HANDBOOK OF COMBINATORIAL OPTIMIZATION
D.-Z. Du and P.M. Pardalos (Eds.) pp. – – –
©1998 Kluwer Academic Publishers

Bin Packing Approximation Algorithms: Combinatorial Analysis

Edward G. Coffman, Jr.
Bell Labs, Lucent Technologies
E-mail: egc@bell-labs.com

Gabor Galambos
Computer Science Department, Teacher's Training College, Szeged
E-mail: galambos@jgytf.u-szeged.hu

Silvano Martello
DEIS, University of Bologna
E-mail: smartello@deis.unibo.it

Daniele Vigo
DEIS, University of Bologna
E-mail: dvigo@deis.unibo.it

Contents

1 Introduction

In the classical version of the bin packing problem one is given a list $L = (a_1, \ldots, a_n)$ of items (or elements) and an infinite supply of bins with capacity C. A function $s(a_i)$ gives the size of item a_i, and satisfies $0 < s(a_i) \leq C$, $1 \leq i \leq n$. The problem is to pack the items into a minimum number of bins under the constraint that the sum of the sizes of the items in each bin is no greater than C. In simpler terms, a set of numbers is to be partitioned into a minimum number of blocks subject to a sum constraint common to each block. We use the bin packing terminology, as it eases considerably the problem of describing and analyzing algorithms.

The mathematical foundations of bin packing began at Bell Laboratories with the work of M. R. Garey and R. L. Graham in the early 70's. They were soon joined by J. D. Ullman, then at Princeton; these three published the first [49] of many papers to appear in the conferences of the computer science theory community over the next 25 years. The second such paper appeared within months; in that paper, D. S. Johnson [68] extended the results in [49] and studied general classes of approximation algorithms. In collaboration with A. Demers, researchers Johnson, Garey, Graham, and Ullman published the first definitive analysis of bin packing approximation algorithms [72]. In parallel with the research producing this landmark paper, Johnson completed his 1973 Ph.D. thesis at

MIT which gave a more comprehensive treatment and more detailed versions of compact proofs in [72].

The pioneering work in [72] opened an extremely rich research area; it soon turned out that this simple model could be used for a variety of different practical problems, ranging from a large number of cutting stock applications to packing trucks with a given weight limit, assigning commercials to station breaks in television programming, or allocating memory in computers. The problem is well-known to be NP-hard (see Garey and Johnson [50]); hence, it is unlikely that efficient (i.e., polynomial time) optimization algorithms can be found for its solution. Researchers have thus turned to the study of approximation algorithms, which do not guarantee an optimal solution for every instance, but attempt to find a near-optimal solution within polynomial time. Together with closely related partitioning problems, bin packing has played an important role in applications of complexity theory and in the theory of approximation algorithms (see, e.g., the recent book of Hochbaum [62]).

Starting with the seminal papers mentioned above, most of the early research focused on combinatorial analysis of algorithms leading to performance guarantees and bounds on worst-case behavior. In particular, letting $A(L)$ be the number of bins used by an algorithm A and letting $OPT(L)$ be the minimum number of bins needed to pack the elements of L, one tries to find a least upper bound on $A(L)/OPT(L)$ over all lists L (for a more formal definition, see Section 2 below). Much of the research can be divided along the boundary between *on-line* and *off-line* algorithms. In the case of on-line algorithms, items are packed in the order they are encountered in the scan of the given list L; the bin in which an item is packed is chosen without knowledge of items not yet encountered in L. These algorithms are the only ones that can be used in certain situations, where the items to be packed arrive in a sequence according to some physical process and have to be assigned to a bin as soon as they arrive. Off-line algorithms have complete information about the entire list throughout the packing process.

In the last two decades progressively more attention has been devoted to the probabilistic analysis of packing algorithms; a recent book by Coffman and Lueker [26] covers the methodology in some detail (see also the book by Hofri [64, Ch. 10]). Nevertheless, combinatorial analysis remains a central research area and, from time to time, numerous new results need to be collected into survey papers. The first comprehensive surveys of bin packing algorithms were by Garey and Johnson [51] in 1981, and by Coffman, Garey, and Johnson [22] in 1984. The next such survey was written some ten years later by Galambos and Woeginger [46] who gave an overview restricted to on-line algorithms. Very recently two new surveys appeared. Csirik and Woeginger [33] concentrate on on-line algorithms, while Coffman, Garey and Johnson [23] extend the coverage to include off-line algorithms as well; both worst-case and average-case analysis are surveyed in [33] and [23].

In this paper, we give a broad summary of results in the one-dimensional bin-packing arena, concentrating on combinatorial analysis; but in contrast to other recent surveys, we devote considerable space to variants of the classical problem. Important new variants continue to arise in many different settings and help account for the thriving interest in bin packing research. We have not attempted to give a self-contained work, e.g., we do not present all algorithms in detail, nor do we give formal proofs; but we refer to certain proof techniques which have been used frequently to establish the most important results. Also, in our coverage of variants, we continue our restriction to partitioning problems in one dimension. Packing in higher dimensions, e.g., strip packing and two-dimensional bin packing, is itself a big subject, one deserving its own survey.

In Section 2, we cover the main definitions and notation. For the classical bin packing problem, we discuss on-line algorithms in Section 3, and off-line algorithms in Section 4. Section 4.4 is a slight abuse of this organization; that section deals with anomalous behavior as it applies to on-line as well as off-line algorithms. Section 5 concentrates on special cases and variants of the classical problem.

2 Definitions

In the classical problem, the bin capacity C is just a scale factor, so we can, without loss of generality, assume the normalization $C = 1$. Unless stated otherwise, this convention is in force throughout. One of the exceptions will be in the section treating a variant of the classical problem in which bins have varying sizes; in that section, bin sizes will be denoted by $s(B)$.

In the algorithmic context, we classify nonempty bins as either *open* or *closed*. Open bins are available for packing additional items. Closed bins are unavailable, and must remain so, i.e., they are never re-opened. It is convenient to regard a completely full bin as open under any given algorithm until it is specifically closed, even though such bins can receive no further items. We denote the bins by $B_1, B_2, \ldots$. When no confusion arises, we may also use B_j to denote the set of items packed in the j-th bin, and $|B_j|$ to denote the number of such items.

We always pack items organized into a list L (or a list L_j in some indexed set of lists). The notation for list *concatenation* is as usual; $L = L_1 L_2 \ldots\ldots L_k$ means that the items of list L_i are followed by the items of list L_{i+1} for each $i = 1, 2, \ldots, k-1$. When the number of items in a list is needed as part of the notation, we use an index in parentheses; $L_{(n)}$ denotes a list of n items.

If B_j is nonempty, we define its current *content* or *level* as

$$c(B_j) = \sum_{a_i \in B_j} s(a_i)$$

A bin (necessarily empty) is *opened* in order to receive its first item. Of course, it may be closed immediately thereafter, depending on the algorithm and the item size. We assume without loss of generality that all algorithms open bins in order of increasing index. Within any collection of nonempty bins, the *earliest opened*, the *leftmost*, and the *lowest indexed* all refer to the same bin.

In general, we consider lists in which the item sizes are taken from the interval $(0, \alpha]$, with $\alpha \in (0, 1]$. Usually, we assume that $\alpha = \frac{1}{r}$, for some integer $r \geq 1$, and many of our results apply only to $r = \alpha = 1$. We can consider two possible measures of the worst-case behavior of an algorithm. Recall that $A(L)$ denotes the number of bins used by algorithm A to pack the elements of L; OPT denotes an optimal algorithm, one that uses a minimum number of bins. Define the set V_α of all lists L for which the maximum size of the items is bounded from above by α. For every $k \geq 1$, let

$$R_A(k, \alpha) = \sup_{L \in V_\alpha} \left\{ \frac{A(L)}{} \right.$$

(The Salzer sequence was defined for $r = 1$.) It was conjectured by Golomb that for $r = 1$ this sequence gives the closest approximation to 1 from below among the approximations by sums of reciprocals of k integers, the basis of the conjecture being

$$\sum_{i=1}^{k} \frac{1}{t_i(1)} + \frac{1}{t_{k+1}(1) - 1} = 1$$

Furthermore, it is easily proved that the following expression is valid for the Golomb sequences:

$$\frac{(r-1)}{t_1(r)} + \sum_{i=2}^{k} \frac{1}{t_i(r)} + \frac{1}{t_{k+1}(r) - 1} = 1 \qquad \text{for any } r \geq 1$$

On the other hand, the following value appears in several results:

$$h_\infty(r) = 1 + \sum_{i=2}^{\infty} \frac{1}{t_i(r) - 1}$$

The first few values of $h_\infty(r)$ are: $h_\infty(1) \approx 1.69103$, $h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$.

3 On-line Algorithms

Recall that a bin-packing algorithm is called *on-line* if it packs each element as soon as it is inspected, without any knowledge of the elements not yet encountered (either the number of them or their sizes). We start with results for some classical algorithms, and then generalize to the *Any-Fit* and the *Almost Any-Fit* classes of on-line algorithms. We will also consider the subclass of *bounded-space* on-line algorithms: An algorithm is bounded-space if the number of open bins at any time in the packing process is bounded by a constant. (The practical significance of this condition is clear; for example, we may have to load trucks at a depot where only a limited number of trucks can be at the loading dock). We will conclude this section with a detailed discussion of lower bounds on the APR's of certain classes of on-line algorithms, but before doing so, we present the most recent variations of old algorithms and the current best-in-class algorithms. We remind the reader that a discussion of the anomalous behavior occurring in on-line algorithms has been deferred to Section 4.4.

3.1 Classical algorithms

We summarize the conventions to be used in this section as follows: In describing an on-line algorithm A for packing list $L = (a_1, \ldots, a_n)$, we assume without loss of generality that

1. All bins are initially empty.
2. Bins are opened in the sequence $B_1, B_2, \ldots$.
3. A begins by packing a_1 into B_1 and thereafter packs items in the order $a_2, a_3, \ldots$.

Occasionally, it will be convenient, just before a decision point, to refer to the *current* item as the next item to be packed; right after A packs a_i, $i < n$, a_{i+1} becomes the new current item.

A simple approach is to pack the bins one at a time according to

Next-Fit (NF): After packing the first item, NF packs each successive item in the bin containing the last item to be packed, if it fits in that bin; if it does not fit, NF closes that bin and packs the current item in an empty bin.

The time complexity of NF is clearly $O(n)$. Note that only one bin is ever open under NF, so it is bounded-space. This advantage is compensated by a relatively poor APR, however.

Theorem 3.1 (Johnson et al. [72]). *We have*

$$R^{\infty}_{\mathrm{NF}}(\alpha) = \begin{cases} 2 & \text{if } \frac{1}{2} < \alpha \leq 1 \\ (1-\alpha)^{-1} & \text{if } 0 < \alpha \leq \frac{1}{2} \end{cases}$$

Fisher [37] discovered an interesting property that NF does not share with other classical approximation algorithms. He proved that NF packs any list and its reverse into the same number of bins. The conspicuous disadvantage of NF is that it closes bins that could be used for packing later items. An immediate improvement would seem to be never to close bins. But then the next question is: If an item can be put into more than one open bin, which bin should be selected? One possible rule drawn from scheduling theory (where it is known as the greedy or largest-processing-time rule) is the following:

Worst-Fit (WF): If there is no open bin in which the current item fits, then WF packs the item in an empty bin. Otherwise, WF packs the current item into an open bin of smallest content in which it fits; if there is more than one such bin, WF chooses the lowest indexed one.

Although we might expect WF to behave better than NF, it does not.

Theorem 3.2 (Johnson [70]). *We have for all* $\alpha \in (0, 1]$

$$R^{\infty}_{\mathrm{WF}}(\alpha) = R^{\infty}_{\mathrm{NF}}(\alpha)$$

To achieve smaller APR's, there are many better rules for choosing from among the open bins. One that quickly comes to mind is:

First-Fit (FF): FF packs the current item in the lowest indexed nonempty bin in which it fits, assuming there is such a bin. If no such bin exists, FF packs the current item in an empty bin.

A natural complement to WF packs each item into a bin that *minimizes* the space left over.

Best-Fit (BF): If there is no open bin in which the current item fits, then BF packs the item in an empty bin. Otherwise, BF packs the current item into an open bin of largest content in which it fits; if there is more than one such bin BF chooses the lowest indexed one.

By adopting appropriate data structures for representing packings, it is easy to verify that the time complexity of these algorithms is $O(n \log n)$. The analysis of the worst-case behavior of the packings they produce is far more complicated. The basic idea of the upper-bound proofs is the *weighting function* technique, which has played a fundamental role in bin packing theory. So, before proceeding with other algorithms, we describe this technique, which was introduced in [49, 72] and subsequently applied in many other papers (see, e.g, [7, 8, 42, 70, 83]).

3.2 Weighting functions

To bound the asymptotic worst-case behavior of an algorithm A, we can try to find a function $W_A : (0,1] \longrightarrow I\!R$ with the properties: (i) There exists a constant $K \geq 0$ such that for any list L

$$\sum_{a \in L} W_A(a) \geq A(L) - K \tag{1}$$

and (ii) there exists a constant K^* such that for any set B of items summing to no more than 1

$$\sum_{a \in B} W_A(a) \leq K^* \tag{2}$$

The value $W_A(a)$ is the *weight* of item a and W_A is a *weighting function* for A. Note that (1) requires that for large packings (large $A(L)$), the average total weight of the items in a bin must satisfy a lower bound close to 1 for all lists L. On the other hand, (2) says that the total weight in any bin of any packing is at most K^*, so for an optimal packing

$$\sum_{a \in L} W_A(a) = \sum_{i=1}^{OPT(L)} \sum_{a \in B_i} W_A(a) \leq K^* OPT(L) \tag{3}$$

Together, (1) and (3) obviously imply the bound $A(L) \leq K^* OPT(L) + K$, and hence $R_A^\infty \leq K^*$. We remark that the technique above is only representative; it is easy to find weaker conditions on the weighting function which will produce the same upper bound for the APR.

The proof of the NF upper bound is not appreciably simplified by a weighting function argument, but it does offer a simple example of such arguments. Consider the case $\alpha = 1$ and define $W_{\mathrm{NF}}(a) = 2s(a)$ for all a. We need but one observation about NF when $NF(L) > 1$: Since the first item of B_{j+1} did not fit in B_j, $1 \leq j < NF(L)$, the sum of the item sizes in $B_j \cup B_{j+1}$ exceeds 1 and hence the sum of the weights of the items in $B_j \cup B_{j+1}$ exeeds 2. Then

$$\begin{aligned}\sum_{i=1}^{NF(L)} \sum_{a \in B_i} W_{\mathrm{NF}}(a) &\geq \sum_{i=1}^{\lfloor NF(L)/2 \rfloor} \sum_{a \in B_{2i-1} \cup B_{2i}} W_{\mathrm{NF}}(a) \\ &\geq 2\lfloor NF(L)/2 \rfloor \geq NF(L) - 1\end{aligned}$$

so (1) holds with $K = 1$. The inequality (2) with $K^* = 2$ is immediate from the definition of W_{NF}, so $R_{\mathrm{NF}}^{\infty} \leq 2$, as desired.

There is no systematic way to find appropriate weighting functions, and the approach can be difficult to work out. For example, consider the proof of the following result.

Theorem 3.3 (Johnson et al. [72]). *We have*

$$R_{\mathrm{FF}}^{\infty}(\alpha) = R_{\mathrm{BF}}^{\infty}(\alpha) = \begin{cases} \frac{17}{10} & \text{if } \frac{1}{2} < \alpha \leq 1 \\ 1 + \lfloor \frac{1}{\alpha} \rfloor^{-1} & \text{if } 0 < \alpha \leq \frac{1}{2} \end{cases}$$

The proof of the $\frac{17}{10}$ upper bound consists of verifying that the following weighting function suffices

$$W(x) = \begin{cases} \frac{6}{5}x & \text{if } 0 \leq x \leq \frac{1}{6} \\ \frac{9}{5}x - \frac{1}{10} & \text{if } \frac{1}{6} < x \leq \frac{1}{3} \\ \frac{6}{5}x + \frac{1}{10} & \text{if } \frac{1}{3} < x \leq \frac{1}{2} \\ \frac{6}{5}x + \frac{2}{5} & \text{if } \frac{1}{2} < x \leq 1 \end{cases}$$

and it requires a substantial effort. (The argument reduces to several pages of case analysis.) Moreover, despite a few hints that emerge in this effort, a clear understanding of how this function was obtained in the first place requires still more effort.

Theorem 3.3 for $\alpha \leq \frac{1}{2}$ can be proved without weighting functions (see [72]), but the reader may find it instructive to prove the $1 + \lfloor \frac{1}{\alpha} \rfloor^{-1}$ upper bound with the weighting function

$$W_{\mathrm{FF}}(a) = \frac{r+1}{r} s(a), \ \alpha = \frac{1}{r}, \ r \geq 2$$

Sequences of specific examples establish lower bounds for $R_A^{\infty}(\alpha)$. In particular, one seeks a sequence of lists $L_{(n_1)}, L_{(n_2)}, \ldots$ satisfying $L_{(n_k)} \in V_\alpha$ $(k = 1, 2, \ldots)$, $\lim_{k \to \infty} OPT(L_{(n_k)}) = \infty$, and for some constant K_*,

$$\lim_{k \to \infty} \frac{A(L_{(n_k)})}{OPT(L_{(n_k)})} = K_*$$

Then $R_A^{\infty}(\alpha) \geq K_*$, and if K_* is equal to K^* of the upper bound analysis, we have the APR for the algorithm considered. Finding worst-case examples can be anywhere from quite easy to quite hard. For example, the reader would have little trouble with NF, but would probably find FF to be quite challenging.

3.3 Any-Fit and Almost Any-Fit algorithms

The algorithms described so far belong to a much larger class of on-line heuristics satisfying similar worst-case properties. It is clear that FF, WF, and BF satisfy the following condition:

Any-Fit constraint: *If $B_1, \ldots, B_j$ are the current non-empty bins, then the current item will not be packed into B_{j+1} unless it does not fit in any of the bins $B_1, \ldots, B_j$.*

We denote the class of on-line heuristics satisfying the Any-Fit constraint by $\mathcal{AF}$. The following result shows that FF and WF are best and worst algorithms in $\mathcal{AF}$, in the APR sense.

Theorem 3.4 (Johnson [70]). *For every algorithm $A \in \mathcal{AF}$ and for every $\alpha \in (0, 1]$*

$$R^{\infty}_{\mathrm{FF}}(\alpha) \leq R^{\infty}_{A}(\alpha) \leq R^{\infty}_{\mathrm{WF}}(\alpha)$$

By a slight tightening of the Any-Fit constraint, we can eliminate the high-APR algorithms like WF and define a class of heuristics all having the same APR.

Almost Any-Fit constraint: *If $B_1, \ldots, B_j$ are the current non-empty bins, and B_k ($k \leq j$) is the unique bin with the* smallest content*, then the current item will not be packed into B_k unless it does not fit in any of the bins to the left of B_k.*

Clearly, WF does not satisfy this condition, but it is easy to verify that both FF and BF do. We denote the class of on-line algorithms satisfying both constraints above by $\mathcal{AAF}$.

Theorem 3.5 (Johnson [70]). $R^{\infty}_{A}(\alpha) = R^{\infty}_{\mathrm{FF}}(\alpha)$ *for every A in $\mathcal{AAF}$ and $\alpha \in (0, 1]$.*

Almost Worst-Fit (AWF) is a modification of WF whereby the current item is always placed in a bin having the second lowest content, if such a bin exists and the current item fits in it; the current item is packed in a bin with smallest content only if it fits nowhere else. Interestingly, though it seems to differ little from WF, AWF has a substantially better APR, since it is in $\mathcal{AAF}$ and hence $R^{\infty}_{\mathrm{AWF}}(\alpha) = R^{\infty}_{\mathrm{FF}}(\alpha)$.

3.4 Bounded-space algorithms

An on-line bin packing algorithm uses *k-bounded space* if, for each item, the choice of where to pack it is restricted to a set of at most k open bins. We obtain bounded-space counterparts of the algorithms of the previous section by specifying a suitable policy for closing bins.

As previously observed, NF uses only 1-bounded space; the only algorithm to use less is the trivial algorithm that puts each item in a separate bin. To improve on the APR of NF, yet stay with bounded-space algorithms, Johnson [69] proposed an algorithm that packs items according to the First-Fit rule, but considers as candidates only the k most recently opened bins; when a new bin has to be opened and there are already k open bins, then the lowest-indexed open bin is closed. We expect that the APR of the resulting algorithm, which is known as *Next-k-Fit* (NF$_k$), tends to $\frac{17}{10}$ as k increases. Finding the exact bound was not an easy task, although Johnson did give a narrow range for the APR. Later, Csirik and Imreh [30] constructed the worst-case sequences, and later still, Mao was able to prove the exact bound.

Theorem 3.6 (Mao [87]). *For any $k \geq 2$,* $\quad R^{\infty}_{\mathrm{NF}_k} = \frac{17}{10} + \frac{3}{10(k-1)}$.

In general, we define a bounded-space algorithm by specifying the packing and closing rules. An interesting class of such rules is based on FF and BF as follows.

- **Packing rules:** the elements are packed following either the First-Fit rule or the Best-Fit rule.

- **Closing rules:** the next bin to close is either the the lowest indexed one, or the one of largest content.

The algorithm that uses packing rule X, closing rule Y, and k-bounded space is denoted by AXY_k, where $X =$ F or B for FF or BF and $Y =$ F or B for the lowest-indexed (First) open bin or the largest-content (Best) open bin. With this terminology NF$_k$ can also be classified as AFF$_k$. Note that, independently of the chosen rules, if $k = 1$ then we always get NF.

Algorithm ABF$_k$ was first analyzed by Mao, who called it *Best-k-Fit.* He proved that, for any fixed $k \geq 2$, this algorithm is slightly better than NF$_k$.

Theorem 3.7 (Mao [86]). *For any $k \geq 2$,* $\quad R^{\infty}_{\mathrm{ABF}_k} = \frac{17}{10} + \frac{3}{10k}$.

The tight asymptotic bound for AFB$_k$ was found by Zhang.

Theorem 3.8 (Zhang [105]). *For any $k \geq 2$,* $\quad R^{\infty}_{\mathrm{AFB}_k} = R^{\infty}_{\mathrm{NF}_k}$.

Finally, consider the algorithm ABB$_k$ whose asymptotic behavior is, rather surprisingly, independent of $k \geq 2$, and equal to that of FF and BF.

Theorem 3.9 (Csirik and Johnson [31]). *If $k \geq 2$ then* $R^{\infty}_{\mathrm{ABB}_k} = \frac{17}{10}$.

Since all of the above algorithms fulfill the Any-Fit constraint with respect to the open bins, the overall bound $R^{\infty}_A \geq \frac{17}{10}$ is to be expected from Theorem 3.4. A better on-line algorithm can only be obtained without the Any-Fit constraint. In the remaining part of this section, we show how the fruitful idea of *reservation techniques* (introduced by Yao [102] for unbounded-space algorithms and discussed in the next section) led to on-line algorithms which are neither in class $\mathcal{AF}$ nor in $\mathcal{AAF}$.

Yao's idea appeared in the work of Lee and Lee [83] who developed the *Harmonic-Fit* algorithm, which will be denoted by HF$_k$ since, for the case $\alpha = 1$, it uses at most k open bins. The algorithm divides the interval $(0,1]$ into subintervals $I_j = (\frac{1}{j+1}, \frac{1}{j}]$ $(1 \le j \le k-1)$ and $I_k = (0, \frac{1}{k}]$. An element is called an *I_j-element* if its size belongs to interval I_j. Similarly, there are k different types of bin: an *I_j-bin* is reserved for I_j-elements only. An I_j-element is always packed into an I_j-bin following the Next-Fit rule, and so at most k bins are open at the same time. Galambos [42] extended the idea to general α. Observe that, if we choose $\alpha \in \left(\frac{1}{r+1}, \frac{1}{r}\right]$, then the number, say M, of bin types exceeds the space bound k by $r-1$; this is because I_j-bins for $j < r$ are never opened. Instead of our notation HF_k with k the space bound, the literature often uses HF_M with $M = k + r - 1$ being the number of bin types.

The general APR can be formulated as follows. (See the end of Section 2 for the definitions of the quantities $t_s(r)$ and $h_\infty(r)$.)

Theorem 3.10 (Lee and Lee [83], Galambos [42]). *Suppose that $L \in V_\alpha$ with $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$ for some positive integer r, and choose any sequence k_s, $s \ge 1$, such that $t_s(r) < k_s + 1 \le t_{s+1}(r)$. Then*

$$\lim_{s\to\infty} R^\infty_{\mathrm{HF}_{k_s}}(\alpha) = \lim_{s\to\infty} \left(1 + \sum_{j=2}^{s} \frac{1}{t_j(r) - 1} + \frac{k_s + r - 1}{(t_{s+1}(r) - 1)(M - 1)}\right) = h_\infty(r)$$

The results in [83, 42] gave tight bounds only for the cases $k = t_j(r) - r + 1$ and $k = t_{j+1}(r) - r$ for integers $j \ge 1$. Also, considering only the $\alpha = 1$ case, one can see that, to obtain an APR better than $\frac{17}{10}$, at least 7 open bins are needed. These observations raised two further questions:

- For the case $\alpha = 1$, is there an on-line, bounded-space algorithm that uses fewer than 7 bins and has an APR better than $\frac{17}{10}$?
- What are tight bounds on HF_k for specific k?

An affirmative answer to the first question was given by Woeginger [100]. Using a more sophisticated interval structure, one based on the Golomb sequences, the performance of his *Simplified Harmonic* (SH_k) algorithm undershot the $\frac{17}{10}$ bound with 6 open bins; precisely, $R^\infty_{\mathrm{SH}_6} \approx 1.69444$. Moreover, Woeginger proved the following deeper, more general result.

Theorem 3.11 (Woeginger [100]). *To achieve the worst-case performance ratio of heuristic HF_k with k open bins and $\alpha = 1$, heuristic SH_k only needs $O(\log\log k)$ open bins.*

The second question was investigated by Csirik and Johnson [31] and van Vliet [97, 98]. They gave tight bounds for the case $\alpha = 1$ with $k = 4$ and 5. Tight bounds for further k remain open problems.

Similarly, the general case has not been discussed exhaustively, and some questions raised by Woeginger [100] are still open:

- What is the smallest k such that there exists an on-line heuristic using k-bounded space and having an APR strictly less than $\frac{17}{10}$?

k	NF$_k$	ABF$_k$	ABB$_k$	HF$_k$ $\leq$	SH$_k$	best
2	2.00000	1.85000	1.70000	2.00000*	2.00000	1.70000
3	1.85000	1.80000	1.70000	1.75000*	1.75000	1.75000
4	1.80000	1.77500	1.70000	1.71429*	1.72222	1.70000
5	1.77500	1.76000	1.70000	1.70000*	1.70000	1.70000
6	1.76000	1.75000	1.70000	1.70000*	1.69444	1.69444
7	1.75000	1.74286	1.70000	1.69444*	1.69388	1.69388
8	1.74286	1.73750	1.70000	1.69388	1.69106	1.69106
9	1.73750	1.73333	1.70000	1.69345	1.69104	1.69104
10	1.73333	1.73000	1.70000	1.69312	1.69104	1.69104
42	1.70732	1.70714	1.70000	1.69106*	1.69103	1.69103
$+\infty$	1.70000	1.70000	1.70000	1.69103	1.69103	1.69103

Table 1: APR values for bounded-space algorithms, rounded to five decimals. The values in column HF$_k$ $\leq$ are upper bounds, and are tight if starred. Note that $42 = t_4(1) - 1$.

- What is the best possible APR for any on-line heuristic using 2-bounded space? (ABB$_2$ achieves a worst-case ratio of $\frac{17}{10}$.)
- If we consider only algorithms that pack the items by the Next-Fit rule according to some fixed partition of $(0, 1]$ into k subintervals, which partition gives the best APR ? (It is known that for $k \leq 2$ the best possible APR is 2 (see Csirik and Imreh [30]), but for $k \geq 3$ no tight bound is known.)

Tables 1 and 2 show the best results known for bounded-space algorithms. Note that the worst-case ratios of all algorithms in Table 1 are never smaller than $h_\infty(1) \approx 1.69103$. As pointed out by Lee and Lee [83] for the $\alpha = 1$ case, bounded-space algorithms cannot do better. The result holds for general α too, as shown by Galambos, i.e.,

Theorem 3.12 (Lee and Lee [83], Galambos [42]). *Every bounded-space on-line bin packing algorithm A satisfies $R_A^\infty(\alpha) \geq h_\infty(r)$ for all α, $\frac{1}{r+1} < \alpha \leq \frac{1}{r}$.*

We note finally that none of the known bounded-space algorithms achieves the lower bound using a finite number of open bins. We will later show (see Section 5.1) that the bound can be achieved with three open bins if repacking among the open bins is allowed, but without such a relaxation the question remains open.

3.5 Variations and the best-in-class

Chronologically, Yao [102] was the first to break through the $h_\infty(1)$ barrier with his *Refined First-Fit* (RFF) algorithm. RFF classifies items into types 1, 2, 3, or 4 according as their sizes are in the respective intervals $(0, \frac{1}{3}]$, $(\frac{1}{3}, \frac{2}{5}]$, $(\frac{2}{5}, \frac{1}{2}]$ and $(\frac{1}{2}, 1]$. RFF packs 4 sequences of bins, one for each type. With one exception, RFF packs type-i items into the sequence of type-i bins using First Fit. The exception is that every sixth type-2 item (with a size

k	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
2	2.00000*	1.50000*	1.33333*	1.25000*	1.20000*	1.18888*
3	1.75000*	1.44444*	1.31250*	1.24000*	1.19444*	1.16326*
4	1.71429*	1.43750	1.31000	1.23888	1.19387	1.16294
5	1.70000*	1.43333	1.30833	1.23809	1.19345	1.16269
6	1.70000*	1.43055	1.30714	1.23750	1.19312	1.16250
7	1.69444*	1.42857	1.30625	1.23703	1.19285	1.16233
8	1.69388	1.42708	1.30555	1.23666	1.19264	1.16220
9	1.69345	1.42592	1.30500	1.23636	1.19246	1.16208
10	1.69312	1.42499	1.30454	1.23611	1.19230	1.16198
$+\infty$	1.69103	1.42307	1.30238	1.23441	1.19102	1.16102

Table 2: APR values for $\mathrm{HF}_k(\frac{1}{r})$. Starred values are tight.

in $(\frac{1}{3}, \frac{2}{5}])$ is thrown in with the type-4 items, i.e., packed by First Fit into the sequence of type-4 bins. It is easily verified that RFF uses unbounded space and has a time complexity $O(n \log n)$. Yao proved that $R^{\infty}_{\mathrm{RFF}} = \frac{5}{3} = 1.666\ldots$. We remark that Yao did not use the usual weighting function technique, but based his proof on enumeration of the elements in each class. Also, the APR remains unchanged if the special treatment given every sixth type-2 item is instead given every mth type-2 item, where m is taken to be one of 7, 8, or 9.

It was immediately clear that this reservation technique was a promising approach, a fact supported by the Harmonic-Fit algorithm discussed in the previous section. The main disadvantage of the latter algorithm is that each I_1-element, even with a size slightly over $\frac{1}{2}$, is packed alone into a bin. An immediate improvement is to try to add other items to these bins. In their algorithm *Refined Harmonic-Fit* (RHF), Lee and Lee [82, 83] modified HF_{20} by subdividing intervals I_1 and I_2 into two subintervals:

$$\begin{aligned} I_1 &= I_{1,s} \cup I_{1,b}, \\ I_2 &= I_{2,s} \cup I_{2,b} \end{aligned}$$

with $I_{2,s} = (\frac{1}{3}, \frac{37}{96}]$, $I_{2,b} = (\frac{37}{96}, \frac{1}{2}]$, $I_{1,s} = (\frac{1}{2}, \frac{59}{96}]$ and $I_{1,b} = (\frac{59}{96}, 1]$. This brought the number of bin types to $M = 22$. The packing strategy for I_j-elements ($3 \le j \le M - 2 = 20$), $I_{1,b}$-elements and $I_{2,b}$-elements is the same as in Harmonic Fit, but the $I_{1,s}$-elements and the $I_{2,s}$-elements are allowed to share the same bins in certain situations. We omit the details and simply point out that the time complexity of RHF is $O(n)$, but that the algorithm is no longer bounded-space. Its APR is given by

Theorem 3.13 (Lee and Lee [83]). $R^{\infty}_{\mathrm{RHF}} \le \frac{373}{228} \approx 1.63596$.

It is not known whether this bound is tight.

In 1989 several improved algorithms were presented by Ramanan et al. [91]. The first one, called *Modified Harmonic-Fit* (MHF), applies Yao's idea in a more sophisticated way.

Instead of choosing $\frac{59}{96}$ as the point dividing $(\frac{1}{2}, 1]$, the problem is handled in a more general fashion. Let the number of bin types satisfy $M \geq 5$, and consider the subdivision of $(0, 1]$:

$$(0,1] = \bigcup_{j=1}^{M-2} I_j$$

with $I_1 = I_{1,s} \cup I_{1,b}$ and $I_2 = I_{2,s} \cup I_{2,b}$ as earlier, but now

$$\begin{array}{rclcrcl} I_{1,s} & = & (\frac{1}{2}, 1-y] & \quad & I_{2,s} & = & (\frac{1}{3}, y] \\ I_{1,b} & = & (1-y, 1] & \quad & I_{2,b} & = & (y, \frac{1}{2}] \end{array}$$

for some y satisfying $\frac{1}{3} < y < \frac{1}{2}$. Initially, the set of empty bins is divided into M infinite classes, each associated with a subinterval. All $I_{1,b}$-bins, $I_{2,s}$-bins, $I_{2,b}$-bins and I_j-bins $(3 \leq j \leq M-2)$ are only used to pack elements from the associated interval (as in Harmonic Fit). $I_{1,s}$-bins on the other hand can contain $I_{1,s}$-elements and some of the $I_{2,s}$-elements and I_j-elements $(j \in \{3, 6, 7, \ldots, M-2\})$. The algorithm also includes more complicated rules for packing and for deciding when items with sizes in different intervals can share the same bin. For this algorithm with $M = 40$, we have the following bounds.

Theorem 3.14 (Ramanan et al. [91]).

$$1.6156146 < \frac{3}{2} + \frac{1}{9} + \frac{1}{222} - \frac{1}{987012} \leq R^\infty_{\mathrm{MHF}} < \frac{3}{2} + \frac{1}{9} + \frac{1}{222} = 1.615615\ldots.$$

The above algorithm was further generalized by Ramanan et al.[91] who introduced a sequence of classes of on-line linear-time algorithms, called $C^{(h)}$; an algorithm A belongs to $C^{(h)}$, for a given $h \geq 1$, if it divides $(0, 1]$ into disjoint subintervals including

$$\begin{array}{rclrcll} I_{1,b} & = & (1-y_1, 1], & I_{2,b} & = & (y_h, \frac{1}{2}], & \\ I_{1,j} & = & (1-y_{j+1}, 1-y_j], & I_{2,j} & = & (y_{h-j}, y_{h-j+1}], & 1 \leq j \leq h, \end{array}$$

and

$$\begin{array}{rclrcll} I_{\lambda,1} & = & (0, \lambda], & I_{\lambda,2} & = & (\lambda, \frac{1}{3}], & 0 < \lambda \leq \frac{1}{3} \end{array}$$

where $\frac{1}{3} = y_0 < y_1 < \ldots < y_h < y_{h+1} = \frac{1}{2}$, and $I_{\lambda,2} = \emptyset$ if $\lambda = \frac{1}{3}$. Elements are classified, as usual, according to the intervals in which their sizes fall.

Note that algorithm MHF belongs to $C^{(1)}$. Ramanan et al. [91] developed an algorithm (*Modified Harmonic-2*, MH2), which is in $C^{(2)}$, and proved that $R^\infty_{\mathrm{MH2}} \leq 1.612\ldots$. The algorithm is quite elaborate and beyond the scope of this survey. The authors discussed further improvements aimed at reducing the APR to 1.59. They also proved the lower bound result

Theorem 3.15 (Ramanan et al. [91]). *There is no on-line algorithm A in $C^{(h)}$ such that $R^\infty_A < \frac{3}{2} + \frac{1}{12} = 1.58333\ldots$*

The HARMONIC+1 algorithm of Richey [92] subdivides intervals $(\frac{1}{2}, 1]$ and $(\frac{1}{3}, \frac{1}{2}]$ very finely (using 76 classes of subintervals!) in order to allow a precise item pairing in these two intervals. It also allows items of size $(\frac{1}{4}, \frac{1}{3}]$ to be mixed with larger items of various sizes and not just with those of size at least $\frac{1}{2}$. This algorithm has the current best APR.

Theorem 3.16 (Richey [92]). $1.5874 \le R^\infty_{\text{HARMONIC+1}} \le 1.587936$.

Richey also introduced a class of algorithms which differs from $C^{(h)}$ in the way items with sizes at most $\frac{1}{3}$ are packed. Richey's main result is that the above lower bound remains valid for any algorithm A in this new class.

3.6 APR lower bounds

In previous sections we mentioned some results concerning lower bounds on the APR, but we have yet to consider the fundamental problem: What is the best an on-line algorithm can do in the asymptotic worst case? In this section, we discuss lower bound results in chronological order.

Most of the existing lower bounds come from the same idea. To obtain a good packing, it seems advisable to first pack the large elements so that either a bin is 'full enough' or its empty space can be reduced by subsequent small elements. Therefore, if we want to force bad behavior on a heuristic algorithm A, we should challenge it with a list in which small items come first. If A adopts a policy that packs these small items tightly, then it will not be able to find a good packing for the large items which may come later. If instead, A leaves space for large items while packing the small ones, then the expected large items might not appear in the list. In both cases the resulting packing will be poor.

To give a more precise description, let us consider a simple example involving two lists L_1 and L_2, each containing n identical items. The size of each element in L_1 is $\frac{1}{2} - \varepsilon$, and the size of each element in L_2 is $\frac{1}{2} + \varepsilon$. We investigate the asymptotic behavior of an arbitrary approximation algorithm A on two lists: L_1 alone and the concatenated list L_1L_2. It is easy to see that $OPT(L_1) = \frac{n}{2}$, and $OPT(L_1L_2) = n$. Let us examine the behavior of our algorithm on L_1: it will pack some elements alone into bins (say x of them) and it will match up the remaining $n - x$. Hence $A(L_1) = \frac{n+x}{2}$. For the concatenated list L_1L_2, when processing the elements of L_2, the best that A can do is to add one element of L_2 to each of x bins containing a single element of L_1, and to pack alone the remaining $n - x$ elements. Therefore, $A(L_1L_2) = \frac{3n-x}{2}$. Since n may be arbitrarily large,

$$R^\infty_A \ge \max\left\{\frac{A(L_1)}{OPT(L_1)}, \frac{A(L_1L_2)}{OPT(L_1L_2)}\right\} = \max\left\{\frac{n+x}{n}, \frac{3n-x}{2n}\right\}$$

for which the minimum is attained when $x = \frac{n}{3}$, implying a lower bound of $\frac{4}{3}$ for the APR of *any* on-line algorithm A.

The above idea can be easily generalized by taking a carefully chosen series of lists $L_1, \ldots, L_k$, and evaluating the performance of a heuristic on the concatenated lists $L_1 \ldots L_j$, $(1 \le j \le k)$. The first step along these lines was made by Yao [102]. He proved a lower bound of $\frac{3}{2}$ based on three lists of equal-size elements, the sizes being $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$ and $\frac{1}{6} - 2\varepsilon$. Using Salzer's sequences, Brown [13] and Liang [85] independently gave a further improvement to $1.53634577\ldots$. (The largest sizes in their sequences were: $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$, $\frac{1}{7} + \varepsilon$, $\frac{1}{43} + \varepsilon$, and $\frac{1}{1807} + \varepsilon$.) Galambos [42] used the Golomb sequences to extend the

r	Lower bound	R^∞_{HF}	R^∞_{FF}	R^∞_{NF}	Best known
1	1.54015	1.69103	1.70000	2.00000	1.58872
2	1.38965	1.42307	1.50000	2.00000	1.42307
3	1.29144	1.30238	1.33333	1.50000	1.30238
4	1.22986	1.23441	1.25000	1.33333	1.23441
5	1.18888	1.19102	1.20000	1.25000	1.19102
6	1.15891	1.16102	1.16667	1.20000	1.16102

Table 3: Lower bounds and R^∞_A for various algorithms.

idea to general α. The proof in [42] was considerably simplified by Galambos and Frenk [43]. Van Vliet gave an exhaustive analysis of the lower bound constructions with a linear programming technique applied to all α: his lower bound is the best so far.

Theorem 3.17 (van Vliet [96]). *For any on-line algorithm A, $R^\infty_A \geq 1.540$.*

Table 3 gives a comparison for several values of $\alpha = \frac{1}{r}$ between the best lower bounds and the corresponding upper bounds for various algorithms. It is interesting to note that the gap between the lower and upper bounds becomes rather small for $r \geq 2$.

Chandra [17] has examined the effect on lower bounds when randomization is allowed in the construction of on-line algorithms, i.e., when coin flips are allowed in determining where to pack items. The performance ratio for randomized algorithm A is now $E[A(L)]/OPT(L)$, where $E[A(L)]$ is the expected number of bins needed by A to pack the items in L. Chandra has shown that there are lists such that this ratio exceeds 1.536 for all randomized on-line algorithms, and so from this limited standpoint, results suggest that randomization is not a valuable tool in the design of on-line algorithms.

4 Off-Line Algorithms

An *off-line* algorithm has all items available for preprocessing, reordering, grouping, etc. before packing. We have seen that most of the classical on-line algorithms achieve their worst-case ratio when the items are packed in increasing order of size (see, e.g., FF and BF), or if small and large items are merged (see, e.g., NF). Thus, one is led to expect improved behavior by a sorting of the items in decreasing order of size. Note that $O(n \log n)$ sorting steps puts us outside the class of linear-time algorithms.

We start with results for approaches that sort the items before executing one of the on-line algorithms. We then consider linear-time heuristics. We conclude this section with approximation schemes and a discussion of the anomalous behavior that is exhibited by many bin packing algorithms, including both on-line and off-line algorithms.

4.1 Algorithms with presorting

When the sorted list is packed according to the Next-Fit rule, we obtain the *Next-Fit Decreasing* (NFD) algorithm. This heuristic was investigated by Baker and Coffman, who proved by a weighting function argument that its APR is slightly better than that of FF and BF:

Theorem 4.1 (Baker and Coffman [8]). *If* $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$ $(r \geq 1)$ *then* $R^{\infty}_{\mathrm{NFD}}(\alpha) = h_{\infty}(r)$.

Packing the sorted list according to First-Fit or Best-Fit gives the algorithms *First-Fit Decreasing* (FFD) and *Best-Fit Decreasing* (BFD), with much better asymptotic worst-case performance.

Theorem 4.2 (Johnson et al. [72]). $R^{\infty}_{\mathrm{FFD}} = R^{\infty}_{\mathrm{BFD}} = \frac{11}{9}$.

The original proof was based on the weighting function technique but subsequent proofs introduced dramatic changes; the giant case-analysis of Johnson [69] was considerably shortened by Baker [6], and recently Yue [104] presented the shortest proof known so far. In parallel, the additive constant was also improved; Johnson had proved that $\mathrm{FFD}(L) \leq \frac{11}{9}OPT(L) + 4$, but Baker reduced the constant to 3, and Yue reduced it to 1.

The behavior of BFD for general α is not known, but that of FFD has been intensively investigated. Johnson et al. [72] analyzed several cases, showing that

$$R^{\infty}_{\mathrm{FFD}}(\alpha) = \begin{cases} \frac{11}{9} & \text{if } \frac{1}{2} < \alpha \leq 1 \\ \frac{71}{60} & \text{if } \frac{8}{29} < \alpha \leq \frac{1}{2} \\ \frac{7}{6} & \text{if } \frac{1}{4} < \alpha \leq \frac{8}{29} \\ \frac{23}{20} & \text{if } \frac{1}{5} < \alpha \leq \frac{1}{4} \end{cases}$$

and conjecturing that, for any integer $m \geq 4$,

$$R^{\infty}_{\mathrm{FFD}}(\tfrac{1}{m}) = F_m := 1 + \tfrac{1}{m+2} - \tfrac{2}{m(m+1)(m+2)}$$

Twenty years later, Csirik [28] proved that the above conjecture is valid only for m even, and that, for m odd,

$$R^{\infty}_{\mathrm{FFD}}(\tfrac{1}{m}) = G_m := 1 + \tfrac{1}{m+2} - \tfrac{1}{m(m+1)(m+2)}$$

In the same year, the complete analysis of FFD for arbitrary values of $\alpha \leq \frac{1}{4}$ was published by Xu [101]. He showed that if m is even, then F_m is the correct APR for any α in $(\frac{1}{m+1}, \frac{1}{m}]$, while for m odd the interval has to be divided into two parts, with

$$R^{\infty}_{\mathrm{FFD}}(\alpha) = \begin{cases} F_m, & \text{if } \frac{1}{m+1} < \alpha \leq d_m \\ G_m, & \text{if } d_m < \alpha \leq \frac{1}{m} \end{cases}$$

where $d_m := (m+1)^2/(m^3 + 3m^2 + m + 1)$.

Recall that, after a presorting stage, all of the above algorithms belong to the Any-Fit class (see Section 3.3). Johnson showed that, after a presort in increasing order, Any-Fit algorithms do not perform well in the worst case; e.g., their APRs must be at least $h_\infty(1)$ when $\alpha = 1$. But presorting in decreasing order is much more useful.

Theorem 4.3 (Johnson [69, 70]). *Any algorithm $A \in \mathcal{AF}$ operating on a list presorted in decreasing-size order must have*

$$\begin{array}{rcccl} \frac{11}{9} & \leq & R_A^\infty(1) & \leq & \frac{5}{4} \\ \frac{1}{m+2} - \frac{2}{m(m+1)(m+2)} & \leq & R_A^\infty(\alpha) & \leq & \frac{1}{m+2}, \quad \text{where } m = \lfloor \frac{1}{\alpha} \rfloor \text{ and } \alpha < 1 \end{array}$$

For a long time, the FFD bound was the smallest proved APR. Johnson [69] made an interesting attempt to obtain a better APR. His *Most-k-Fit* (MF$_k$) algorithm takes elements from both ends of the sorted list, packing bins one at a time. At any step, after trying to place the largest unpacked element into the current bin, the algorithm attempts to fill up the remaining space in the bin using the smallest k (or fewer) as yet unpacked items. As soon as the available space becomes smaller than the smallest unpacked element, the algorithm starts packing a new bin. The algorithm has time complexity $O(n^k \log n)$, so it is practical only for small k. Johnson conjectured that $\lim_{k\to\infty} R_{\mathrm{MF}_k}^\infty = \frac{10}{9}$, but almost twenty years later the conjecture was contradicted by Friesen and Langston [40], who gave examples for which $R_{\mathrm{MF}_k}^\infty \geq \frac{5}{4}$, $k \geq 2$.

Yao [102] devised the first improvement to FFD. He presented a complicated $O(n^{10} \log n)$ algorithm, called *Refined First-Fit Decreasing* (RFFD), with worst-case ratio $R_{\mathrm{RFFD}}^\infty \leq \frac{11}{9} - 10^{-7}$. Following this result, further efforts were made to develop better off-line algorithms. Garey and Johnson [52] proposed the *Modified First-Fit Decreasing* (MFFD) algorithm. The main idea is to supplement FFD with an attempt to improve that part of the packing containing bins with items of sizes larger than $\frac{1}{2}$, by trying to pack in these bins pairs of items (to be called S items) with sizes in $(\frac{1}{6}, \frac{1}{3}]$. The non-FFD decisions of MFFD occur only during the packing of S items. At the time these items come up for packing, the bins currently containing a single item larger than $\frac{1}{2}$ are packed first, where possible, in decreasing-gap order as follows. In packing the next such bin, MFFD first checks whether there are two still-unpacked S items that can fit into the bin; if not, MFFD finishes out the remaining packing just like FFD. Otherwise, the smallest available S item is packed first in the bin; the largest remaining available S item that fits with it is packed second. The running time of MFFD is not appreciably larger than that for FFD, but Garey and Johnson proved that

Theorem 4.4 (Garey and Johnson [52]). $R_{\mathrm{MFFD}}^\infty = \frac{71}{60} = 1.18333\ldots$

Another modification of FFD was presented by Friesen and Langston [40]. Their *Best Two-Fit* (B2F) algorithm starts by filling one bin at a time, greedily; when no further element fits into the current bin, and the bin contains more than one element, an attempt is made to replace the smallest one by two unpacked elements with sizes at least $\frac{1}{6}$. When all the unpacked elements have sizes smaller than $\frac{1}{6}$, the standard FFD algorithm is applied. Friesen and Langston proved that $R^{\infty}_{\text{B2F}} = \frac{5}{4}$, which is worse than $\frac{11}{9}$. However, they further showed that a combined algorithm (CFB), which runs both B2F and FFD and takes the better packing, has an improved APR.

Theorem 4.5 (Friesen and Langston [40]). $1.16410\ldots = \frac{227}{195} \leq R^{\infty}_{\text{CFB}} \leq \frac{6}{5} = 1.2$.

4.2 Linear time, randomization, and comparisons

The off-line algorithms analyzed so far have time complexity at least $O(n \log n)$. It is also interesting to see what can be obtained in linear time – in particular, without sorting. The first such heuristic was constructed by Johnson [70]. His *Group-X Fit Grouped* (GXFG) algorithm depends on the choice of a set of breakpoints X, defined by a sequence of real numbers $0 = x_0 < x_1 < \ldots < x_p = 1$. For a given X, the algorithm partitions the items, according to their size, into at most p classes, and renumbers them in such a way that items of the same class are consecutive and classes are ordered by decreasing maximum size. The bins are also collected into p groups according to their *actual gap*, defined as the maximum x_j such that the current empty space in the bin is at least x_j. The items are packed using the Best-Fit rule with respect to the actual gaps. The algorithm can be implemented so as to require linear time, and has the following APR.

Theorem 4.6 (Johnson [70]). *For all $m \geq 1$, if X contains $\frac{1}{m+2}$, $\frac{1}{m+1}$ and $\frac{1}{m}$, then $R^{\infty}_{\text{GXFG}}(\alpha) = \frac{m+2}{m+1}$ for all $\alpha \leq 1$ such that $m = \left\lfloor \frac{1}{\alpha} \right\rfloor$.*

For $\alpha = 1$ the above theorem gives $R^{\infty}_{\text{GXFG}} = \frac{3}{2}$, a bound subsequently improved by Martel. His algorithm, H_4, uses a set $X = \{\frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$, but it does not reorder the items. It instead inserts them into heaps, and uses a linear search for the median-size item. The packing strategy makes use of an elaborate pairing technique.

Theorem 4.7 (Martel [88]). $R^{\infty}_{\text{H}_4} = \frac{4}{3}$.

Very recently, Bèkèsi and Galambos [11] applied the Martel idea to a set $X = \{\frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{3}{8}, \frac{1}{2}, \frac{2}{3}, \frac{4}{5}\}$, and improved the bound to $\frac{5}{4}$. Making experimental comparisons, they further showed that this linear-time algorithm is faster than the less complicated FFD rule even for small problem instances. They also mentioned that, by using more breakpoints, one can further improve the above result, but the resulting algorithms would be linear-time with a constant term so large that they would not be useful in practice.

To this point, if we consider the established tight bounds, the best is MFFD. However, very recently Kenyon [76] presented an algorithm that *could* be the new best-in-class, assuming randomization is allowed; her *Best-Fit Randomized* (BFR) algorithm produces a series of packings by applying the Best-Fit strategy to random permutations of the items. If we measure the efficiency by the expected number of bins, $E(A(L))$, then we have

$$\frac{227}{195} \leq \limsup_{k \to \infty} \left\{ \max \frac{E(A(L))}{OPT(L)} : OPT(L) = k \right\} \leq \frac{3}{2}$$

Algorithm	Time	$R_A^\infty(1)$	$R_A^\infty(2)$	$R_A^\infty(3)$	$R_A^\infty(4)$
NFD	$O(n)$	1.691...	1.424...	1.302...	1.234
FFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.15
BFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.15
GXFD	$O(n)$	1.5	1.333...	1.25	1.2
MFFD	$O(n \log n)$	1.183...	1.183...	1.183...	1.15
H_4	$O(n)$	1.333...	NA	NA	NA
H_7	$O(n)$	1.25	NA	NA	NA
B2F	$O(n \log n)$	1.25	NA	NA	NA
CFB	$O(n \log n)$	1.2	$\leq 1.183...$	$\leq 1.183...$	≤ 1.15

Table 4: Worst-case ratios of off-line algorithms. NA means "not analized".

Although this is worse than R_{MFFD}^∞, no example has been found so far such that the expected ratio is greater than 1.144.

Table 4 summarizes the best worst-case ratios of off-line algorithms (H_7 denotes the algorithm of Bèkèsi and Galambos [11]).

As a final note on fast off-line algorithms, we direct the reader to the work of Anderson, Mayr, and Warmuth [3] for the implementation of approximation algorithms on parallel architectures. They show that, with $n/\log n$ processors in the EREW PRAM model, a packing can be obtained in parallel in $O(\log n)$ time which has the same asymptotic $\frac{11}{9}$ bound as FFD.

4.3 Asymptotic approximation schemes

In 1980, Yao [102] raised an interesting question: Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm A must satisfy the lower bound $R_A^\infty \geq 1 + \varepsilon$? Fernandez de la Vega and Lueker [36] answered the question in the negative by constructing an asymptotic linear approximation scheme for bin packing. In their important paper, LP (linear programming) relaxations were first introduced as a technique for devising bin packing approximation algorithms. (See Section 5.2 for an integer programming formulation of the bin packing problem.) In this section, we first discuss their result and then mention some improvements proposed by Johnson and by Karmarkar and Karp. Further discussion can be found in [23, 63].

The main idea in [36] is the following. Given an ε, $0 < \varepsilon < 1$, define ε_1 so that $\frac{\varepsilon}{2} < \varepsilon_1 \leq \frac{\varepsilon}{\varepsilon+1}$. Instead of packing the given list L, we pack a concatenation of three lists $L_1 L_2 L_3$ determined as follows.

- L_1 contains all the elements of L with sizes smaller than ε_1.
- L_2 is a list of $(m-1)h$ dummy elements with "rounded" sizes, corresponding to the $(m-1)h$ smallest elements of $L \setminus L_1$, where $m = \left\lceil \frac{4}{\varepsilon^2} \right\rceil$ and $h = \left\lfloor \frac{|L \setminus L_1|}{m} \right\rfloor$. The elements of L_2 have only $m-1$ different sizes, and, for each size s, the list has h elements; the sizes of the corresponding h elements in $L \setminus L_1$ are no greater than s.

- L_3 contains the remaining elements of $L \setminus L_1$, i.e., those not having a corresponding rounded element in L_2. We have $|L_3| \leq 2h - 1$.

By construction, for each group of h elements of L_2 having the same size s, there exists a distinct group of h elements of $L \setminus L_1$ having sizes at least s. It follows that $OPT(L_2) \leq OPT(L \setminus L_1)$.

It is first proved that for a given ε, one can construct, in linear time via an LP relaxation, a packing for the elements of L_2 that requires no more than $OPT(L \setminus L_1) + \frac{4}{\varepsilon}$ bins. The next step is to pack each element of L_3 into a separate bin, so the number of open bins at this point is bounded by

$$OPT(L \setminus L_1) + \frac{4}{\varepsilon} + 2h - 1 < OPT(L \setminus L_1) + \frac{4}{\varepsilon} + |L \setminus L_1|\frac{\varepsilon^2}{2} \leq OPT(L \setminus L_1)(1+\varepsilon) + \frac{4}{\varepsilon}$$

(using the fact that $OPT(L \setminus L_1) \geq \frac{\varepsilon}{2}|L \setminus L_1|$).

Finally, the items of L_1 are added to the current packing, one bin at a time, using the Next-Fit rule. If no new bin is opened in this phase, the resulting packing has the desired performance. If at least one new bin is opened, we know that all the bins, except possibly the last one, are filled to at least $(1 - \varepsilon_1)$, and hence the total number of bins is bounded by

$$\frac{OPT(L)}{1-\varepsilon_1} + 1 \leq OPT(L)(1+\varepsilon) + 1 \leq OPT(L)(1+\varepsilon) + \frac{4}{\varepsilon}$$

Combining this with an analysis of the time to pack L_2, Fernandez de la Vega and Lueker proved the following result.

Theorem 4.8 (Fernandez de la Vega and Lueker [36]). *For any $\varepsilon > 0$, there exists a bin packing algorithm A with asymptotic worst-case ratio $R_A^\infty \leq 1 + \varepsilon$, requiring time $C_\varepsilon + Cn \log \frac{1}{\varepsilon}$, where C_ε depends only on ε and C is an absolute constant.*

Although the time complexity of the above approximation scheme is polynomial in the number of elements, it is exponential in $\frac{1}{\varepsilon}$.

The first improvement was given by Johnson [71], who observed that, if ε is allowed to grow suitably slowly with $OPT(L)$, one can use the above approach to construct a polynomial-time algorithm A such that $A(L) \leq OPT(L) + o(OPT(L))$; incorporating a scheme suggested by Karmarkar and Karp, he achieved $A(L) \leq OPT(L) + O(OPT(L)^{1-\delta})$, where δ is a positive constant. Thus, as a corollary to Theorem 4.8, there exists a polynomial-time approximation algorithm for bin packing with an APR equal to 1.

Karmarkar and Karp [74] made further improvements and produced an asymptotic fully polynomial-time approximation scheme. They brought several new techniques into play. The ellipsoid method applied to an LP relaxation drove the approach; a feasible packing was determined by an extension of the approach of Fernandez de la Vega and Lueker. A function T, estimated below, describes the time complexity of the LP problem in terms of properties of the problem instance. Let $c(L)$ denote the total size of the items in L and recall that k denotes the number, perhaps infinite, of possible item sizes. The main results are several different forms of asymptotic optimality and their associated running-time bounds. Recalling that the absolute error for an algorithm A is simply $A(L) - OPT(L)$, we have the following

Theorem 4.9 (Karmarkar and Karp [74]). *For each row in the table below, there is an approximation algorithm with the given running-time and absolute-error bounds; the approximation parameters in the last two rows satisfy $\alpha \in [0,1]$ and $\epsilon > 0$.*

running-time bound	*absolute-error bound*
$O(T_n(c(L)))$	$O(\log^2 OPT(L))$
$O(T_n(k))$	$O(\log^2 k)$
$O(T_n(c(L)^{1-\alpha})$	$O(OPT(L)^{\alpha})$
$O(T_n(\varepsilon^{-2}))$	$\varepsilon OPT(L) + O(\varepsilon^{-2})$

where

$$T_n(v) = O(v^8 \log v \log^2 n + v^4 n \log v \log n)$$

The bound on $T_n(v)$ is not especially attractive, so this approach as it stands is not likely to be very practical, although the authors mention that a mixture of their technique with a column generation method [53, 54] may be very efficient in practice.

4.4 Anomalies

For a given algorithm A, one usually expects that if a list is made shorter, or its items made smaller, then the number of bins needed by A to pack the list could not increase; and if the reductions were large enough, then the number of bins required would actually decrease. This is certainly true of optimal algorithms, but as this section shows, it is not the case for many of the better on-line and off-line approximation algorithms. This anomalous behavior can explain the difficulty in analyzing algorithms; with such behavior, inductive arguments can not normally be expected to work. The following instances illustrate bin packing anomalies. Define

$$\begin{aligned} L &= (0.7,\ 0.68,\ 0.5399,\ 0.3201,\ 0.15,\ 0.14,\ 0.08,\ 0.08,\ 0.08,\ 0.08,\ 0.08,\ 0.07) \\ L' &= (0.7,\ 0.68,\ 0.5399,\ 0.32,\ 0.15,\ 0.14,\ 0.08,\ 0.08,\ 0.08,\ 0.08,\ 0.08,\ 0.07) \end{aligned}$$

and note that the two instances differ only in the fourth element, which is slightly smaller in L'. BF and BFD produce the same packing of L:

$$\begin{aligned} c(B_1) &= 0.7 + 0.15 + 0.08 + 0.07 \\ c(B_2) &= 0.68 + 0.08 + 0.08 + 0.08 + 0.08 \\ c(B_3) &= 0.5399 + 0.3201 + 0.14 \end{aligned}$$

BF and BFD also produce the same packing of L', which is larger by one bin:

$$\begin{array}{rcl} c(B_1) & = & 0.7+0.15+0.14 \\ c(B_2) & = & 0.68+0.32 \\ c(B_3) & = & 0.5399+0.08+0.08+0.08+0.08+0.08 \\ c(B_4) & = & 0.07 \end{array}$$

We say that a list $L_{(n)} = (a_1, a_2, \ldots, a_n)$ *dominates* a list $L'_{(m)} = (a'_1, a'_2, \ldots, a'_m)$ if $n \geq m$ and $s(a_i) \geq s(a'_i)$ for $i = 1, 2, \ldots, m$. An algorithm A is *monotonic* if $A(L) \geq A(L')$ whenever L dominates L'. An *anomalous* algorithm is one that is not monotonic.

Graham [58] and Johnson [69] observed that algorithms FF and FFD are anomalous. However, our current insights into the anomalous behavior of bin packing algorithms are due mainly to Murgolo. In the notation of Section 3.4 (with W meaning Worst-Fit), he proved

Theorem 4.10 Murgolo [90]: *Algorithms NF and NF_2 are monotonic but each of BF, BFD, WF, WFD, ABF_2, AWF_2, and AFF_k with $k \geq 3$ is anomalous.*

Murgolo also obtained upper and lower bounds on the behavior of anomalous algorithms. An algorithm is called *conservative* if it never opens a new bin unless the current element can not fit into an open bin. For conservative algorithms, the following bound applies.

Theorem 4.11 Murgolo [90]: *If L' dominates L and A is conservative then $A(L') \leq 2A(L)$. In addition, if $A \in \{FFD, BFD\}$ then $A(L') \leq \frac{11}{9}A(L) + 4$.*

He also proved the following complementary results.

Theorem 4.12 Murgolo [90]: *There exist arbitrarily long lists L_i and L'_i, with L'_i dominating L_i, $(i = 1, 2, 3)$, such that*

$$\begin{array}{llrcl} \textit{if } A \in \{BF, BFD\} & \textit{then} & A(L_1) & \geq & \frac{43}{42}A(L'_1), \\ \textit{if } A \in \{FF\} & \textit{then} & A(L_2) & \geq & \frac{76}{75}A(L'_2), \\ \textit{if } A \in \{WF, WFD\} & \textit{then} & A(L_3) & \geq & \frac{16}{15}A(L'_3). \end{array}$$

5 Variations and Special Lists

In this section, we discuss problems in which special types of lists, alternative objective functions, or any of various constraints on items or packings are considered. We will see that in some cases the problem becomes easier, in the sense of approximability, while in others it becomes harder. Similarly, adaptations of classical heuristics give more or less attractive results depending on the new problem.

5.1 Semi-on-line algorithms

We know that the APR of on-line algorithms cannot be less than 1.540... (see Section 3.6). For almost twenty years only the pure on-line and off-line algorithms were analyzed, and no attention was paid to algorithms lying between these two classes. In a more general setting, we can consider giving the algorithm more information about the list and/or more freedom with respect to the pure on-line case. Here, we will consider semi-on-line algorithms that can

- repack elements;
- look ahead to later elements before assigning the current one;
- assume some preordering of the elements.

We first consider the case where repacking is allowed. We have seen in Section 3.4 that no known on-line bounded-space algorithm reaches the bound $h_\infty(1)$ using finitely many open bins. We will see that this is possible with semi-on-line algorithms.

In 1985, Galambos [41] made a first step in this direction. His *Buffered Next-Fit* (BNF) algorithm uses two open bins, say B_1 and B_2. The arriving elements are initially packed into B_1, until the first element arrives for which B_1 does not have enough space. This element and those currently packed in B_1 are then reordered by decreasing size and repacked in B_1 and B_2 following the Next-Fit rule. B_1 is now closed, B_2 is renamed B_1, and a new bin B_2 is opened. Using a weighting function approach, Galambos [41] proved that $h_\infty(1) \leq R^\infty_{\mathrm{BNF}} \leq \frac{18}{10}$.

Galambos and Woeginger [45] generalized the above idea, adopting a better weighting function. Let $w(B)$ be the sum of the weights associated with the items currently packed in bin B. Their *Repacking* algorithm (REP$_3$) uses three open bins. When a new item a_i arrives, the following steps are performed: (i) a_i is packed into an empty bin; (ii) all the elements in the three open bins are repacked by the FFD strategy, with the result that either one bin becomes empty or at least one bin B has $w(B) \geq 1$; (iii) all bins B with $w(B) \geq 1$ are closed and replaced by new empty bins. In [45], it was proved that this repacking in fact helps, since $R^\infty_{\mathrm{REP}_3} = h_\infty(1)$. It is not known whether the same result can be obtained with two bins.

Gambosi, Postiglione and Talamo [47] were the first to beat the 1.540 on-line bound via repacking. They gave two algorithms. In the first one, A$_1$, the interval $(0, 1]$ is divided into 4 subintervals, and the elements are packed into bins in a Harmonic-like way. As each new item is packed, groups of small elements can be repacked so as to fill up gaps in bins that are not full enough. By using appropriate data structures, this repacking is performed in constant time. The algorithm has linear time and space complexity, and it has an APR, $R^\infty_{\mathrm{A}_1} \leq \frac{3}{2}$. In the second algorithm, A$_2$, the unit interval is divided into 6 subintervals, and, as each new item is encountered, the elements are repacked more carefully, in $O(\log n)$ time, by means of a pairing technique analogous to that introduced by Martel (see Section 4.2). The time complexity of A$_2$ is $O(n \log n)$ and its APR is $R^\infty_{\mathrm{A}_2} \leq \frac{4}{3}$.

Ivkovic and Lloyd [65] gave a further improvement achieving a $\frac{5}{4}$ worst-case ratio. Their algorithm is much more complicated than the previous ones, as it was designed for handling the dynamic packing case (see Section 5.4). They also proved a lower bound for the special class of semi-on-line algorithms that only use *atomic* repacking moves; such a move is limited to the transfer of a single item from one bin to another. Then

Theorem 5.1 (Ivkovic and Lloyd [66]). *For any semi-on-line algorithm A that makes only a bounded number of atomic repacking moves at each step, the APR satisfies $R_A^\infty \geq \frac{4}{3}$.*

Quite recently, Ivkovic and Lloyd [67] presented approximation schemes for their dynamic, semi-on-line model, applying the techniques of Fernandez de la Vega and Lueker and those of Karmarkar and Karp cited in Section 4.3.

We now consider the case in which the algorithm is allowed to look ahead, in the sense that, when an element arrives, it is not necessary to pack it immediately; one is allowed to collect further elements whose sizes can effect the packing decision. In order to avoid degeneration to an off-line algorithm, we will consider the case of *bounded* look-ahead. Grove [59] proposed a k-bounded algorithm which had, in addition, a capacity (or *warehouse*) constraint W. The algorithm can delay the packing of item a_i until it has collected all subsequent elements $a_{i+1}, \ldots, a_j$ such that $\sum_{r=i}^{j} a_r \leq W$. For any fixed k and W the $h_\infty(1)$ lower bound (see Theorem 3.12) remains valid, but Grove's *Revised Warehouse* (RW) algorithm reaches the bound if W is sufficient large. In his proof, Grove uses a weighting function argument.

Finally, we examine the rarely considered class of algorithms in which it is assumed that the input list is *presorted.* Because of the lower bound constructions, it is easy to see that, if the list is presorted by increasing size, the on-line lower bounds remain valid. Moreover, the Johnson result holds: if the list is preordered by decreasing item size then $\frac{11}{9} \leq R_A^\infty \leq \frac{5}{4}$ for any algorithm $A \in \mathcal{AF}$ (see Theorem 4.3). This begged the broader question of lower bounds: how good can an arbitrary algorithm be? A partial answer is given by the following result.

Theorem 5.2 (Csirik, Galambos and Turan [29]). *If the list is preordered by decreasing size, then $R_A^\infty \geq \frac{8}{7}$ for all algorithms A.*

This is the best lower bound known.

5.2 Item-size restrictions

Perhaps the simplest (and most practical) restriction is simply that we have a finite number k of item sizes, say $s_1, \ldots, s_k$, and thus a finite number N of feasible item configurations in a bin. This class of problems arose in the work on approximation schemes [36, 74], and was considered in its own right by Blazewicz and Ecker [12]. Let the ith configuration be denoted by $p_i = (p_{i1}, \ldots, p_{ik})$ where p_{ij} is the number of items of size s_j in p_i. By the definition of feasibility, $\sum_{j=1}^{k} p_{ij} s_j \leq 1$ for all i. Let b_i be the number of bins in a packing of a given list L that contain configuration p_i, and let n_j be the number of items of size

s_j in L. Then a solution to the bin packing problem for L is a solution to the integer programming formulation

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} b_i \\ & \sum_{i=1}^{N} p_{ij} b_i \quad \geq n_j \qquad\qquad (j = 1, \ldots, k) \\ & \qquad\quad b_i \quad \geq 0 \text{ and integer } (i = 1, \ldots, N) \end{aligned}$$

Note that, if $\frac{1}{r}$ lower bounds the item sizes, then $N = O(k^{r-1})$, so the constraint matrix above has k rows and $O(k^{r-1})$ columns. This problem can be solved in time polynomial in the number of constraints (see Lenstra [84]), which is a constant independent of the number of items. The optimal overall packing can then be constructed in linear time.

If we do not need the exact solution, then we can use the classical approach of Gilmore and Gomory [53, 54]) and compute LP relaxations (see Section 4.3); this method gives solutions that are at most k off the optimal in significantly less time.

We consider next the classical problem restricted to lists L of items whose sizes form a *divisible sequence*, i.e., the distinct sizes $s(a_1) > s(a_2) > \ldots$ taken on by the items are such that $s(a_{i+1})$ divides $s(a_i)$ for all $i \geq 1$. The number of items of each size is arbitrary. Given the bin capacity C, the pair (L, C) is *weakly divisible* if L has divisible item sizes and *strongly divisible* if in addition the largest item size $s(a_1)$ in L divides C.

This variant has practical applications in computer memory allocation, where device capacities and memory block sizes are commonly restricted to powers of 2. It is important to recognize such applications when they are encountered, because approximation algorithms for many types of NP-hard bin packing problems generate significantly better packings when items satisfy divisibility constraints. We emphasize here the cases where the restriction leads to algorithms that are asymptotically optimal. The problem was studied by Coffman, Garey and Johnson, who proved the following result for classical off-line algorithms.

Theorem 5.3 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then NFD and FFD packings are optimal.*

Indeed, the residual capacity in a bin is always either zero or at least as large as the last (smallest) item packed in the bin. The last packed item is at least as large as any remaining (unpacked) item, so either the bin is totally full or it has room for the next item. Therefore the FFD and NFD algorithms have the same behavior: they initialize a new bin only when the previous one is totally full, so the packings they produce are identical and perfect.

The optimality of FFD holds in the less restrictive case of the following theorem as well.

Theorem 5.4 (Coffman, Garey and Johnson [20]). *If (L, C) is weakly divisible, then an FFD packing is always optimal.*

For strongly divisible instances, we can also obtain optimal performance without sorting the items.

Theorem 5.5 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then an FF packing is always optimal.*

It would be interesting to examine how approximation algorithms behave with other special size sequences (e.g., Fibonacci numbers), as mentioned by Chandra, Hirschler and Wong [16] in the context of memory allocation.

5.3 Variable size bins

Taking a complementary approach to the approximation schemes of Section 4.3, Hochbaum and Shmoys [61] define an *ε-dual approximation algorithm* for bin packing, which we denote by M_ε. For a given $\epsilon > 0$, M_ε finds in polynomial time a packing of L in $OPT(L)$ bins, each of capacity $C = 1 + \epsilon$. A primary objective of M_ε is the approximation scheme for the capacity minimization problem discussed in Section 5.6, but such algorithms also find use in bin packing settings where precise bin capacities vary or are unknown.

We remark that the above dual approach can be likened to a search for near-feasible, optimal solutions rather than feasible, near-optimal solutions. The construction of M_ε exploits a reduction of the problem to bin packing with a finite number of item sizes, and hence the integer program formulation in Section 5.2. A recent, general discussion of the details has been given by Hochbaum [63], who describes a version of M_ε that requires only linear running time (with constants depending on ε).

Consider now the case where we are given different bin types, $B_1, B_2, \ldots, B_k$ with sizes $1 = s(B_1) > s(B_2) > \ldots > s(B_k)$. For each type, an unlimited number of bins is available, and the aim is to pack a list L of elements a_i, with $s(a_i) \in [0, 1]$, into a subset of bins having the smallest total size. Let $s(A, L)$ denote the total size of the bins used by algorithm A to pack the items of L. Then

$$R_A^\infty = \lim_{k \to +\infty} \sup \left\{ \frac{s(A, L)}{s(OPT, L)} : s(OPT, L) \geq k \right\}$$

The problem has practical importance in many of the classical bin packing applications, e.g., cutting-stock problems, the assignment of commercials to variable size breaks in television or radio broadcasting, memory allocation for computer operating systems, etc.

At first glance, one might expect variable size bin packing to be harder than the classical problem. However, this need not be true in general. For example, if we have bin types for all sizes between $\frac{1}{2}$ and 1, then packing large elements with sizes at least $\frac{1}{2}$ can be done without wasted space. Similarly, we can pack smaller elements together without any waste, so long as their sum is at least $\frac{1}{2}$.

For general sets of bin sizes, an on-line algorithm must select the bin size for packing the current element whenever a new bin is opened. Friesen and Langston [39] proposed an on-line algorithm based on the Next-Fit rule, which always selects the largest bin-size (*Next-Fit, using Largest possible bins*, NFL), and proved that its APR is 2. Kinnersley and Langston [78] showed that the same APR is obtained by adopting the First-Fit rule, both for an algorithm that uses the largest possible bins and for an algorithm that uses the smallest bins. Very recently, Burkard and Zhang [14] pointed out that this APR characterizes a large class of algorithms that use one of the above bin-opening rules.

Kinnersley and Langston [78] analyzed other fast on-line algorithms. They proposed a scheme based on a user-specified *fill factor*, $f \geq \frac{1}{2}$, and proved that this strategy guarantees an APR smaller than $\frac{3}{2} + \frac{f}{2} \leq 1.75$. Zhang [106] proved that, with $f = \frac{1}{2}$, the APR is $\frac{17}{10}$.

Csirik [27] investigated an algorithm based on the Harmonic-Fit strategy (*Variable Harmonic-Fit*, VH), and showed that its APR is at most $h_\infty(1) \approx 1.69103$. For special collections of bin types the algorithm may perform better; for example, Csirik proved that if there are only two types of bins, with $s(B_1) = 1$ and $s(B_2) = \frac{7}{10}$, we have $R^\infty_{\mathrm{VH}} = \frac{7}{5}$. This result is very interesting since the bound is smaller than the 1.54 on-line lower bound of van Vliet [96] for the classical problem (see Section 3.6), and implies that, for certain sets of two or more bin sizes, on-line algorithms can behave better than those restricted to a single bin size. This raised further, still unanswered questions. If there are only two different bin types, which combination of sizes produces the smallest APR? What lower bounds depending on bin sizes can be proved? What can be said about the problem with at least three bin sizes?

Csirik's VH algorithm suffers from the same "illness" as the classical HF_k algorithm (see Section 3.4); it reaches the $h_\infty(1)$ bound only for a very large number of open bins. More precisely, let $M > 1$ be a positive integer, and suppose that the algorithm can use l different bin types. Let $M_j = \lceil Ms(B_j) \rceil$ $(j = 1, \ldots, l)$, and denote by VH_M the resulting VH algorithm, which is a k-bounded space algorithm with $k = \sum_{j=1}^{l} M_j - l + 1$. If $M_l \leq 5$ then $R^\infty_{\mathrm{VH}_M} \geq \frac{17}{10}$.

Burkard and Zhang [14] investigated other bounded-space algorithms for variable-size bin packing. They distinguished three different components: the opening rule, packing rule, and closing rule. The opening rule was fixed as follows. If the current item has size $s(a_i) > \frac{1}{2}$ and there exist bin sizes smaller than 1 that can accommodate a_i, then the rule opens the smallest such bin; otherwise, it opens a bin of size 1. The packing rules analyzed were First-Fit (the F rule) and Best-Fit (the B rule). The closing rule closes a bin of size less than 1, if one exists; otherwise, it closes either the lowest indexed bin (the first-bin or F rule) or the most nearly full bin (the best-bin or B rule). With this terminology, $\mathrm{V}XY_k$ denotes the k-bounded algorithm that incorporates packing rule X and closing rule Y. Burkard and Zhang showed that, among the four resulting algorithms (VFF_k, VFB_k, VBF_k and VBB_k), VBB_k is the best, and that the following holds.

Theorem 5.6 (Burkard and Zhang [14]). *$R^\infty_{\mathrm{VBB}_k} = \frac{17}{10}$. Moreover, $R^\infty_{\mathrm{VH}_M} \geq R^\infty_{\mathrm{VBB}_k}$ if and only if $M_l < 7$, with $k \geq 6l + 1$.*

The authors proved the theorem by a weighting function technique. They also mentioned that, with a small modification in the weighting function, they could prove that $R^\infty_{\mathrm{VFB}_k} =$

$\frac{17}{10} + \frac{3}{10(k-1)}$ for any $k \geq 2$. So, one of the more interesting questions remains open: Does there exist an on-line algorithm with an APR strictly less than $h_\infty(1)$ for all collections of bin sizes?

For the off-line case, few results have been proved. Friesen and Langston [39] presented two algorithms based on the First-Fit Decreasing strategy. The first one, *FFD using Largest bins, and Repack* (FFDLR), begins by packing the presorted elements into the largest bins, then repacks the contents of each open bin into the smallest possible empty bin. The second algorithm, *FFD using Largest bins and Shift* (FFDLS), improves on the first phase of FFDLR: whenever the bin (say of type B_j) where the current item has been packed contains an item of size at least $\frac{1}{3}$, the contents of the bin are shifted, if possible, to the smallest empty bin (say of type B_h) such that $s(B_h) \geq \tilde{c} \geq \frac{3}{4}s(B_h)$, where $\tilde{c}$ denotes the total item size packed into the current bin. Friesen and Langston proved that $R^\infty_{\text{FFDLR}} = \frac{3}{2}$ and $R^\infty_{\text{FFDLS}} = \frac{4}{3}$.

Murgolo [89] proposed an efficient approximation scheme. He first gave an algorithm with a running time linear in the number of items but exponential in $\frac{1}{\varepsilon}$ and the number of bin sizes. Then, following the ideas in Karmarkar and Karp [74], he solved a linear programming formulation of the problem by the ellipsoid method, thus obtaining a fully polynomial-time approximation scheme.

Very recently, Zhang [107] considered a variant of the on-line version of the problem, in which item-size information is known in advance, but no information on bin sizes is available, except that each bin size is known to be no less than the size of the largest item. Bins arrive one at a time, and we have to decide which items to pack into the current bin. Zhang proved that the analogues of the classical NF, FF, NFD and FFD algorithms all have APR equal to 2, and left as an open question whether one can devise an algorithm with an APR better than 2.

It is not difficult to extend the strong divisibility results of the previous section to the variable-size bin case. If the pair $(L_i, s(B_i))$, where L_i is the sublist of L containing the elements with size not exceeding $s(B_i)$, is strongly divisible for all $i = 1, \ldots, k$, then the following algorithm produces an optimal packing (see Coffman, Garey and Johnson [20]). First use the FFD algorithm to pack items in bins of type B_1, until either all items are packed or the total size of the unpacked items is less than $s(B_1)$. In the latter case, if the size of the largest unpacked item exceeds $s(B_2)$, place all these items in a bin of type B_1; otherwise, repeat the process for the remaining items and bins of types $B_2, B_3, \ldots$. The case of weak divisibility has not been treated.

5.4 Dynamic bin packing

In this section we consider a generalization in which each item a_i is characterized by a triple $(b(a_i), d(a_i), s(a_i))$, where $b(a_i)$ is the start (arrival) time, $d(a_i)$ ($d(a_i) > b(a_i)$) is the departure time, and, as usual, $s(a_i)$ is the size. Item a_i remains in the packing during the time interval $[b(a_i), d(a_i))$. We assume that $b(a_i) \leq b(a_j)$ if $i < j$. The problem calls for the minimization of the maximum number, $OPT_D(L)$, of bins ever required in dynamically packing list L when repacking of the current set of items is allowed each time a new item arrives. More formally, if $A(L,t)$ denotes the number of bins used by algorithm A to pack the current set of items at time t, the objective is to minimize

$$A(L) = \max_{0 \le t \le b(a_n)} A(L,t)$$

Note that, in this case, an opened bin can later become empty, so the classical open/close terminology is no longer valid. The definition of the APR is straightforward:

$$R_A^\infty = \lim_{n\to\infty} \sup \left\{ \frac{A(L)}{OPT_D(L)} : |L| = n \right\}$$

The problem models an important aspect of multiprogramming operating systems: the dynamic memory allocation for paged or other virtual memory systems (see, e.g., Coffman [19]), or data storage problems where the bins correspond to storage units (e.g., disk cylinders or tracks), and the items correspond to records which must be stored for certain specified periods of time (see Coffman, Garey and Johnson [22]).

Research on dynamic packing is at the boundary between bin packing and dynamic storage allocation. The most significant difference between the two areas lies in the repacking assumption of dynamic bin packing; repacking is disallowed in dynamic storage allocation, so fragmentation of the occupied space can occur. Coffman, Garey and Johnson [22] studied two versions of the classical FF algorithm. The first is a direct generalization of the classical algorithm. The second is a variant (*Modified First-Fit*, MFF) in which the elements with $s(a_i) \ge \frac{1}{2}$ are handled separately, in an attempt to pair each of them with smaller elements.

Theorem 5.7 (Coffman, Garey and Johnson [22]). *If* $\frac{1}{k+1} < \max\{s(a_i)\} \le \frac{1}{k}$, *then*

$\frac{11}{4} \le R_{\mathrm{FF}}^\infty(k) \le \frac{5}{2} + \frac{3}{2}\ln\frac{\sqrt{13}-1}{2} = 2.89674\ldots$ *if* $k = 1$;

$\frac{k+1}{k} + \frac{1}{k^2} \le R_{\mathrm{FF}}^\infty(k) \le \frac{k+1}{k} + \frac{1}{k-1}\ln\frac{k^2}{k^2-k+1}$ *if* $k \ge 2$;

$2.77 \le R_{\mathrm{MFF}}^\infty(1) \le \frac{5}{2} + \ln\frac{4}{3} = 2.78768\ldots.$

For $k = 2$, we get $\frac{7}{4} \le R_{\mathrm{FF}}^\infty(2) \le 1.78768\ldots.$ Although these results are worse than their classical counterparts, relative performance is much better than one might think. This can be seen in the following lower bounds.

Theorem 5.8 (Coffman, Garey and Johnson [22]). *For any on-line dynamic bin packing algorithm* A, $R_A^\infty \ge \frac{5}{2}$ *and* $R_A(k) \ge 1 + \frac{k+2}{k(k+1)}$ *if* $k \ge 2$, *which gives* $R_A(2) \ge 1.666\ldots.$

If we restrict our attention to strongly divisible instances (see Section 5.2), then matters improve, as expected. The gap between the general lower bound and that for FF disappears. Indeed, we have

Theorem 5.9 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible and L is such that $s(a_1) > s(a_2) > \ldots > s(a_k)$ $(k \geq 2)$, then the APR of the dynamic version of FF is*

$$R^{\infty}_{\mathrm{FF}} = \prod_{i=1}^{k-1} \left(1 + \frac{s(a_i)}{C} - \frac{s(a_{i+1})}{C}\right)$$

Moreover, for any on-line algorithm A that operates only on strongly divisible instances, $R^{\infty}_{A} \geq R^{\infty}_{\mathrm{FF}}$.

By a straightforward inductive argument, it can be shown that the continued product has the upper bound

$$\lim_{k\to+\infty} \left(1 + \frac{1}{2^{k-2}}\right) \prod_{i=1}^{k-2} \left(1 + \frac{1}{2^i}\right) = 2.384\ldots$$

5.5 Bounds on the number of items per bin

The present section deals with the generalization in which the maximum number of items packed into a bin is bounded by a positive integer p. The problem has practical applications in multiprocessor scheduling with a single resource constraint, when the number p of processors is fixed, or in multitasking operating systems where the number of tasks is bounded. In the context of these applications, Krause, Shen and Schwetman [79] modified classical algorithms so as to deal with the restricted number of items per bin. Their on-line algorithm, pFF, is FF with an additional check on the number of items in open bins. They proved that, if $p \geq 3$, then

$$\frac{27}{10} - \frac{37}{10p} \leq R^{\infty}_{\mathrm{pFF}} \leq \frac{27}{10} - \frac{24}{10p}.$$

As p tends to ∞, the bound is much worse than the corresponding APR of the unrestricted problem (2.7 versus 1.7). Whether this upper bound can be improved remains an open question.

Their second (off-line) algorithm (*Largest Memory First*, LMF) is an adaptation of FFD. They proved that $R^{\infty}_{\mathrm{LMF}} = 2 - \frac{2}{p}$ if $p \geq 2$. Here too the gap between the unrestricted and the restricted case is large (2 versus $\frac{11}{9}$). The authors' search for an algorithm with better worst-case behavior resulted in the *Iterated Worst-Fit Decreasing* (IWFD) algorithm. IWFD starts by sorting the items according to nonincreasing size, and by opening q empty bins, where $q := \lceil \max(\frac{n}{p}, \sum_{j=1}^{n} s(a_j)) \rceil$ is an obvious lower bound on $OPT(L)$. Then (i) IWFD tries to place the items into these q bins using the Worst-Fit rule (an item is placed into the open bin whose current number of items is less than p and whose current content is minimum, breaking ties by highest bin index); (ii) whenever the current item a_i does not fit in any of the q bins, q is increased by 1, all items are removed from the bins and the processing is restarted from (i). If one implements this approach using a binary search on q, the time complexity of IWFD is $O(n \log^2 n)$. Krause, Shen and Schwetman proved that the algorithm behaves very well in certain special cases:

- if $p = 2$ then $IWFD(L) = OPT(L)$ for any list L;
- if in the final schedule no capacity constraint is operative, i.e., no open bin has residual capacity smaller than the size of the smallest item, then $IWFD(L) = OPT(L)$;
- if in the final schedule no cardinality constraint is operative, i.e., no open bin containing p items has residual capacity at least equal to the size of the smallest item then

Theorem 5.10 (Galambos and Woeginger [44]). *There exists a sequence of nonnegative numbers $\varepsilon_1, \epsilon_2, \ldots$, with $\epsilon_m > 0$, $m \geq 4$ and $\epsilon_m \to 0$ as $m \to \infty$, such that $R(m) \leq 2 - \frac{1}{m} - \varepsilon_m$.*

This result encouraged further research; important milestones on the way to our current position on the problem are as follows. Bartal, Karloff and Rabani [9] presented an algorithm with a worst-case ratio $2 - \frac{1}{70} = 1.98571\ldots$ for all $m \geq 70$. Earlier, Faigle, Kern and Turan [35] proved lower bounds for different values of m. They pointed out that for $m \leq 3$ the LS algorithm is optimal among on-line algorithms, and they proved a $1 + \frac{1}{\sqrt{2}} = 1.70710\ldots$ lower bound for $m \geq 4$. Then these bounds were improved, as shown below.

Theorem 5.11 (Chen, van Vliet and Woeginger [18]). *For all $m = 80 + 8k$ with k a nonnegative integer, $R(m) \geq 1.83193$.*

Theorem 5.12 (Bartal, Karloff and Rabani [10]). *For all $m \geq 3454$, $R(m) \geq 1.8370$.*

Until recently, the following result gave the best upper bound for $R(m)$, and was a generalization of the methods in [10].

Theorem 5.13 (Karger, Phillips and Torng [73]). *$R(m) \leq 1.945$ for all m.*

The tightest results currently known are those of Albers [2] who recently proved the next two bounds.

Theorem 5.14 (Albers [2]). *$R(m) \leq 1.923$ for all m.*

Just as previous authors, Albers recognized that, during the packing process, a good algorithm must try to avoid packings in which the content of each of the bins is about the same, for in such cases many small items followed by a large item can create a poor worst-case ratio. Albers' new algorithm attempts to maintain throughout the packing process $\lfloor \frac{m}{2} \rfloor$ bins with small total content, and $\lceil \frac{m}{2} \rceil$ bins with large total content. The precise aim is always to have a total content in the small-content bins that is at most γ times the total content in the large-content bins. With an optimal choice of γ, one obtains a worst-case ratio of at most 1.923.

For her new lower bound, Albers proved

Theorem 5.15 (Albers[2]). *$R(m) \geq 1.852$ for all $m \geq 80$.*

Attempts to further reduce the general gap of about .07 continue. For the special case $m = 4$, it is proved in [18] that $1.73101 \leq R(4) \leq \frac{52}{30} = 1.7333\ldots$, but the exact value of $R(4)$ remains an open problem. Another enticing open problem is: Does $R(m) < R(m+1)$ hold for any m?

Let us turn now to the off-line case. By preordering the elements according to nonincreasing size and applying the LS algorithm, we obtain the so-called *Longest Processing Time* (LPT) algorithm for $P||C_{\max}$. Graham [57] proved that $R_{\text{LPT}} = \frac{4}{3} - \frac{1}{3m}$. The *Multifit* algorithm by Coffman, Garey and Johnson [21] adopts a different strategy, leading to better worst-case behavior. The algorithm determines, by binary search over an interval with crude but easily established lower and upper bounds, the smallest capacity C such that the FFD algorithm can pack all the elements into m bins. Coffman, Garey and Johnson proved that, if k binary search iterations are performed, then the algorithm, denoted by MF_k, requires $O(n \log n + kn \log m)$ time and has an absolute worst-case ratio satisfying

$$R_{\mathrm{MF}_k} \leq 1.22 + 2^{-k}$$

Friesen [38] improved the bound uniform in k to 1.2, but Yue [103] later settled the question by proving that this bound is $\frac{13}{11}$. Friesen and Langston [39] developed a different version of the Multifit algorithm, proving that its worst-case ratio is bounded by $\frac{72}{61} + 2^{-k}$.

A different off-line approach was proposed by Graham [57]. His algorithm Z_k optimally packs the $\min\{k, n\}$ largest elements and completes the solution by packing the remaining elements, if any, according to the LS algorithm. Graham proved that

$$R_{\mathrm{Z}_k} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{k}{m} \right\rfloor}$$

and that the bound is tight when m divides k. Algorithm Z_k implicitly defines an approximation scheme. By selecting $k = k_\varepsilon = \lceil \frac{m(1-\varepsilon)-1}{\varepsilon} \rceil$, we obtain $R_{\mathrm{Z}_{k_\varepsilon}} \leq 1 + \varepsilon$. The running time is $O(n \log n + m^{m(1-\varepsilon)/\varepsilon})$ and so the method is unlikely to be practical. The result was improved by Sahni [93], in the sense that the functional dependence on m and ε was reduced substantially. His algorithm A_ε has running time $O(n(\frac{n^2}{\varepsilon})^{m-1})$ and satisfies $R_{\mathrm{A}_\varepsilon} \leq 1 + \varepsilon$; hence it is a fully polynomial-time approximation scheme for any fixed m.

For m even moderately large, the approach of Hochbaum and Shmoys [61] may offer major improvements. They developed an ε-approximate algorithm that removed the running-time dependence on m and reduced the dependence on n. The dependence on $\frac{1}{\varepsilon}$ is exponential, which is to be expected, since a fully polynomial-time approximation scheme would imply $P = NP$. Their algorithm (like MF_k, for example) is based on the binary search of an interval, but it uses the ε-dual approximation algorithm of Section 5.3 at each iteration. Hochbaum and Shmoys show that, after k steps of the binary search, the bin capacity is at most $(1+\varepsilon)(1+2^{-k})$ times optimal. From this fact and the properties of M_ε given in Section 5.3, one obtains a linear-time approximation scheme for the capacity minimization problem. (See also Hochbaum's [63] more extensive discussion of this interesting technique.)

5.7 Maximizing the number of items packed

In this section, we consider another variant in which the number of bins is fixed; the objective now is to pack a maximum cardinality subset of a given list $L_{(n)}$. The problem arises in computing applications, when we want to maximize the number of records stored in one or more levels of a memory hierarchy, or to maximize the number of tasks performed on multiple processors within a given time interval.

We extend the classical bin packing notation in the obvious way and define

$$\widehat{R}_A(m,n) = \min\left\{\frac{A(L_{(n)},m)}{OPT(L_{(n)},m)}\right\}$$

so that the APR is defined by

$$\widehat{R}_A^\infty(m) = \lim_{n\to\infty} \inf \widehat{R}_A(m,n)$$

and has values less than or equal to 1.

The problem was first studied by Coffman, Leung and Ting [25], who adapted the *First-Fit Increasing* (FFI) heuristic. The items are preordered according to nondecreasing size, the m bins are opened, and the current item is packed into the lowest indexed bin into which it fits, stopping the process as soon as an item is encountered that does not fit into any bin. It is proved in [25] that $\widehat{R}_{\text{FFI}}^\infty(m) = \frac{3}{4}$ for all m.

The *Iterated First-Fit Decreasing* (IFFD) algorithm was investigated by Coffman and Leung [24]. The algorithm sorts the items by nonincreasing size, and iteratively tries to pack all the items into the m bins following the First-Fit rule. Whenever the current item cannot be packed, the process is stopped, the largest item is removed from the list, and a new FFD iteration is performed on the shortened list. The search terminates as soon as FFD is able to pack all the items of the current list. Coffman and Leung showed that IFFD performs at least as well as FFI on every list, and that $\frac{6}{7} \le \widehat{R}_{\text{IFFD}}^\infty(m) \le \frac{7}{8}$. The time complexity of IFFD is $O(n\log n + mn\log m)$, but a tight APR is not known.

Let us now consider lower bounds on the APR. If an algorithm A packs items $a_1,\dots,a_t$ from a list $L = (a_1,\dots,a_n)$ so that $s(a_j) > 1 - s(B_i)$ for any i and for any $j > t$ (i.e., no unpacked item fits into any bin), then it is said to be a *prefix algorithm.* Note that both FFI and and IFFD are prefix algorithms).

Theorem 5.16 (Coffman, Leung and Ting [25]). *Let A be a prefix algorithm and let $k = min_{1\le i\le m}\{|B_i|\}$ be the least number of items packed into any bin. Then*

$$\widehat{R}_A^\infty(m) \ge \frac{mk}{mk+m-1}$$

Moreover, the bound is achievable for all $m \ge 1$ and $k \ge 1$.

The case of divisible item sizes was investigated by Coffman, Garey and Johnson [20]. They proved that both FFI and IFFD produce optimal packings if (L,C) is strongly divisible. IFFD remains optimal even if (L,C) is weakly divisible, but FFI does not.

The case of variable sized bins was analyzed by Langston [80], who proved that, if the bins are rearranged by nonincreasing size, then $\widehat{R}_{\text{FFI}}^\infty(m) = \frac{1}{2}$ and $\frac{2}{3} \le \widehat{R}_{\text{IFFD}}^\infty(m) \le \frac{8}{11}$ for all m.

5.8 Maximizing the number of full bins

In this variant of the problem the objective is to pack a list of items into a maximum number of bins subject to the constraint that the content of each bin be no less than a given threshold T. Each such bin is said to be full. Potential practical applications are (i) the packing of canned goods so that each can contains at least its advertised net weight, and (ii) the stimulation of economic activity during a recession by allocating tasks to a maximum number of factories, all working at or beyond the minimal feasible level.

This problem is yet another dual of bin packing. A comprehensive treatment can be found in Assmann's Ph.D. thesis [4]; Assmann's collaboration with Johnson, Kleitman and Leung [5] established the main results. They first analyzed an on-line algorithm, a dual version of NF (*Dual Next-Fit*, DNF): pack the elements into the current bin B_j until $s(B_j) \geq T$, then open a new empty bin B_{j+1} as the current bin. If the current bin is not full when all items have been packed, then merge its contents with those of other full bins. All on-line algorithms are allowed this last repacking step to ensure that all bins are full. It is proved in [5] that $\widetilde{R}^{\infty}_{\mathrm{DNF}} = \frac{1}{2}$, where

$$\widetilde{R}^{\infty}_{A} = \lim_{m\to\infty} \inf \left(\min \left\{ \frac{A(L)}{OPT(L)} : OPT(L) = m \right\} \right)$$

Assmann et al. [5] also studied a parametrized dual of FF, called *Dual First-Fit* (DFF$[r]$). Given a parameter r ($1 < r < 2$), the current item a_i is packed into the first bin B_j for which $c(B_j) + s(a_i) \leq rT$. If there are at least two nonempty and unfilled bins (i.e., bins B_j with $0 < c(B_j) < T$) when all items have been packed, an item is removed from the rightmost such bin and added to the leftmost, thus filling it. Finally, if a single nonempty and unfilled bin remains, then its contents are merged with those of previously packed bins. It was shown that $\widetilde{R}^{\infty}_{\mathrm{DFF}[r]} = \frac{1}{2}$, although a better bound might have been expected. But some years later Csirik and Totik proved that DNF is optimal among the on-line algorithms.

Theorem 5.17 (Assmann et al. [5] and Csirik and Totik [32]). *We have that*

$$\widetilde{R}^{\infty}_{\mathrm{DFF}[r]} = \frac{1}{2}$$

and that there is no on-line dual bin packing algorithm A for which $\widetilde{R}^{\infty}_{A} > \frac{1}{2}$

Turning now to off-line algorithms, we mention first the observation of Assmann et al. [5] that presorting does not help the adaptations of algorithms Next-Fit Decreasing and Next-Fit Increasing, for which $\widetilde{R}^{\infty}_{\mathrm{NFD}} = \widetilde{R}^{\infty}_{\mathrm{NFI}} = \frac{1}{2}$. On the other hand, the off-line version DFFD$[r]$ of DFF$[r]$, obtained by presorting the items according to nonincreasing size, has better performance.

Theorem 5.18 (Assmann et al. [5]). *$\widetilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \frac{2}{3}$ if $\frac{4}{3} < r < \frac{3}{2}$ and* $\lim_{r\to 1} \widetilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \lim_{r\to 2} \widetilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \frac{1}{2}$

The *Iterated Lowest-Fit Decreasing* (ILFD) algorithm was also investigated in [5]. It is analogous to algorithm IFFD described in Section 5.7. The items are preordered by nonincreasing size. At each iteration, a prefixed number m of bins is considered, and a Lowest-Fit packing is obtained by iteratively assigning the current item to the bin with minimum contents. Binary search on m determines the maximum value for which all m bins are filled. Since n is an obvious upper bound on the optimal solution value, it is easily seen that the algorithm has $O(n \log^2 n)$ time complexity. Moreover,

Theorem 5.19 (Assmann et al. [5]). $\widetilde{R}^{\infty}_{\mathrm{ILFD}} = \frac{3}{4}$.

It is not difficult to see that DNF does not produce optimal packings even when (L, C) is strongly divisible. However, we do have the following optimality results.

Theorem 5.20 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then the dual version of NFD (DNFD) produces an optimal packing. For weakly divisible lists DNFD is no longer optimal, but ILFD is.*

Finally, we observe that obtaining an approximation scheme for the dual bin packing problem is still a research challenge. The approach used by Fernandez de la Vega and Lueker [36] and Karmarkar and Karp [74] (see Section 4.3), which eliminates the effect of small items on worst-case behavior, does not appear applicable, as in this dual problem, small items can play an important role in filling small gaps.

5.9 Further directions

5.9.1 Partial orders

In this generalization, precedence constraints among the elements are given, where precedence refers to the relative ordering of bins. Let $\prec$ denote the partial order giving the precedence constraints. Then $a_i \prec a_j$ means that, if a_i and a_j are packed in B_r and B_s, respectively, then $r \leq s$. Call the model *strict* if $r < s$ replaces $r \leq s$ is this definition.

Two practical applications have been considered in the literature. The first one is the assembly line balancing problem, in which the assembly line consists of identical workstations (the bins) where the products stop for a period of time equal to the bin capacity. The item sizes are the durations of tasks to be performed, and a partial order is imposed: $a_i \prec a_j$ means that the workstation to which a_i is assigned cannot be downstream of the one to which a_j is assigned. The second application arises in multiprocessing scheduling; here each item corresponds to a unit-duration process having a memory (or other resource) requirement equal to the item size. The bin capacity measures the total memory availability. In the given partial order, $a_i \prec a_j$ imposes the requirement that a_i must be executed before a_j finishes. The objective is then to find a feasible schedule that finishes the set of processes in minimum time (number of bins).

The first problem was studied by Wee and Magazine [99] and the second by Garey et al. [48]. In both cases the *Ordered First-Fit Decreasing* (OFFD) algorithm was applied. An item is called *available* if all its immediate predecessors have already been packed. At each stage, the set of currently available items is sorted according to nonincreasing size, and each item is packed into the lowest indexed bin where it fits and no precedence constraint is violated. Note that, if no partial order is given, this algorithm produces the same packing as FFD. In general, however, its worst-case behavior is considerably worse. The APR is $R^{\infty}_{\mathrm{OFFD}} = 2$, except in the strict model, where $R^{\infty}_{\mathrm{OFFD}} = \frac{27}{10}$.

5.9.2 Clustered Items

In this generalization, a function $d_{ij} = f(a_i, a_j)$ is given, measuring the 'distance' between items a_i and a_j). A distance constraint D is also given; two items a_i and a_j may be packed into the same bin only if $d_{ij} \leq D$. The problem has several obvious practical applications in contexts where geographical location constraints are present. Chandra, Hirschler and Wong [16] studied different cases, their main result being for the case where the items in a bin must all reside within the same unit square. They proposed a geometric algorithm A a component sets (items) in the magazine can be in production. The problem is to plan the overall production process so as to minimize the number of set-ups.

S. Khanna [77] mentions, in connection with studies of multimedia communications, that graph packing is an interesting special case. Items are edges (pairs of vertices) in a given graph G. An edge is packed in a bin if both of the vertices to which it is incident are in the bin, and so the problem is to pack the edges of G into as few bins as possible subject to the constraint that there can be at most C vertices in any bin. The approximability of this problem has yet to be studied.

References

[1] M. Adler, P. B. Gibbons, and Y. Matias. Scheduling space sharing for internet advertising. Technical report, Bell Labs, Lucent Technologies, Murray Hill, NJ 07974, 1997.

[2] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th Annual ACM Symp. Theory of Comput.*, pages 130–139, 1997.

[3] R. J. Anderson, E. W. Mayr, and M. K. Warmuth. Parallel approximation algorithms for bin packing. *Inf. and Comput.*, 82:262–277, 1989.

[4] S. F. Assmann. *Problems in Discrete Applied Mathematics.* PhD thesis, Mathematics Department MIT, Cambridge, MA, 1983.

[5] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J. Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5:502–525, 1984.

[6] B. S. Baker. A new proof for the first-fit decreasing bin-packing algorithm. *J. Algorithms*, 6:49–70, 1985.

[7] B. S. Baker, D. J. Brown, and H. P. Katseff. A 5/4 algorithm for two-dimensional packing. *J. Algorithms*, 2:348–368, 1981.

[8] B. S. Baker and E. G. Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM J. Algebraic Discr. Meth.*, 2(2):147–152, 1981.

[9] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th Annual ACM Symp. Theory of Comput.*, pages 51–58, Victoria, Canada, 1992.

[10] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Inform. Process. Lett.*, 50:113–116, 1994.

[11] J. Bèkèsi and G. Galambos. A 5/4 linear time bin packing algorithm. Technical Report OR-97-2, Teachers Trainer College, Szeged, Hungary, 1997.

[12] J. Blazewicz and K. Ecker. A linear time algorithm for restricted bin packing and scheduling problems. *Oper. Res. Lett.*, 2:80–83, 1983.

[13] D. J. Brown. A lower bound for on-line one-dimensional bin-packing algorithms. Technical Report R-864, University of Illinois, Coordinated sc. lab., Urbana, 1979.

[14] R. E. Burkard and G. Zhang. Bounded space on-line variable-sized bin packing. *Acta Cybern.*, 13:63–76, 1997.

[15] L. M. A. Chan, D. Simchi-Levi, and J. Bramel. Worst-case analyses, linear programming, and the bin-packing problem. Unpublished manuscript, 1994.

[16] A. K. Chandra, D. S. Hirschler, and C. K. Wong. Bin packing with geometric constraints in computer network design. *Oper. Res.*, 26:760–772, 1978.

[17] B. Chandra. Does randomization help in on-line bin packing? *Inform. Process. Lett.*, 43:15–19, 1992.

[18] B. Chen, A. van Vliet, and G.J. Woeginger. New lower and upper bounds for on-line scheduling. *Oper. Res. Lett.*, 16:221–230, 1994.

[19] E. G. Coffman, Jr. An introduction to combinatorial models of dynamic storage allocation. *SIAM Rev.*, 25(3):311–325, 1983.

[20] E. G. Coffman, Jr., M. Garey, and D. S. Johnson. Bin packing with divisible item sizes. *J. Complexity*, 3:405–428, 1987.

[21] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17, 1978.

[22] 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.

[23] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publ. Company, 1997.

[24] E. G. Coffman, Jr. and J. Y.-T. Leung. Combinatorial analysis of an efficient algorithm for processor and storage allocation. *SIAM J. Comput.*, 8(2):202–217, 1979.

[25] E. G. Coffman, Jr., J. Y.-T. Leung, and D. W. Ting. Bin packing: Maximizing the number of pieces packed. *Acta Inform.*, 9:263–271, 1978.

[26] E. G. Coffman, Jr. and G. S. Lueker. *Probabilistic analysis of packing and partitioning algorithms*. John Wiley & Sons, New York, 1991.

[27] J. Csirik. An on-line algorithm for variable-sized bin packing. *Acta Inform.*, 26:697–709, 1989.

[28] J. Csirik. The parametric behaviour of the first fit decreasing bin-packing algorithm. *J. Algorithms*, 15:1–28, 1993.

[29] J. Csirik, G. Galambos, and G. Turan. Some results on bin-packing. In *Proc. EURO VI Conf.*, page 52, Vienna, Austria, 1983.

[30] J. Csirik and B. Imreh. On the worst-case performance of the next-k-fit bin-packing heuristic. *Acta Cybern.*, 9: f r fi I, 9u I n f f fi I, Austn7

[43] G. Galambos and J. B. G. Frenk. A simple proof of Liang's lower bound for on-line bin packing and the extension to the parametric case. *Discr. Appl. Math.*, 41:173–178, 1993.

[44] G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM J. Comput.*, 22(2):345–355, 1993.

[45] G. Galambos and G. J. Woeginger. Repacking helps in bounded space on-line bin-packing. *Computing*, 49:329–338, 1993.

[46] G. Galambos and G. J. Woeginger. On-line bin packing – a restricted survey. *Z. Oper. Res.*, 42:25–45, 1995.

[47] G. Gambosi, A. Postiglione, and M. Talamo. New algorithms for on-line bin packing. In R. Petreschi, G. Ausiello, and D. P. Bovet, editors, *Algorithms and Complexity*, pages 44–59. World Scientific, Singapore, 1990.

[48] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *J. Combin. Theory, Ser. A*, 21:257–298, 1976.

[49] M. R. Garey, R. L. Graham, and J. D. Ullmann. Worst-case analysis of memory allocation algorithms. In *Proc. 4th Annual ACM Symp. Theory of Comput.*, pages 143–150, New York, 1972.

[50] M. R. Garey and D. S. Johnson. *Computers and intractability (A Guide to the theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[51] M. R. Garey and D. S. Johnson. Approximation algorithm for bin-packing problems: a survey. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithm in Combinatorial Optimization*, pages 147–172. Springer Verlag, New York, 1981.

[52] M. R. Garey and D. S. Johnson. A 71/60 theorem for bin packing. *J. Complexity*, 1:65–106, 1985.

[53] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Oper. Res.*, 9:849–859, 1961.

[54] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem – (Part II). *Oper. Res.*, 11:863–888, 1963.

[55] S. W. Golomb. On certain nonlinear recurring sequences. *American Math. Monthly*, 70:403–405, 1963.

[56] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Syst. Tech. J.*, 45(45):1563–1581, 1966.

[57] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):263–269, 1969.

[58] R. L. Graham. Bounds on multiprocessing anomalies and related packing algorithms. In *Proc. 1972 Spring Joint Computer Conf.*, pages 205–217, Montvale NJ, 1972. AFIPS Press.

[59] E. F. Grove. Onli

[72] 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. Comput.*, 3:256–278, 1974.

[73] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20:400–430, 1996.

[74] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. 23rd Annual IEEE Symp. Found. Comput. Sci.*, pages 312–320, 1982.

[75] H. Kellerer and U. Pferschy. An on-line alorithm for cardinality constrained bin packing problem. Technical report, Universitaet Graz und TU Graz, 1997.

[76] C. Kenyon. Best-fit bin-packing with random order. In *Proc. 7th Annual ACM-SIAM Symp. Discr. Algorithms*, pages 359–364, Philadelphia, 1996.

[77] S. Khanna. Private communication.

[78] N. G. Kinnersley and M. A. Langston. On-line variable sized bin packing. *Discr. Appl. Math.*, 22:143–148, 1988.

[79] K. L. Krause, Y. Y. Shen, and H. D. Schwetman. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *J. ACM*, 22(4):522–550, 1975.

[80] M. A. Langston. Improved 0/1 interchanged scheduling. *BIT*, 22:282–290, 1982.

[81] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Logistics of production and inventory*, volume 4 of *Handbooks in operations research and management science*, pages 445–522. North-Holland, Amsterdam, 1993.

[82] C. C. Lee and D. T. Lee. A new algorithm for on-line bin-packing. Technical Report 83-03-FC-02, Department of Electrical Engineering and computer Science Northwestern University, Evanston, IL, 1983.

[83] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985.

[84] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[85] F. M. Liang. A lower bound for on-line bin packing. *Inform. Process. Lett.*, 10(2):76–79, 1980.

[86] W. Mao. Best-k-fit bin packing. *Computing*, 50:265–270, 1993.

[87] W. Mao. Tight worst-case performance bounds for next-k-fit bin packing. *SIAM J. Comput.*, 22(1):46–56, 1993.

[88] C. U. Martel. A linear time bin-packing algorithm. *Oper. Res. Lett.*, 4(4):189–192, 1985.

[89] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput.*, 16(1):149–161, 1987.

[90] F. D. Murgolo. Anomalous behaviour in bin packing algorithms. *Discr. Appl. Math.*, 21:229–243, 1988.

[91] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. On-line bin packing in linear time. *J. Algorithms*, 10:305–326, 1989.

[92] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discr. Appl. Math.*, 34:203–227, 1991.

[93] S. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23:116–127, 1976.

[94] H. E. Salzer. The approximation of number as sums of reciprocals. *American Math. Monthly*, 54:135–142, 1947.

[95] D. Simchi-Levi. New worst-case results for the bin packing problem. *Naval Res. Log. Quart.*, 41:579–585, 1994.

[96] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.

[97] A. van Vliet. *Lower and Upper Bounds for On-line Bin Packing and Scheduling Heuristic.* PhD thesis, Erasmus University, Rotterdam, 1995.

[98] A. van Vliet. On the asymptotic worst case behavoir of harmonic fit. *J. Algorithms*, 20:113–136, 1996.

[99] T. S. Wee and M. J. Magazine. Assembly line balancing as generalized bin packing. *Oper. Res. Lett.*, 1(2):56–58, 1982.

[100] G. J. Woeginger. Improved space for bounded-space on-line bin-packing. *SIAM J. Discr. Math.*, 6:575–581, 1993.

[101] K. Xu. *A bin-packing problem with Item Sizes in the Interval* $(0, \alpha]$ *for* $\alpha \leq \frac{1}{2}$. PhD thesis, Chinese Academy of Sciences, Institute of Applied Mathematics, Beijing, China, 1993.

[102] A. C.-C. Yao. New algorithms for bin packing. *J. ACM*, 27(2):207–227, 1980.

[103] M. Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Ann. Oper. Res.*, 24:233–259, 1991.

[104] M. Yue. A simple proof of the inequality $FFD(L) \leq \frac{11}{9}OPT(L) + 1 \forall L$ for the FFD bin packing algorithm. *Acta Math. App. Sinica*, 7(4):321–331, 1991.

[105] G. Zhang. Tight worst-case performance bound for AFB_k. Technical Report 15, Inst. of Applied Mathematics. Academia Sinica, Beijng, China, 1994.

[106] G. Zhang. Worst-case analysis of the FFH algorithm for on-line variable-sized bin paking. *Computing*, 56:165–172, 1996.

[107] G. Zhang. A new version of on-line variable-sized bin packing. *Discr. Appl. Math.*, 72:193–197, 1997.