Acta Cybernetica **18** (2007) 47–60.

A Classification Scheme for Bin Packing Theory

Edward G. Coffman, Jr.* and János Csirik†

Abstract

Classifications of published research place new results in a historical context and in so doing identify open problems. An example in wide use classifies results in scheduling theory according to a scheme originated by Graham, Lawler, Lenstra and Rinnooy Kan [10]. A similar effort was made by Dyckhoff [6] for cutting and packing problems. Such classification schemes can be combined with comprehensive bibliographies, e.g., the one provided for scheduling theory by Bruckner[1]. This paper describes a novel classification scheme for bin packing which is being applied by the authors to an extensive (and growing) bibliography of the theory. Problem classifications are supplemented by compact descriptions of the main results and of the corresponding algorithms. The usefulness of the scheme is extended by an online search engine. With the help of this software, one is easily able to determine whether results already exist for applications that appear to be new, and to assist in locating the cutting edge of the general theory.

1 Introduction

For given positive reals $a_1, \ldots, a_n$ and $b_1, b_2, \ldots$, classical bin packing algorithms partition some subset of $\{a_1, \ldots, a_n\}$ into blocks $B_1, B_2, \ldots, B_j$ such that the *levels* $\ell(B_i) := \sum_{a_k \in B_i} a_k$ satisfy the sum constraints $\ell(B_i) \leq b_i$, $1 \leq i \leq j$. This definition embraces several packing problems, depending on the way the subset of the a_i's and the integer j are chosen. In bin packing terms, the a_i are called *items*, the blocks B_i are called *bins* with respective *capacities* or *sizes* b_i, and the partitions are called *packings*; 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, we adopt the harmless abuse whereby a_i denotes both the name and the size of the i-th item. The generic symbol for packing is $\mathcal{P}$; the

*Department of Electrical Engineering, Columbia University, 1312 S.W. Mudd, 500 West 120th Street, New York, NY 10027, USA. E-mail: egc@ee.columbia.edu

†Department of Computer Science, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary. E-mail: csirik@inf.u-szeged.hu

[1]Available at http://www.mathematik.uni-osnabrueck.de/research/OR/class/

number of items in $\mathcal{P}$ is denoted by $|\mathcal{P}|$ and the norm of the packing is defined as the sum of the sizes of the nonempty bins: $\|\mathcal{P}\| := \sum_{\ell(B_i)>0} b_i$. In the majority of problems being classified, the entire list is packed, so $|\mathcal{P}| = n$ and the index j of the last occupied bin is the packing measure of interest. It is also common to have all bin sizes the same, in which case the bin size is denoted simply by b and $a_i \leq b$ is assumed for all i. Further, when b functions only as a scale factor, it is usually normalized to 1; *in this case, the norm reduces simply to the number of bins in the packing*, i.e., $j = \|\mathcal{P}\|$. The term *wasted space* has the obvious meaning, $\|\mathcal{P}\| - \sum_{i=1}^{n} a_i$.

With bin sizes given by context, let $\mathcal{P}_A(L)$ denote the packing of L produced by algorithm A. In the literature, one finds the notation $A(L)$ representing properties such as $\|\mathcal{P}_A(L)\|$; but since $A(L)$ may denote different properties for different problems (the same algorithm A may apply to problems with different objective functions), we will need the alternative notation on occasion. The more general forms with b specified are $\mathcal{P}(L, b)$ and $A(L, b)$, but the bin size will be omitted whenever it has been normalized to 1. The minimum of $\|\mathcal{P}(L)\|$ over all partitions of L satisfying the sum constraints will have the notation: $OPT(L) := \min_{\mathcal{P}} \|\mathcal{P}(L)\|$, the notation $OPT(L)$ suffering from the same ambiguity as before, i.e., the objective function to which it applies is determined by context. Moreover, in contrast with other algorithm notation, OPT does not denote a unique algorithm.

The classical theory refers to the study of algorithms satisfying various operating constraints which try to minimize, usually only approximately, the number of bins $\|\mathcal{P}(L)\|$ under the sum constraints $\ell(B_i) \leq 1$. *Dual bin packing* changes the sum constraints to $\ell(B_i) \geq 1$ and asks for a packing which *maximizes* the number of bins under these new constraints. Dual bin packing is often called bin covering, a term that we will use here. These combinatorial optimization problems are 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 problems in the classifications below.

To fix ideas, consider the Next Fit (NF), First Fit (FF), and Best Fit (BF) approximation algorithms for classical bin packing. Each algorithm packs all the items of L one by one in the sequence $a_1, a_2, \ldots, a_n$. NF packs items in B_1 until it encounters an item, say a_i, for which $a_i > 1 - \sum_{1 \leq j < i} a_j$; that is, a_i does not fit in the space left over by $a_1, \ldots, a_{i-1}$. At that point B_1 is *closed* in the sense that no further items can be packed in B_1, and a_i is placed as the first item in B_2. This bin-by-bin process repeats, packing the items $a_i, a_{i+1}, \ldots, a_n$ into $B_2, B_3, \ldots$, and continues until no items remain to be packed. The bin being packed at any given step is called the *open* bin. Under FF and BF all bins remain open throughout the packing process. At the i-th step under FF, a_i is packed in the lowest indexed bin with sufficient space (of course, this may have to be the empty bin just beyond the last nonempty bin). At the i-th step under BF, a_i is packed into a bin in which it fits best, i.e., with the least space left over. In case two or more bins satisfy this criterion, the lowest indexed of these bins is chosen.

Another dual of classical bin packing, called *multiprocessor* or *makespan scheduling*, takes the number, m, of bins to be constant and minimizes the capacity b such

that L can be packed into m bins of capacity b; again, the norm $\|\mathcal{P}(L)\| = mb$ is minimized, but in this case via b for fixed m. List scheduling (LS) is a classical algorithm for this problem and is organized like WORST FIT (a misnomer in the makespan context): the next item to be packed is put in a least-full bin, with ties resolved in favor of lower indexed bins.

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). Our incursion into scheduling problems will be limited to the most elementary duals and applications of bin packing problems, such as the one above.

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 optimal performance. (Detailed definitions will be provided shortly.) The term *performance guarantee* puts a more positive slant on results of this type[2] . Probability models also enjoy wide use, and are growing in popularity, as they bring out typical, *average-case* behavior rather than the, normally quite rare, worst-case behavior. In probabilistic or stochastic 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 difficulties inherent to these problems, even for elementary algorithms, one must often settle for weaker results, such as bounds on tail probabilities and asymptotic (large-n) estimates of expected values.

An analysis combining aspects of both the combinatorial and probabilistic approaches is that of *stochastic bin packing*, in which a typical problem is to find a packing algorithm that optimizes the expected value of some performance measure. These problems are almost always substantially more difficult extensions of problems that are already quite difficult. The classification scheme will have very few opportunities to cite such results.

The scheme for classifying problems and solutions will take the form of five fields: arena, objective function, class of algorithms, results, and constraints. The arena field describes the nature of the bins[3], such as whether they have variable capacities; the objective function to be minimized or maximized under sum constraints refers to the number of bins of fixed capacity, the capacity of a fixed number of bins, etc.; the class of algorithms refers to paradigms such as online, offline, bounded space, etc. as well as to algorithmic approaches such as *grouping* and *fitting*, to be described in Section 3; the results field specifies performance in terms of absolute or asymptotic worst case ratios, problem complexity, etc.; and constraints refer

[2]So also does the term *competitive analysis*, which usually refers to a worst case analysis comparing an on-line approximation algorithm with an optimal offline algorithm.

[3]The present classification of one dimensional problems will eventually be extended to higher dimensions, in which case the arena field will also specify problem dimensionality (e.g., packing 2-dimensional bins and strip packing.

to limitations in problem parameters, such as a minimum placed on item sizes, a restriction of all data to be integers, and so on. The classification scheme is intended for general use as a compact means for referring to packing problems; however, in the entries of the bibliography, the classification will be supplemented by a brief description of the algorithms studied and the results (typically, but not always, bounds of some kind) derived for the algorithms.

In what follows, Section 2 covers typical results and performance measures, Section 3 describes fundamental algorithms, and then Section 4 contains the details of the classification scheme. The many annotated examples in Section 5 are meant to familiarize the reader with classification criteria and their limitations.

An updated, classified bibliography with a search engine will be available at http://www.inf.u-szeged.hu/∼csirik.

2 Results

There are many forms results take, but the most common in combinatorial analysis are *performance ratios* or *guarantees*, which give the performance of an approximation algorithm relative to an optimal algorithm. Hereafter, dependence of performance ratios on α means that all item sizes satisfy $a_i \leq \alpha$; this dependence is omitted if there is no upper bound on item size, i.e. $\alpha = b$. For classical bin packing, the *asymptotic* worst-case ratio (or bound) for algorithm A is defined as

$$R_A^{\infty}(\alpha) := \limsup_{k\to\infty} R_A^{(k)}(\alpha)$$

with

$$R_A^{(k)}(\alpha) := \sup_{L:OPT(L)=k} \left\{ \frac{A(L)}{k} \right\}$$

where $OPT(L)$ refers to the optimal offline result. A less formal but more instructive definition describes $R_A^{\infty}(\alpha)$ as the smallest multiplicative constant such that for some additive constant $K < \infty$,

$$A(L) \leq R_A^{\infty}(\alpha) \cdot OPT(L) + K$$

for all L.

The *absolute* worst-case ratio is simply

$$R_A(\alpha) := \sup_L \left\{ \frac{A(L)}{OPT(L)} \right\}.$$

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.

The performance guarantees for covering have a complementary form. The asymptotic ratio is

$$R_A^\infty := \liminf_{k\to\infty} R_A(k)$$

where

$$R_A(k) := \inf_{L:OPT(L)=k} \left\{ \frac{A(L)}{k} \right\}$$

and the absolute ratio is

$$R_A := \inf_L \left\{ \frac{A(L)}{OPT(L)} \right\}$$

Note that the covering ratios are bounded above by 1; the better algorithms have the larger ratios.

Similar performance guarantees are defined for scheduling and a number of other problems. As can be seen, the ratio notation above is generic; the context will determine which definition is in force. When all item sizes are at most the item size parameter α, these bounds are denoted by $R_A^\infty(\alpha)$ and $R_A(\alpha)$.

The determination of time complexities of fundamental algorithms and their extensions or adaptations is usually routine. The analysis of parallel algorithms for computing packings is an example where deriving time complexities is not routine. 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 [9], denoted by PTAS (or FPTAS). In simplified terms, a typical form of such results is illustrated by: "Algorithm A produces packings with $O(\epsilon)$ expected wasted space and has a running time polynomial in $1/\epsilon$."

Average-case results may be in the form of expected ratios like $\mathbf{E}R_A(L)$ or simply expected performance $\mathbf{E}A(L)$, usually in terms of $\mathbf{E}OPT(L)$. (These comparisons need not be the same of course.) In many cases, tails of the distributions are estimated in the process of deriving estimates for expected values.

3 Fundamental algorithms

A number of such algorithms will be incorporated directly into the classification notation. These include the FIT algorithms FF, BF, and WF which we have already described. In some cases only the algorithmic approach or structure will be described, with extensive details omitted. The four structures most often used in defining algorithms are described below.

3.1 Fitting algorithms

These refer not only to those just mentioned, but also their offline decreasing counterparts denoted by NFD, FFD, BFD, and WFD, where the D stands for decreasing.

In each case, the algorithm begins with an ordering of L by decreasing item size. The respective algorithms are then applied to the reordered list. The notation for the corresponding *increasing* counterparts simply replaces the D by an I.

Bounded-space algorithms are a subcategory of fitting algorithms and are specified in many cases by a fitting rule, either FF or BF, and a closing rule. The closing rule is invoked when the next item to be packed does not fit into any of the open bins, in which case one of the open bins must be closed and a new bin opened. The choices for the bin to close are the lowest indexed bin (the First bin) and a bin with the highest level (a Best bin).

3.2 Grouping algorithms

Grouping is a standard technique that has been studied at great length with many variations. Essentially, it refers to schemes that pack/schedule items based on group membership, where groups are defined by item size. A primary example called HARMONIC and denoted by H_k is based on a partition of the interval $(0, 1]$ into k subintervals, where the partitioning points are $1/2, 1/3, \ldots, 1/k$. Each of these subintervals corresponds to a different group, and each has its own open bin; items belonging to a given group/subinterval are packed only into the corresponding open bin. If a new item arrives that does not fit into the open bin of its group, the bin is closed and a new bin of that type is opened. Thus, the packing of items in each group is an NF packing. Grouping has been defined on other than the H_k intervals, and it has been combined with various greedy fitting algorithms. HARMONIC has received so much coverage in the literature that we adopt, along with the FIT acronyms, the symbol H_k as part of the notation.

3.3 Iterative algorithms

Iterating an algorithm designed for good performance under one objective function may be an effective algorithm under another objective function. For example, consider approximation algorithms for the problem of minimizing schedule makespans. One could iterate BFD on an increasing sequence of "candidate" makespans (bin capacities) until one is tried with success, which then yields the desired approximation of the minimum makespan. For iterative versions of fundamental algorithms we use the prefix I. Thus, the algorithm just mentioned would be called IBFD. In a similar approach, IWFD could be used as an approximation algorithm for classical bin packing.

3.4 Limiting item sizes or the number packed per bin

If the number of distinct item sizes is limited, to N say, then when N is relatively small, substantial improvements in algorithm design and performance are possible. Moreover, finite (if not actually small) N loses no generality in practice. This assumption leads naturally to integer-program formulations. For example, consider

classical bin packing and define a *configuration* as any subset of items (with replication allowed) with a total size at most 1. Let C_{jk} be the number of items of the j-th size in the k-th configuration, and let $t_k \equiv t_k(\mathcal{P})$ be the number of bins of $\mathcal{P}$ with the k-th configuration. If there is a total of M possible configurations, and if there are m_j items of the j-th size in an instance I of the bin packing problem, then finding the size (norm) of an optimal packing is solving the following integer program for I : *minimize* $\sum_{k=1}^{M} t_k$ *subject to* $\sum_{k=1}^{M} t_k C_{jk} = m_j,\ j = 1, \ldots, N,$ *and* $t_k \geq 0,\ k = 1, \ldots M.$

Limiting the number of items per bin is a similar restriction, one that has often been used to greatly simplify average-case analysis. For example, for classical bin packing, there are simple algorithms packing at most 2 items per bin which yield smallest possible asymptotic estimates of expected wasted space, when item sizes are drawn independently and uniformly at random from $[0, 1]$.

4 Classification scheme

The notation takes the form

arena|objective_function|algorithm_class|results|constraints

This section gives the current lists of entries for each field, with definitions where needed. The special terms or abbreviations adopted for entries will be given in bold face.

4.1 Arena

The basic arena as a sequence of one-dimensional bins has already been described. When sum constraints apply, and all bins have the same size b, then the arena field will be empty. When this field is not empty, terms like the following will appear.

1. **variable** b_i means that there is more than one bin size and that there is an unlimited supply for each size.

2. **open_end** refers to problems in which sum constraints are relaxed as follows: bin B_i can always accommodate an item if $\ell(B_i) < b_i$ but it is closed as soon as $\ell(B_i) \geq b_i$. Other notions of exceeding bin capacity will fall under the general term **overfill**.

4.2 Objective function

This function will most often be implicit in a term adopted for the corresponding combinatorial optimization problem.

1. **pack** refers to the classical problem of minimizing $\|\mathcal{P}(L)\|$ subject to the sum constraints $\ell(B_i) \leq 1$.

2. **makespan** refers to the problem of minimizing the common bin capacity needed to pack L into a given number of bins. The *bin-stretching* problem is a special case of the makespan problem in which the value of the bin size in the optimal packing is known in advance. For this problem, the term *stretch* will be appended to the performance-guarantee notation.

3. **deadline** abbreviates *deadline scheduling* and refers to the problem of finding schedules in which a maximum cardinality subset of the tasks in L finish by a given deadline (capacity) b on a given number m of processors.

4. **pack_cover** refers to the dual bin packing problem of maximizing $\|\mathcal{P}(L)\|$ subject to the dual constraints $\ell(B_i) \geq 1$.

5. **schedule_cover** refers to the dual makespan scheduling problem of maximizing the makespan b for fixed m such that $\ell(B_i) \geq b$.

   In principle, there are covering versions of deadline scheduling as well, but we have encountered no research on these problems. One such problem is:

6. **deadline_cover** names the problem of minimizing the total size of the subset of tasks needed to cover a given number m of processors with a given deadline b.

4.3 Algorithm class

1. **offline** algorithms have no constraints beyond the intrinsic sum constraints; an offline algorithm simply maps the entire list L into a packing $\mathcal{P}(L)$. Effectively, all items are known in advance, so the ordering of L plays no role.

2. **online** 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 $a_1, \ldots, a_i$. Note that NF, FF, and BF are all online.

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. Note that FF and BF are not bounded space algorithms, but NF is, with $k = 1$. A more precise definition and further discussion of these algorithms appear later.

4. **linear-time** algorithms have $O(n)$ running time. In fact, a more precise statement can be made: all such algorithms classified here take constant time to pack each item. NF is clearly a linear-time algorithm, but FF and BF are not.

   The three characterizations above are orthogonal. But the literature suggests that the following convention will allow us to use one term in classifying algorithms most of the time: *Bounded space implies linear time and linear time implies online.* Exceptions will be noted explicitly; below (under **repack**) we will see how offline algorithms can be linear time.

5. **greedy.** Any algorithm in a broad class of algorithms variously called reasonable, fair, any-fit, or greedy is required to pack the current item into an open bin with sufficient space, in case such a bin exists; in particular, it can not choose to open a new bin in this case. We use the term greedy exclusively to describe such algorithms. Scheduling algorithms satisfying a similar constraint are sometimes called *conservative* or *work conserving*.

6. **repack.** There have been a number of studies devoted to packing problems which allow the repacking (possibly limited in some way) of items – moving an item, say a_i, from one bin to another based on the sizes of items $a_j,\ j > i$.

7. **dynamic** packing introduces the time dimension; an instance L of this problem consists of a sequence of triples (a_i, r_i, d_i) with r_i and d_i denoting arrival and departure times, respectively. Under packing algorithm A, $A(L,t)$ denotes the number of bins occupied at time t, i.e. the number of bins occupied by those items a_i for which $r_i < t < d_i$.

Conventions: Along with the algorithm class, the algorithm will be specified when possible. In many cases, the algorithm will be an adaptation or variant of some well-known algorithm, like FF for example, in which case the specification will have the form **FF variant**.

4.4 Results

Almost all results fall into the broad classes mentioned in Section 2.

1. Asymptotic worst case ratios, where $\mathbf{R}_{\mathbf{A}}^{\infty}$ is the general entry with A specified where appropriate.

2. Absolute worst case, with $\mathbf{R}_{\mathbf{A}}$ being the entry.

3. Average case: A probabilistic analysis, usually leading only to expected values, is indicated. The entries adopt standard notation such as $\mathbf{E}A(L_n)$ or $\Pr\{A(L_n) > x\}$. In parentheses, a distribution or class of distributions will be given. Examples include $U(0,\alpha)$ (the uniform distribution on $[0,\alpha]$) and $\Delta(0,\alpha)$ (the triangular distribution on $[0,\alpha]$, but if no distribution is specified then it is assumed to be general. Standard terms like **unimodal** and **decreasing** (referring to a density function), etc. will be encountered. Item sizes are assumed to be independent random variables in all cases, unless stated otherwise.

4. Where possible, complexity of the problem will be given in the standard notation of problem complexity.

5. Complexity of the algorithm refers to running-time complexity and will be signalled by the entry **running-time**.

Conventions: 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), and a paper given an average case classification may also have worst case results; here also, both classifications will be noted only if both worst-case and average-case analysis play major roles in the paper. Approximation schemes are classified as complexity results and have entries like **PTAS, FPTAS** as noted earlier.

4.5 Constraints

These typically introduce further limitations on the problem instance, or further properties of the algorithm classification.

1. **mutex** stands for mutual exclusion and introduces constraints in the form of a sequence of pairs (a_i, a_j), $i \neq j$, signifying that a_i and a_j can not be put in the same bin.

2. **items/bin** $\leq k$ gives a bound on the number of items that can be packed in a bin.

3. $a_i \leq \alpha$ **or** $a_i \geq \alpha$. These denote bounds on item sizes, the former being far more common in the literature. In the former case α is usually part of the result notation (see the next subsection) so in these cases, it is omitted from the classification. *Throughout the bibliography, the symbol α is reserved for this purpose.* These cases are often called parametric cases in the literature.

   A constraint that may refer as much to analysis as to algorithm design calls for discrete sets of items; as such it is not a significant practical constraint.

4. **discrete** which means that item sizes are all multiples of $1/k$ with $b = 1$. Equivalently, the bin size could be taken as some integer b and item sizes restricted to the set $\{1, \ldots, b\}$.

5. **restricted sizes** refers to the problem where the number of different item sizes is finite.

6. The symbol ***** refers to features or properties not classifiable within the scheme.

Conventions: There are further interesting extensions which occur only in a few papers. In these cases we will use a special notation; a short description in each case will be given as a remark.

5 Examples

The following examples should help familiarize the reader with the classification technique.

1. Reference [4] gives an average-case analysis of the classical, bounded-space Next Fit algorithm for bin packing:

   $$|\text{pack}|\text{bounded\_space}|\mathbf{E}NF(L_n), U(0,1)$$

   Result: $\mathbf{E}NF(L) = \frac{4}{3}\mathbf{E}OPT(L) + O(1)$, where item sizes are independent draws from U(0,1).

   Recall that the empty arena component implies that all bin levels are bounded above by 1, the common bin size.

2. The classification of [13] shows a nonempty arena field:

   $$\text{variable } b_i|\text{pack}|\text{offline}|\ \text{PTAS}$$

3. The classification of [12] gives another such example:

   $$\text{open\_end}|\text{pack}|\text{online; offline}|R_A^\infty \text{ bound; FPTAS}$$

   Results: For the open-end bin packing problem any online algorithm must have an asymptotic worst-case ratio of at least 2. Next Fit achieves this ratio. There is a fully polynomial approximation scheme for this problem.

4. The classification of [2] illustrates a makespan problem:

   $$|\text{makespan}|\text{online}|R_A \text{ stretch}$$

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

5. Reference [3] shows a deadline objective function:

   $$|\text{deadline}|\text{offline; WFI, FFI}|R_A$$

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

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

   $$|\text{pack\_cover}|\text{online}|R_A^\infty \text{ bound}$$

   Results: Asymptotic bound: $R_A^\infty \leq 1/2$ for any online algorithm A. There exists an asymptotically optimal online algorithm.

7. The classification of [7] is

$$|\text{pack}|\text{offline; combined BF, FFD variant}|R_A^\infty$$

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:

$R_{B2F}^\infty = 5/4, \quad \frac{227}{195} \le R_{CBF}^\infty \le \frac{6}{5}$

Note that the word 'variant' may be simplistic in that it occasionally hides details of relatively complicated algorithms.

8. Reference [8] shows an example for combination of two algorithm classes:

$$|\text{pack}|\text{bounded\_space, repack}|R_A$$

Algorithm: REP_3: an adaptation of FFD using three open bins at any time.

Result: $R_{REP_3}^\infty \approx 1.69...$.

9. The classification of [11] illustrates a constraint:

$$|\text{pack}|\text{offline; FF variant}|R_A^\infty|\text{items/bin} \le k$$

Result:

$$\left(\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil\right) \le R_{FF_k}^\infty \le \left(\frac{27}{10} - \frac{24}{10k}\right),$$

where FF_k is the obvious adaptation of FF.

10. For [14], the classification mentions yet another constraint:

$$|\text{pack}|\text{bounded\_space; } \text{H}_k \text{ variant}|R_A^\infty|O(\log k) \text{ open bins}$$

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

11. Classification of [1] aggregates several results:

$$|\text{pack\_cover}|\text{offline, online; NF, FFD, IWFD variants}|R_A^\infty$$

Algorithms: Adaptation of Next Fit called DNF (a new bin is opened when the current open bin, say B, first overflows with item, say a_i, but in this case a_i stays in B); of First Fit Decreasing with a parameter r (FFD_r); and of an iterated version of Worst Fit (IWFD).

Results:

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

References

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

$$|\text{pack\_cover}|\text{offline}, \text{online}; \text{NF, FFD, IWFD variants}|R_A^\infty$$

[2] Y. Azar and O. Regev. On-line bin-stretching. *Theor. Comp. Sci.*, 268:17–41, 2001.

$$|\text{makespan}|\text{online}|R_A \text{ stretch}$$

[3] E. G. Coffman, Jr., J. Y. Leung, and D. W. Ting. Bin packing: maximizing the number of pieces packed. *Acta Informatica*, 9:263–271, 1978.

$$|\text{deadline}|\text{offline}; \text{WFI, FFI}|R_A$$

[4] E. G. Coffman, Jr., K. So, M. Hofri, and A. C. Yao. A stochastic model of bin-packing. *Inf. and Cont.*, 44:105–115, 1980.

$$|\text{pack}|\text{bounded\_space}|\mathbf{E}NF(L_n), U(0,1)$$

[5] J. Csirik and V. Totik. On-line algorithms for a dual version of bin packing. *Disc. Appl. Math.*, 21:163–167, 1988.

$$|\text{pack\_cover}|\text{online}|R_A^\infty \text{ bound}$$

[6] H. Dyckhoff. A typology of cutting and packing problems. *Eur. J. Oper. Res.*, 44:145–159, 1990.

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

$$\text{pack}|\text{offline; combined BF, FFD variant}|R_A^\infty$$

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

$$|\text{pack}|\text{bounded\_space}; \text{repack}|R_A$$

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, New York, New York, 1979.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals Disc. Math.*, 5:287–326, 1979.

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

$$|\text{pack}|\text{offline}; \text{FF variant}|R_A^\infty|\text{items/bin} \leq k$$

[12] J. Y.-T. Leung, M. Dror, and G. H. Young. A note on an open-end bin packing problem. *J. of Scheduling*, 4:201–207, 2001.

$$\text{open\_end}|\text{pack}|\text{online}; \text{offline}|R_A^\infty \text{ bound; FPTAS}$$

[13] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput.*, 16:149–161, 1988.

$$\text{variable } b_i|\text{pack}|\text{offline}| \text{ PTAS}$$

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

$$|\text{pack}|\text{bounded\_space; } \mathrm{H}_k \text{ variant}|R_A^\infty|O(\log k) \text{ open bins}$$