Bin Packing with Item Fragmentation

Nir Menakerman **Raphael Rom**

Department of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 32000, Israel
{mnir@techunix, rom@ee}.technion.ac.il

Abstract

We investigate a variant of the bin packing problem in which items may be fragmented into smaller size pieces called fragments. While there are a few applications to bin packing with item fragmentation, our model of the problem is derived from a scheduling problem present in data over CATV networks. Fragmenting an item is associated with a cost which renders the problem NP-hard. We study two possible cost functions and as a result get two variants of bin packing with item fragmentation. In the first variant, called *bin packing with size-increasing fragmentation*, we are asked to pack a list of items into a minimum number of unit capacity bins. Each item may be fragmented in which case overhead units are added to the size of every fragment. In the second variant we are asked to pack a list of items into a fixed number of unit capacity bins. Each item has a size and a cost and fragmenting an item increases its cost but does not change its size. The goal is to minimize the total cost. We call this variant *bin packing with size-preserving fragmentation.*

We develop several algorithms for the problem of bin packing with item fragmentation and investigate their performance. The algorithms we present are based on well known bin packing algorithms such as Next-Fit and First-Fit Decreasing, as well as of other algorithms.

1. Introduction

Because of its applicability to a large number of applications and because of its theoretical interest bin packing has been widely researched and investigated (see, e.g., [4], [6], [9] and [2] for a comprehensive survey). In the classical one-dimensional bin packing problem, we are given a list of items $L=(a_1,a_2,...,a_n)$, each with a size $s(a_i)\in(0,1]$ and are asked to pack them into a minimum number of unit capacity bins. Since the problem, as many of its derivatives, is NP-hard many approximation algorithms have been developed for it (see, e.g., [3], [7], [8] and [1] for a survey). The common assumption in bin packing problems is that an item may not be fragmented into smaller pieces. There are several applications, however, in which this assumption does not hold. The subject of item fragmentation in bin packing problems received almost no attention so far. This paper concentrates on aspects that were heretofore never researched, such as developing algorithms for the problem and investigating their performance.

The variant of bin packing presented in this paper is derived from a scheduling problem present in data over CATV (community antenna television) networks. In particular we refer to Data-Over-Cable Service Interface Specification (DOCSIS), standard of the Multimedia Cable Network System (MCNS) standard committee, for a detailed description see [12]. When using CATV networks for data communication the data subscribers are connected via a cable modem to the headend. The headend is responsible for the scheduling of all transmissions in the upstream direction (from the cable modem to the headend). Scheduling is done by dividing the upstream, in time, into a stream of numbered mini-slots. The headend receives requests from the modems for allocation of datagram transmission. The length of each datagram can vary and may require a different number of mini-slots. From time to time, the headend publishes a *MAP* in which it allocates mini-slots to one modem or a group of modems. The scheduling problem is that of allocating the mini-slots to be published in the MAP, or in other words, how to order the datagrams transmission in the best possible way.

The headend must consider two kinds of datagram allocations:

i. *Fixed Location* - Allocations for connections with timing demands, such as a CBR (constant bit rate) connection. These connections must be scheduled so as to ensure delivering the guaranteed service. Fixed location datagrams are therefore scheduled in fixed, periodically located mini-slots.

ii. *Free Location* - Allocations for connections without timing demands, such as a best effort connection. Free location datagrams can use any of the mini-slots.

The headend therefore performs the allocation in two stages: in the first stage it schedules, or allocates, all fixed location datagrams. We assume that after the fixed allocations have been made, a gap of U mini-slots is left between successive fixed allocations. In the second stage all free location datagrams are scheduled. The free allocations must fit into the gaps left by the fixed allocations.

The relation to the bin packing problem should now be clear. The items are the free location datagrams that should be scheduled, each of which may require a different number of mini-slots. The bins are defined by the gaps between every two successive fixed allocations in the MAP. The goal is to use the available mini-slots in the MAP, in the best way.

One of the capabilities of the system is the ability to break a datagram into smaller pieces called *fragments*. When a datagram is fragmented, i.e., transmitted in non successive mini-slots, extra bits are added to the original datagram to enable the reassembly of all the fragments at the headend. In a typical CATV network one mini-slot is added to every fragment.

The scheduling problem provided our initial motivation for exploring the problem of bin packing with item fragmentation. We point out that the practical scheduling problem may actually be somewhat more complicated since the bins (gaps between fixed allocation) may not be of uniform size. We studied the case of variable size bins in [11], but due to size constraints we cannot present the results here. The bottom line however is that, as far as worst case results are concerned, there is no significant difference between the two cases. We also came across other applications to which bin packing with item fragmentation may be applicable (e.g., scheduling Bandwidth on Demand in IP satellite networks and applications in VLSI circuit design). This has motivated us to consider other variants of the problem. In this paper we present two variants which differ in the cost they associate with fragmentation.

To make the problem of bin packing with item fragmentation nontrivial a cost must be associated with fragmentation. We study two possible cost functions and as a result get two variants of bin packing with item fragmentation. In the first variant, called *bin packing with size-increasing fragmentation*, the cost function adds one (or more) overhead unit to the size of every fragment. In the second variant the cost function increases the cost of an item upon each fragmentation, but does not change its size. We call this variant, *bin packing with size preserving fragmentation*. The scheduling problem, where the cost is due to the extra overhead bits that are added to each fragment, serves as a model to the first variant. The second variant is suitable for problems where the cost associated with fragmentation is a result of extra processing time or reassembly delay. The two cost functions we present here are not the only possible cost functions. In other applications, for example, the cost may be related to the size of the item, a combination of the two cost functions is also possible. It is interesting to note that when the cost associated with fragmentation is ignored the packing problem becomes trivial, and when the cost is very high, it does not pay to fragment items and we face the classical problem. Hence, the problem is interesting with the middle-range costs. It has been shown in [10] that for non zero cost the ability to fragment items does not reduce the complexity of the problem, that is, the problem of bin packing with item fragmentation is NP-hard.

We present a worst case analysis of both variants of bin packing with item fragmentation. We begin by showing that both variants are NP-Hard in the strong sense. We then devise approximation algorithms for the problem and investigate their performance. We restrict our attention to practical bin packing algorithms, i.e., of low order polynomial running time, and examine both *online* and *offline* algorithms. Online algorithms are applicable to cases where the items arrive in some order and must be assigned to the bins as soon as they arrive. Offline algorithms assume the entire list of items is known before the packing begins. We devise algorithms which are based on well known bin packing algorithms but include the capability of fragmenting items. We investigate the performance of algorithms such as Next-Fit (NF) and First-Fit Decreasing (FFD), as well as of other algorithms. As we mentioned, this paper present only worst case analysis of the algorithms. Our work include average case analysis of some of the algorithms but this analysis could not be included in this paper.

Our work has both theoretical and practical value. The problem of bin packing with item fragmentation has not been explored yet and many interesting questions are left open. This paper provides the basic definition of the problem, present practical algorithms and investigate their worst case performance. Future work may refer to subjects such as devising a polynomial time approximation scheme (PTAS) for the problem, providing average case results and considering other variants (e.g., variable size bins and items bigger than the bin size).

The reminder of the paper is organized as follows. In Section 2 we address the problem of bin packing with size-increasing fragmentation. Section 3 is devoted to the problem of bin packing with size-preserving fragmentation.

2. Bin Packing with Size-Increasing Fragmentation

In this section we study the variant of bin packing with item fragmentation where fragmentation increases the size of an item. We define the problem similar to the classical bin packing problem. The classical bin packing problem deals with equal-sized (unit capacity) bins and a list of items each of which can fit in every bin. To handle fragmentation, we use a discrete version of the problem and add a fragmentation cost function that adds overhead units to each fragment. We proceed to formally define the problem.

Bin Packing with Size-Increasing Fragmentation (BP-SIF): We are given a list of items $L=(a_1,a_2,...,a_n)$, each with a size $s(a_i) \in \{1,2,...,U\}$. The items must be packed into a minimum number of bins, which are all the size of U units. When packing a fragment of an item, one unit of overhead is added to the size of *every* fragment.

Example: Assume the bin size is U=10 and we are given three items of sizes 5, 6 and 7. We can pack the items in only two bins in the following way: We pack the items of sizes 5 and 7 one in a bin and fragment the item of size 6 into two fragments of sizes 2 and 4. When we pack the fragments we add one overhead unit to each, so their sizes become 3 and 5, respectively. We pack the fragment of size 5 with the item of size 5 and the fragment of size 3 with the item of size 7.

Performance Ratio: We use the same definition as is typically used in analyzing the classical problem. For a given list L and algorithm A, let $A(L)$ be the number of bins used when algorithm A is applied to list L, let $OPT(L)$ denote the optimum number of bins for a packing of L, and let $R_A(L) \equiv A(L) / OPT(L)$. The ***asymptotic worst case performance ratio*** R_A^∞ is defined to be:

$$R_A^\infty \equiv \inf\{r \geq 1\text{: } for\ some\ N > 0\ ,\ R_A(L) \leq r\ for\ all\ L\ with\ OPT(L) \geq N\}\ . \qquad (1)$$

The bin packing problem is know to be NP-Hard in the strong sense [13]. We show that the complexity of BP-SIF is the same.

***Claim**: BP-SIF is NP-Hard in the strong sense.*

Proof: We denote by D(BP-SIF) the decision version of BP-SIF and show that it is NP-Complete in the strong sense. We do so by reducing the 3-PARTITION problem to a restricted instance of D(BP-SIF). The 3-PARTITION problem (defined formally below) is known to be NP-Complete in the strong sense [5].

3-PARTITION: given a list L of $n=3m$ integers: $w_1,w_2,...,w_n$ and a bound $B \in Z^+$ such that $B/4 < w_j < B/2$ for $j = 1,...,n$ and $\sum_{j=1}^{n} w_j = mB$, can L be partitioned into m disjoint subsets $S_1,...,S_m$ such that $\sum_{j\in S_i} w_j = B$ for $i = 1,...,m$?

We define D(BP-SIF) as follows: given a list of items L, a size $s(a) \in Z^+$ for each item $a \in L$, a positive integer bin capacity U and a positive integer K, is there a feasible packing of L in K bins of size U? Any instance I of 3-PARTITION can be polynomially transformed into an equivalent instance I' of D(BP-SIF) by setting $U=B$ and $K=m$. To realize the two decision problems are equivalent note that, since the total size of all items is equivalent to the total capacity of all bins, in any "yes" instance of D(BP-SIF) all n items are packed without fragmentation and the packing is therefore also valid for 3-PARTITION. Clearly the packing of any "yes" instance of 3-PARTITION is also valid for D(BP_SIF). This means that the "yes" and "no" instances of the two problems are equivalent. ◊

When packing a list of n items into m bins, the maximum number of fragmentations possible is $n \cdot m$ (each item is fragmented over all bins). From the definition of the problem it is obvious that a good algorithm should try to perform the minimum number of fragmentations. Therefore we would only like to consider algorithms that do not fragment items unnecessarily.

Definition 1: An algorithm A is said to *prevent unnecessary fragmentation* if it follows the following two rules:

1. No unnecessary fragmentation: An item (or fragment of an item) is fragmented only if it is to be packed into a bin that cannot contain it. In case of fragmentation, the item (or fragment) is divided into two fragments. The first fragment must fill one of the bins. The second fragment is packed according to the packing rules of the algorithm.
2. No unnecessary bins: An item is packed in a new bin only if it cannot fit in any of the open bins used by A.

Algorithms that prevent unnecessary fragmentation have the following property.

Lemma 1*: For any algorithm A, that prevents unnecessary fragmentations -* $R_A^\infty \le \frac{U}{U-2}$, $U > 2$.

Proof: Assume that the number of bins used by the algorithm is $A(L) = m$. Since A prevents unnecessary fragmentation it can perform at most m–1 fragmentations while packing m bins (no fragmentation in the last bin), regardless of the size of the list. Each fragmentation adds 2 units of overhead at the most. Therefore, in the worst case, $2(m–1)$ units of overhead are added to the total size of all items. Note that in this case only the last bin may be left unfilled. Assuming the optimal packing does not fragment any item, the number of bins used by it satisfies:

$$OPT(L) \ge \frac{(m-1)U-2(m-1)+1}{U} = (m-1)\frac{U-2}{U} + \frac{1}{U} \quad (2)$$

The asymptotic performance ratio follows:

$$R_A(L) = \frac{A(L)}{OPT(L)} \le \frac{mU}{(m-1)(U-2)+1} \xrightarrow{m\to\infty} R_A^\infty \le \frac{U-2}{U}$$ ◊

Remark: For the more general case, where r units of overhead (instead of one) are added to the size of every fragment, it can be shown by similar arguments, that: $R_A^{\infty} \le \frac{U}{U-2r}, \quad U > 2r$.

We now have an upper bound on the performance ratio of any algorithm. In the remainder of this section we investigate specific algorithms to find their actual performance ratio. For a given algorithm A we define a version of the algorithm that allows item fragmentation and denote it by A_f. We investigate the worst case performance ratio of the following algorithms: NF_f, NFD_f (NFI_f) and FFD_f (BFD_f).

2.1. Next-Fit with item fragmentation - NF_f

The NF_f algorithm is defined similar to the NF algorithm.

Algorithm NF_f - In each stage there is only one open bin. The items are packed, according to their order in the list L, into the open bin. When an item does not fit in the open bin, it is fragmented into two parts. The first part fills the open bin and the bin is closed. The second part is packed into a new bin which becomes the open bin. Offline version of the algorithm sorts the items in decreasing (NFD_f) or increasing (NFI_f) order before packing the list.

The NF_f algorithm is very simple, can be implemented to run in linear time and requires only one open bin (bounded space). However, as we show next, similar to the classical problem, the performance ratio the algorithm achieves is the worst possible.

Theorem 1: *For algorithm* NF_f - $R_{NFf}^{\infty} = \frac{U}{U-2}, \ \forall\ U \ge 6$.

Proof: Lemma1 provides an upper bound on the performance ratio of the algorithm. We present an example that proves the lower bound. Let us first consider the case where the bin size U is an even number. As a worst case example we choose the following list L: The first item is of size $U/2$, the next $(\frac{U}{2}-2)$ items are of size 1. The rest of the list repeats this pattern kU times. An example of such a list, for U=6, is given in Figure 1. The optimal packing avoids fragmentations by packing bins with two items of size $U/2$ or U items of size 1. The total number of bins used is: $OPT(L) = \frac{U}{2}k + (\frac{U}{2}-2)k = (U-2)k$. On the other hand, algorithm NF_f fragments each item of size $U/2$ (except for the first item). Overall $2(kU-1)$ units of overhead are added to the packing and therefore the number of bins used by NF_f is:

$$NF_f(L) = \left\lceil \frac{U}{2}k + 2\frac{kU-1}{U} + (\frac{U}{2}-2)k \right\rceil = \left\lceil Uk - \frac{2}{U} \right\rceil = Uk\,. \tag{3}$$

A worst case example for the case where U is an odd number is similar. The first item in L is of size $(U-1)/2$, the next $(\frac{U-1}{2}-1)$ items are of size 1. The rest of the list repeats this pattern kU times. It is easy to verify that, as in the previous example, $OPT(L)=(U-2)k$, while $NF_f(L)=kU$.

Therefore for each $U \ge 6$, the asymptotic performance ratio is: $R_{NFf}^{\infty} = \frac{NF_f(L)}{OPT(L)} = \frac{U}{U-2}$. ◊

When the bin size is very small the above proof does not hold. For the values $2 < U \le 5$ we can show [11] that the worst case asymptotic performance ratio is: $R_{NFf}^{\infty} = \frac{3}{2}$.

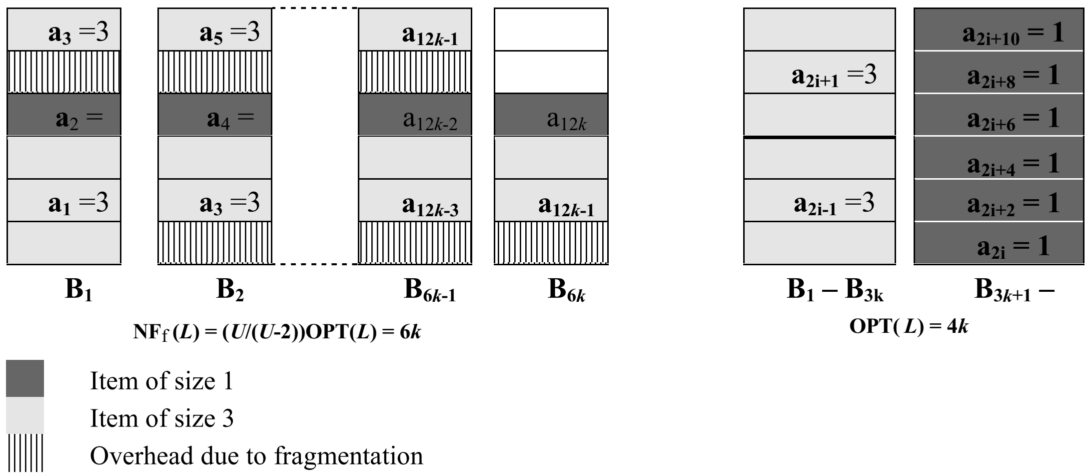

Figure 1: An example for the packing of NF_f for U=6, the list is L = {3,1,3,1....3,1}.

It is interesting to compare the NF_f algorithm to the classic NF algorithm for which the asymptotic worst case performance ratio is: $R^{\infty}_{NF} = \frac{2U}{U+1}$, $\forall\ U \geq 1$. While the performance ratio of NF is increasing with U the performance ratio of NF_f is decreasing with U. This is intuitive since as the bin size gets larger the effective cost of fragmentation gets smaller.

2.1.1. NFD_f and NFI_f Algorithms

Given the poor performance of the NF_f algorithm, one may ask whether sorting the items before applying the NF_f packing, would yield better results. It turns out that, in the worst case, algorithms NFD_f and NFI_f are very similar to the NF_f algorithm. The actual performance ratio of these algorithms depends on the bin size U, but in all cases it is not far from the ratio of NF_f. To avoid dealing with each value of U separately, we present an example for a general value. The example serves as a lower bound on the performance ratio for any value of U, but as we shell see in some cases the ratio may be even worse. Our list of items is made of k items of size U-2 and k items of size 2. The optimal packing uses k bins where each bin contains two items, one of each kind. The NFD_f algorithm first packs k items, of size U-2, into k bins. Then k-1 items of size 2 are packed into $\lceil 2(k-1)/U \rceil$ bins if U is even, or $\lceil 2(k-1)/(U-1) \rceil$ bins if it is odd. The NFI_f algorithm will use the same number of bins, since the only difference is that the items are packed in reverse order. This simple example gives us the following lower bounds:

$$R^{\infty}_{NFD_f} = R^{\infty}_{NFI_f} \geq \frac{U+2}{U}, \text{ for any even } U \geq 6. \tag{4}$$

$$R^{\infty}_{NFD_f} = R^{\infty}_{NFI_f} \geq \frac{U+1}{U-1}, \quad \text{for any odd } U \geq 5. \tag{5}$$

The above example served as a lower bound. We now demonstrate that in some cases the NFD_f and NFI_f algorithms can perform just as bad as NF_f. The first example is for U=5 and a list L={3,...,3,2,...,2}, for which $R^{\infty}_{NFD_f} = R^{\infty}_{NFI_f} = R^{\infty}_{NF_f} = \frac{3}{2}$. To see that such examples are not restricted to low values of bin size, consider U=32 and choose a list of 15k items of size 10 and 15k items of size 6. In the optimal packing the content of each bin is { 10, 10, 6, 6}, therefore $OPT(L)$=7.5k.

The total number of bins used by the algorithms is: $\text{NFD}_f(L)= \text{NFI}_f(L)=8k$, since all bins (save two) contains two units of overhead. The performance ratio in this case is the worst possible:

$$R^{\infty}_{\text{NFI}_f} = R^{\infty}_{\text{NFD}_f}\Big|_{U=32} = \frac{8k}{7.5k} = \frac{32}{30} = \frac{U}{U-2} \,. \tag{6}$$

On the other hand for some values of bin size, NFI_f and NFD_f have a better performance ratio than NF_f. For example when U=6, $R^{\infty}_{\text{NFD}_f} = R^{\infty}_{\text{NFI}_f} = \frac{4}{3}$, while $R^{\infty}_{\text{NF}_f} = \frac{3}{2}$.

2.2. First-Fit Decreasing with item fragmentation - FFD_f

We now develop an algorithm based on the FFD heuristic. Let us first describe algorithm FFD_f which packs items from a list L, into a *fixed* number of m bins.

Algorithm FFD_f - First-Fit Decreasing with item fragmentation: The algorithm packs the items in decreasing order. An item is packed into the lowest indexed bin into which it fits. If an item does not fit into any bin it is fragmented. When fragmenting an item the first fragment fills the lowest indexed bin that is as yet not full. If the second fragment can be packed without fragmentation, it is packed into the lowest indexed bin into which it fits, otherwise another fragmentation is performed according to the above rule.

Remark: Other definitions are possible. For example, upon fragmentation we may choose to insert the second fragment back to the list. Another possibility is to first go over the whole list and pack items without fragmentation and only then pack the remaining items into the available free space.

Note that FFD_f may not be able to pack all the items in L. To ensure all items are packed, we use the following iterative algorithm:

Algorithm FFD_f Iterative (FFD_f-I) - The FFD_f-I algorithm tries to pack the list L into a fixed number of m bins. If it fails it increases m by one and tries again. Let $s(L)$ be the sum of all items in L, the first value of m is: $m_1=\lceil s(L)/U \rceil$, which is the minimum number of bins possible.

The algorithm performs the following steps:

1. Set $m = m_1= \lceil s(L)/U \rceil$.
2. Try to pack the list L into m bins using the FFD_f algorithm.
3. If all items were packed stop.
4. Otherwise set m=m+1 and go to step 2.

It is interesting to see if FFD_f-I improves the performance ratio of NF_f. As we shell see the improvement is significant for small values of bin size.

Theorem 2*: The asymptotic performance ratio of the* FFD_f-I *algorithm satisfies:*

(i) $R^{\infty}_{\text{FFD}_f\text{-I}} \le \frac{U}{U-1}$ *when* $U \le 15$

(ii) $\frac{U}{U-1} < R^{\infty}_{\text{FFD}_f\text{-I}} < \frac{U}{U-2}$ *when* $U \ge 16$

Proof: To prove the theorem we use a property which we call the *border bin property*. We assume FFD_f-I has m bins to pack and number the bins $B_1,...,B_m$ according to the order they are opened by the algorithm. Looking at the level (used space) of the bins at a certain time during algorithm execution, we say that bin B_j, $j<m$, is a *border bin* at that time if its level satisfies: $l(B_j) \neq l(B_{j+1})$.

Claim: *At any time before the first item is fragmented, the packing of* FFD_f-I *contains at most* $2U$ *border bins.*

Proof: The FFD_f-I algorithm packs the items in decreasing size order. We consider the number of border bins before the first item is fragmented. Denote by $BR(k)$ the number of border bins, after k different sizes of items have been packed by FFD_f-I. Clearly $BR(1) \leq 2$, since items of the same size are packed in a similar way. Whenever each additional size is packed the number of border bins may increase by at most two, therefore, $BR(k) \leq BR(k\text{-}1)+2$. Since there are at most U different sizes, $BR(U) \leq 2U$, and the claim follows. ◊

We now proceed to prove the theorem for each range separately.

(i) $U \leq 15$: We establish $\frac{U}{U-1}$ as an upper bound on the asymptotic performance ratio. We can show that as long as the number of bins with 2 units of overhead is small, i.e., O(1), the asymptotic performance ratio cannot exceed $\frac{U}{U-1}$ (the proof is similar to that of Lemma 1). We make the following observation:

Claim: *The final packing of the* FFD_f-I *algorithm* ***cannot*** *contain more than O(1) bins with 2 units of overhead, if one of the following conditions is met:*

1) *Before the first fragmentation occurs, the free space in the bins is 2 or less.*
2) *The fragmented items are of size 4 or less.*

Proof: Consider the moment before the first fragmentation is performed. The first condition is trivial since at most 1 unit of overhead can be packed when the free space in the bins is 2. To prove the second condition, knowing that the free space in the bins is at least 3, we need only test the case of items of size 4 (and bins with free space 3). When an item of size 4 is fragmented over two bins with free space of size 3, only one unit of overhead is packed in each bin, together with a fragment of size 2. The only way to get a bin with 2 units of overhead, is to fragment an item of size 4 over three bins which must have free space of sizes 2,3,3 or 3,2,3; in this case the last bin of each triplet contains 2 units of overhead. However, the number of such triplets is O(1), since each of them contains at least one border bin and according to the *border bins property*, there are at most $2U$ border bins. It follows that the number of bins containing 2 overhead units is less than $2U$, which is O(1) for $m>>U$. ◊

It is easy to verify that the conditions set by the above claim cannot be extended, since fragmenting items of size 5 over bins of size 3 results in one third of the bins containing 2 units of overhead. Therefore, in order to get a significant number of bins with 2 units of overhead, items of size 5 or more must be fragmented. This means that the list should contain items of size 6 or more. Note that if FFD_f-I fragments items of sizes $s(a_i) \geq \frac{U}{2}$, they are also fragmented by the optimal packing. We conclude that in order to create a difference of more than one overhead unit, between the optimal packing and the packing of FFD_f-I, two items of size $s(a_i) \geq 6$ must be packed in a bin and leave a free space of at least 3. To do so, the bin size must satisfy $U \geq 15$. However, in the case of U=15, items of size 5 are packed without fragmentation and therefore the value $\frac{U}{U-1}$ is an upper bound on the asymptotic performance ratio for $U \leq 15$.

(ii) $U \geq 16$: We first prove the lower bound and then the upper bound.

Claim*: For the* FFD_f-I *algorithm -* $\frac{U}{U-1} < R^{\infty}_{\text{FFD}_\text{f}\text{-I}}$.

Proof : For each value $U \geq 16$, we present an example where the performance ratio exceeds $\frac{U}{U-1}$. We first consider the case where U is even and then the case where U is odd.

Case 1: Even U=2u. We choose the following item list- L: k items of size u-2, then k items of size u-3 and finally k items of size 5. The optimal packing fills k bins each with one item of each size, $OPT(L)$=k. When FFD_f-I is applied to L the first k bins contain all items except for the last k/4 items of size 5. In the best case (U is a multiple of 5) no more overhead is produced and $\frac{5k}{4U}$ more bins are used. The total number of bins required by the algorithm is: $\text{FFD}_\text{f}\text{-I}(L) \geq (k + 5k/4U)$. The performance ratio in this case is:

$$R^{\infty}_{\text{FFD}_\text{f}\text{-I}} \geq \frac{4U+5}{4U} > \frac{U}{U-1},\ \forall even\ U \geq 16. \tag{7}$$

Case 2: Odd U=2u+1. This time we choose the following item list: k items of size u-1, then k items of size u-3 and finally k items of size 5. The packing is similar to the one in the last example and the total number of bins used by the algorithm is: $\text{FFD}_\text{f}\text{-I}(L) \geq (k + 7k/6U)$. The example for the odd case provides a slightly lower bound:

$$R^{\infty}_{\text{FFD}_\text{f}\text{-I}} \geq \frac{6U+7}{6U} > \frac{U}{U-1}, \forall odd\ U \geq 17. \quad \Diamond \tag{8}$$

We now turn to the upper bound on the performance ratio. We show that for any bin size U, the performance ratio is smaller than $\frac{U}{U-2}$, which is the upper bound set by Lemma 1.

Claim: *For the* FFD_f-I *algorithm* - $R^{\infty}_{\text{FFD}_\text{f}\text{-I}} < \frac{U}{U-2}$.

Proof : In order to prove the claim we show that the algorithm cannot produce a packing where each bin (maybe except for a negligible number) contains 2 units of overhead. The *border bins property* (page **Error! Bookmark not defined.**) implies that just before the first item is fragmented, the bins are arranged in long sequences where the free space in all bins is equal. The items are also packed in long sequences. Let us assume the free space in a sequence is x and the size of the items packed is y. Obviously $x<y$ otherwise the items are not fragmented. Note that when an item is fragmented over more than 2 bins, only the first and last bins can contain 2 units of overhead. Therefore it is enough to consider only the case where an item is fragmented over two bins. Assume that item number k is fragmented over two bins such that bin number 1 contains a fragment of size α and bin number 2 contains a fragment of size $y-\alpha$. The next item (number k+1) is also fragmented and a fragment of size $x-y+\alpha-2$ is packed in bin number 2. In order to create a repeated cycle of fragmentations the size of the first fragment of each item must be equal, that is $\alpha = x-y+\alpha-2$. This is true for $y = x-2$, but for that value the items are not fragmented in the first place. Since we can not create a repeated cycle of equal size fragmentations, the size of the first fragment packed in a bin increases, until it reach the size of the bin, in which case only one unit of overhead is packed in the bin. This means that at least 1 of every y (the size of the fragmented item) bins contains only one unit of overhead. Since $y \le U/2$ we can establish that:

$$R^{\infty}_{\text{FFD}_\text{f}\text{-I}} \le \frac{U}{U-2+\frac{2}{U}} < \frac{U}{U-2}. \quad \Diamond \tag{9}$$

This concludes the proof of **Error! Reference source not found.**. ◊

The improvement of FFD_f-I over NF_f is significant for low values of bin size ($U \le 15$), which are the most meaningful values (since the performance ratio is decreasing with the bin size). Moreover, FFD_f-I is superior for any value of U. We make the following remarks:

i) For $U \ge 16$, equations **Error! Reference source not found.** and (8) enable us to set a tighter bound on the performance ratio, than what is stated in the theorem.

ii) For each of the following values: $U \in \{7, 9, 10, 11, 13, 14, 15\}$, it is possible to find an example where the ratio is $\frac{U}{U-1}$, hence $R^{\infty}_{\text{FFD}_\text{f}\text{-I}} = \frac{U}{U-1}$, [11].

iii) When $U \le 5$, $R^{\infty}_{\text{FFD}_\text{f}\text{-I}} = 1$, however when $U \in \{6,8\}$ $R^{\infty}_{\text{FFD}_\text{f}\text{-I}} > 1$. This is interesting, since for the classical problem the performance ratio of FFD is: $R^{\infty}_{\text{FFD}} = 1$.

iv) We may define algorithm BFD_f-I in a similar way to FFD_f-I, only this algorithm is based on the Best-Fit Decreasing heuristic. We can show that the BFD_f-I algorithm has the same performance ratio as FFD_f-I [11].

3. Bin Packing with Size-Preserving Fragmentation

In this section we study a different fragmentation cost function in which fragmenting an item does not increase its size. Instead, we assume that packing an item is associated with a cost and fragmentation increases this cost. The cost of packing an item (or fragmenting it) depends on the application. As an example, take the scheduling problem but assume that fragmenting a datagram does not increase its size, because the format of the datagram already includes the fragmentation fields. On the other hand fragmentation requires additional resources from the system (CPU, memory) and takes longer to process. In other applications, such as in stock -cutting problems, it may simply cost to fragment an item (cut a piece of pipe for example) or put it back together. We proceed to formally define the problem.

Bin Packing with Size-Preserving Fragmentation (BP-SPF): We are given a list of n items $L=(a_1,a_2,...,a_n)$, each with a size $s(a_i)\in\{1,2,...,U\}$ and a cost $c(a_i)\in Z^+$. The items must be packed into m identical bins, of size U. It is possible to fragment any item, in which case one unit is added to its cost but does not change its size. The goal is to minimize the total cost.

Denote by $s(L)$ and $c(L)$ the total size and cost of all items, respectively. To ensure all items can be packed, we assume $s(L)\le mU$.

Performance: There are several ways to evaluate the performance of an algorithm for the problem. We observe that since the cost of fragmentation is not related to the size or cost of an item, the additional cost of an algorithm depends only on the number of fragmentations it performs. We therefore chose to evaluate the performance of an algorithm by its *overhead*. For a given list L and algorithm A, let $c(A,L)$ be the total cost of algorithm A, let $c(OPT,L)$ denote the optimal (minimal) cost and define the overhead of A as: $OH_A(L)\equiv c(A,L)-c(OPT,L)$. For the case of $OPT(L)=m$, we define the worst case overhead of algorithm A, OH_A^m, as:

$$OH_A^m \equiv \inf\{h:\ OH_A(L)\le h\ \ for\ \ all\ \ L\ with\ \ OPT(L)=m\}\ .$$

(10)

We first show that the complexity of BP-SPF is similar to that of BP-SIF.

***Claim**: BP-SPF is NP-Hard in the strong sense.*

Proof: The proof is similar to that of BP-SIF. We define the decision version D(BP-SPF) as follows: given a list of items L, a size $s(a)\in Z^+$ and a cot $c(a)\in Z^+$ for each item $a\in L$, a positive integer bin capacity U and two positive integers, K and C, is there a feasible packing of L in K bins of size U such that the total cost is C? Any instance I of 3-PARTITION can be polynomially transformed into an equivalent instance I' of D(BP-SPF) by setting $U=B$, $K=m$ and $C=c(L)$ (which implies that for any "yes" instance all n items are packed without fragmentation). ◊

We consider only algorithms that prevent unnecessary fragmentation (see Definition 1). Such algorithms have the following property.

Lemma 2: *For any algorithm A, that prevents unnecessary fragmentations -* $OH_A^m \le m-1$.

Proof: Since A prevents unnecessary fragmentations, it may perform at most m–1 fragmentations when packing m bins. The maximum cost of algorithm A is therefore: $c(A,L) = c(L) + (m-1)$. Clearly $c(OPT,L) \ge c(L)$, which means that for any list L: $OH_A(L) \le m-1$. ◊

We now examine the performance of the $\mathrm{NF_f}$ and $\mathrm{FFD_f}$ algorithms (defined in Sections 2.1 and 2.2, respectively). We show that the performance of the $\mathrm{NF_f}$ algorithm is the worst possible while $\mathrm{FFD_f}$ performs better.

Theorem 3: *The overhead of algorithm* $\mathrm{NF_f}$ *for every* $m \ge 2$ *is -* $OH_{\mathrm{NF_f}}^m = m-1,\ \forall\ 2 \le U$.

Proof: Lemma 2 provides an upper bound. We show that this is also the lower bound. As a worst case example choose the following list of items: one item of size U–1 followed by m–1 items of size U. The optimal packing causes no fragmentations by packing each item in a bin. The $\mathrm{NF_f}$ algorithm fragments all the items of size U. The total overhead is therefore m–1. ◊

We now turn to the $\mathrm{FFD_f}$ algorithm. We expect $\mathrm{FFD_f}$ to perform better than $\mathrm{NF_f}$ and this is indeed true when the bin size U is small. However, we show that if the bin size is not bounded, the worst case overhead of $\mathrm{FFD_f}$ is the maximum possible, that is, there exist a list L for which, for any value of m, $c(\mathrm{FFD_f},L) - c(OPT,L) = m-1$.

Claim: *For the* $\mathrm{FFD_f}$ *algorithm, for every* $m \ge 2$ *and* $2m+16 < U$ *-* $OH_{\mathrm{FFD_f}}^m = m-1$.

Proof: We choose a bin size satisfying: $U > 2m+16$. The list, L, is made of k repetitions of the following set: L'={U/2+2, U/2+1, U/4+2, U/4+1, U/4–3, U/4–3}. Two bins of size U are needed to pack L', therefore m=2k bins are needed to pack L. The optimal packing cause no fragmentations by packing k bins with items of size U/2+2, U/4+1 and U/4-3, the other k bins are packed with items of size U/2+1, U/4+2 and U/4-3. The $\mathrm{FFD_f}$ algorithm packs the items in the following way: The first 5k items are packed without fragmentations. The free space in the first k bins is U/4–4, in the remaining k bins the free space is 1. Since the free space in all bins is smaller than the size of the items, the items are fragmented. Each item except the last is fragmented over two bins. When packing the last item, bins $B_1,...B_{k-1}$ are full, bin B_k has free space of U/4-k-3 and each bin $B_{k+1},...,B_{2k}$ has free space of 1. As a result, the last item is fragmented over the remaining free space, causing k more fragmentations. The total number of fragmentations is m–1 and the overhead is therefore $OH_{\mathrm{FFD_f}}^m = m-1$. ◊

To hold for every value of m the above claim requires an unbounded bin size. Since we are mainly interested in asymptotic behavior, i.e., U<<m, this is clearly not a practical assumption. For the more reasonable cases where m>U we have the following property of $\mathrm{FFD_f}$.

Claim: *For the* $\mathrm{FFD_f}$, *if* $m > U$ *then there is* $N > 0$ *for which -* $OH_{\mathrm{FFD_f}}^m < m-1,\ \forall N < m$.

Guidelines of Proof: We show that if $U<<m$ either we have $c(A,L) < c(L) + (m-1)$ or, in cases where $c(A,L) = c(L) + (m-1)$, we have $c(OPT,L) > c(L)$. The overhead is therefore always smaller than m–1.

References

[1] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, Algorithm Design for Computer System Design, pages 49-106. Springer-Verlag, Wien, 1984.

[2] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum (ed), PSW publishing, Boston. Approximation Algorithms for NP-Hard Problems, pages 46-93. 1996.

[3] D. K. Friesen and F. S. Kuhl. Analysis of a hybrid algorithm for packing unequal bins. SIAM J. of Computing, vol. 17, pp. 23-40, 1988.

[4] D. K. Friesen and M. A. Langston. Analysis of a compound bin packing algorithm. SIAM J. Disc. Math, vol. 4, pp. 61-79, 1991.

[5] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., San Francisco, 1979.

[6] 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 J. of Computing, 3:299-325, 1974.

[7] D. S. Johnson. Fast algorithms for bin packing. Journal of computer and system Science. vol. 8, pp. 272-314, 1974.

[8] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin packing problem. In Proc. 23rd Ann. Symp on Foundations of Computer Science, pp. 312-320, 1982.

[9] C. C. Lee and D. T. Lee. A simple on-line packing algorithm. J. ACM, vol. 32, pp. 562-572, 1985.

[10] Mandal-CA, Chakrabarti-PP and Ghose-S. Complexity of fragmentable object bin packing and an application. Computers and Mathematics with Applications. vol.35, no.11, pp. 91-7, 1998.

[11] Nir Menakerman and Raphael Rom. Packing problems with item fragmentation. Technical report. Unpublished.

[12] Multimedia Cable Network System Ltd., "Data-Over-Cable Service Interface Specifications - Radio Frequency Interface Specification", July 2000, http://www.cablelabs.com.

[13] J. D. Ullman. Complexity of sequencing Problems. Computer and Job-Shop Scheduling Theory. Wiley and Sons. 1976.