstract.

References

[1] E. G. Coffman, Jr., M. R. Garey and D. S. Johnson, Approximation Algorithms for bin packing: A survey, in *Approximation Algorithms for NP-hard Problems*, D. S. Hochbaum (Editor), PWS Publishing Company, Boston, MA.

[2] W. Fernandez de la Vega and G. S. Lueker, Bin packing can be solved within $1 + \epsilon$ in linear time, *Combinatorica*, (1981) 1:349-355.

[3] D. K. Friesen and M. A. Langston, Variable sized bin packing, *SIAM J. Computing* (1986) 222-230.

[4] *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum (editor), PWS Publishing Company, Boston, MA.

[5] N. Karmarkar and R. M. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, *IEEE FOCS* (1982) 312-320.

[6] M. Grotschel, L. Lovasz and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* **1** (1981) 169-198.