Bin Packing Approximation Algorithms: Survey and Classification

Edward G. Coffman, János Csirik, Gábor Galambos, Silvano Martello and Daniele Vigo

Contents

E.G. Coffman
Department of Computer Science, Columbia University, New York, NY, USA
e-mail: coffman@cs.columbia.edu; egc@ee.columbia.edu

J. Csirik
Department of Computer Algorithms and Artificial Intelligence, University of Szeged, Szeged, Hungary
e-mail: csirik@inf.u-szeged.hu

G. Galambos
Faculty of Education, Department of Computer Science, University of Szeged, Szeged, Hungary
e-mail: galambos@jgypk.u-szeged.hu

S. Martello (✉) • D. Vigo
DEI "Guglielmo Marconi", University of Bologna, Bologna, Italy
e-mail: silvano.martello@unibo.it; daniele.vigo@unibo.it

P.M. Pardalos et al. (eds.), *Handbook of Combinatorial Optimization*,
DOI 10.1007/978-1-4419-7997-1_35, © Springer Science+Business Media New York 2013

Abstract

The survey presents an overview of approximation algorithms for the classical bin packing problem and reviews the more important results on performance guarantees. Both on-line and off-line algorithms are analyzed. The investigation is extended to variants of the problem through an extensive review of dual versions, variations on bin sizes and item packing, as well as those produced by additional constraints. The bin packing papers are classified according to a novel scheme that allows one to create a compact synthesis of the topic, the main results, and the corresponding algorithms.

1 Introduction

In the classical version of the bin packing problem, one is given an infinite supply of bins with capacity C and a list L of n items with sizes no larger than C: the problem is to pack the items into a minimum number of bins so that the sum of the sizes 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. Bin packing rather than partitioning terminology will be used, as it eases considerably the problem of describing and analyzing algorithms.

The mathematical foundations of bin packing were first studied at Bell Laboratories by M.R. Garey and R.L. Graham in the early 1970s. They were soon joined by J.D. Ullman; then at Princeton, these three published the first [103] of many papers to appear in the conferences of the computer science theory community over the next 40 years. The second such paper appeared within months; in that paper, D.S. Johnson [125] extended the results in [103] and studied general classes of approximation algorithms. In collaboration with A. Demers at Princeton, researchers Johnson, Garey, Graham, and Ullman published the first definitive analysis of bin packing approximation algorithms [129]. 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 the abbreviated proofs in [129].

The pioneering work in [129] opened an extremely rich research area; it soon turned out that this simple model could be used for a wide 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 $\mathcal{NP}$-hard (see, e.g., Garey and Johnson [100]); 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 both the combinatorial and average-case analysis of approximation algorithms (see, e.g., the volume edited by Hochbaum [115]).

Starting with the seminal papers mentioned above, most of the early research focused on combinatorial analysis of algorithms leading to bounds on worst-case behavior, also known as performance guarantees. 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 items 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 Sect. 2). 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 a scan of 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. For example, the items to be packed may 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.

Since the early 1980s, progressively more attention has been devoted to the probabilistic analysis of packing algorithms. A book by Coffman and Lueker [46] covers the methodology in some detail (see also the book by Hofri [119, Chap. 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 [101]

in 1981 and by Coffman et al. [50] in 1984. The next such survey was written some 10 years later by Galambos and Woeginger [96] who gave an overview restricted to on-line algorithms. New surveys appeared at the end of the 1990s. Csirik and Woeginger [62] concentrated on on-line algorithms, while Coffman et al. [52] extended the coverage to include off-line algorithms, and Coffman et al. [53] considered both classes of algorithms in an earlier version of the present survey. Worst-case and average-case analysis are surveyed in [62] and [52]. More recently, classical bin packing and its variants were covered in Coffman and Csirik [43] and in Coffman et al. [54, 55].

This survey gives a broad summary of results in the one-dimensional bin packing arena, concentrating on combinatorial analysis, but in contrast to other surveys, greater space is devoted 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. The survey does not attempt to give a self-contained work, for example, it does not present all algorithms in detail nor does it give formal proofs, but it refers to certain proof techniques which have been used frequently to establish the most important results. Also, in the coverage of variants, the restriction to partitioning problems in one dimension is maintained. Packing in higher dimensions, for example, strip packing and two-dimensional bin packing, is itself a big subject, one deserving its own survey. The interested reader is referred to the recent survey by Epstein and van Stee [79].

Another important feature of this survey is that it adopts the novel classification scheme for bin packing papers recently introduced by Coffman and Csirik [44]. This scheme allows one to create a compact synthesis of the topic, the main results, and the corresponding algorithms.

Section 2 covers the main definitions and as well as a full description of the classification scheme. For the classical bin packing problem, on-line (and semi-on-line) algorithms are discussed in Sect. 3 and off-line algorithms in Sect. 4. (Section 4.4 is a slight departure from this organization: it deals with anomalous behavior both for on-line and off-line algorithms.)

The second part of this survey concentrates on special cases and variants of the classical problem. Variations on bin size are considered in Sect. 5, dual versions in Sect. 6, variations on item packing in Sect. 7, and additional conditions in Sect. 8. Finally, Sect. 9 gives a series of examples to familiarize the reader with the classification scheme. The classification of the papers considered in the survey is also given, when appropriate, in the Bibliography.

2 Definitions and Classification

This section introduces more formally all of the relevant notation required to define, analyze, and classify bin packing problems. Starting with the classical problem, the notation is then extended to more general versions, and finally, the new classification scheme is described. Additional definitions needed by specific variants are introduced in the appropriate sections.

2.1 Basic Definitions for Classical Bin Packing

The classical bin packing problem is defined by an infinite supply of bins with capacity C and a list $L = (a_1, \ldots, a_n)$ of items (or elements). A value $s_i \equiv s(a_i)$ gives the size of item a_i and satisfies $0 < s_i \leq C,\ 1 \leq i \leq n$. 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 the classical problem, the bin capacity C is just a scale factor, so without loss of generality, one can adopt the normalization $C = 1$. Unless stated otherwise, this convention is in force throughout. One of the exceptions will be in the sections treating variants of the classical problem in which bins B_j have varying sizes. In those sections, bin sizes will be denoted by $s(B_j)$.

In the algorithmic context, nonempty bins will be classified as either *open* or *closed*. Open bins are available for packing additional items. Closed bins are unavailable and can never be reopened. 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. Denote the bins by $B_1, B_2, \ldots$. When no confusion arises, B_j will also denote the set of items packed in the j-th bin, and $|B_j|$ will denote the number of such items.

The items are always organized into a list L (or a list L_j in some indexed set of lists). The notation for list *concatenation* is as usual; $L = L_1 L_2 \ldots L_k$ 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, an index in parentheses is used: $L_{(n)}$ denotes a list of n items.

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

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

A bin (necessarily empty) is *opened* when it receives its first item. Of course, it may be closed immediately thereafter, depending on the algorithm and the item size. It is assumed, 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, in the considered lists, the item sizes are taken from the interval $(0, \alpha]$, with $\alpha \in (0, 1]$. It is usually assumed that $\alpha = \frac{1}{r}$, for some integer $r \geq 1$, and many results apply only to $r = \alpha = 1$.

There are two principal 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

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

Then the *asymptotic worst-case ratio* (or *asymptotic performance ratio*, APR) as a function of α is given by

$$R_A^\infty(\alpha) = \overline{\lim}_{k\to\infty} R_A(k,\alpha).$$

Clearly, $R_A^\infty(\alpha) \geq 1$, and this number measures the quality of the packings produced by algorithm A compared to optimal packings in the worst case. In an equivalent definition, $R_A^\infty(\alpha)$ is a smallest number such that there exists a constant $K \geq 0$ for which

$$A(L) \leq R_A^\infty(\alpha)OPT(L) + K.$$

for every list $L \in V_\alpha$. Hereafter, if α is left unspecified, the APR of algorithm A refers to $R_A^\infty \equiv R_A^\infty(1)$.

The second way to measure the worst-case behavior of an algorithm A is the *absolute worst-case ratio* (AR)

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

The comparison of algorithms by asymptotic bounds can be strikingly different from that by absolute bounds. Generally speaking, the number of items n must be sufficiently large (how large will depend on the algorithm) for the asymptotic bounds to be the better measure for purposes of comparison. Note that the ratios are bounded below by 1; the better algorithms have the smaller ratios. When algorithm A is an on-line algorithm, the asymptotic ratio is also called the *competitive ratio*.

There are some further proposals to measure the quality of a packing, like *differential approximation measure* (see Demange et al. [66], [67]), *random-order ratio* (see Kenyon [136]), *relative worst-order ratio* (see Boyar and Favrholdt [27]), and *accommodation function* (see Boyar et al. [28]).

It is finally worth mentioning a special sequence t_i of integers which was investigated by Sylvester [170] in connection with a number theoretic problem and later generalized by Golomb [107]. (In describing the performance of various algorithms in later sections, such sequence will occasionally be considered.) For an integer $r \geq 1$, define

$$\begin{aligned} t_1(r) &= r+1 \\ t_2(r) &= r+2 \\ t_{i+1}(r) &= t_i(r)(t_i(r)-1)+1, \quad \text{for } i \geq 2. \end{aligned}$$

(The Sylvester 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 also 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 the following: $h_\infty(1) \approx 1.69103$, $h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$.

2.2 General Definitions

This survey covers results on several variants of the one-dimensional bin packing problem. In such variants, the meaning of a number of notations can be different, so more general definitions are necessary. These will be introduced in the present section, together with the basic ideas behind the adopted classification scheme.

The notion of packing items into a sequence of initially empty bins helps visualize algorithms for constructing partitions. It is also helpful in classifying algorithms according to the various constraints under which they must operate in practice. The items are normally given in the form of a sequence or *list* $L = (a_1, \ldots, a_n)$, although the ordering in many cases will not have any significance. To economize on notation, a harmless abuse is adopted whereby s_i denotes the name as well as the size of the i-th item. The generic symbol for packing is P, the number of items in P is denoted by $|\mathsf{P}|$, the sum of the sizes of the items in P is denoted by $c(\mathsf{P})$, and the number of bins in P is denoted by $\#\mathsf{P}$. In the classical bin packing optimization problem, the objective is to find a packing P of all the items in L such that $\#\mathsf{P}$ is minimized over all partitions of L satisfying the sum constraints. Recall that C is only a scale factor in this problem; it is usually convenient to replace the s_i by s_i/C and take $C = 1$.

Let $\mathsf{P}_A(L)$ denote the packing of L produced by algorithm A. In the literature, one finds the notation $A(L)$ representing metrics such as $\#\mathsf{P}$, but since $A(L)$ may denote different metrics for different problems (the same algorithm A may apply to problems with different objective functions), the alternative notation will be

necessary on occasion. The minimum of #P over all partitions P of L satisfying the sum constraints will have the notation $OPT(L) := \min_{\pi(L)} \#\mathsf{P}(L)$, where $\pi(L)$ denotes the set of all such partitions. The notation $OPT(L)$ suffers from the same ambiguity as before, that is, the objective function to which it applies is determined by context. Moreover, in contrast to other algorithm notation, *OPT* does not denote a unique algorithm.

There are two variants of classical bin packing that arise immediately from the definition. In one, a general bin capacity C and a number m of bins are part of the problem instance, and the problem is to find a maximum cardinality subset of $\{s_i\}$ that can be packed into m bins of capacity C. In the other, all n items must be packed into m bins, and the problem is to find a smallest bin capacity C such that there exists a packing of all the items in m bins of capacity C. The first problem suggests yet another in which the total size of the items in a subset $\mathcal{S}$ is the quantity of interest rather than the number $|\mathcal{S}|$ of items. Problems fixing the number of bins fall within scheduling theory whose origins in fact predate those of bin packing theory. In scheduling theory, which is very large in its own right, makespan scheduling is more likely to be described as scheduling a list of tasks or jobs (items) on m identical processors (bins) so as to minimize the schedule length or makespan (bin capacity). This incursion into scheduling problems will be limited to the most elementary variants and applications of bin packing problems, such as those above.

Bin covering problems are also included in bin packing theory for classification purposes, and these change the sum constraints to $c(B_j) \geq 1$, where again, bin capacity is normalized to 1. The classical covering problem asks for a partition, C, called a *cover*, which *maximizes* the number #C of bins satisfying the new constraint. Both the packing and covering combinatorial optimization problems are $\mathcal{NP}$-hard. With problems defined on restricted item sizes or number of items per bin being the major exceptions, this will be the case for nearly all problem variants in the classification scheme. Note that there are immediate variants to bin covering, just as there were for bin packing. For example, one can take the number m of bins to be fixed and ask for a *minimum* cardinality subset of $\{a_i\}$ from which a cover of m bins can be obtained. Or one can consider the problem of finding a *largest* capacity C such that L can be made to cover m bins of capacity C.

Order-of-magnitude estimates of time complexities of fundamental algorithms and their extensions are usually easy to derive. The analysis of parallel algorithms for computing packings is an example where deriving time complexities is not always so easy. However, the research in this area, in which results take the form of complexity measures, has been very limited.

Several results quantify the trade-off between the running time of algorithms and the quality of the packings. They produce polynomial-time (or fully polynomial-time) approximation schemes [100], denoted by PTAS (or FPTAS). In simplified terms, a typical form of such results is illustrated by "Algorithm A produces packings with $O(\varepsilon)$ wasted space and has a running time that is polynomial in $1/\varepsilon$."

The most common approach to the analysis of approximation algorithms has been *worst-case* analysis by which the worst possible performance of an algorithm is compared with the performance of an optimization algorithm. (Detailed definitions will be provided shortly.) The term *performance guarantee* puts a more positive slant on results of this type. So also does the term *competitive analysis*, which usually refers to a worst-case analysis comparing an on-line approximation algorithm with an optimal off-line algorithm. Probability models also enjoy wide use and have grown in popularity, as they bring out typical, *average-case* behavior rather than the normally rare worst-case behavior. In probabilistic analysis, algorithms have random inputs; the items are usually assumed to be independent, identically distributed random variables. For a given algorithm A, $A(L_n)$ is a random variable whose distribution becomes the goal of the analysis. Because of the major differences in probabilistic results and in the analysis that produces them, they are not covered in this survey, which focuses exclusively on combinatorial analysis. For a general treatment of average-case analysis of packing and related partitioning problems, see the text by Coffman and Lueker [46].

2.3 Classification Scheme

The scheme for classifying problems and solutions is aimed at giving the reader a good idea of the results in bin packing theory to be found in any given paper on the subject. Although in many, if not most cases, it is impractical to describe every result contained in a paper, an indication of the main results should be useful to the reader.

The classification uses four fields, presented in the form

problem | algorithm class | results | parameters

For a brief preview, observe that, normally, the *problem* is to minimize or maximize a metric under sum constraints and refers, for example, to the number of bins of fixed capacity and the capacity of a fixed number of bins. The *algorithm class* refers to paradigms such as on-line, off-line, or bounded space. The *results* field specifies performance in terms of absolute or asymptotic worst-case ratios, problem complexity, etc. Lastly, the *parameters* field describes restrictions on problem parameters, such as a limit on item sizes or on the number of items per bin, and a restriction of all data to finite or countable sets. In many cases, there are no additional parameters to specify, in which case the *parameters* field will be omitted. The three remaining fields will normally be nonempty.

In the following, a detailed description of the classification scheme is introduced. Section 9 gives a number of annotated examples to familiarize the reader with the proposed classification. The special terms or abbreviations adopted for entries will be given in bold face. In the large majority of cases, the entry will have a mnemonic quality that makes the precise meaning of an entry clear in spite of its compact form.

2.3.1 Problem

The combinatorial optimization problems of interest can easily be expressed in the notation given so far, but for readability, abbreviations of the names by which the problems are commonly known will be used. Most, but not all, of the problems below were discussed earlier. Moreover, some of them are not treated in this survey and are listed for the sake of completeness.

1. **pack** refers to the problem of minimizing $\#\mathsf{P}(L)$. The default sum constraints are $c(B_j) \leq 1$. But they are $c(B_j) \leq s(B_j)$, if variable bin capacities $s(B_j)$ are considered, and more simply $c(B_j) \leq C$ if there is only a common bin capacity C.
2. **maxpack** problems invert the problem above, that is, $\#\mathsf{P}$ is to be maximized. Clearly, such problems trivialize unless there is some constraint placed on opening new bins. The tacit assumption will be that packings under approximation algorithms must obey a conservative, any-fit constraint: a new bin cannot be opened for an item s_i unless s_i cannot fit into any currently open bin. Optimal packings by definition must be such that, for some permutation of the bins, no item in B_i fits into B_j for any $j < i$.
3. **mincap** refers to the problem of minimizing the common bin capacity needed to pack L into a given number m of bins. *Bin-stretching* analysis applies if the problem instances are restricted to those for which an optimization rule can pack L into m unit-capacity bins. Under this assumption, in the analysis of algorithm A, one asks how large must the bin capacity be (how much must it be stretched) for algorithm A to pack L into the same number of bins.
4. **maxcard(subset)** has the number m of bins and their common capacity C as part of the problem instance. The problem is to find a largest cardinality subset of L that can be packed into m bins of capacity C.
5. **maxsize(subset)** has the number m of bins and their common capacity C as part of the problem instance. The problem is to find a subset of L with maximum total item size that can be packed into m bins of capacity C.
6. **cover** refers to the problem of finding a partition of L into bins that maximizes $\#\mathsf{P}(L)$ subject to the constraints $c(B_j) \geq 1$. The partition is called a *cover*.
7. **capcover** refers to the problem of finding, for a given number m of bins, the *maximum* capacity C such that a covering of m bins of capacity C can be obtained from L.
8. **cardcover(subset)** is the related problem of minimizing the cardinality of the subset of L needed to cover a given number m of bins with a given capacity C.

2.3.2 Algorithm Class

1. **on-line** algorithms sequentially assign items to bins, in the order encountered in L, without knowledge of items not yet packed. Thus, the bin to which a_i is assigned is a function only of the sizes $s_1, \ldots, s_i$.
2. **off-line** algorithms have no constraints beyond the intrinsic sum constraints; an off-line algorithm simply maps the entire list L into a packing $\mathsf{P}(L)$. All items are known in advance, so the ordering of L plays no role.

3. **bounded-space** algorithms decide where an item is to be packed based only on the current contents of at most a finite number k of bins, where k is a parameter of the algorithm. A more precise definition and further discussion of these algorithms appear later.
4. **linear-time** algorithms have $O(n)$ running time. More precisely, all such algorithms take constant time to pack each item.

   The three characterizations above are orthogonal. But the literature suggests that the following convention allows to use one term in classifying algorithms most of the time: *bounded space implies linear time and linear time implies on-line.* Exceptions will be noted explicitly: it will be seen below (under **repack**) how off-line algorithms can be linear time.
5. **open-end** refers to certain cover approximation algorithms. An open bin B_j, that is, one for which $c(B_j) < s(B_j)$ and further packing is allowed, must be closed just as soon as $c(B_j) \geq s(B_j)$. (The item causing the "overflow" is left in the bin.)
6. **conservative** algorithms are those required to pack the current item into an open bin with sufficient space, whenever such a bin exists; in particular, it cannot choose to open a new bin. Scheduling algorithms satisfying a similar constraint are sometimes called *work conserving*.
7. **repack** refers to packing problems which allow the repacking (possibly limited in some way) of items, that is, moving an item, say s_i, from one bin to another.
8. **dynamic** packing introduces the time dimension; an instance L of this problem consists of a sequence of triples (s_i, b_i, d_i) with b_i and d_i denoting arrival and departure times, respectively. Under a packing algorithm A, $A(L, t)$ denotes the number of bins occupied at time t, that is, the number of bins occupied by those items a_i for which $b_i \leq t < d_i$.

2.3.3 Results

Almost all results fall into the broad classes mentioned in Sect. 2.1.

1. $\mathbf{R}_{\mathbf{A}}^{\infty}$ is the asymptotic worst-case ratio, with algorithm A specified when appropriate. When only a bound is proved, the word **bound** is appended.
2. $\mathbf{R}_{\mathbf{A}}$ is the absolute worst-case ratio. When only a bound is proved, the word **bound** is appended.
3. Where possible, complexity of the problem will be given in the standard notation of problem complexity. Approximation schemes are classified as complexity results and have entries like **PTAS and FPTAS** as noted earlier.
4. Complexity of the algorithm may also be a result; it refers to running-time complexity and will be signaled by the entry **running time**.

A paper classified as a worst-case analysis may also have complexity results (but not conversely, unless both types of results figure prominently in the paper, in which case both classifications will be given).

2.3.4 Parameters

These typically correspond to generalizations whereby limitations can be placed on the problem instance, or further properties of the algorithm classification. In some

cases, problems may be simplified by certain limitations, such as a specific upper limit of two to the number of items per bin.

1. $\{\mathbf{B_i}\}$ means that there can be more than one bin size and an unlimited supply of each size.
2. **stretch** refers to certain asymptotic bounds which compare the number of bins of capacity C needed to pack L by algorithm A to the number of unit-capacity bins needed by an optimal packing.
3. **mutex** stands for mutual exclusion and introduces constraints, where needed, in the form of a sequence of pairs (s_i, s_j), $i \neq j$, meaning that s_i and s_j cannot be put in the same bin.
4. $\mathbf{card(B) \leq k}$ gives a bound on the number of items that can be packed in a bin.
5. $\mathbf{s}_i \leq \alpha$ **or** $\mathbf{s}_i \geq \alpha$ denotes bounds on item sizes, the former being far more common in the literature. In some cases, α is specialized to a discrete parameter $1/k$, k an integer. The problems with item-size restrictions are called *parametric* cases in the literature.
6. **restricted** $\mathbf{s_i}$ refers to simplified problems where the number of different item sizes is finite.
7. **discrete** calls for discrete sets of item sizes, in particular, item sizes that, for a given integer r, are all multiples of $1/r$ with $C = 1$. Equivalently, the bin size could be taken as r and item sizes restricted to the set $\{1, \ldots, j\}$ for some $j \in \{1, \ldots, r\}$. While this may not be a significant practical constraint, it will affect the difficulty of the analysis and may create significant changes in the results.
8. **controllable** means that one has the possibility not to pack an item as is, but to first take some decision about it, such as rejecting it or splitting it into parts.

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). This review starts with results for some classical algorithms and then generalizes to the *Any-Fit* and the *Almost Any-Fit* classes of on-line algorithms. The subclass of *bounded-space* on-line algorithms will also be considered: 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; e.g., one may have to load trucks at a depot where only a limited number of trucks can be at the loading dock). This section will be concluded by a detailed discussion of lower bounds on the APRs of certain classes of on-line algorithms, but before doing so, relevant variations of old algorithms and the current best-in-class algorithms will be presented. Anomalous behavior occurring in on-line algorithms is discussed in Sect. 4.4.

3.1 Classical Algorithms

In describing an on-line algorithm, it will occasionally be convenient, just before a decision point, to refer to the next item to be packed as the *current* item; right after A packs a_i, $i < n$, a_{i+1} becomes the 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 1 (Johnson et al. [129]) *One has*

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

Fisher [86] 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 one might expect WF to behave better than NF, it does not.

Theorem 2 (Johnson [127]) *For all* $\alpha \in (0, 1]$

$$R_{\mathrm{WF}}^{\infty}(\alpha) = R_{\mathrm{NF}}^{\infty}(\alpha).$$

To achieve smaller APRs, 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. FF is neither linear time nor bounded space.

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

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, it is convenient to describe this technique, which was introduced in [103, 129] and subsequently applied in many other papers (see, e.g., [13, 92, 127, 143]).

3.2 Weighting Functions

To bound the asymptotic worst-case behavior of an algorithm A, one can try to find a function $W_A : (0, 1] \longrightarrow \mathbb{R}$ 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 under A's *weighting function* W_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

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

Together, (1) and (3) obviously imply the bound $A(L) \leq K^* OPT(L) + K$, and hence $R_A^\infty \leq K^*$. Note 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, where $s(a)$ denotes the size of item a. An observation is needed about NF when $\mathrm{NF}(L) > 1$: Since the first item of B_{j+1} did not fit in B_j, $1 \leq j < \mathrm{NF}(L)$, 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 $B_j \cup B_{j+1}$ exceeds 2. Then

$$\sum_{j=1}^{\mathrm{NF}(L)} \sum_{a \in B_j} W_{\mathrm{NF}}(a) \geq \sum_{j=1}^{\lfloor \mathrm{NF}(L)/2 \rfloor} \sum_{a \in B_{2j-1} \cup B_{2j}} W_{\mathrm{NF}}(a)$$

$$\geq 2\lfloor \mathrm{NF}(L)/2 \rfloor \geq \mathrm{NF}(L) - 1,$$

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 (Johnson et al. [129])

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

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

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

and it requires a substantial effort. (The argument encompasses several pages of case analysis.) Yet, 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 for $\alpha \leq \frac{1}{2}$ can be proved without weighting functions (see [129]), 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^{∞}. 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 $R_A^\infty \geq K_*$, and if K_* is equal to K^* of the upper bound analysis, one has 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 nonempty bins, then the current item will not be packed into B_{j+1} unless it does not fit in any of the bins $B_1, \ldots, B_j$.*

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

Theorem 4 (Johnson [127]) *For every algorithm $A \in \mathcal{AF}$ and for every $\alpha \in (0, 1]$*

$$R_{\mathrm{FF}}^\infty(\alpha) \leq R_A^\infty(\alpha) \leq R_{\mathrm{WF}}^\infty(\alpha).$$

By a slight tightening of the Any-Fit constraint, one 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 nonempty 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. The class of on-line algorithms satisfying both constraints above will be denoted by $\mathcal{AAF}$.

Theorem 5 (Johnson [127]) $R_A^\infty(\alpha) = R_{\mathrm{FF}}^\infty(\alpha)$ *for every A in $\mathcal{AAF}$.*

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 $R_{\mathrm{AWF}}^\infty(\alpha) = R_{\mathrm{FF}}^\infty(\alpha) = \frac{17}{10}$.

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. One obtains 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 [126] 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. It can be expected 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 [58] constructed the worst-case sequences, and then Mao was able to prove the exact bound:

Theorem 6 (Mao [153]) *For any* $k \geq 2$, $\quad R^{\infty}_{\mathrm{NF}_k} = \frac{17}{10} + \frac{3}{10(k-1)}$.

In general, a bounded-space algorithm is defined 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 lowest indexed one or one of largest content.

The algorithm that uses packing rule X, closing rule Y, and k-bounded space is denoted by $\mathrm{A}XY_k$, 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 one always gets NF.

Algorithm ABF_k was first analyzed by Mao, who called it *Best-k-Fit*. He proved that for any fixed k, this algorithm is slightly better than NF_k.

Theorem 7 (Mao [152]) *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 [184] (see also the paper version [187]).

Theorem 8 (Zhang [187]) *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 9 (Csirik and Johnson [60]) *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 4. A better on-line algorithm can only be obtained without the Any-Fit constraint. In the remaining part of this section, it is shown how the fruitful idea of *reservation techniques* (introduced by Yao [180] 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 [143] 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 bin types: 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 [92] extended the idea to general α. Observe that, by selecting $\alpha \in \left(\frac{1}{r+1}, \frac{1}{r}\right]$, 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 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 Sect. 2 for the definitions of the quantities $t_s(r)$ and $h_\infty(r)$.)

Theorem 10 (Lee and Lee [143], Galambos [92]) *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 [92, 143] 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 seven 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 [175]. Using a more sophisticated interval structure, one based on the Golomb sequences, the performance of his *Simplified Harmonic* (SH_k) algorithm improved on the $\frac{17}{10}$ bound with six open bins; precisely, $R^{\infty}_{\mathrm{SH}_6} \approx 1.69444$. Moreover, Woeginger proved the following deeper, more general result.

Theorem 11 (Woeginger [175]) *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 [60] (see also [59]) and van Vliet [172, 173]. They gave tight bounds for the case $\alpha = 1$ with $k = 4$ and 5. Tight bounds for further k remain open problems.

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 $42 = t_4(1) - 1$

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	1.70000
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

Similarly, the general case has not been discussed exhaustively, and some questions raised by Woeginger [175] 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 $\frac{17}{10}$?
- What is the best possible APR for any on-line heuristic using 2-bounded space? (ABB_2 achieves a worst-case ratio of $\frac{17}{10}$.)
- By considering 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 [58]), 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 [143] for the $\alpha = 1$ case, bounded-space algorithms cannot do better. The result holds for general α too, as shown by Galambos, that is,

Theorem 12 (Lee and Lee [143], Galambos [92]) *Every bounded-space on-line bin packing algorithm A satisfies $R_A^\infty(\alpha) \geq h_\infty(r)$ for all α, $\frac{1}{r+1} < \alpha \leq \frac{1}{r}$.*

None of the known bounded-space algorithms achieves the lower bound using a finite number of open bins. It will be later shown (see Sect. 3.7) 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 [180] 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

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

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

accordingly 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 four 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 in $(\frac{1}{3}, \frac{2}{5}]$) is thrown in with the type-4 items, that is, 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 $R^{\infty}_{\mathrm{RFF}} = \frac{5}{3} = 1.666\ldots$. 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 m-th 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 [142, 143] 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. The details are omitted. Note however that the time complexity of RHF is $O(n)$, but the algorithm is no longer bounded space. Its APR is given by

Theorem 13 (Lee and Lee [143]) $R^{\infty}_{\mathrm{RHF}} \le \frac{373}{228} \approx 1.63596$.

It is not known whether this bound is tight.

In 1989, several improved algorithms were presented by Ramanan et al. [159]. 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

$$I_{1,s} = (\tfrac{1}{2}, 1 - y] \quad I_{2,s} = (\tfrac{1}{3}, y]$$
$$I_{1,b} = (1 - y, 1] \quad I_{2,b} = (y, \tfrac{1}{2}]$$

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 $M = 40$, the following bounds hold.

Theorem 14 (Ramanan et al. [159])

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

The above algorithm was further generalized by Ramanan et al. [159] 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

$$I_{1,b} = (1 - y_1, 1], \qquad I_{2,b} = (y_h, \tfrac{1}{2}],$$
$$I_{1,j} = (1 - y_{j+1}, 1 - y_j], \quad I_{2,j} = (y_{h-j}, y_{h-j+1}], \quad 1 \leq j \leq h,$$

and

$$I_{\lambda,1} = (0, \lambda], \qquad I_{\lambda,2} = (\lambda, \tfrac{1}{3}], \qquad 0 < \lambda \leq \tfrac{1}{3}$$

where $\frac{1}{3} = y_0 < y_1 < \cdots < y_h < y_{h+1} = \frac{1}{2}$, 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. [159] developed an algorithm (*Modified Harmonic-2*, MH2), which is in $C^{(2)}$ and proved that $R_{\mathrm{MH2}}^{\infty} \leq 1.612\ldots$. 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 15 (Ramanan et al. [159]) *There is no on-line algorithm A in $C^{(h)}$ such that $R_A^\infty < \frac{3}{2} + \frac{1}{12} = 1.58333\ldots$.*

The HARMONIC+1 algorithm of Richey [160] 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 $\frac{1}{2}$. Seiden [163] showed that such result was flawed and gave a new algorithm, HARMONIC++, whose asymptotic performance ratio is at most 1.58889, which is the current best APR for on-line bin packing.

3.6 APR Lower Bounds

In previous sections, some results concerning lower bounds on the APR were mentioned, but the fundamental problem was not considered yet: what is the best an on-line algorithm can do in the asymptotic worst case? In this section, lower bound results are discussed 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, in order to force bad behavior on a heuristic algorithm A, one 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, 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 $\frac{1}{2} + \varepsilon$. The asymptotic behavior of an arbitrary approximation algorithm A will be investigated 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 $OPT(L_1L_2) = n$. Consider the behavior of this 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,

$$R_A^\infty \geq \max\left\{\frac{A(L_1)}{OPT(L_1)}, \frac{A(L_1L_2)}{OPT(L_1L_2)}\right\} = \max\left\{\frac{n+x}{n}, \frac{3n-x}{2n}\right\}$$

Table 3 Lower bounds and R_A^∞ for various algorithms

r	Lower bound	R_{HF}^∞	R_{FF}^∞	R_{NF}^∞	Best known
1	1.54037	1.69103	1.70000	2.00000	1.58889
2	1.38966	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	1.18881	1.19102	1.20000	1.25000	1.19102
6	1.15982	1.16102	1.16667	1.20000	1.16102

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 $L_1 \ldots L_j$, $(1 \leq j \leq k)$. The first step along these lines was made by Yao [180]. 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 Sylvester's sequences, Brown [32] and Liang [147] independently gave a further improvement to 1.53634577 .... (The largest sizes in their sequences were as follows: $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$, $\frac{1}{7} + \varepsilon$, $\frac{1}{43} + \varepsilon$, and $\frac{1}{1{,}807} + \varepsilon$.) Galambos [92] used the Golomb sequences to extend the idea to general α. The proof in [92] was considerably simplified by Galambos and Frenk [93]. van Vliet gave an exhaustive analysis of the lower bound constructions with a linear programming technique applied to all α and gave the following lower bound:

Theorem 16 (van Vliet [171]) *For any on-line algorithm A,* $R_A^\infty \geq 1.54015\ldots$

The current best-in-class, $R_A^\infty \geq 1.54037$, was given by Balogh et al. [15]. 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 $r \geq 2$.

Faigle, Kern, and Turán [83] proved that if there are only two item sizes, then no on-line algorithm can be better that 4/3.

Chandra [38] has examined the effect on lower bounds when randomization is allowed in the construction of on-line algorithms, that is, 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.

3.7 Semi-on-line Algorithms

The APR of on-line algorithms cannot break through the $1.540\ldots$ barrier (see Sect. 3.6). For almost 20 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, one can consider giving the algorithm more information about the list and/or more freedom with respect to the pure on-line case. This section deals with *semi-on-line* (SOL) algorithms that can

– Repack elements
– Look ahead to later elements before assigning the current one
– Assume some preordering of the elements

First consider the case where repacking is allowed. SOL algorithms allowing only a restricted number of elements to be repacked at each step are called *c-repacking SOL* algorithms. It was seen in Sect. 3.4 that no known on-line bounded-space algorithm reaches the bound $h_\infty(1)$ using finitely many open bins. It will be shown that this is possible with SOL algorithms.

In 1985, Galambos [91] 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, it was proved that $h_\infty(1) \le R^\infty_{\mathrm{BNF}} \le \frac{18}{10}$.

Galambos and Woeginger [95] 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, sorted by nonincreasing size, are repacked by the FF strategy, with the result that either one bin becomes empty or at least one bin B has $w(B) \ge 1$; and (iii) all bins B with $w(B) \ge 1$ are closed and replaced by new empty bins. In [95], 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 et al. [99] (see also [97] and [98]) were the first to beat the 1.540 on-line bound via repacking. They gave two algorithms. Both of them use an unusual step to repack the elements: the small elements are grouped into a "bundle" of $O(n)$ elements, which can be repacked in a single step.

In the first algorithm, A_1, the interval $(0, 1]$ is divided into four 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 – in a bundle – 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, $R^\infty_{\mathrm{A}_1} \le \frac{3}{2}$. In the second algorithm, A_2, the unit interval is divided into six 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 Sect. 4.2). The time complexity of A_2 is $O(n \log n)$, and its APR is $R^{\infty}_{A_2} \leq \frac{4}{3}$.

Ivkovic and Lloyd [122] gave a further improvement on SOL algorithms 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. The dynamic bin packing problem will be considered in Sect. 7.1 in detail, but here it is enough to know that in case of dynamic packing, deletions of some elements are allowed in each step and $A(L)$ is considered as the maximum number of occupied bins during the packing. Ivkovic and Lloyd proved a $\frac{4}{3}$ lower bound for the c-repacking SOL algorithms. This result was improved by Balogh et al. [14].

Theorem 17 (Balogh et al. [14]) *For any c-repacking SOL algorithm A, the APR satisfies* $R^{\infty}_{A} \geq 1.3871\ldots$.

Ivkovic and Lloyd [121] also presented approximation schemes for their dynamic, SOL model, applying the techniques of Fernandez de la Vega and Lueker and those of Karmarkar and Karp cited in Sect. 4.3.

Consider now 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 relaxation to off-line algorithms, consider the case of *bounded* lookahead. Grove [111] 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 12) remains valid, but Grove's *Revised Warehouse* (RW) algorithm reaches the bound if W is sufficiently large. In his proof, Grove uses a weighting function argument.

Another subclass of the lookahead SOL problems arises if the input list is divided into a number of batches. If an algorithm works on a batched list, then it has to pack each batch as an off-line list (i.e., lookahead is only possible within the current batch). However, while packing elements of the current batch, the items of earlier batches cannot be moved. If the number of batches is bounded by m, the problem is called *m-batched bin packing* (BBP) problem. Gutin et al. [112] were the first to study the BBP problem: they investigated in depth the 2-BBP problem and proved a lower bound of $1.3871\ldots$ for this case.

The rarely considered class of algorithms in which it is assumed that the input list is *presorted* is finally considered. Because of the lower bound constructions, it is easy to see that, if the list is presorted by increasing item size, the on-line lower bounds remain valid. Moreover, the Johnson result holds: if the list is presorted by decreasing item size, then $\frac{11}{9} \leq R^{\infty}_{A} \leq \frac{5}{4}$ for any algorithm $A \in \mathcal{AF}$ (see Theorem 21). This begged the broader question of lower bounds: how good can an arbitrary algorithm be? Based on two different lists, a partial answer was given in

the early 1980s by Csirik et al. [64]. They proved that if the list is presorted by decreasing item size, then $R_A^\infty \geq \frac{8}{7}$ for all algorithms A. Although this approach seemed to be very simple (the authors used only two different list types), no progress was made until very recently Balogh et al. [15] gave an improved lower bound:

Theorem 18 (Balogh et al. [15]) *If the list is presorted by decreasing item size, then* $R_A^\infty \geq \frac{54}{47}$ *for all algorithms A.*

4 Off-line Algorithms

An *off-line* algorithm has all items available for preprocessing, reordering, grouping, etc. before packing. It has been 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 the $O(n \log n)$ sorting step makes the algorithm no longer linear time.

The review starts with results for approaches that sort the items before executing one of the on-line algorithms. Linear-time heuristics are then considered. The section is concluded by approximation schemes and by 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, one obtains 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 19 (Baker and Coffman [13]) *If* $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$ $(r \geq 1)$, *then* $R_{\mathrm{NFD}}^\infty(\alpha) = h_\infty(r)$.

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 20 (Johnson et al. [129]) $R_{\mathrm{FFD}}^\infty = R_{\mathrm{BFD}}^\infty = \frac{11}{9}$.

The original proof was based on the weighting function technique, but subsequent proofs introduced dramatic changes; the giant case analysis made in 1973 by Johnson [126] was considerably shortened by Baker [12] in 1985, and in 1991, Yue

[183] 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. Then, in 1997, Li and Yue [146] further reduced the constant to $\frac{7}{9}$ and conjectured the tight value to be $\frac{5}{9}$. However, in 2007, Dósa [68] closed the issue by proving that the tight value is $\frac{6}{9}$.

The behavior of BFD for general α is not known, but that of FFD has been intensively investigated. Johnson et al. [129] 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 [57] 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 [178] (see also [179]), who 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 Sect. 3.3). Johnson showed that, after a presort in increasing order, Any-Fit algorithms do not perform well in the worst case; for example, their APRs must be at least $h_{\infty}(1)$ when $\alpha = 1$. But presorting in decreasing order is much more useful.

Theorem 21 (Johnson [126, 127]) *Any algorithm $A \in \mathcal{AF}$ operating on a list presorted in decreasing-size order must have*

$$\tfrac{11}{9} \leq R^{\infty}_{A}(1) \leq \tfrac{5}{4}$$
$$\tfrac{1}{m+2} - \tfrac{2}{m(m+1)(m+2)} \leq R^{\infty}_{A}(\alpha) \leq \tfrac{1}{m+2}, \quad \textit{where } m = \lfloor \tfrac{1}{\alpha} \rfloor \textit{ and } \alpha < 1.$$

For a long time, the FFD bound was the smallest proved APR. Johnson [126] 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 $\lim_{k\to\infty} R^{\infty}_{\mathrm{MF}_k} = \frac{10}{9}$, but almost 20 years later, the conjecture was contradicted by Friesen and Langston [90], who gave examples for which $R^{\infty}_{\mathrm{MF}_k} \geq \frac{5}{4}, k \geq 2$.

Yao [180] 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 $R^{\infty}_{\mathrm{RFFD}} \leq \frac{11}{9} - 10^{-7}$. Following this result, further efforts were made to develop better off-line algorithms. Garey and Johnson [102] 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 22 (Garey and Johnson [102]) $R^{\infty}_{\mathrm{MFFD}} = \frac{71}{60} = 1.18333\ldots.$

Another modification of FFD was presented by Friesen and Langston [90]. 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 $R^{\infty}_{\mathrm{B2F}} = \frac{5}{4}$, 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 23 (Friesen and Langston [90]) $1.16410\ldots = \frac{227}{195} \leq R^{\infty}_{\mathrm{CFB}} \leq \frac{6}{5} = 1.2.$

Concerning the absolute worst-case ratio, Johnson et al. [129] had already conjectured that, if the number of bins in the optimal solution is more than 20, then the absolute worst-case ratio of FF is no more than 1.7. The first results were given by

Johnson et al. [129] and Simchi-Levi [169]. They showed that FF and BF have an absolute performance bound not greater than 1.75 and that FFD and BFD have an absolute performance ratio of 1.5. The latter is the best possible for the classical bin packing problem, unless $\mathcal{P} = \mathcal{NP}$. Xie and Liu [177] improved the bound for FF to 1.737. Zhang et al. [188] provided a linear-time bounded-space off-line approximation algorithm with an absolute worst-case ratio of 1.5 and a linear-time bounded-space on-line algorithm with an absolute worst-case ratio of 1.75. Berghammer and Reuter [24] gave a different linear-time algorithm with an absolute worst-case ratio of 1.5.

4.2 Linear Time, Randomization, and Other Approaches

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 accomplished in linear time – in particular, without sorting. The first such heuristic was constructed by Johnson [127]. His *Group-X-Fit Grouped* (GXFG) algorithm depends on the choice of a set of breakpoints X, defined by a sequence of real numbers $0 = x_0 < x_1 < \cdots < x_p = 1$. 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 24 (Johnson [127]) *For all $m \geq 1$, if X contains $\frac{1}{m+2}$, $\frac{1}{m+1}$, and $\frac{1}{m}$, then $R^{\infty}_{\mathrm{GXFG}}(\alpha) = \frac{m+2}{m+1}$ for all $\alpha \leq 1$ such that $m = \lfloor \frac{1}{\alpha} \rfloor$.*

For $\alpha = 1$, the above theorem gives $R^{\infty}_{\mathrm{GXFG}} = \frac{3}{2}$, 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 25 (Martel [154]) $R^{\infty}_{\mathrm{H}_4} = \frac{4}{3}$.

Later, Békési et al. [23] 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.

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

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	$\leq 1.183\ldots$	$\leq 1.183\ldots$	≤ 1.15

Table 4 summarizes the tightest worst-case ratios of off-line algorithms (H_7 denotes the algorithm of Békési and Galambos [22]).

As a final note on fast off-line algorithms, the reader is referred to the work of Anderson et al. [4] 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.

Caprara and Pferschy [34] proposed a simple (although non-polynomial) algorithm, discussed in Sect. 7.2, for which they proved an upper bound on the worst-case ratio of $4/3 + \ln 4/3 = 1.62102\ldots$.

4.3 Asymptotic Approximation Schemes

In 1980, Yao [180] 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 [84] 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 Sect. 8.1 for an integer programming formulation of the bin packing problem.) In this section, their result is first discussed, and then some improvements proposed by Johnson and by Karmarkar and Karp are introduced. Further discussion can be found in [52, 116].

The main idea in [84] 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, the algorithm packs 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 $m = \lceil \frac{4}{\varepsilon^2} \rceil$ 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 $L \setminus L_1$, that is, those not having a corresponding rounded element in L_2, hence $|L_3| \leq 2h-1$.

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, this implies 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 26 (Fernandez de la Vega and Lueker [84]) *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 [128], 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 26, there exists a polynomial-time approximation algorithm for bin packing with an APR equal to 1.

Karmarkar and Karp [132] 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 $A(L) - OPT(L)$, one has the following.

Theorem 27 (Karmarkar and Karp [132]) *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* $\varepsilon > 0$*:*

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 [105, 106] may be very efficient in practice.

Taking a complementary approach, Hochbaum and Shmoys [118] (see also [117]) define an *ε-dual approximation algorithm* for bin packing, denoted in the following by M_ε. For a given $\varepsilon > 0$, M_ε finds in polynomial time a packing of L in $OPT(L)$ bins, each of capacity $C = 1 + \varepsilon$. A primary objective of M_ε is the approximation scheme for the capacity minimization problem discussed in Sect. 6.1, but such algorithms also find use in bin packing settings where precise bin capacities vary or are unknown.

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 Sect. 5.2. A general discussion of the details has been given by Hochbaum [116], who describes a version of M_ε that requires only linear running time (with constants depending on ε).

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 for 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 cannot normally be expected to work. The following instances illustrate bin packing anomalies. Define

$$L = (0.7, 0.68, 0.5399, 0.3201, 0.15, 0.14, 0.08, 0.08, 0.08, 0.08, 0.08, 0.07)$$
$$L' = (0.7, 0.68, 0.5399, 0.32, \quad 0.15, 0.14, 0.08, 0.08, 0.08, 0.08, 0.08, 0.07)$$

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:

$$c(B_1) = 0.7 + 0.15 + 0.08 + 0.07$$
$$c(B_2) = 0.68 + 0.08 + 0.08 + 0.08 + 0.08$$
$$c(B_3) = 0.5399 + 0.3201 + 0.14$$

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

$$c(B_1) = 0.7 + 0.15 + 0.14$$
$$c(B_2) = 0.68 + 0.32$$
$$c(B_3) = 0.5399 + 0.08 + 0.08 + 0.08 + 0.08 + 0.08$$
$$c(B_4) = 0.07$$

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 the size of each element a_i is no less than the size of the corresponding element a'_i. 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 [110] and Johnson [126] observed that algorithms FF and FFD are anomalous. However, the current insights into the anomalous behavior of bin packing algorithms are due mainly to Murgolo. In the notation of Sect. 3.4 (with W meaning Worst-Fit), he proved

Theorem 28 (Murgolo [156]) *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 cannot fit into an open bin. For conservative algorithms, the following bound applies.

Theorem 29 (Murgolo [156]) *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 30 (Murgolo [156]) *There exist arbitrarily long lists L_i and L_i', with L_i' dominating L_i, $(i = 1, 2, 3)$, such that*

$$\begin{aligned} &if\ A \in \{\text{BF}, \text{BFD}\} \quad then\ A(L_1) \geq \tfrac{43}{42}A(L_1'), \\ &if\ A \in \{\text{FF}\} \qquad\quad then\ A(L_2) \geq \tfrac{76}{75}A(L_2'), \\ &if\ A \in \{\text{WF}, \text{WFD}\}\ then\ A(L_3) \geq \tfrac{16}{15}A(L_3'). \end{aligned}$$

The following sections discuss problems in which special types of lists, alternative objective functions, or any of various constraints on items or packings are considered. It will be seen 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 Variations on Bin Sizes

5.1 Variable-Sized Bins

Consider the case where one is given different bin types, $B_1, B_2, \ldots, B_k$ with sizes $1 = s(B_1) > s(B_2) > \cdots > s(B_k)$. For each type, an unlimited number of bins are available, and the aim is to pack a list L of elements a_i, with $s_i \in (0, 1]$, 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, for example, cutting stock problems, the assignment of commercials to variable size breaks in television or radio broadcasting, and memory allocation for computer operating systems.

At first glance, one might expect variable-sized bin packing to be harder than the classical problem. However, this need not be true in general. For example, if there are 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, smaller elements can be packed 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 [89] 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 [138] 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. Burkard and Zhang [33] pointed out that this APR characterizes a large class of algorithms that use one of the above bin-opening rules.

Kinnersley and Langston [138] 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 [185] proved that with $f = \frac{1}{2}$, the APR is $\frac{17}{10}$.

Csirik [56] 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 $s(B_2) = \frac{7}{10}$, then $R^\infty_{\mathrm{VH}} = \frac{7}{5}$.

This result is very interesting since the bound is smaller than the 1.54 on-line lower bound of Balogh et al. [15] for the classical problem (see Sect. 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.

Csirik's VH algorithm suffers from the same "illness" as the classical HF_k algorithm (see Sect. 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 $M_j = \lceil M s(B_j) \rceil$ $(j = 1, \ldots, l)$, 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 $R^\infty_{\mathrm{VH}_M} \geq \frac{17}{10}$.

This raised further 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 (or an arbitrary number of) bin sizes?

Some of the above questions were answered by Seiden [162], who revisited the VH algorithm. For the case where there are two bin sizes, he developed an algorithm that provides a lower bound for any fixed α. Specifically, he proved the following result:

$$1.37530 < \frac{78{,}392{,}621}{57{,}000{,}000} \leq \inf_{\alpha \in (0,1]} R^\infty_{OPT}(\alpha) \leq \frac{395{,}101{,}163}{287{,}280{,}000} < 1.37532.$$

He additionally proved that the upper bound $h_\infty(1) \approx 1.69103$, given in [56], is tight and that algorithm VH is optimal among bounded-space algorithms.

To obtain this result, Seiden used the weighting function technique and developed "twin" mathematical programs, which have the same sets of conditions but different objective functions. He used the mathematical programs to get lower and upper bounds for the asymptotic performance ratio of VH. He proved the asymptotic optimality of VH, although the precise value could not be computed.

The third question remained open even in the case of three different sizes.

Burkard and Zhang [33] investigated other bounded-space algorithms for variable-sized 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 $s_i > \frac{1}{2}$ 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 31 (Burkard and Zhang [33]) $R^{\infty}_{\mathrm{VBB}_k} = \frac{17}{10}$. *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 [89] 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 $R^{\infty}_{\mathrm{FFDLR}} = \frac{3}{2}$ and $R^{\infty}_{\mathrm{FFDLS}} = \frac{4}{3}$.

Seiden et al. [164] made great strides by improving the upper bound and giving the first – and to this very day the only – general lower bound for the case with two bin sizes. First, they introduced a parameter $\mu \in (\frac{1}{3}, \frac{1}{2})$. They gave an algorithm $\mathrm{VH1}(\mu)$ for all $\alpha \in (0, 1)$, and an algorithm $\mathrm{VH2}(\mu)$, only defined for $\alpha > \max\{\frac{1}{2(1-\mu)}, \frac{1}{3\mu}\}$. Both algorithms are combinations of the Harmonic and Refined Harmonic algorithms. The upper bound was achieved by optimizing μ over each choice of α, and the authors chose the best value among $\mathrm{VH1}(\mu)$, $\mathrm{VH2}(\mu)$, and

VH. The resulting upper bound is at most $\frac{373}{228} < 1.63597$ for all α. Note that this is the asymptotic performance ratio of the RHF algorithm in the classic bin packing context. The lower bound result is summarized by the following theorem.

Theorem 32 (Seiden et al. [164]) *Any on-line algorithm for the variable-sized bin packing problem with two bin sizes has asymptotic performance ratio at least* $\frac{495,176,908,800}{370,749,511,199} > 1.33561$.

The result is quite good since the largest gap between the two bounds is 0.18193 for $\alpha = 0.9071$, and the smallest gap is 0.003371 for $\alpha = 0.6667$.

Murgolo [157] 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 [132], he solved a linear programming formulation of the problem by the ellipsoid method, thus obtaining a fully polynomial-time approximation scheme.

Zhang [186] 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 one has 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 an APR equal to 2 and left as an open question whether one can devise an algorithm with an APR better than 2.

Kang and Park [130] investigated a problem where there are m different bin sizes, and the total cost of a unit of each bin size does not increase with the bin size, that is,

$$\frac{c_{i_1}}{s(B_{i_1})} \leq \frac{c_{i_2}}{s(B_{i_2})}, \quad \text{if} \quad 1 \leq i_1 < i_2 \leq m.$$

They analyzed two algorithms, *Iterative First-Fit Decreasing* (IFFD) and *Iterative Best-Fit Decreasing* (IBFD), for three cases. If both the bin and item sizes are divisible, then both algorithms are optimal. If only the bin sizes are divisible, then $R_{\mathrm{IFFD}} = R_{\mathrm{IBFD}} = \frac{11}{9}$, whereas for the general (non-divisible) case, $R_{\mathrm{IFFD}} = R_{\mathrm{IBFD}} = \frac{3}{2}$. In addition, they proved that an algorithm by Coffman et al. [51] for the variable-sized bin case does not provide the optimal solution.

The case of weak divisibility has not been treated.

5.2 Resource Augmentation

The idea of *resource augmentation* was introduced by Csirik and Woeginger [63]. The motivation is that the worst-case examples are "pathological," that is, if for a unit-capacity bin one considers items with size $\frac{1}{2} + \varepsilon$ with a sufficiently small ε, then only one item per bin can be packed. If the size of the bins is slightly increased, then two items can be packed in a bin, thus sparing a lot of capacity. (This is also called *bin stretching*.)

In this problem, an algorithm A has to pack a list L of elements in $(0, 1]$ into a minimal number of bins of capacity $C \geq 1$. Denote by $A_C(L)$ the number of bins used by A for packing L into bins of capacity C. Then the APR is defined as follows:

$$R_C^\infty = \lim_{OPT_1(L)\to\infty} \sup_L A_C(L),$$

where $OPT_1(L)$ is the optimal (off-line) packing of L into unit-capacity bins. Csirik and Woeginger considered on-line bounded-space algorithms and gave a complete analysis. For each C, a sequence $T(C) = \langle t_1, t_2, \ldots, \rangle$ of positive integers is defined:

$$t_1 = \lfloor 1 + C \rfloor \qquad \text{and} \qquad r_1 = \frac{1}{C} - \frac{1}{t_1}$$

and, for $i = 2, 3, \ldots$,

$$t_i = \left\lfloor 1 + \frac{1}{r_{i-1}} \right\rfloor \qquad \text{and} \qquad r_i = r_{i-1} - \frac{1}{t_i}.$$

It is clear that this is a simple transformation of the Sylvester sequence, so

$$\frac{1}{C} = \sum_{i=1}^{\infty} \frac{1}{t_i}.$$

They defined a new function

$$\rho(C) = \sum_{i=1}^{\infty} \frac{1}{t_i - 1}$$

for which it is easy to prove that $\rho(C)$ is strictly decreasing on $[1, \infty)$, and $\rho(1) = h_\infty(1) \approx 1.69103$, $\rho(2) \approx 0.69103$. Furthermore, the following inequalities hold:

$$\frac{1}{m} \leq \rho(m) \leq \frac{1}{m-1} \text{ for integer } m \geq 2.$$

For a given $l \geq 3$, they defined t_l intervals: $\mathcal{I}_j = \left(\frac{C}{j+1}, \frac{C}{j}\right]$ for $j = 1, \ldots, t_l - 1$, and $\mathcal{I}_{t_l} = \left(0, \frac{C}{t_l}\right]$. For these intervals, they constructed a Harmonic-Fit type algorithm. There is one active bin for every interval $\mathcal{I}_j$, and all items from the interval $\mathcal{I}_j \cap (0, 1]$ are packed into it using a Next-Fit rule. Using a weighting function (which is an adaptation of the one defined for the classical Harmonic-Fit algorithm), they proved the following theorem.

Theorem 33 (Csirik and Woeginger [63]) *For every bin size $C \geq 1$, there exist on-line, bounded-space bin packing algorithms with APRs arbitrarily close to $\rho(C)$.*

For every bin size $C \geq 1$, the bound $\rho(C)$ cannot be beaten by an on-line bounded-space bin packing algorithm.

Azar and Regev [7] presented two on-line algorithms, each guaranteeing a worst-case performance of 5/3 for any number of bins. They also combined the algorithms to obtain an algorithm with a tight worst-case performance of $13/8 = 1.625$. This turns out to be an interesting result, as the best lower bound for any algorithm is equal to 4/3 for $m \geq 2$.

On-line bounded-space bin packing with limited repacking was considered by Epstein and Kleiman [75], who extended to this case the ideas of Galambos and Woeginger [95]. They defined a semi-on-line algorithm, *Resource Augmented* $\mathrm{REP}_3(C)$ ($\mathrm{RAR}_3(C)$), which is allowed to repack the elements within three active bins, and proved the following theorem.

Theorem 34 (Epstein and Kleiman [75]) *For every bin size $C \geq 1$, and for any on-line bounded-space bin packing algorithm A that allows repacking within k active bins, $R_A^{\infty}(C) \geq \rho(C)$. Algorithm* $\mathrm{RAR}_3(C)$ *has the best possible APR.*

Epstein and van Stee [80] extended the study to on-line algorithms, investigating both lower and upper bounds. For the upper bound, they started with an analysis of the harmonic type algorithms. Recall that by introducing two new interval endpoints, $\Delta > 1/2$ and $1 - \Delta$, one can combine one small element with an item in the interval $(1/2, \Delta]$. Also recall that in this case, only a fraction of the bins belonging to interval i can be used to reserve space for a (possibly later arriving) element from the interval $(1/2, \Delta]$. This fraction is denoted as α^i. (The values of α^is obviously depend on the number of intervals, and they are the crucial point of any such algorithm.) On this basis, they developed four algorithms, which are applied to the intervals $[1, 6/5)$, $[6/5, 4/3)$, $[4/3, 12/7)$, and $[12/7, 2)$, respectively.

An analytical solution was given only for the *Tiny Modified-Fit* algorithm, which was applied to interval $[12/7, 2)$. To examine the worst-case behavior of the other algorithms, they defined two weighting functions and constructed a linear program $P(f)$. By applying it to the cost function, they could use the method introduced by Seiden [163], thus obtaining upper bounds for the asymptotic behavior of the algorithms. The cost functions were not given explicitly, but program P was solved for many C values, and a diagram was given to show the upper bounds.

Concerning lower bounds, Epstein and van Stee [80] followed the argument laid down by van Vliet [171]. In this case too, an LP was generated for many values of C, and appropriate sublists were obtained through specialized greedy algorithms.

Chan et al. [37] further extended the study to dynamic bin packing. They considered the case where the items may depart at any time and where moving items among the open bins is disallowed. They first investigated Any-Fit algorithms, proving the following theorem.

Theorem 35 (Chan et al. [37]) *Consider the resource augmentation problem with bin capacity C, and let A be an Any-Fit algorithm. Then A can achieve 1-competitiveness if and only if $C = 2$.*

They also gave a new lower bound, as a function of C, for on-line algorithms. Let m be the largest integer such that $C - \frac{1}{m} < 1$, and define, for any positive integer $1 \le i \le m$,

$$\alpha(1) = m!(m-1)! \qquad \beta(i) = \sum_{j=1}^{i} \alpha(j) \qquad \alpha(i) = \frac{\beta(i-1)}{(m-i+2)(m-i+1)}.$$

Theorem 36 (Chan et al. [37]) *Let A be an on-line algorithm. Then*

$$R_{\mathrm{A}}^{\infty}(C) \ge \max\left\{\frac{\beta(m)}{m!(m-1)!}, \frac{2}{C}\right\}.$$

They also analyzed certain classical algorithms:

Theorem 37 (Chan et al. [37])

$$R_{\mathrm{FF}}^{\infty}(C) \le \min\left\{\frac{2C+1}{2C-1}, \frac{5-C}{2C-1}, \frac{C^2+3}{C(2C-1)}\right\},$$

$$R_{\mathrm{BF}}^{\infty}(C) = \frac{1}{C-1}$$

and

$$\min\left\{\frac{2C+2}{2C}, \frac{C^2-8C+20}{C(4C)}\right\} \le R_{\mathrm{WF}}^{\infty}(C) \le \frac{4}{C^2}. \tag{4}$$

Very recently, Boyar et al. [31] gave a complete APR analysis of algorithms WF and NF for the resource augmentation case. They showed that WF is strictly better than NF in the interval $1 < C < 2$ but has the same APR for all $C \ge 2$:

Theorem 38 (Boyar et al. [31]) *Let $t_C = \lfloor \frac{1}{C-1} \rfloor$. Then*

$$R_{\mathrm{WF}}^{\infty}(C) = \begin{cases} \frac{2C}{3C-2}, & \text{if } C \in [1,2] \\ \frac{1}{C-1}, & \text{if } C \in [2,\infty) \end{cases}$$

and

$$R_{\mathrm{NF}}^{\infty}(C) = \frac{2t_C^2 C - 2t_C^2 - 4t_C + 2 + 2Ct_C}{t_C^2 C + 2Ct_C - t_C^2 - 3t_C - 2 + 2C}.$$

6 Dual Versions

6.1 Minimizing the Bin Capacity

Suppose that the number of bins is fixed, and let the objective be a smallest C such that a given list L of items can be packed in m bins of capacity C. This dual of bin packing actually predates bin packing and is known as multiprocessor or parallel-machine scheduling. It is a central problem of scheduling theory and is surveyed in depth elsewhere (see, e.g., Lawler et al. [141] and Hall [114]), so only relevant results and research directions will be highlighted. The problem is referred to as $P||C_{\max}$ in scheduling notation.

Optimal and heuristic capacities normally differ by at most a maximum item size, so absolute worst-case ratios rather than asymptotic ones are considered. The first analysis of approximation algorithms was presented by Graham [108, 109], who studied the worst-case performance of various algorithms. For the on-line case, he investigated the Worst-Fit algorithm (better known as the *List Scheduling*, LS algorithm), which initially opens m empty bins and then iteratively packs the current item into the bin having the minimum current content (breaking ties in favor of the lowest bin index). Graham proved that $R_{\mathrm{LS}} = 2 - \frac{1}{m}$, a bound that for many years was thought to be best among on-line algorithms. Galambos and Woeginger were first to prove that the bound could be improved. Let $A(L, m)$ denote the bin capacity needed under A for m bins, and define

$$R_A(m) = \sup_L \left\{ \frac{A(L,m)}{OPT(L,m)} \right\} \quad \text{and} \quad R(m) = \inf_A R_A(m).$$

Theorem 39 (Galambos and Woeginger [94]) *There exists a sequence of nonnegative numbers $\varepsilon_1, \varepsilon_2, \ldots$, with $\varepsilon_m > 0$, $m \geq 4$ and $\varepsilon_m \to 0$ 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 the current position on the problem are as follows. Bartal et al. [19] presented an algorithm with a worst-case ratio $2 - \frac{1}{70} = 1.98571\ldots$ for all $m \geq 70$. Earlier, Faigle et al. [83] 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 40 (Chen et al. [41]) *For all $m = 80 + 8k$ with k a nonnegative integer, $R(m) \geq 1.83193$.*

Theorem 41 (Bartal et al. [20]) *For all $m \geq 3{,}454$, $R(m) \geq 1.8370$.*

For some time, the following result gave the best upper bound for $R(m)$ and was a generalization of the methods in [20].

Theorem 42 (Karger et al. [131]) $R(m) \leq 1.945$ *for all* m.

A tighter result was found by Albers [2], who proved the next two bounds.

Theorem 43 (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 44 (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 Chen et al. [41] 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?

The off-line case is now considered. By preordering the elements according to nonincreasing size and applying the LS algorithm, one obtains the so-called *Largest Processing Time* (LPT) algorithm for $P||C_{\max}$. Graham [109] proved that $R_{\mathrm{LPT}} = \frac{4}{3} - \frac{1}{3m}$. The *Multifit* algorithm by Coffman et al. [47] 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 $O(n \log n + k n \log m)$ time and has an absolute worst-case ratio satisfying

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

Friesen [87] improved the bound uniform in k to 1.2, but Yue [182] later settled the question by proving that this bound is $\frac{13}{11}$. Friesen and Langston [89] 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 [109]. 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

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

and that the bound is tight when m divides k. Algorithm Z_k implicitly defines an approximation scheme. By selecting $k = k_\varepsilon = \lceil \frac{m(1-\varepsilon)-1}{\varepsilon} \rceil$, one obtains $R_{Z_{k_\varepsilon}} \leq 1 + \varepsilon$. 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 [161], 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 $R_{A_\varepsilon} \leq 1 + \varepsilon$; hence it is a fully polynomial-time approximation scheme for any fixed m.

For m even moderately large, the approach of Hochbaum and Shmoys [118] offers 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 polynomial dependence would imply $\mathcal{P} = \mathcal{NP}$. Their algorithm is based on the binary search of an interval (like MF_k), but it uses the ε-dual approximation algorithm of Sect. 4.3 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 Sect. 4.3, one obtains a linear-time approximation scheme for the capacity minimization problem. (See also Hochbaum's [116] for a more extensive discussion of this interesting technique.) The Hochbaum and Shmoys result was generalized by Alon et al. [3], who gave conditions under which a number of scheduling problems on parallel machines admit a polynomial-time approximation scheme.

6.2 Maximizing the Number of Items Packed

This section considers another variant in which the number of bins is fixed; the objective is now to pack a maximum cardinality subset of a given list $L_{(n)}$. The problem arises in computing applications, when one wants 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.

The classical bin packing notation is extended in an obvious way by defining

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

so that the APR is defined by

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

and has values less than or equal to 1.

First consider off-line algorithms. The problem was first studied by Coffman et al. [48], 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 [48] that $\hat{R}^{\infty}_{\mathrm{FFI}}(m) = \frac{3}{4}$ for all m.

The *Iterated First-Fit Decreasing* (IFFD) algorithm was investigated by Coffman and Leung [45]. 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 $\frac{6}{7} \leq \hat{R}^{\infty}_{\mathrm{IFFD}}(m) \leq \frac{7}{8}$. The time complexity of IFFD is $O(n \log n + mn \log m)$, but a tight APR is not known.

Lower bounds on the APR are now considered. If an algorithm A packs items $a_1, \ldots, a_t$ from a list $L = (a_1, \ldots, a_n)$ so that $s_i > 1 - s(B_j)$ for any j and for any $i > t$ (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 45 (Coffman et al. [48]) *Let A be a prefix algorithm and let $k = \min_{1 \leq j \leq m}\{|B_j|\}$ be the least number of items packed into any bin. Then*

$$\hat{R}^{\infty}_{A}(m) \geq \frac{mk}{mk + m - 1}.$$

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

The case of divisible item sizes was investigated by Coffman et al. [51]. 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 first analyzed by Langston [140], who proved that if the bins are rearranged by nonincreasing size, then $\hat{R}^{\infty}_{\mathrm{FFI}}(m) = \frac{1}{2}$ and $\frac{2}{3} \leq \hat{R}^{\infty}_{\mathrm{IFFD}}(m) \leq \frac{8}{11}$ for all m. Friesen and Kuhl [88] gave a new efficient algorithm, which is a hybrid of two algorithms defined earlier: *First-Fit Decreasing* (FFD) and *Best-Two-Fit* (B2F). The hybrid is iterative: it attempts to pack smaller and smaller suffixes of its list of items until it succeeds in packing the entire suffix. During each attempt, the hybrid partitions the current list of items and packs one part by B2F and then the other part by FFD. They proved an APR of $\frac{3}{4}$ for this hybrid.

Coming to the on-line case, Azar et al. [9] (see also [8]) studied the behavior of so-called fair and unfair algorithms. When there is a fixed number of bins, an on-line algorithm is *fair* if it rejects an item only if it does not fit in any bin, while an *unfair* algorithm is allowed to discard an item even if it fits. They show that an unfair algorithm may get better performance compared with the performance achieved by the fair version. In particular, they show that an unfair variant of

First-Fit can pack approximately $2/3$ of the items for sequences for which an optimal off-line algorithm can pack all the items, while the standard (fair) First-Fit algorithm has an asymptotically tight performance arbitrarily close to $5/8$. Later, Epstein and Favrholdt [74] proved that the APR of any fair, deterministic algorithm is $\frac{1}{2} \leq R_A \leq \frac{2}{3}$ and that a class of algorithms including BF has an APR of $\frac{n}{2n-1}$.

6.3 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 [5]; Assmann's collaboration with Johnson, Kleitman and Leung (see [6]) 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 [6] that $\tilde{R}^{\infty}_{\mathrm{DNF}} = \frac{1}{2}$, where

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

Assmann et al. [6] 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 $c(B_j) + s_i \leq rT$. 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 $\tilde{R}^{\infty}_{\mathrm{DFF}[r]} = \frac{1}{2}$, 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 46 (Assmann et al. [6] and Csirik and Totik [61]) $\tilde{R}^{\infty}_{\mathrm{DFF}[r]} = \frac{1}{2}$ *and there is no on-line dual bin packing algorithm A for which* $\tilde{R}^{\infty}_{A} > \frac{1}{2}$.

Turning now to off-line algorithms, first consider the observation of Assmann et al. [6] that presorting does not help the adaptations of algorithms Next-Fit Decreasing and Next-Fit Increasing, for which $\tilde{R}^{\infty}_{\mathrm{NFD}} = \tilde{R}^{\infty}_{\mathrm{NFI}} = \frac{1}{2}$. 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 47 (Assmann et al. [6]) $\tilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \frac{2}{3}$ *if* $\frac{4}{3} < r < \frac{3}{2}$ *and* $\lim_{r\to 1} \tilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \lim_{r\to 2} \tilde{R}^{\infty}_{\mathrm{DFFD}[r]} = \frac{1}{2}$.

The *Iterated Lowest-Fit Decreasing* (ILFD) algorithm was also investigated in [6]. It is analogous to algorithm IFFD described in Sect. 6.2. 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 48 (Assmann et al. [6]) $\tilde{R}^{\infty}_{\mathrm{ILFD}} = \frac{3}{4}$.

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

Theorem 49 (Coffman et al. [51]) *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.*

Concerning approximation schemes for the dual bin packing problem, first observe that the approach used by Fernandez de la Vega and Lueker [84] and Karmarkar and Karp [132] (see Sect. 4.3), which eliminates the effect of small items on worst-case behavior, does not appear applicable, as in the dual problem, small items can play an important role in filling small gaps. Csirik et al. [65] have given a PTAS for this problem. Later, an FPTAS was presented by Jansen and Solis-Oba [124].

7 Variations on Item Packing

7.1 Dynamic Bin Packing

The idea of *Dynamic Bin Packing* (DBP) was introduced by Coffman et al. [49]. In the case of dynamic packing, deletion of some elements is allowed at each step, and $A(L)$ is defined as the maximum number of bins used during the packing.

Consider the generalization in which each item a_i is characterized by a triple (s_i, b_i, d_i), where b_i is the start (arrival) time, d_i (with $d_i > b_i$) is the departure time, and, as usual, s_i is the size. Item a_i remains in the packing during the time interval $[b_i, d_i)$. Assume that $b_i \leq b_j$ 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

$$A(L) = \max_{0 \le t \le b_n} A(L,t).$$

Note that, in this case, an open 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 [42]), 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 et al. [49]).

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 et al. [49] 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 $s_i \ge \frac{1}{2}$ are handled separately, in an attempt to pair each of them with smaller elements.

Theorem 50 (Coffman et al. [49]) *If* $\frac{1}{k+1} < \max\{s_i\} \le \frac{1}{k}$, *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 $k = 2$, one gets $\frac{7}{4} \le R_{\mathrm{FF}}^\infty(2) \le 1.78768\ldots$. 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 51 (Coffman et al. [49]) *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* $R_A(2) \ge 1.666\ldots$.

By restricting attention to strongly divisible instances (see Sect. 8.1), then matters improve, as expected. The gap between the general lower bound and that for FF disappears. Indeed,

Theorem 52 (Coffman et al. [51]) *If (L, C) is strongly divisible and L is such that $s_1 > s_2 > \cdots > s_k$ $(k \geq 2)$, then the APR of the dynamic version of FF is*

$$R_{\mathrm{FF}}^{\infty} = \prod_{i=1}^{k-1} \left(1 + \frac{s_i}{C} - \frac{s_{i+1}}{C}\right).$$

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

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

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

Other algorithms were developed, which, in addition to deletion, allow at least one of the following capabilities: repacking, look ahead, and preordering. For those algorithms which may use all of the above operations, Ivkovic and Lloyd [122] (see also [120]) introduced the *fully dynamic bin packing* (FDBP) term and presented an algorithm, MMP, which is $\frac{5}{4}$-competitive.

7.2 Selfish Bin Packing

Consider the case in which the items are controlled by selfish agents, and the cost of a bin is split among the agents in proportion to the fraction of the occupied bin space their items require. Namely, if an agent packs its item a of size $s(a)$ into bin B, then its cost is $\frac{s(a)}{c(B)}$, where $c(B)$ is the content of the bin. In other words, the selfish agents would like their items to be packed in a bin that is as full as possible. Hence, given a feasible solution, the agents are interested in bins B' such that $s(a) + c(B') \leq 1$ and $\frac{s(a)}{s(a)+c(B')} < \frac{s(a)}{c(B)}$. If such a bin exists, then a unilaterally migrates from B to B', as in this new feasible packing its cost decreases. When no such bin exists for any agent, a stable state has been reached. The *social optimum* is to minimize the number of bins used, which is the aim of the standard BP problem.

This problem corresponds to a noncooperative game, and the stable state is a pure *Nash Equilibrium* (NE). There is a *Strong Nash Equilibrium* (SNE) if there is no subset of agents, which can profit by jointly moving their items to different bins so that all agents in the subset obtain a strictly smaller cost.

More precisely, a noncooperative strategic game is a tuple $G = \langle N, (S_i)_{i \in N}, (c_i)_{i \in N} \rangle$ where N is a finite set of players. Each player has a finite set S_i of strategies and a cost function c_i, and each player has complete information about the strategy of all players. Players choose their strategies independently

of each other. A combination of strategies chosen by the players is denoted by $s = (x_j)_{j \in N} \in \times_{j \in N} S_j$ and is called a *strategy profile*. The *social cost* of a game is an objective function $SC(s) : X \longrightarrow \mathbb{R}$, that gives the cost of an outcome of the game for a strategy profile $s \in X$, where X denotes the set of all strategy profiles. The *social optimum* of the game is then $OPT(G) = \min_{s \in X} SC(s)$.

The quality of an NE is measured by the *Price of Anarchy* (PoA) and the *Price of Stability* (PoS), which are defined as follows:

$$PoA = \sup_{s \in NE(G)} \frac{SC(s)}{OPT(G)} \qquad PoS = \inf_{s \in NE(G)} \frac{SC(s)}{OPT(G)}.$$

By using the SNE, one gets the *Strong Price of Anarchy* (SPoA) and the *Strong Price of Stability* (SPoS):

$$SPoA = \sup_{s \in SNE(G)} \frac{SC(s)}{OPT(G)} \qquad SPoS = \inf_{s \in SNE(G)} \frac{SC(s)}{OPT(G)}.$$

For the bin packing problem asymptotic measures are used:

$$PoA(BP) = \limsup_{OPT(G) \to \infty} \sup_{G \in BP} PoA(G) \qquad PoS(BP) = \limsup_{OPT(G) \to \infty} \inf_{G \in BP} PoS(G)$$

and

$$SPoA(BP) = \limsup_{OPT(G) \to \infty} \sup_{G \in BP} SPoA(G) \quad SPoS(BP) = \limsup_{OPT(G) \to \infty} \inf_{G \in BP} SPoS(G).$$

To stress the connection with game theory the problem is called the *bin packing game*. The first publication which discussed this problem from a game theoretical perspective was presented by Biló (see [25]). He showed that starting from an arbitrary feasible packing, one can always reach an NE using an exponential number of migration steps. This result implies that, for every instance of the bin packing game, there exists an optimal packing which provides a NE, that is, $PoS(BP) = 1$. He also proved that computing the best NE is $\mathcal{NP} - hard$. Yu and Zhang [181] constructed an $O(n^4)$-time Recursive First-Fit Decreasing (RFFD) algorithm for finding an NE. They also investigated the $PoA(BP)$ and proved the following theorem:

Theorem 53 (Yu and Zhang [181]) *For the bin packing game,*

$$1.6416 \approx \sum_{l=1}^{\infty} \frac{1}{2^{\frac{l(l-1)}{2}}} \leq PoA(BP) \leq \frac{41 + \sqrt{145}}{32} \approx 1.6575.$$

Epstein and Kleiman [76] improved the upper bound to 1.64286 and conjectured that the best upper bound is equal to the lower bound.

Bin packing algorithms look for a good global solution; hence, in general, they do not produce a Nash equilibrium. An exception is the *Subset Sum* (SS) algorithm by Caprara and Pferschy [34], which produces an NE by repeatedly solving a knapsack problem to pack a bin as full as possible. Unfortunately, the knapsack problem is not polynomial unless $\mathcal{P} = \mathcal{NP}$. Indeed, Epstein and Kleiman [76] on the one hand proved equality among the $SPoS(BP)$ and the $SPoA(BP)$ and the asymptotic behavior of the Subset Sum algorithm and on the other hand showed that finding an SNE is $\mathcal{NP}$-hard.

Caprara and Pferschy proved that $1.6062 \leq R_{SS}^{\infty} \leq 1.6210$. So, by comparing $PoA(BP)$ and $SPoA(BP)$, it appears that they have similar values. Therefore, the efficiency in the bin packing game is only due to selfishness, and coalitions do not help. In addition, $PoS(BP)$ is significantly better than $SPoS(BP)$, which in turn equals $SPoA(BP)$, implying that in the bin packing game the best equilibrium is less efficient if coalition among agents is allowed.

7.3 Bin Packing with Rejection

This variation is motivated by the following problem, arising in file management. Suppose that files are used by a local system. Either a file is downloaded in advance or the system downloads it only when it is actually requested. The first option needs space on a local server, and it has a local transmission cost, while a program uses the file. The second option has a communication cost that one incurs, while the system downloads the file from an external server. An algorithm which manages the local file system has the choice of either packing a file (item) on the local storage device or (according to the costs) downloading it when needed. The problem can be modeled as the following bin packing problem. A pair (s_i, r_i) is associated with each item i $(i = 1, \ldots, n)$, where s_i is the size and r_i is the *rejection cost* of the item. An algorithm has the choice of either packing an element into the bins or rejecting it. The objective is to minimize the number of bins used, plus the sum of all rejection costs. (If all rejection costs are larger than 1 then every item will be packed, and a standard problem arises.) This model was introduced by Dósa and He [69], who considered both off-line and on-line algorithms, and investigated their absolute and asymptotic behavior.

For off-line algorithms, Dósa and He gave an algorithm with absolute worst-case ratio 2. They also observed that the lower bound for the standard problem remains valid, that is, there is no algorithm with absolute worst-case ratio better than $\frac{3}{2}$, unless $\mathcal{P} = \mathcal{NP}$. The lower bound was reached by Epstein [73] (see also [70]). Based on the ratio r_i/s_i, Dósa and He classified the elements into subclasses and used a sophisticated greedy-type algorithm, RFF4, for which they proved that $R_{RFF4}^{\infty} \leq \frac{3}{2}$.

For the on-line case, they investigated an algorithm, called RFF1, and proved that the absolute worst-case ratio of the best on-line algorithm lies in the interval $(2.343, 2.618)$. For the asymptotic competitive ratio, they introduced another algorithm, RFF2, and proved the following:

Theorem 54 (Dósa and He [69]) *For any positive integer* $m \geq 2$, $R^{\infty}_{RFF2}(m) \leq \frac{7m-3}{4m-2}$. *Hence there exists an on-line algorithm with an APR arbitrarily close to* $7/4 = 1.75$.

Not surprisingly, this result was beaten by a variant of the HF algorithm proposed by Epstein [73]. Her RejH$_k$ algorithm classifies the items as HF. Then thresholds are defined for the rejection choice as follows. Suppose that the next item is $a_i \in I_j$, where $I_j = (\frac{1}{j+1}, \frac{1}{j}]$ for some $1 \leq j \leq k$. Item a_i will be rejected if either $j = k$ and $r_i \leq \frac{k}{k-1}s_i$ or $j < k$ and $r_i \leq \frac{1}{j}$. Using the weighting function technique, Epstein proved the following theorem.

Theorem 55 (Epstein [73]) *The APR of RejH$_k$ tends to* $h_{\infty}(1) = 1.6901$ *if* $k \to \infty$. *No algorithm with a constant number of open bins can have a smaller APR.*

For the unbounded-space case, the latter result was improved by a *Rejective Modified Harmonic* (MHR) algorithm, which is an adaptation of the Modified Harmonic analyzed by Ramanan et al. [159]:

Theorem 56 (Epstein [73]) *The APR of* MHR *is at most* $\frac{538}{333}$.

Starting from the classical results of Fernandez de la Vega and Lueker [84] and Hochbaum and Shmoys [118], Epstein [73] also proposed an APTAS for bin packing with rejection, having time complexity $O(n^{O((\varepsilon^{-4})^{\varepsilon^{-1}})})$. Using some results on the multiple knapsack problem (see Chekuri and Khanna [40] and Kellerer [134]) a faster, $O(n^{O(\varepsilon^{-2})})$ time, approximation scheme was constructed by Bein et al. [21].

7.4 Item Fragmentation

Menakerman and Rom [155] investigated a variant of the bin packing problem in which items may be subdivided into pieces of smaller size, called *fragments*. Fragmenting an item is associated with a cost, which makes the problem $\mathcal{NP}$-hard. They studied two possible cost functions. In the first variant, called *Bin Packing with Size-Increasing Fragmentation* (BP-SIF), whenever an item is fragmented, overhead units are added to the size of every fragment. In the second variant, called *Bin Packing with Size-Preserving Fragmentation* (BP-SPF), each item has a size and a cost, and, whenever it is fragmented, one overhead unit is added to its cost without changing its total size. (Actually, problem BP-SIF was first introduced by Mandal et al. [150] who showed that it is $\mathcal{NP}$-hard.)

It is supposed that the item sizes are positive integers and the bins are all of size C. It is obvious that a good algorithm should try to perform the minimum number of fragmentations. Menakerman and Rom developed algorithms that do not fragment items unnecessarily. More precisely, an algorithm is said to prevent unnecessarily fragmentation if it follows the next two rules:

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

For the BP-SIF problem, it is proved that for any algorithm A that prevents unnecessary fragmentation, $R_A^\infty \leq \frac{C}{C-2}$ for every $C > 2$. For the obvious generalization of the Next-Fit algorithm, called *Next-Fit with item fragmentation* (NF_f), a $\frac{C}{C-2}$ bound for all $C \geq 6$ was proved. For a more sophisticated algorithm, which is actually an iterated version of First-Fit Decreasing, they proved that the APR is not greater than $\frac{C}{C-1}$ when $C \leq 15$, while it is between $\frac{C}{C-1}$ and $\frac{C}{C-2}$ when $C \geq 16$.

As to the BP-SPF problem, the performance of an algorithm for a given list L, $OH_A(L)$ is measured by its overhead, that is, by the difference between the cost of the solution produced by the algorithm and the cost of an optimal solution. The worst-case performance of an algorithm is

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

Menakerman and Rom [155] proved that for any algorithm A that prevents unnecessary fragmentation, $OH_A^m \leq m - 1$ and that an appropriate modification of Next-Fit reaches this bound. First-Fit Decreasing performs better when the bin size is small.

Naaman and Rom [158] considered two generalizations of the BP-SPF. The first one is quite simple: r overhead units are added to the size of every fragment. This changes the performance of Next-Fit to $\frac{C}{C-2r}$. The other generalization is more challenging: a set B of m bins is given, whose (integer) bin sizes may be different. Denote by $\overline{C} = \frac{1}{m}\sum_{j=1}^{m} s(B_j)$ the average bin size. In this case, the asymptotic worst-case bound for Next-Fit is $\frac{\overline{C}}{\overline{C}-2r}$ for every $\overline{C} > 4r$.

Shachnai et al. [168] studied slightly modified versions of BP-SIF and BP-SPF. For the BP-SIF, a header of fixed size Δ is attached to each (whole or fragmented) item, that is, the size required for packing a_i is $s_i + \Delta$. When an item is fragmented, each fragment gets the header. For the BP-SPF, the number of possible splits is bounded by a given value. They presented a dual PTAS and an APTAS for each of the problems. The dual PTASs pack all the items in $OPT(L)$ bins of size $(1 + \varepsilon)$, while the APTASs use at most $(1 + \varepsilon)OPT(L) + 1$ bins. All of these schemes have running times that are polynomial in n and exponential in $1/\varepsilon$. They also showed that both problems admit a dual AFPTAS, which packs the items in $OPT(L) + O(1/\varepsilon^2)$ bins of size $1 + \varepsilon$. Shachnai and Yehezkely [167] developed fast AFPTASs for both problems. Their schemes pack the items in $(1 + \varepsilon)OPT(L) + O(\varepsilon^{-1} \log \varepsilon^{-1})$ bins in a time that is linear in n and polynomial in $1/\varepsilon$.

Epstein and van Stee [81] investigated a version where the items may be split without extra cost, but each bin may contain at most k (parts of) items, for a given constant k. They provided a polynomial-time approximation scheme and a dual approximation scheme for this problem.

7.5 Fragile Objects

Bansal et al. [17] (see also [16]) investigated two interrelated optimization problems: bin packing with fragile objects and frequency allocation in cellular networks (which is a special case of the former problem). In this problem, each item has two attributes: size and fragility. The goal is to pack the items so that the sum of the item sizes in each bin is not greater than the fragility of any item in the bin. They provided a 2-approximation algorithm for the problem of minimizing the number of bins and proved a lower bound of $\frac{3}{2}$ on the asymptotical approximation ratio. They also considered the approximation with respect to fragility and gave a 2-approximation algorithm.

Chan et al. [35] considered the on-line version of the problem and proved that the asymptotic competitive ratio of any on-line algorithm is at least 2. They also considered the case where the ratio between maximum and minimum fragility is bounded by a value k, showing that the APR of an Any-Fit algorithm (including First-Fit and Best-Fit) is at least k. For the case where k is bounded by a constant, they developed a class of on-line algorithms which achieves an APR of $\frac{1}{4} + 3\frac{r}{2}$ for any $r > 1$.

7.6 Packing Sets and Graphs

Let the bin capacity C be a positive integer and suppose that the items are sets of elements drawn from some universal set. A collection of items can fit into a bin if and only if their union has at most C elements. As usual, the object is to pack the items in the smallest number of bins. Good approximation algorithms have yet to be analyzed for this problem; adaptations of the classical bin packing approximation algorithms do not have finite APRs. One observes immediately that while classical algorithms prioritize items based on cardinality and position in sequence, a good algorithm for set packing will have to consider the intersections among a given set of items, an orthogonal basis for prioritizing the items. It is thought that approximation algorithms with a finite APR for this problem are unlikely to exist, but a rigorous treatment of this question has yet to appear.

Dror (Dror, Private communication) points out that scheduling setups in a manufacturing process is an application of set packing. As an example, he describes a machine that produces many types of circuit boards, each requiring a given set of components. The machine has a magazine (bin) that holds at most C components. At any time, only the circuit boards with their 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 setups.

Khanna (Khanna, Private communication) mentions that, in connection with studies of multimedia communications, 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.

Katona [133] investigated a special case of graph packing. A graph is called *p-polyp* if it consists of p simple paths of the same length and sharing a common end vertex. *Polyp Packing* is the following generalization of bin packing: pack a set of paths of different lengths into a minimum number of edge disjoint polyps. He proved that the problem is $\mathcal{NP}$-hard and gave bounds on the performance of a modification of First-Fit.

8 Additional Conditions

8.1 Item-Size Restrictions

Perhaps the simplest (and most practical) restriction is simply to 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 studies on approximation schemes (see [84,132]) and was considered in its own right by Blazewicz and Ecker [26]. Let the i-th 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 y_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:

$$\min \sum_{i=1}^{N} y_i$$

$$\sum_{i=1}^{N} p_{ij} y_i \geq n_j \qquad (j = 1, \ldots, k)$$

$$y_i \geq 0 \text{ and integer } (i = 1, \ldots, N).$$

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 [144]), which is a constant independent of the number of items. The optimal overall packing can then be constructed in linear time.

If the exact solution is not needed, then one can use the classical approach of Gilmore and Gomory [105, 106]) and compute LP relaxations (see Sect. 4.3); this method gives solutions that are at most k off the optimal in significantly less time.

Gutin et al. [113] investigated an on-line problem in which one only has two different element sizes. They gave a $\frac{4}{3}$ lower bound for this problem and provided an asymptotically optimal algorithm for it. Epstein and Levin [77] focused on the parametric case, where both item sizes are bounded from above by $\frac{1}{k}$ for some natural number $k \geq 1$.

Next consider the classical problem restricted to lists L of items whose sizes form a *divisible sequence*, that is, the distinct sizes $s_1 > s_2 > \ldots$ taken on by the items are such that s_{i+1} divides s_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_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 $\mathcal{NP}$-hard bin packing problems generate significantly better packings when items satisfy divisibility constraints. In some cases, 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 57 (Coffman et al. [51]) *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 58 (Coffman et al. [51]) *If (L, C) is weakly divisible, then an FFD packing is always optimal.*

For strongly divisible instances, optimal performance can also be obtained without sorting the items.

Theorem 59 (Coffman et al. [51]) *If (L, C) is strongly divisible, then an FF packing is always optimal.*

In the *Unit-Fractions Bin Packing Problem*, n items are given, and the size of each item i is $\frac{1}{w_i}$, with w_i a positive integer ($1 \leq i \leq n$). This problem was initially studied by Bar-Noy et al. [18]. Let $H = \lceil \sum_{i=1}^{n} \frac{1}{w_i} \rceil$ be a lower bound on the required number of bins. They presented an off-line algorithm which uses at most $H + 1$ bins. For the on-line version, they analyzed an algorithm which uses $H + O(\sqrt{H})$ bins. It is also proved that any on-line algorithm for the unit-fractions bin packing problem uses at least $H + \Omega(\ln H)$ bins.

Dynamic bin packing problems with unit-fractions items were considered by Chan et al. [36]. They investigated the family of Any-Fit algorithms and proved the following results:

$$R_{\mathrm{BF}}^{\infty} = R_{\mathrm{WF}}^{\infty} = 3$$

$$2.45 \leq R_{\mathrm{FF}}^{\infty} \leq 2.4985.$$

In addition they proved the following lower bound result:

Theorem 60 (Chan et al. [36]) *For a dynamic bin packing problem with unit-fractions items, there is no on-line algorithm A with* $R_A^{\infty} < 2.428$.

It would be interesting to examine how approximation algorithms behave with other special size sequences (e.g., Fibonacci numbers), as mentioned by Chandra et al. [39] in the context of memory allocation.

8.2 Cardinality Constrained Problems

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. If the maximum number of items in a bin is bounded by p, then the problem is called the *p-Cardinality Constrained Bin Packing Problem.* Observe which is the difficulty when one tries to construct a good algorithm for this problem. Without the cardinality constraint, a huge number of small elements slightly affect the number of occupied bins: either they can be packed together into few bins, or they can be packed into bins which are "almost full." In the considered case, having packed many small elements results in almost empty bins, because of the cardinality constraint. Therefore, when a large item arrives, it has to be packed into a new bin even if it would fit into an already open bin.

In the context of the above applications, Krause et al. [139] 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

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

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.

On-line algorithms were also studied by Babel et al. [11] (see also [10]), who investigated four algorithms, denoted as $A_1, \ldots, A_4$.

- Algorithm A_1 operates with the BF strategy while packing the subsequent element. Different classes of blocked bins are defined according to the number of items in a bin. The algorithm tries to pack the element first into all blocked bins that satisfy a threshold condition, then into all formerly blocked and currently unblocked bins, and finally into the remaining bins. It is proved that for every $p \geq 3$,

$$R(p) = 2 + \frac{p + k_p(k_p - 3)}{(p - k_p + 1)k_p}, \qquad \text{where} \qquad k_p = \left[\frac{-p + \sqrt{p^3 - 2p}}{p - 2}\right],$$

  so $\lim_{p\to\infty} A_1 = 2$.
- In algorithm A_2, a bin is closed if $c(B) \geq \frac{1}{2}$ and it contains at least $\frac{p}{2}$ items. Furthermore, a pair of bins, B_1 and B_2, is also closed if $c(B_1) + c(B_2) \geq 1$ and $|B1| + |B2| \geq k$. The open bins are subdivided into three classes, and the algorithm tries to pack the current item, in turn, into bins of such classes. If the item does not fit in any of these bins, a new bin is open. It is proved that for every $p \geq 3$, algorithm A_2 has an APR of 2.
- Algorithm A_3 deals with the case $p = 2$. The basic idea of the algorithm is that, when a small item is packed, a balance should be kept between the number of bins with only one small item and the number of bins which have two such items. It is proved that $R_{A_3} = 1 + \frac{1}{\sqrt{5}}$.
- For the 3-cardinality problem, a Harmonic-type algorithm is defined, with four bin classes. The score intervals are as follows: $I_1 = (0; \frac{1}{3}]$, $I_2 = (\frac{1}{3}, \frac{1}{2}]$, $I_3 = (\frac{1}{2}, \frac{2}{3}]$, and $I_4 = (\frac{2}{3}, 1]$. The resulting Algorithm A_4 is a much more complicated version of the classical HF algorithm and has an asymptotic competitive ratio of 1.8.

  Two lower bounds are also given:

Theorem 61 (Babel et al. [11]) *There is no* 2*-cardinality on-line bin packing algorithm A with APR* $R_A < \sqrt{2}$.

Theorem 62 (Babel et al. [11]) *There is no* 3*-cardinality on-line bin packing algorithm A with APR* $R_A < \frac{3}{2}$.

For unbounded-space algorithms, some further cases were investigated by Epstein [71]. She considered a Harmonic-type algorithm that defines five

subintervals and packs the items into bins with a more sophisticated procedure. Her algorithm for the case $p = 3$ improves the APR of [11] to $\frac{7}{4} = 1.75$. For the cases $p = 4$, $p = 5$ and $p = 6$, she refined the basic algorithm, designing three algorithms with APRs $\frac{71}{38} = 1.86842$, $\frac{771}{398} = 1.93719$, and $\frac{284}{144} = 1.99306$. She proved that an algorithm with an APR strictly better than 2 cannot be bounded space (unless $p \leq 3$).

For the bounded-space case, Epstein defined the *Cardinality Constrained Harmonic-p* (CCHp) algorithm. She considered the Sylvester sequence, defined the sum

$$S_p = \sum_{i=1}^{p} \left\{ \frac{1}{t_i - 1}, \frac{1}{p} \right\},$$

and proved the following theorems:

Theorem 63 (Epstein [71]) *The value of S_p is a strictly increasing function of $p \geq 2$ such that $\frac{3}{2} \leq S_p < h_\infty(1) + 1$ and $lim_{p\to\infty} S_p = h_\infty(1) + 1 \approx 2.69103$.*

Theorem 64 (Epstein [71]) *For every $p \geq 2$, the APR of CCHp is S_p, and no on-line algorithm which uses bounded space can have a better competitive ratio.*

In addition, Epstein extended her results to the variable-sized and the resource-augmentation cases (see Sects. 5.1 and 5.2).

An off-line algorithm (*Largest Memory First*, LMF), based on an adaptation of FFD, was studied by Krause et al. [139]. They proved that $R^{\infty}_{\mathrm{LMF}} = 2 - \frac{2}{p}$ if $p \geq 2$. Here 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 $q := \lceil \max(\frac{n}{p}, \sum_{i=1}^{n} s_i) \rceil$ 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 $\mathrm{IWFD}(L) = OPT(L)$ for any list L.
- If in the final schedule no capacity constraint is operative, that is, no open bin has residual capacity smaller than the size of the smallest item, then $\mathrm{IWFD}(L) = OPT(L)$.
- If in the final schedule no cardinality constraint is operative, that is, no open bin containing p items has residual capacity at least equal to the size of the smallest item, then $R^{\infty}_{\mathrm{IWFD}} < \frac{4}{3}$.

For the general case, where the final packing contains both bins for which the capacity constraint is operative and bins for which it is not, the constants are

significantly worse but still small: $\frac{4}{3} \leq R^{\infty}_{\mathrm{IWFD}} \leq 2$. The authors conjecture that $R^{\infty}_{\mathrm{IWFD}} = \frac{4}{3}$.

For a long time, it was an open question whether there exists a heuristic algorithm with an APR better than 2. Kellerer and Pferschy [135] proposed a new algorithm based on a method that proved to be fruitful in scheduling theory, namely, one that performs a binary search on the possible $OPT(L)$ values. At each search step, they run a procedure that tries to pack all items into a number of bins equal to $\frac{3}{2}$ times the current search value. The procedure basically tries to solve a multiprocessor scheduling problem using the classical LPT rule with a cardinality constraint. The time complexity of this *Binary Search* (BS) algorithm is $O(n \log^2 n)$, and the improved bound is $\frac{4}{3} \leq R^{\infty}_{\mathrm{BS}} \leq \frac{3}{2}$. Although the gap has been reduced, the exact bound is still not known.

8.3 Class Constrained Bin Packing

In the *Class Constrained Bin Packing Problem* (CLCP), each item a_i has two parameters: the size s_i and the color c_i, where $s_i \in (0, 1]$ and a color is represented by a positive integer, that is, $c_i \in [1, q]$. A further parameter $Q \in \mathbb{N}$ is associated with the problem as a bound for the number of the different colors within a bin. Different items may have the same color, and the items with the same color are classified into *color classes.* The objective is to pack the items into the minimum number of bins $\{B_1, B_2, \ldots, B_m\}$, such that $\sum_{a_j \in B_i} s_j \leq 1$ for $i = 1, \ldots, m$ and each such bin has items from at most Q color classes. Clearly, if $q = n$, then one gets the p-cardinality constrained problem (see Sect. 8.2), and if $q \leq Q$, then a classical bin packing problem arises. Shachnai and Tamir [165] proved that the problem is $\mathcal{NP}$-hard even in the case of identical item sizes.

For the case of on-line bounded-space algorithms, Xavier and Miyazawa [176] presented inapproximability results. More precisely, they proved that if $s_i \in [K_1, K_2]$, with $0 < K_1 < K_2 \leq 1$, for all $i = 1, \ldots, n$, then any bounded-space on-line algorithm has an APR of $\Omega(\frac{1}{QK_1})$.

Shachnai and Tamir [166] studied unbounded-space algorithms for the case of identically sized items. Suppose that $s_i \in (\frac{1}{r+1}, \frac{1}{r}]$ for each a_i: they denoted by $S_{Q,r}(k, h)$ the set of those lists which have $kQ < q \leq (k+1)Q$ and $hr < n \leq (h+1)n$, where n is the number of elements in the list. They proved that for any deterministic algorithm A,

$$R_A \geq 1 + \frac{k + 1 - \lceil \frac{kQ+1}{r} \rceil}{h+1},$$

and that the lower bound can be reached by the FF algorithm.

Sachnai and Tamir also introduced the notion of *Color-Set algorithm* (CS). This algorithm partitions the colors into $\lceil q/Q \rceil$ color sets and packs the items of each color set in a greedy way. Note that only one bin is open for each color-set at a time.

They proved that $R_{CS} < 2$. The set of Any-Fit algorithms was also investigated, and it was shown that $R_A \leq \min(r/Q, Q+1)$ for any such algorithm. They further proved that the absolute worst-case ratio of the Next-Fit algorithm is $R_{\mathrm{NF}} = r/Q$.

The idea of CS was used to develop new algorithms. Color classes are divided *on-line* into sets of Q colors each, in order of appearance. Each open bin is assigned one color set. As a new item arrives, the algorithm tries to pack it into a bin having the color of the item. If the FF rule is used, then one gets the *Color-Set First-Fit* (CSFF) algorithm, while the NF rule gives the *Color-Set Next-Fit* (CSNF) algorithm. For the case of identical item sizes considered in [165], the authors proved that the competitive ratio of both CSFF and CSNF is at most 2.

The case $Q = 2$ was exhaustively studied by Epstein et al. [82], who proved that the upper bound for identical items is tight for algorithms FF and CSFF. Furthermore, they showed that, for arbitrary item sizes, the competitive ratio is at least $9/4$. A 1.5652 lower bound was also given for any on-line algorithm with arbitrary item sizes, thus showing that, even if $Q = 2$, the on-line algorithms for CLCP behave worse than those for the classical bin packing. For what concerns upper bounds, they proved that for arbitrary values of Q, $R_{CSFF}\infty \leq 3 - \frac{1}{Q}$.

Xavier and Miyazawa [176] investigated the FF algorithm and proved that $R^{\infty}_{\mathrm{FF}} \in [2.7, 3]$. They defined algorithm $\mathcal{A}_Q$, which subdivides the items, in a harmonic way, into three classes, and packs the next item, according to its size, into the class it belongs to. The algorithm also considers the color constraint Q. It was proved that $R^{\infty}_{\mathcal{A}_Q} \in (2.666, 2.75]$. Epstein et al. [82] developed an algorithm which has a competitive ratio of 2.65492, valid for every Q.

For fixed values of q, there are approximability results for CLCP. Note that, if q is fixed, then Q is also constant since $Q \leq q$. For equal item sizes, Shachnai and Tamir [165] constructed a dual PTAS whose time complexity is polynomial in n. The scheme uses the optimal number of bins, but the bin capacity is $1 + \varepsilon$. Xavier and Miyazawa [176] developed an APTAS. They also gave an AFPTAS with time complexity $O(\frac{n}{\varepsilon^2})$. Epstein et al. [82] proved that if q is considered as a parameter, then there is no APTAS for CLCP for any value of Q. They also gave an AFPTAS for the case of constant q, with time complexity $O(\frac{n}{\varepsilon^2})$.

8.4 LIB Constraints

In certain applications, the positions of the items within a bin are restricted: a larger item cannot be put on top of a smaller one. This leads to the variant with the LIB (Larger Item on the Bottom) constraint. The problem was first discussed by Finlay and Manyem [85] and Manyem et al. [151]. They studied the Next-Fit algorithm and proved that it cannot achieve a finite APR. For the First-Fit algorithm, they showed that $2 \leq R^{\infty}_{\mathrm{FF}} \leq 3$. They also analyzed a variant of Harmonic-Fit (HARM) which partitions the items into classes following the classical rule and packs them by applying to each class of bins the FF rule (instead of NF, which performs

very badly). They showed that $2 \leq R^{\infty}_{\mathrm{HARM}}$. The parametric case of HARM was considered in Epstein [72], who proved the following result:

Theorem 65 (Epstein [72]) *Let $r > 0$ be a positive integer, and suppose that $\frac{1}{r+1} < \max\{s_i\} \leq \frac{1}{r}$. Let k denote the number of bin classes. Then $R_{\mathrm{HARM}}(r) = \Theta(k - r + 1)$ for any fixed value of $k < \infty$.*

This result has an interesting consequence: contrary to the standard problem, for which the performance of the Harmonic-Fit algorithm improves when the number of bin classes is increased, in this case, the competitive ratio worsens, and algorithm HARM has an unbounded ratio as $k \to \infty$. Unfortunately, neither BF nor WF nor AWF behave better than HARM, that is, their APRs are also unbounded. In the same paper, Epstein analyzed the FF algorithm, proving that $R_{\mathrm{FF}}(1) = 2.5$ and $R_{\mathrm{FF}}(r) = 2 + \frac{1}{r}$ if $r \geq 2$. Very recently, her result was improved by Dósa, Tuza, and Ye (Dósa et al., 2010, Private communication), who proved that $R_{\mathrm{FF}}(1) = 2 + \frac{1}{6}$ and $R_{\mathrm{FF}}(r) = 2 + \frac{1}{r(r+2)}$ if $r \geq 2$.

As to lower bounds, very little is known. Clearly, the FFD algorithm automatically fulfills the LIB constraint, so $\frac{11}{9}$ is a lower bound for any semi-on-line algorithm which packs preordered elements. The only nontrivial lower bound was obtained by Epstein [72]:

Theorem 66 (Epstein [72]) *The competitive ratio of any on-line algorithm for bin packing with LIB constraints is at least* 2. *The result is valid for the parametric case as well, for any value of $r \geq 1$.*

8.5 Bin Packing with Conflicts

In this problem, a set of items $L = a_1, a_2, \ldots, a_n$ with sizes $s_1, s_2, \ldots, s_n$ and a conflict graph $G(L, E)$ are given. The objective is to find the minimum number of independent sets such that the sum of the sizes within each set is at most one. The problem is a generalization of both the classical bin packing problem (when $E = \emptyset$) and the graph coloring problem (when $s_i = 0$ for all i). In the on-line version of the problem, when a new item arrives, one gets its size and the edges of the conflict graph which connect it to previous items. Instead of examining the APR, most investigations focused on the absolute worst-case ratio AR, since the conflict graph that proves the worst case is valid both for small and large instances. Therefore, the APR may not be better than the AR.

It is known that graph coloring is hard to approximate for general graphs (see, e.g., Lovász et al. [149]); the problem at hand has been studied for specific graphs. The early studies on the *bin packing with conflicts* (BPC) dealt with *perfect graphs*, since the coloring problem is polynomially solvable for them. Jansen and Öhring [123] gave an algorithm with an APR of 2.7. Epstein and Levin [78] improved this result with an algorithm having the performance guarantee $h_{\infty}(1) + 1 = 2.690103$.

Jansen and Öhring also proposed an algorithm that uses the so-called *Precoloring Extension Problem* (PEP) as a preprocessing stage to get a partition of the conflict graph. In the PEP, one is given an undirected graph $G = (V, E)$ and k distinct vertices $v_1, v_2, \ldots, v_k$. The aim is to find a minimum coloring f of G such that $f(v_i) = i$ for $i = 1, 2, \ldots, k$. The minimum number of colors is denoted by $\chi_I(G)$. The PEP is polynomially solvable for a number of graph classes (interval graphs, forests, K-trees, etc.), whereas it is $\mathcal{NP}$-hard for bipartite graphs. Let $\mathcal{C}$ be the class of those graphs for which the PEP is polynomially solvable for any induced subgraph of G.

Let $G \in \mathcal{C}$ be a conflict graph and let $BIG = \{a_j \in L : s_j > \frac{1}{2}\}$. The algorithm by Jansen and Öhring uses BIG as the set of pre-colored vertices and then determines a feasible coloring of G using $\chi_I(G)$ colors: in this coloring, each pair of items in BIG has different colors. Finally, it uses a bin packing heuristic for each color class. Denote this algorithm by $\mathcal{A}(H)$, where H is the bin packing heuristic. They proved the following theorem:

Theorem 67 (Jansen and Öhring [123]) *For the BPC problem, the* $\mathcal{A}(FFD)$ *has a performance ratio* $2.4231 = h_\infty(2) + 1 \leq R_{\mathcal{A}(FFD)} \leq 2.5$.

Later, Epstein and Levin [78] revisited the algorithm, proving that the lower bound of $R_{\mathcal{A}(FFD)}$ is tight. Instead of using the PEP for preprocessing, they used a matching-based preprocessing (MP) algorithm, for which they proved that $R_{MP} = 2.5$, and that the result remains valid for all classes of graphs for which there exists a polynomial-time algorithm to find a coloring with a minimum number of colors. They also suggested a greedy algorithm which tries to pack two or three independent items into a single bin and then applies FFD to each color class. They proved that the approximation ratio of this algorithm is $\frac{7}{3}$.

As already mentioned, the PEP is $\mathcal{NP}$-hard for bipartite graphs. However, Jansen and Öhring [123] proved that by defining a simple algorithm which finds a coloring of the conflict graph with two colors, and packs each color class with NF, one gets an AR of 2. They also pointed out that the AR does not change if NF is replaced by FFD. Using a more complicated pairing technique, Epstein and Levin [78] presented an algorithm with approximation ratio exactly equal to $\frac{7}{4}$.

For the on-line version, the BPC is hard to approximate for many classes of graphs. Success depends on the on-line coloring phase. For *interval graphs*, Kierstead and Trotter [137] presented an on-line coloring algorithm, which uses $3\omega - 2$ colors in the worst case, where ω is the maximum clique size of the graph. Epstein and Levin [78] pointed out that, using NF for each color class, one obtains a 5-competitive algorithm. They also proved a combination of FFD with the algorithm in [137] produces an algorithm with competitive ratio 4.7. For interval graphs, there exists a lower bound for any on-line algorithm:

Theorem 68 (Epstein and Levin [78]) *The competitive ratio of any on-line algorithm for BPC on interval graphs is at least* $\frac{155}{36} \approx 4.30556$.

The theorem is a direct consequence of two interesting lemmas:

Lemma 1 (Epstein and Levin [78]) *Let c be a lower bound on the APR of any on-line algorithm for classical bin packing, which knows the value OPT in advance. Then the AR of any on-line algorithm for BPC on interval graphs is at least $3 + c$.*

Lemma 2 (Epstein and Levin [78]) *Any on-line algorithm for classical bin packing, which knows the value OPT in advance, has an AR of at least $\frac{47}{36} \approx 1.30556$.*

The on-line problem for other graph classes is still open.

8.6 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 $r \leq s$ in 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 [174] and the second one by Garey et al. [104]. 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 $R^{\infty}_{\mathrm{OFFD}} = \frac{27}{10}$.

Liu and Sidney [148] considered the case in which there is an *ordinal assumption*: initially, the actual sizes of the items are unknown, but their ordering is known, that is, $s_1 \geq s_2 \geq \cdots \geq s_n \geq 0$. Now assume that perfect knowledge of some of the sizes can be "purchased" by specifying the ordinal position of the desired sizes. It is shown that a worst-case performance ratio of ρ (where $\rho \geq 2$ is an

integer) can be achieved if knowledge of $\lfloor \ln(n(\rho-1)+1)/\ln\rho \rfloor$ weights can be purchased.

8.7 Clustered Items

In this generalization, a function $f(a_i, a_j)$ 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 $f(a_i, a_j) \leq D$. The problem has several obvious practical applications in contexts where geographical location constraints are present. Chandra et al. [39] 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 and proved that $3.75 \leq R_{\mathrm{A}}^{\infty} \leq 3.8$.

8.8 Item Types

In studying the packing of advertisements on Web pages, Adler et al. [1] encountered a variant of bin packing in which items are classified into *types*; the set of types is given, and there are no constraints on the number of items of any type. The problem is to pack the items into as few bins as possible subject to the constraint that no bin can have two items of the same type. They design optimal algorithms for restricted cases and then bound the performance of these algorithms viewed as approximations to the general problem.

9 Examples of Classification

In this section, the classification scheme proposed in Sect. 2.3 is resumed. The following examples should help familiarize the reader with it:

1. The classification of [157] shows a bin-size extension of **pack**.

$$pack|off\text{-}line|FPTAS|\{B_i\}$$

   Results: An approximation algorithm for variable-sized bin packing is presented which for any given positive ε produces a scheme with approximation ratio $1+\varepsilon$. This algorithm has time complexity polynomial in the number of items, the number of bins, and $1/\varepsilon$.

2. The classification of [145] gives another such example:

$$pack|on\text{-}line, off\text{-}line, open\text{-}end|R_A^{\infty} bound; FPTAS$$

   Results: For this hybrid of *pack* and *cover*, it is shown that any open-end version of on-line algorithms must have an asymptotic worst-case ratio

of at least 2. Next-Fit achieves this ratio. There is a fully polynomial-time approximation scheme for this problem.

3. The classification of [7] illustrates a capacity minimization problem with bin-stretching analysis:

$$mincap|on\text{-}line|R_A bound|stretching$$

Results: A combined algorithm achieves a worst-case bound of 1.625. The best lower bound for any on-line algorithm is 4/3.

4. Reference [48] studies a subset-cardinality objective function:

$$maxcard(subset)|off\text{-}line|R_A$$

Results: $R_{\mathrm{WFI}} = \frac{1}{2}$, $R_{\mathrm{FFI}} = \frac{3}{4}$.

5. Reference [61] is classified as a covering problem.

$$cover|on\text{-}line|R_A^{\infty} bound$$

Results: Asymptotic bound: $R_A^{\infty} \leq 1/2$ for any on-line algorithm A. There exists an asymptotically optimal on-line algorithm.

6. The classification of [90] is

$$pack|off\text{-}line|R_A^{\infty} bound$$

Algorithm: Combined Best-Fit (CBF) which takes the better of the First-Fit Decreasing solution and Best Two-Fit (B2F) solution, where the latter algorithm is a grouping version of Best-Fit limiting the number of items per bin.
Results: $R_{\mathrm{B2F}}^{\infty} = 5/4, \quad \frac{227}{195} \leq R_{CBF}^{\infty} \leq \frac{6}{5}$
Note that the word "variant" may be simplistic in that it occasionally hides details of relatively complicated algorithms.

7. Reference [95] shows an example for the combination of two algorithm classes:

$$pack|bounded\text{-}space, repack|R_A^{\infty} bound$$

Algorithm: REP_3, an adaptation of FFD using at most three open bins.
Result: $R_{\mathrm{REP}_3}^{\infty} \approx 1.69\ldots$.

8. The classification of [139] illustrates a case where one is allowed to constrain the number of items per bin.

$$pack|on\text{-}line, off\text{-}line|R_A^{\infty} bound|card(B) \leq k$$

Result:

$$\left(\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil\right) \leq R_{\mathrm{FF}_k}^{\infty} \leq \left(\frac{27}{10} - \frac{24}{10k}\right),$$

where FF_k is the obvious adaptation of FF.

9. For [175], the classification mentions yet another constraint:

$$pack|bounded\text{-}space, on\text{-}line|R_A^\infty$$

Result: $R^\infty_{SH_k}(k) = R^\infty_{H_k}$, where SH_k is a simplified version of H_k that uses only $O(\log k)$ open bins at any time.

10. Classification of [6] aggregates several results:

$$cover|on\text{-}line, off\text{-}line, open\text{-}end|R_A^\infty$$

Algorithms: Open-end variant of Next-Fit called DNF, of First-Fit Decreasing with a parameter r (FFD_r), and of an iterated version of Worst-Fit (IWFD).
Results:

$$R^\infty_{\mathrm{DNF}} = \frac{1}{2}, FFD_r^\infty = \frac{2}{3} \text{ for all } r, \frac{4}{3} \le r \le \frac{3}{2}, \text{ and } R^\infty_{\mathrm{IWFD}} = \frac{3}{4}.$$

11. A more recent such publication is [29, 30]:

$$maxpack|on\text{-}line, off\text{-}line|R_A^\infty|s_i \le 1/k$$

Results:
   a. No off-line approximation algorithm can have an asymptotic bound larger than 17/10, a bound achieved by FFD.
   b. For the off-line case, $R^\infty_{\mathrm{FFI}} = 6/5$, $R^\infty_{\mathrm{FFI}}(1/k) = 1 + 1/k,\ k \ge 2$
   c. For the on on-line case, $R^\infty_{\mathrm{FF}}(1/k) = R^\infty_{\mathrm{BF}}(1/k) = 1 + 1/(k-1),\ k \ge 3$.
   d. No deterministic algorithm A has a finite asymptotic bound R_A^∞ for either the *card*(subset)_min or*c*(subset)_min problems, except for the latter problem in the parametric case with $k > 1$, in which case the bound is again $1 + 1/(k-1)$.

12. An analysis with similarities to bin stretching is used in [31] to compare WF and NF.

$$pack|on\text{-}line|R_A^\infty|stretching$$

Results: A derivation of the asymptotic ratios for WF and NF under the assumption that they use bins of capacity C while *OPT* uses unit-capacity bins shows that WF and NF have the same asymptotic bound for all $C \ge 1$, except when $1 < C < 2$, in which case WF has the smaller bound.

Acknowledgements The second author was supported by Project "TÁMOP-4.2.1/B-09/1/KONV-2010-0005 - Creating the Center of Excellence at the University of Szeged," supported by the European Union and cofinanced by the European Regional Development Fund.

Cross-References

▸Advances in Scheduling Problems
▸Complexity Issues on PTAS
▸Online and Semi-online Scheduling

Recommended Reading

1. M. Adler, P.B. Gibbons, Y. Matias, Scheduling space sharing for internet advertising. J. Sched. **5**, 103–119 (2002)
   - $pack|$*off-line*$|$*running-time*$|mutex$.
2. S. Albers, Better bounds for on-line scheduling, in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 130–139
   - $mincap|$*on-line*$|R_A$*bound*.
3. N. Alon, Y. Azar, G.J. Woeginger, T. Yadid, Approximation schemes for scheduling, in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA (SIAM, 1997), pp. 493–500
   - $mincap|$*off-line*$|PTAS$.
4. R.J. Anderson, E.W. Mayr, M.K. Warmuth, Parallel approximation algorithms for bin packing. Inf. Comput. **82**, 262–277 (1989)
   - $pack|$*off-line*$|$*running-time*.
5. S.F. Assmann, *Problems in Discrete Applied Mathematics*. PhD thesis, Mathematics Department MIT, Cambridge, MA, 1983
6. S.F. Assmann, D.S. Johnson, D.J. Kleitman, J.Y.-T. Leung, On a dual version of the one-dimensional bin packing problem. J. Algorithms **5**, 502–525 (1984)
   - $cover|$*on-line*, *off-line*, *open-end*$|R_A^\infty$.
7. Y. Azar, O. Regev, On-line bin-stretching. Theor. Comput. Sci. **268**, 17–41 (2001)
   - $mincap|$*on-line*$|R_A$*bound*$|stretching$.
8. Y. Azar, J. Boyar, L.M. Favrholdt, K.S. Larsen, M.N. Nielsen, Fair versus unrestricted bin packing, in *SWAT '00, 7th Scandinavian Workshop on Algorithm Theory*, Bergen, Norway. Lecture Notes in Computer Science, vol. 1851 (Springer, 2000), pp. 200–213. This is the preliminary version of [9]
9. Y. Azar, J. Boyar, L. Epstein, L.M. Favrholdt, K.S. Larsen, M.N. Nielsen, Fair versus unrestricted bin packing. Algorithmica **34**, 181–196 (2002)
   - $maxcard(subset)|$*on-line*, $conservative|R_A$*bound*.
10. L. Babel, B. Chen, H. Kellerer, V. Kotov, On-line algorithms for cardinality constrained bin packing problems, in *ISAAC 2001*, Christchurch, New Zealand. Lecture Notes in Computer Science, vol. 2223 (Springer, 2001), pp. 695–706. This is the preliminary version of [11]
11. L. Babel, B. Chen, H. Kellerer, V. Kotov, On-line algorithms for cardinality constrained bin packing problems. Discret. Appl. Math. **143**, 238–251 (2004)
   - $pack|$*on-line*$|R_A^\infty|s_i \leq 1/k$.
12. B.S. Baker, A new proof for the first-fit decreasing bin-packing algorithm. J. Algorithms **6**, 49–70 (1985)
   - $pack|$*off-line*$|R_A^\infty$.
13. B.S. Baker, E.G. Coffman Jr., A tight asymptotic bound for next-fit-decreasing bin-packing. SIAM J. Algebra. Discret. Methods **2**, 147–152 (1981)
   - $pack|$*off-line*$|R_A^\infty|s_i \leq 1/k$.
14. J. Balogh, J. Békési, G. Galambos, M.C. Markót, Improved lower bounds for semi-online bin packing problems. Computing **84**, 139–148 (2009)
   - $pack|$*on-line*, $repack|R_A^\infty$*bounds*.
15. J. Balogh, J. Békési, G. Galambos, New lower bounds for certain bin packing algorithms, in *WAOA 2010*, Liverpool, UK. Lecture Notes in Computer Science, vol. 6534 (Springer, 2011), pp. 25–36
   - $pack|$*on-line*, *off-line*$|R_A^\infty$*bound*.
16. N. Bansal, Z. Liu, A. Sankar, Bin-packing with fragile objects, in *2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, vol. 223 (Montréal, Québec, Canada, 2001), pp. 38–46
17. N. Bansal, Z. Liu, A. Sankar, Bin-packing with fragile objects and frequency allocation in cellular networks. Wirel. Netw. **15**, 821–830 (2009)

- $pack|$*off-line*$|R_A^{\infty}|controllable$.

18. A. Bar-Noy, R.E. Ladner, T. Tamir, Windows scheduling as a restricted version of bin packing. ACM Trans. Algorithms **3**, 1–22 (2007)
    - $pack|$*off-line*$|R_A|discrete$.
19. Y. Bartal, A. Fiat, H. Karloff, R. Vohra, New algorithms for an ancient scheduling problem, in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, Victoria, Canada, 1992, pp. 51–58
    - $mincap|$*on-line*$|R_A$*bound*.
20. Y. Bartal, H. Karloff, Y. Rabani, A better lower bound for on-line scheduling. Inf. Process. Lett. **50**, 113–116 (1994)
    - $mincap|$*on-line*$|R_A$*bound*.
21. W. Bein, J.R. Correa, X. Han, A fast asymptotic approximation scheme for bin packing with rejection. Theor. Comput. Sci. **393**, 14–22 (2008)
    - $pack|$*off-line*$|PTAS|\{B_i\}, controllable$.
22. J. Békési, G. Galambos, A 5/4 linear time bin packing algorithm. Technical report OR-97-2, Teachers Trainer College, Szeged, Hungary, 1997. This is the preliminary version of [23]
23. J. Békési, G. Galambos, H. Kellerer, A 5/4 linear time bin packing algorithm. J. Comput. Syst. Sci. **60**, 145–160 (2000)
    - $pack|$*off-line*, *linear-time*$|R_A^{\infty}$.
24. R. Berghammer, F. Reuter, A linear approximation algorithm for bin packing with absolute approximation factor 3/2. Sci. Comput. Program. **48**, 67–80 (2003)
    - $pack|$*linear-time*$|R_A$.
25. V. Bilo, On the packing of selfish items, in *Proceedings of the 20th International Parallel and Distributed Processing Sysmposium (IPDPS)*, Rhodes Island, Greece (IEEE, 2006), pp. 25–29
    - $pack|repack|R_A^{\infty}$.
26. J. Blazewicz, K. Ecker, A linear time algorithm for restricted bin packing and scheduling problems. Oper. Res. Lett. **2**, 80–83 (1983)
    - $mincap, pack|$*off-line*, *linear-time*$|$*running-time*$|restricted$.
27. J. Boyar, L.M. Favrholdt, The relative worst order ratio for online algorithms. ACM Trans. Algorithms **3**, 1–24 (2007)
    - $pack, maxcard(subset)|$*on-line*.
28. J. Boyar, K.S. Larsen, M.N. Nielsen, The accomodation function: a generalization of the competitive ratio. SIAM J. Comput. **31**, 233–258 (2001)
    - $maxcard(subset)|$*on-line*$, conservative|R_A$*bound*$|restricted$.
29. J. Boyar, L. Epstein, L.M. Favrholdt, J.S. Kohrt, K.S. Larsen, M.M. Pedersen, S. Wøhlk, The maximum resource bin packing problem, in *Fundamentals of Computation Theory*. Lecture Notes in Computer Science, vol. 3623 (Springer, Berlin/New York, 2005), pp. 387–408. This is the preliminary version of [30]
30. J. Boyar, L. Epstein, L.M. Favrholdt, J.S. Kohrt, K.S. Larsen, M.M. Pedersen, S. Wøhlk, The maximum resource bin packing problem. Theor. Comput. Sci. **362**, 127–139 (2006)
    - $maxpack|$*on-line*, *off-line*$|R_A^{\infty}|s_i \leq 1/k$.
31. J. Boyar, L. Epstein, A. Levin, Tight results for Next Fit and Worst Fit with resource augmentation. Theor. Comput. Sci. **411**, 2572–2580 (2010)
    - $pack|$*on-line*$|R_A^{\infty}|stretching$.
32. D.J. Brown, A lower bound for on-line one-dimensional bin-packing algorithms. Technical report R-864, University of Illinois, Coordinated Science Laboratory, Urbana, IL, 1979
    - $pack|$*on-line*$|R_A^{\infty}$*bound*.
33. R.E. Burkard, G. Zhang, Bounded space on-line variable-sized bin packing. Acta Cybern. **13**, 63–76 (1997)
    - $pack|$*bounded-space*$|R_A^{\infty}|\{B_i\}$.
34. A. Caprara, U. Pferschy, Worst-case analysis of the subset sum algorithm for bin packing. Oper. Res. Lett. **32**, 159–166 (2004)
    - $pack|$*off-line*$|R_A^{\infty}$*bound*$|s_i \leq 1/k$.

35. W.-T. Chan, F.Y.L. Chin, D. Ye, G. Zhang, Y. Zhang, Online bin packing of fragile objects with application in cellular networks. JOCO **14**(4), 427–435 (2007)
• $pack|on\text{-}line|R_A^\infty|controllable$.
36. J.W. Chan, T. Lam, P.W.H. Wong, Dynamic bin packing of unit fractions items. Theor. Comput. Sci. **409**, 521–529 (2008)
• $pack|on\text{-}line, dynamic|R_A^\infty bounds|discrete$.
37. J.W. Chan, P.W.H. Wong, F.C.C. Yung, On dynamic bin packing: an improved lower bound and resource augmentation analysis. Algorithmica **53**, 172–206 (2009)
• $pack|on\text{-}line, dynamic|R_A^\infty bound|discrete, stretching$.
38. B. Chandra, Does randomization help in on-line bin packing? Inf. Process. Lett. **43**, 15–19 (1992)
• $pack|on\text{-}line|R_A^\infty bound$.
39. A.K. Chandra, D.S. Hirschler, C.K. Wong, Bin packing with geometric constraints in computer network design. Oper. Res. **26**, 760–772 (1978)
• $pack|off\text{-}line|R_A^\infty$.
40. C. Chekuri, S. Khanna, A polynomial time approximation scheme for the multiple knapsack problem. SIAM J. Comput. **35**, 713–728 (2005)
41. B. Chen, A. van Vliet, G.J. Woeginger, New lower and upper bounds for on-line scheduling. Oper. Res. Lett. **16**, 221–230 (1994)
• $mincap|on\text{-}line|R_A bounds$.
42. E.G. Coffman Jr., An introduction to combinatorial models of dynamic storage allocation. SIAM Rev. **25**, 311–325 (1983)
43. E.G. Coffman Jr., J. Csirik, Performance guarantees for one-dimensional bin packing, in *Handbook of Approximation Algorithms and Metaheuristics*, chapter 32, ed. by T. Gonzales (Taylor and Francis Books/CRC, Boca Raton, 2006), pp. 32–1–32–18
44. E.G. Coffman Jr., J. Csirik, A classification scheme for bin packing theory. Acta Cybern. **18**, 47–60 (2007)
• $pack|on\text{-}line, off\text{-}line|R_A^\infty, R_A$.
45. E.G. Coffman Jr., J.Y.-T. Leung, Combinatorial analysis of an efficient algorithm for processor and storage allocation. SIAM J. Comput. **8**, 202–217 (1979)
• $maxcard(subset)|off\text{-}line|R_A bound$.
46. E.G. Coffman Jr., G.S. Lueker, *Probabilistic Analysis of Packing and Partitioning Algorithms* (Wiley, New York, 1991)
47. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, An application of bin-packing to multiprocessor scheduling. SIAM J. Comput. **7**, 1–17 (1978)
• $mincap|off\text{-}line|R_A bound$.
48. E.G. Coffman Jr., J.Y.-T. Leung, D.W. Ting, Bin packing: maximizing the number of pieces packed. Acta Inform. **9**, 263–271 (1978)
• $maxcard(subset)|off\text{-}line|R_A$.
49. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Dynamic bin packing. SIAM J. Comput. **12**, 227–258 (1983)
• $pack|dynamic, repack|R_A^\infty bound|s_i \leq 1/k$.
50. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation algorithms for bin-packing: An updated survey, in *Algorithm Design for Computer System Design*, ed. by G. Ausiello, M. Lucertini, P. Serafini (Springer, Wien, 1984), pp. 49–106
51. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Bin packing with divisible item sizes. J. Complex. **3**, 405–428 (1987)
• $pack, cover, maxcard(subset)|on\text{-}line, off\text{-}line, dynamic|complexity|\{B_i\}, restricted$.
52. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: a survey, in *Approximation Algorithms for NP-Hard Problems*, ed. by D.S. Hochbaum (PWS Publishing Company, Boston, 1997), pp. 46–93
53. E.G. Coffman Jr., G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: Combinatorial analysis, in *Handbook of Combinatorial Optimization*, ed. by D.-Z. Du, P.M. Pardalos (Kluwer, Boston, 1999)

54. E.G. Coffman Jr., J. Csirik, J.Y.-T. Leung, Variable-sized bin packing and bin covering, in *Handbook of Approximation Algorithms and Metaheuristics*, chapter 34, ed. by T. Gonzales (Taylor and Francis Books/CRC, Boca Raton, 2006), pp. 34–1–34–11
55. E.G. Coffman Jr., J. Csirik, J.Y-T. Leung, Variants of classical one-dimensional bin packing, in *Handbook of Approximation Algorithms and Metaheuristics*, chapter 33, ed. by T. Gonzales (Taylor and Francis Books/CRC, Boca Raton, 2006), pp. 33–1–33–13
56. J. Csirik, An on-line algorithm for variable-sized bin packing. Acta Inform. **26**, 697–709 (1989)
    - $pack|on\text{-}line|R_A^\infty|\{B_i\}$.
57. J. Csirik, The parametric behaviour of the first fit decreasing bin-packing algorithm. J. Algorithms **15**, 1–28 (1993)
    - $pack|off\text{-}line|R_A^\infty|s_i \leq 1/k$.
58. J. Csirik, B. Imreh, On the worst-case performance of the Next-k-Fit bin-packing heuristic. Acta Cybern. **9**, 89–105 (1989)
    - $pack|bounded\text{-}space|R_A^\infty bounds$.
59. J. Csirik, D.S. Johnson, Bounded space on-line bin-packing: best is better than first, in *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, 1991, pp. 309–319. This is the preliminary version of [60]
60. J. Csirik, D.S. Johnson, Bounded space on-line bin-packing: best is better than first. Algorithmica **31**, 115–138 (2001)
    - $pack|bounded\text{-}space|R_A^\infty$.
61. J. Csirik, V. Totik, On-line algorithms for a dual version of bin packing. Discret. Appl. Math. **21**, 163–167 (1988)
    - $cover|on\text{-}line|R_A^\infty bound$.
62. J. Csirik, G.J. Woeginger, Online packing and covering problems, in *Online Algorithms: The State of the Art*, ed. by A. Fiat, G.J. Woeginger. Lecture Notes in Computer Science, vol. 1442 (Springer, Berlin, 1998), pp. 154–177
63. J. Csirik, G.J. Woeginger, Resource augmentation for online bounded space bin packing. J. Algorithms **44**, 308–320 (2002)
    - $pack|bounded\text{-}space|R_A^\infty bound|stretching$.
64. J. Csirik, G. Galambos, G. Turan, Some results on bin-packing, in *Proceedings of the EURO VI Conference*, Vienna, Austria, 1983, pp. 52
    - $pack|on\text{-}line, off\text{-}line|R_A^\infty$.
65. J. Csirik, D.S. Johnson, C. Kenyon, Better approximation algorithms for bin covering, in *Proceedings of the Twelft Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington, DC, 2001, pp. 557–566
    - $cover|off\text{-}line|PTAS$.
66. M. Demange, P. Grisoni, V.T. Paschos, Differential approximation algorithms for some combinatorial optimization problems. Theor. Comput. Sci. **209**, 107–122 (1998)
    - $pack|off\text{-}line$.
67. M. Demange, J. Monnot, V.T. Paschos, Bridging gap between standard and differential polynomial approximation: the case of bin-packing. Appl. Math. Lett. **12**, 127–133 (1999)
    - $pack|off\text{-}line|R_A^\infty$.
68. G. Dósa, The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leq (11/9)\ OPT(I) + 6/9$, in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*ed. by B. Chen, M. Paterson, G. Zhang. Lecture Notes in Computer Science, vol. 4614 (Springer, Berlin, 2007), pp. 1–11
    - $pack|off\text{-}line|R_A^\infty$.
69. G. Dósa, Y. He, Bin packing problems with rejection penalties and their dual problems. Inf. Comput. **204**, 795–815 (2006)
    - $pack, cover|on\text{-}line, off\text{-}line|R_A^\infty, R_A|controllable$.
70. L. Epstein, Bin packing with rejection revisited, in *WAOA*, Zurich, Switzerland. Lecture Notes in Computer Science, vol. 4368 (Springer, 2006), pp. 146–159. This is the preliminary version of [73]

71. L. Epstein, Online bin packing with cardinality constraints. SIAM J. Discret. Math. **20**, 1015–1030 (2007)
• $pack|bounded\text{-}space|R_A^\infty|\{B_i\}, stretching, card(B) \leq k.$
72. L. Epstein, On online bin packing with LIB constraints. Nav. Res. Logist. (NRL) **56**, 780–786 (2009)
• $pack|on\text{-}line|R_A^\infty|s_i \leq 1/k, controllable.$
73. L. Epstein, Bin packing with rejection revisited. Algorithmica **56**, 505–528 (2010)
• $pack|off\text{-}line, bounded\text{-}space|R_A^\infty bound, PTAS|controllable.$
74. L. Epstein, L.M. Favrholdt, On-line maximizing the number of items packed in variable-sized bins. Acta Cybern. **16**, 57–66 (2003)
• $maxcard(subset)|on\text{-}line, conservative|R_A|\{B_i\}.$
75. L. Epstein, E. Kleiman, Resource augmented semi-online bounded space bin packing. Discret. Appl. Math. **157**, 2785–2798 (2009)
• $pack|bounded\text{-}space, repack|R_A^\infty|stretching.$
76. L. Epstein, E. Kleiman, Selfish bin packing. Algorithmica (2011). To appear (online first: doi:10.1007/s00453-009-9348-6)
• $pack|repack|R_A^\infty bounds.$
77. L. Epstein, A. Levin, More on online bin packing with two item sizes. Discret. Optim. **5**(4), 705–713 (2008)
• $pack|on\text{-}line|R_A^\infty|restricted, s_i \leq 1/k.$
78. L. Epstein, A. Levin, On bin packing with conflicts. SIAM J. Optim. **19**, 1270–1298 (2008)
• $pack|on\text{-}line, off\text{-}line|R_A|mutex.$
79. L. Epstein, R. van Stee, Multidimensional packing problems, in *Handbook of Approximation Algorithms and Metaheuristics*, chapter 35, ed. by T. Gonzales (Taylor and Francis Books/CRC, Boca Raton, 2006), pp. 35–1–35–15
80. L. Epstein, R. van Stee, Online bin packing with resource augmentation. Discret. Optim. **4**, 322–333 (2007)
• $pack|on\text{-}line|R_A^\infty bound|stretching.$
81. L. Epstein, R. van Stee, Approximation schemes for packing splittable items with cardinality constraints, in *WAOA*, Eilat, Israel. Lecture Notes in Computer Science, vol. 4927 (Springer, 2008), pp. 232–245
• $pack|off\text{-}line|PTAS|controllable.$
82. L. Epstein, C. Imreh, A. Levin, Class constrained bin packing revisited. Theor. Comput. Sci. **411**, 3073–3089 (2010)
• $pack|on\text{-}line, off\text{-}line|R_A^\infty, FPTAS|restricted, card(B) \leq kcolors.$
83. U. Faigle, W. Kern, Gy. Turán, On the performance of on-line algorithms for partition problems. Acta Cybern. **9**, 107–119 (1989)
• $pack, mincap|on\text{-}line|R_A^\infty bound|restricted.$
84. W. Fernandez de la Vega, G.S. Lueker, Bin packing can be solved within $1+\epsilon$ in linear time. Combinatorica **1**, 349–355 (1981)
• $pack|off\text{-}line|PTAS.$
85. L. Finlay, P. Manyem, Online LIB problems: heuristics for bin covering and lower bounds for bin packing. RAIRO Rech. Oper. **39**, 163–183 (2005)
• $pack, cover|on\text{-}line|R_A^\infty|controllable.$
86. D.C. Fisher, Next-fit packs a list and its reverse into the same number of bins. Oper. Res. Lett. **7**, 291–293 (1988)
• $pack|bounded\text{-}space.$
87. D.K. Friesen, Tighter bounds for the multifit processor scheduling algorithm. SIAM J. Comput. **13**, 170–181 (1984)
• $mincap|off\text{-}line|R_A^\infty bound.$
88. D.K. Friesen, F.S. Kuhl, Analysis of a hybrid algorithm for packing unequal bins. SIAM J. Comput. **17**, 23–40 (1988)
• $maxcard(subset)|off\text{-}line|R_A|\{B_i\}.$

89. D.K. Friesen, M.A. Langston, Variable sized bin packing. SIAM J. Comput. **15**, 222–230 (1986)
• $pack|on\text{-}line|R_A^\infty|\{B_i\}$.
90. D.K. Friesen, M.A. Langston, Analysis of a compound bin-packing algorithm. SIAM J. Discret. Math. **4**, 61–79 (1991)
• $pack|off\text{-}line|R_A^\infty bound$.
91. G. Galambos, A new heuristic for the classical bin-packing problem. Technical report 82, Institute fuer Mathematik, Augsburg, 1985
• $pack|repack|R_A^\infty bound|s_i \leq 1/k$.
92. G. Galambos, Parametric lower bound for on-line bin-packing. SIAM J. Algebra. Discret. Meth. **7**, 362–367 (1986)
• $pack|on\text{-}line|R_A^\infty bound|s_i \leq 1/k$.
93. G. Galambos, J.B.G. Frenk, A simple proof of Liang's lower bound for on-line bin packing and the extension to the parametric case. Discret. Appl. Math. **41**, 173–178 (1993)
• $pack|on\text{-}line|R_A^\infty bound|s_i \leq 1/k$.
94. G. Galambos, G.J. Woeginger, An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. SIAM J. Comput. **22**, 345–355 (1993)
• $mincap|on\text{-}line|R_A$.
95. G. Galambos, G.J. Woeginger, Repacking helps in bounded space on-line bin-packing. Computing **49**, 329–338 (1993)
• $pack|bounded\text{-}space, repack|R_A^\infty bound$.
96. G. Galambos, G.J. Woeginger, On-line bin packing – a restricted survey. Z. Oper. Res. **42**, 25–45 (1995)
97. G. Gambosi, A. Postiglione, M. Talamo, New algorithms for on-line bin packing, in *Algorithms and Complexity*, ed. by R. Petreschi, G. Ausiello, D.P. Bovet (World Scientific, Singapore, 1990), pp. 44–59. This is the preliminary version of [99]
98. G. Gambosi, A. Postiglione, M. Talamo, On-line maintenance of an approximate bin-packing solution. Nord. J. Comput. **4**, 151–166 (1997)
• $pack|repack|R_A^\infty$.
99. G. Gambosi, A. Postiglione, M. Talamo, Algorithms for the relaxed online bin-packing model. SIAM J. Comput. **30**, 1532–1551 (2000)
• $pack|on\text{-}line, repack|R_A^\infty$.
100. M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman, New York, 1979)
101. M.R. Garey, D.S. Johnson, Approximation algorithm for bin-packing problems: a survey, in *Analysis and Design of Algorithm in Combinatorial Optimization*, ed. by G. Ausiello, M. Lucertini (Springer, New York, 1981), pp. 147–172
102. M.R. Garey, D.S. Johnson, A 71/60 theorem for bin packing. J. Complex. **1**, 65–106 (1985)
• $pack|off\text{-}line|R_A^\infty$.
103. M.R. Garey, R.L. Graham, J.D. Ullmann, Worst-case analysis of memory allocation algorithms, in *Proceedings of the 4th Annual ACM Symposium Theory of Computing*, Denver, CO (ACM, New York, 1972), pp. 143–150
104. M.R. Garey, R.L. Graham, D.S. Johnson, A.C.-C. Yao, Resource constrained scheduling as generalized bin packing. J. Comb. Theory Ser. A **21**, 257–298 (1976)
• $pack|on\text{-}line|R_A^\infty bound|s_i \leq 1/k, controllable$.
105. P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem. Oper. Res. **9**, 849–859 (1961)
106. P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem – (Part II). Oper. Res. **11**, 863–888 (1963)
107. S.W. Golomb, On certain nonlinear recurring sequences. Am. Math. Mon. **70**, 403–405 (1963)
108. R.L. Graham, Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)

109. R.L. Graham, Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)
• $mincap|$*on-line*, *off-line*$|R_A$.
110. R.L. Graham, Bounds on multiprocessing anomalies and related packing algorithms, in *Proceedings of 1972 Spring Joint Computer Conference* (AFIPS Press, Montvale, 1972), pp. 205–217
• $pack, mincap|$*on-line*, *off-line*$|R_A$*bound*.
111. E.F. Grove, Online bin packing with lookahead, in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA (SIAM, 1995), pp. 430–436
• $pack|$*on-line*, $repack|R_A^\infty$*bound*.
112. G. Gutin, T.R. Jensen, A. Yeo, Batched bin packing. Discret. Optim. **2**, 71–82 (2005)
• $pack|$*on-line*, $repack|R_A^\infty$.
113. G. Gutin, T.R. Jensen, A. Yeo, On-line bin packing with two item sizes. Algorithm. Oper. Res. **1**, 72–78 (2006)
• $pack|$*on-line*$|R_A|restricted$.
114. L.A. Hall, Approximation algorithms for scheduling, in *Approximation Algorithms for NP-Hard Problems*, ed. by D.S. Hochbaum (PWS Publishing Company, Boston, 1997), pp. 1–45
115. D.S. Hochbaum (ed.), *Approximation Algorithms for NP-Hard Problems* (PWS Publishing Company, Boston, 1997)
116. D.S. Hochbaum, Various notions of approximation: good, better, best, and more, in *Approximation Algorithms for NP-Hard Problems*, ed. by D.S. Hochbaum (PWS Publishing Company, Boston, 1997), pp. 389–391
117. D.S. Hochbaum, D.B. Shmoys, A packing problem you can almost solve by sitting on your suitcase. SIAM J. Algebra. Discret. Methods **7**, 247–257 (1986)
• $pack|$*off-line*$|complexity|restricted$.
118. D.S Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**, 144–162 (1987)
• $mincap|$*off-line*$|R_A$.
119. M. Hofri, *Analysis of Algorithms* (Oxford University Press, New York, 1995)
120. Z. Ivković, E. Lloyd, Fully dynamic algorithms for bin packing: being myopic helps, in *Proceedings of the 1st European Symposium on Algorithms*. Lecture Notes in Computer Science, vol. 726 (Springer, New York, 1993), pp. 224–235. This is the preliminary version of [122]
121. Z. Ivković, E. Lloyd, Partially dynamic bin packing can be solved within $1+\epsilon$ in (amortized) polylogarithmic time. Inf. Process. Lett. **63**, 45–50 (1997)
• $pack|dynamic|PTAS$.
122. Z. Ivković, E. Lloyd, Fully dynamic algorithms for bin packing: being (mostly) myopic helps. SIAM J. Comput. **28**, 574–611 (1998)
• $pack|dynamic, repack|R_A^\infty$.
123. K. Jansen, S. Öhring, Approximation algorithms for time constrained scheduling. Inf. Comput. **132**, 85–108 (1997)
• $pack|$*off-line*$|R_A$*bound*$|mutex$.
124. K. Jansen, R. Solis-Oba, An asymptotic fully polynomial time approximation scheme for bin covering. Theor. Comput. Sci. **306**, 543–551 (2003)
• $cover|$*off-line*$|FPTAS$.
125. D.S. Johnson, Fast allocation algorithms, in *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, New York, 1972, pp. 144–154
126. D.S. Johnson, *Near-Optimal Bin Packing Algorithms*. PhD thesis, MIT, Cambridge, MA, 1973
127. D.S. Johnson, Fast algorithms for bin packing. J. Comput. Syst. Sci. **8**, 272–314 (1974)
• $pack|$*on-line*, *off-line*$|R_A^\infty|s_i \leq 1/k$.
128. D.S. Johnson, The NP-completeness column: an ongoing guide. J. Algorithms **3**, 89–99 (1982)

129. D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J. Comput. **3**, 256–278 (1974)
• $pack|on\text{-}line, off\text{-}line|R_A^\infty, R_A|s_i \leq 1/k$.
130. J. Kang, S. Park, Algorithms for the variable sized bin packing problem. Eur. J. Oper. Res. **147**, 365–372 (2003)
• $pack|off\text{-}line|R_A^\infty|\{B_i\}$.
131. D.R. Karger, S.J. Phillips, E. Torng, A better algorithm for an ancient scheduling problem. J. Algorithms **20**, 400–430 (1996)
• $mincap|on\text{-}line|R_A$.
132. N. Karmarkar, R.M. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, in *Proceedings of the 23rd Annual IEEE Symposium on Foundations Computer Science*, Chicago, IL, 1982, pp. 312–320
• $pack|off\text{-}line|FPTAS$.
133. G.Y. Katona, Edge disjoint polyp packing. Discret. Appl. Math. **78**, 133–152 (1997)
• $pack|on\text{-}line|R_A^\infty bounds$.
134. H. Kellerer, A polynomial time approximation scheme for the multiple knapsack problem, in *RANDOM-APPROX*, Berkeley, CA. Lecture Notes in Computer Science, vol. 1671 (Springer, 1999), pp. 51–62
135. H. Kellerer, U. Pferschy, Cardinality constrained bin-packing problems. Ann. Oper. Res. **92**, 335–348 (1999)
• $pack|off\text{-}line|R_A^\infty|card(B) \leq k$.
136. C. Kenyon, Best-fit bin-packing with random order, in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms* Atlanta, GA (ACM/SIAM, Philadelphia, 1996), pp. 359–364
• $pack|on\text{-}line$.
137. K.A. Kierstead, W.T. Trotter, An extremal problem in recursive combinatorics. Congr. Numer. **33**, 143–153 (1981)
138. N.G. Kinnersley, M.A. Langston, Online variable-sized bin packing. Discret. Appl. Math. **22**, 143–148 (1988–1989)
• $pack|on\text{-}line|R_A^\infty|\{B_i\}$.
139. K.L. Krause, Y.Y. Shen, H.D. Schwetman, Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J. ACM **22**, 522–550 (1975)
• $pack|on\text{-}line, off\text{-}line|R_A^\infty bound|card(B) \leq k$.
140. M.A. Langston, Improved 0/1 interchanged scheduling. BIT **22**, 282–290 (1982)
• $maxcard(subset)|off\text{-}line|R_A^\infty|\{B_i\}$.
141. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in *Logistics of Production and Inventory*, ed. by S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin. Handbooks in Operations Research and Management Science, vol. 4 (North-Holland, Amsterdam, 1993), pp. 445–522
142. C.C. Lee, 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
143. C.C. Lee, D.T. Lee, A simple on-line bin-packing algorithm. J. ACM **32**, 562–572 (1985)
• $pack|on\text{-}line|R_A^\infty bound$.
144. H.W. Lenstra Jr., Integer programming with a fixed number of variables. Math. Oper. Res. **8**, 538–548 (1983)
145. J.Y.-T. Leung, M. Dror, G.H. Young, A note on an open-end bin packing problem. J. Sched. **4**, 201–207 (2001)
• $pack|on\text{-}line, off\text{-}line, open\text{-}end|R_A^\infty bound; FPTAS$.
146. R. Li, M. Yue, The proof of $FFD(L) \leq 11/9\, OPT(L) + 7/9$. Chin. Sci. Bull. **42**, 1262–1265 (1997)
• $pack|off\text{-}line|R_A^\infty$.

147. F.M. Liang, A lower bound for on-line bin packing. Inf. Process. Lett. **10**, 76–79 (1980)
- $pack|$*on-line*$|R_A^\infty$*bound*.

148. W.-P. Liu, J.B. Sidney, Bin packing using semi-ordinal data. Oper. Res. Lett. **19**, 101–104 (1996)
- $pack|$*off-line*$|R_A^\infty$.

149. L. Lovász, M. Saks, W.T. Trotter, An on-line graph-coloring algorithm with sublinear performance ratio. Discret. Math. **75**, 319–325 (1989)
150. C.A. Mandal, P.P. Chakrabarti, S. Ghose, Complexity of fragmentable object bin packing and an application. Comput. Math. Appl. **35**, 91–97 (1998)
- $pack|$*off-line*$|$*running-time*$|controllable$.

151. R.L. Manyem, P. Salt, M.S. Visser, Approximation lower bounds in online lib bin packing and covering. J. Autom. Lang. Comb. **8**, 663–674 (2003)
- $pack|$*on-line*$|R_A^\infty$*bound*$|controllable$.

152. W. Mao, Best-k-fit bin packing. Computing **50**, 265–270 (1993)
- $pack|$*bounded-space*$|R_A^\infty$.

153. W. Mao, Tight worst-case performance bounds for next-k-fit bin packing. SIAM J. Comput. **22**, 46–56 (1993)
- $pack|$*bounded-space*$|R_A^\infty$.

154. C.U. Martel, A linear time bin-packing algorithm. Oper. Res. Lett. **4**, 189–192 (1985)
- $pack|$*off-line*, *linear-time*$|R_A^\infty$.

155. N. Menakerman, R. Rom, Bin packing with item fragmentation, in *WADS*, Providence, RI. Lecture Notes in Computer Science, vol. 2125 (Springer, 2001), pp. 313–324
- $pack|$*on-line*, *off-line*$|R_A|controllable$.

156. F.D. Murgolo, Anomalous behaviour in bin packing algorithms. Discrte. Appl. Math. **21**, 229–243 (1988)
- $pack|$*on-line*, *off-line*, $conservative$.

157. F.D. Murgolo, An efficient approximation scheme for variable-sized bin packing. SIAM J. Comput. **16**, 149–161 (1988)
- $pack|$*off-line*$|FPTAS|\{B_i\}$.

158. N. Naaman, R. Rom, Packet scheduling with fragmentation, in *INFOCOM 2002*, New York, NY (IEEE, 2002)
- $pack|$*on-line*, *off-line*$|R_A|controllable$.

159. P. Ramanan, D.J. Brown, C.C. Lee, D.T. Lee, On-line bin packing in linear time. J. Algorithms **10**, 305–326 (1989)
- $pack|$*on-line*, *linear-time*$|R_A^\infty$*bound*.

160. M.B. Richey, Improved bounds for harmonic-based bin packing algorithms. Discret. Appl. Math. **34**, 203–227 (1991)
- $pack|$*on-line*, *linear-time*$|R_A^\infty$*bound*.

161. S. Sahni, Algorithms for scheduling independent tasks. J. ACM **23**, 116–127 (1976)
162. S.S. Seiden, An optimal online algorithm for bounded space variable-sized bin packing. SIAM J. Discret. Math. **14**, 458–470 (2001)
- $pack|$*on-line*, *bounded-space*$|R_A^\infty$*bound*$|\{B_i\}$.

163. S.S. Seiden, On the online bin packing problem. J. ACM **49**, 640–671 (2002)
- $pack|$*on-line*$|R_A^\infty$*bound*.

164. S.S. Seiden, R. van Stee, L. Epstein, New bounds for variable-sized online bin packing. SIAM J. Comput. **33**, 455–469 (2003)
- $pack|$*on-line*$|R_A^\infty$*bound*$|\{B_i\}$.

165. H. Shachnai, T. Tamir, Polynomial time approximation schemes for class-constrained packing problems. J. Sched. **4**, 313–338 (2001)
- $pack|$*off-line*$|PTAS|card(B) \leq k$*colors*.

166. H. Shachnai, T. Tamir, Tight bounds for online class-constrained packing. Theor. Comput. Sci. **321**, 103–123 (2004)
- $pack|$*on-line*$|R_A|card(B) \leq k$*colors*.

167. H. Shachnai, O. Yehezkely, Fast asymptotic FPTAS for packing fragmentable items with costs, in FCT, Budapest, Hungary. Lecture Notes in Computer Science, vol. 4639 (Springer, 2007), pp. 482–493
• $pack|$*off-line*$|FPTAS|controllable$.
168. H. Shachnai, T. Tamir, O. Yehezkely, Approximation schemes for packing with item fragmentation. Theory Comput. Syst. **43**, 81–98 (2008)
• $pack|$*off-line*$|PTAS|controllable$.
169. D. Simchi-Levi, New worst-case results for the bin packing problem. Nav. Res. Logist. Q. **41**, 579–585 (1994)
• $pack|$*on-line*, *off-line*$|R_A$*bound*.
170. J. Sylvester, On a point in the theory of vulgar fractions. Am. J. Math. **3**, 332–335 (1880)
171. A. van Vliet, An improved lower bound for on-line bin packing algorithms. Inf. Process. Lett. **43**, 277–284 (1992)
• $pack|$*on-line*$|R_A^\infty$*bound*$|s_i \leq 1/k$.
172. A. van Vliet, *Lower and Upper Bounds for On-Line Bin Packing and Scheduling Heuristic*. PhD thesis, Erasmus University, Rotterdam, 1995
173. A. van Vliet, On the asymptotic worst case behavoir of harmonic fit. J. Algorithms **20**, 113–136 (1996)
• $pack|$*on-line*, *bounded-space*$|R_A^\infty$*bound*$|s_i \leq 1/k$.
174. T.S. Wee, M.J. Magazine, Assembly line balancing as generalized bin packing. Oper. Res. Lett. **1**, 56–58 (1982)
• $pack|$*off-line*$|R_A^\infty$.
175. G.J. Woeginger, Improved space for bounded-space, on-line bin-packing. SIAM J. Discret. Math. **6**, 575–581 (1993)
• $pack|$*on-line*, *bounded-space*$|R_A^\infty$.
176. E.C. Xavier, F.K. Miyazawa, The class constrained bin packing problem with applications to video-on-demand. Theor. Comput. Sci. **393**, 240–259 (2008)
• $pack|$*on-line*, *off-line*$|R_A^\infty, PTAS|card(B) \leq k$.
177. J. Xie, Z. Liu, New worst-case bound of first-fit heuristic for bin packing problem, Unpublished manuscript.
• $pack|$*on-line*$|R_A$*bound*.
178. 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
179. K. Xu, The asymptotic worst-case behavior of the FFD heuristics for small items. J. Algorithms **37**, 237–246 (2000)
• $pack|$*off-line*$|R_A^\infty|s_i \leq 1/k$.
180. A.C.-C. Yao, New algorithms for bin packing. J. ACM **27**, 207–227 (1980)
• $pack|$*on-line*, *off-line*$|R_A^\infty$.
181. G. Yu, G. Zhang, Bin packing of selfish items, in *WINE 2008*, Shanghai, China. Lecture Notes in Computer Science, vol. 5385 (Springer, 2008)
• $pack|repack|R_A^\infty$*bounds*.
182. M. Yue, On the exact upper bound for the multifit processor scheduling algorithm. Ann. Oper. Res. **24**, 233–259 (1991)
• $mincap|$*off-line*$|R_A$.
183. 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. Appl. Sin. **7**, 321–331 (1991)
• $pack|$*off-line*$|R_A^\infty$.
184. G. Zhang, Tight worst-case performance bound for AFB_k bin packing. Technical report 15, Institute of Applied Mathematics. Academia Sinica, Beijng, China, 1994. This is the preliminary version of [187]
185. G. Zhang, Worst-case analysis of the FFH algorithm for on-line variable-sized bin paking. Computing **56**, 165–172 (1996)
• $pack|$*on-line*$|R_A^\infty|\{B_i\}$.

186. G. Zhang, A new version of on-line variable-sized bin packing. Discret. Appl. Math. **72**, 193–197 (1997)
- $pack|on\text{-}line, off\text{-}line|R_A^{\infty}|\{B_i\}$.

187. G. Zhang, M. Yue, Tight performance bound for AFB_k bin packing. Acta Math. Appl. Sin. Engl. Ser. **13**, 443–446 (1997)
- $pack|bounded\text{-}space|R_A^{\infty}$.

188. G. Zhang, X. Cai, C.K. Wong, Linear time-approximation algorithms for bin packing. Oper. Res. Lett. **26**, 217–222 (2000)
- $pack|on\text{-}line, off\text{-}line|R_A$.