Bin packing problem

From Wikipedia, the free encyclopedia

Not to be confused with bin picking.

Covering/packing-problem pairs	
Covering problems	**Packing problems**
Minimum set cover	Maximum set packing
Minimum edge cover	Maximum matching
Minimum vertex cover	Maximum independent set
Bin covering	**Bin packing**
Polygon covering	Rectangle packing

V · T · E

The **bin packing problem**[1][2][3][4] is an optimization problem, in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that minimizes the number of bins used. The problem has many applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups in media and technology mapping in FPGA semiconductor chip design.

Computationally, the problem is NP-hard, and the corresponding decision problem - deciding if items can fit into a specified number of bins - is NP-complete. Despite its worst-case hardness, optimal solutions to very large instances of the problem can be produced with sophisticated algorithms. In addition, many approximation algorithms exist. For example, the first fit algorithm provides a fast but often non-optimal solution, involving placing each item into the first bin in which it will fit. It requires Θ(*n* log *n*) time, where *n* is the number of items to be packed. The algorithm can be made much more effective by first sorting the list of items into decreasing order (sometimes known as the first-fit decreasing algorithm), although this still does not guarantee an optimal solution, and for longer lists may increase the running time of the algorithm. It is known, however, that there always exists at least one ordering of items that allows first-fit to produce an optimal solution.[5]

There are many variations of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. The bin packing problem can also be seen as a special case of the cutting stock problem. When the number of bins is restricted to 1 and each item is characterised by both a volume and a value, the problem of maximizing the value of items that can fit in the bin is known as the knapsack problem.

A variant of bin packing that occurs in practice is when items can share space when packed into a bin. Specifically, a set of items could occupy less space when packed together than the sum of their individual sizes. This variant is known as VM packing[6] since when virtual machines (VMs) are packed in a server, their total memory requirement could decrease due to pages shared by the VMs that need only be stored once. If items can share space in arbitrary ways, the bin packing problem is hard to even approximate. However, if the space sharing fits into a hierarchy, as is the case with memory sharing in virtual machines, the bin packing problem can be efficiently approximated.

Another variant of bin packing of interest in practice is the so-called online bin packing. Here the items of different volume are supposed to arrive sequentially, and the decision maker has to decide whether to select and pack the currently observed item, or else to let it pass. Each decision is without recall. In contrast, offline bin packing allows rearranging the items in the hope of achieving a better packing once additional items arrive. This of course requires additional storage for holding the items to be rearranged.

Contents [hide]

Formal statement [edit]

In *Computers and Intractability*[7]:226 Garey and Johnson list the bin packing problem under the reference [SR1]. They define its decision variant as follows.

Instance: Finite set I of items, a size $s(i) \in \mathbb{Z}^+$ for each $i \in I$, a positive integer bin capacity B, and a positive integer K.

Question: Is there a partition of I into disjoint sets $I_1, \dots, I_K$ such that the sum of the sizes of the items in each I_j is B or less?

Note that in the literature often an equivalent notation is used, where $B = 1$ and $s(i) \in \mathbb{Q} \cap (0,1]$ for each $i \in I$. Furthermore, research is mostly interested in the optimization variant, which asks for the smallest possible value of K. A solution is *optimal* if it has minimal K. The K-value for an optimal solution for a set of items I is denoted by $\mathrm{OPT}(I)$ or just OPT if the set of items is clear from the context.

A possible integer linear programming formulation of the problem is:

$$\text{minimize } K = \sum_{j=1}^{n} y_j$$

$$\text{subject to } K \ge 1,$$

$$\sum_{i \in I} s(i) x_{ij} \le B y_j, \quad \forall j \in \{1,\dots,n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in I$$

$$y_j \in \{0,1\}, \quad \forall j \in \{1,\dots,n\}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in I \, \forall j \in \{1,\dots,n\}$$

where $y_j = 1$ if bin j is used and $x_{ij} = 1$ if item i is put into bin j.[8]

Hardness of bin packing [edit]

The bin packing problem is strongly NP-complete. This can be proven by reducing the strongly NP-complete 3-partition problem to bin packing.[7]

Furthermore, there can be no approximation algorithm with absolute approximation ratio smaller than $3/2$ unless $P = NP$. This can be proven by a reduction from the partition problem:[9] given an instance of Partition where the sum of all input numbers is 2 *T*, construct an instance of bin-packing in which the bin size is *T*. If there exists an equal partition of the inputs, then the optimal packing needs 2 bins; therefore, every algorithm with approximation ratio smaller than 3/2 must return less than 3 bins, which must be 2 bins. In contrast, if there is no equal partition of the inputs, then the optimal packing needs at least 3 bins.

On the other hand, bin packing is solvable in pseudo-polynomial time for any fixed number of bins K, and solvable in polynomial time for any fixed bin capacity B.[7]

Approximation algorithms for bin packing [edit]

To measure the performance of an approximation algorithm there are two approximation ratios considered in the literature. For a given list of items L the number $A(L)$ denotes the number of bins used when algorithm A is applied to list L, while $\mathrm{OPT}(L)$ denotes the optimum number for this list. The absolute worst-case performance ratio R_A for an algorithm A is defined as

$$R_A \equiv \inf\{r \ge 1 : A(L)/\mathrm{OPT}(L) \le r \text{ for all lists } L\}.$$

On the other hand, the asymptotic worst-case ratio R_A^∞ is defined as

$$R_A^\infty \equiv \inf\{r \ge 1 : \exists N > 0, A(L)/\mathrm{OPT}(L) \le r \text{ for all lists } L \text{ with } \mathrm{OPT}(L) \ge N\}.$$

Equivalently, R_A^∞ is the smallest number such that, for some constant *K*, for all lists *L*:[4]

$$A(L) \le R_A^\infty \cdot OPT(L) + K.$$

Additionally, one can restrict the lists to those for which all items have a size of at most α. For such lists, the bounded size performance ratios are denoted as $R_A(\text{size} \le \alpha)$ and $R_A^\infty(\text{size} \le \alpha)$.

Approximation algorithms for bin packing can be classified into two categories:

1. Online heuristics, that consider the items in a given order and place them one by one inside the bins. These heuristics are also applicable to the offline version of this problem.
2. Offline heuristics, that modify the given list of items e.g. by sorting the items by size. These algorithms are no longer applicable to the online variant of this problem. However, they have an improved approximation guarantee while maintaining the advantage of their small time-complexity. A sub-category of offline heuristics is asymptotic approximation schemes. These algorithms have an approximation guarantee of the form $(1+\varepsilon)\mathrm{OPT}(L) + C$ for some constant that may depend on $1/\varepsilon$. For an arbitrarily large $\mathrm{OPT}(L)$ these algorithms get arbitrarily close to $\mathrm{OPT}(L)$. However, this comes at the cost of a (drastically) increased time complexity compared to the heuristical approaches.

Online heuristics [edit]

In the online version of the bin packing problem, the items arrive one after another and the (irreversible) decision where to place an item has to be made before knowing the next item or even if there will be another one. A diverse set of offline and online heuristics for bin-packing have been studied by David S. Johnson on his Ph.D. thesis.[10]

Single-class algorithms [edit]

There are many simple algorithms that use the following general scheme:

- For each item in the input list:
  1. If the item fits into one of the currently open bins, then put it in one of these bins;
  2. Otherwise, open a new bin and put the new item in it.

The algorithms differ in the criterion by which they choose the open bin for the new item in step 1 (see the linked pages for more information):

- **Next Fit (NF)** always keeps a single open bin. When the new item does not fit into it, it closes the current bin and opens a new bin. Its advantage is that it is a bounded-space algorithm, since it only needs to keep a single open bin in memory. Its disadvantage is that its asymptotic approximation ratio is 2. In particular, $NF(L) \le 2 \cdot \mathrm{OPT}(L) - 1$, and for each $N \in \mathbb{N}$ there exists a list L such that $\mathrm{OPT}(L) = N$ and $NF(L) = 2 \cdot \mathrm{OPT}(L) - 2$.[10] Its asymptotic approximation ratio can be somewhat improved based on the item sizes: $R_{NF}^\infty(\text{size} \le \alpha) \le 2$ for all $\alpha \ge 1/2$ and $R_{NF}^\infty(\text{size} \le \alpha) \le 1/(1-\alpha)$ for all $\alpha \le 1/2$. For each algorithm A that is an AnyFit-algorithm it holds that $R_A^\infty(\text{size} \le \alpha) \le R_{NF}^\infty(\text{size} \le \alpha)$.
- **Next-k-Fit (NkF)** is a variant of Next-Fit, but instead of keeping only one bin open, the algorithm keeps the last k bins open and chooses the first bin in which the item fits. Therefore, it is called a *k-bounded space* algorithm.[11] For $k \ge 2$ the NkF delivers results that are improved compared to the results of NF, however, increasing k to constant values larger than 2 improves the algorithm no further in its worst-case behavior. If algorithm A is an AlmostAnyFit-algorithm and $m = \lfloor 1/\alpha \rfloor \ge 2$ then $R_A^\infty(\text{size} \le \alpha) \le R_{N2F}^\infty(\text{size} \le \alpha) = 1 + 1/m$.[10]
- **First-Fit (FF)** keeps all bins open, in the order in which they were opened. It attempts to place each new item into the *first* bin in which it fits. Its approximation ratio is $FF(L) \le \lfloor 1.7\mathrm{OPT} \rfloor$, and there is a family of input lists L for which $FF(L)$ matches this bound.[12]
- **Best-Fit (BF)**, too, keeps all bins open, but attempts to place each new item into the bin with the *maximum* load in which it fits. Its approximation ratio is idenetical to that of FF, that is: $BF(L) \le \lfloor 1.7\mathrm{OPT} \rfloor$, and there is a family of input lists L for which $BF(L)$ matches this bound.[13]
- **Worst-Fit (WF)** attempts to place each new item into the bin with the *minimum* load. It can behave as badly as Next-Fit, and will do so on the worst-case list for that $NF(L) = 2 \cdot \mathrm{OPT}(L) - 2$. Furthermore, it holds that $R_{WF}^\infty(\text{size} \le \alpha) = R_{NF}^\infty(\text{size} \le \alpha)$. Since WF is an AnyFit-algorithm, there exists an AnyFit-algorithm such that $R_{AF}^\infty(\alpha) = R_{NF}^\infty(\alpha)$.[10]
- **Almost Worst-Fit (AWF)** attempts to place each new item inside the *second most empty* open bin (or emptiest bin if there are two such bins). If it does not fit, it tries the most empty one. It has an asymptotic worst-case ratio of $17/10$.[10]

In order to generalize these results, Johnson introduced two classes of online heuristics called *any-fit algorithm* and *almost-any-fit* algorithm:[4]:470

- In an **AnyFit (AF)** algorithm, if the current nonempty bins are $B_1,\dots,B_j$, then the current item will not be packed into B_{j+1} unless it does not fit in any of $B_1,\dots,B_j$. The FF, WF, BF and AWF algorithms satisfy this condition. Johnson proved that, for any AnyFit algorithm A and any α:

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

- In an **AlmostAnyFit (AAF)** algorithm, if the current nonempty bins are $B_1,\dots,B_j$, and of these bins, B_k is the unique bin with the smallest load, then the current item will not be packed into B_k, unless it does not fit into any of the bins to its left. The FF, BF and AWF algorithms satisfy this condition, but WF does not. Johnson proved that, for any AAF algorithm A and any α:

$R_A^\infty(\alpha) = R_{FF}^\infty(\alpha)$ In particular: $R_A^\infty = 1.7$.

Refined algorithms [edit]

Better approximation ratios are possible with heuristics that are not AnyFit. These heuristics usually keep several classes of open bins, devoted to items of different size-ranges (see the linked pages for more information):

- **Refined-first-fit bin packing** (RFF) partitions the item sizes into four ranges: $(1/2,1]$, $(2/5,1/2]$, $(1/3,2/5]$, and $(0,1/3]$. Similarly, the bins are categorized into four classes. The next item $i \in L$ is first assigned to its corresponding class. Inside that class, it is assigned to a bin using first-fit. Note that this algorithm is not an Any-Fit algorithm since it may open a new bin despite the fact that the current item fits inside an open bin. This algorithm was first presented by Andrew Chi-Chih Yao,[14] who proved that it has an approximation guarantee of $RFF(L) \le (5/3) \cdot \mathrm{OPT}(L) + 5$ and presented a family of lists L_k with $RFF(L_k) = (5/3)\mathrm{OPT}(L_k) + 1/3$ for $\mathrm{OPT}(L) = 6k+1$.
- **Harmonic-k** partitions the interval of sizes $(0,1]$ based on a Harmonic progression into $k-1$ pieces $I_j := (1/(j+1), 1/j]$ for $1 \le j < k$ and $I_k := (0, 1/k]$ such that $\bigcup_{j=1}^{k} I_j = (0,1]$.

  This algorithm was first described by Lee and Lee.[15] It has a time complexity of $\mathcal{O}(|L|\log(|L|))$ and at each step, there are at most k open bins that can be potentially used to place items, i.e., it is a k-bounded space algorithm. For $k \to \infty$, its approximation ratio satisfies $R_{Hk}^\infty \approx 1.6910$, and it is asymptotically tight.
- **Refined-harmonic** combines ideas from Harmonic-k with ideas from Refined-First-Fit. It places the items larger than $1/3$ similar as in Refined-First-Fit, while the smaller items are placed using Harmonic-k. The intuition for this strategy is to reduce the huge waste for bins containing pieces that are just larger than $1/2$. This algorithm was first described by Lee and Lee.[15] They proved that for $k = 20$ it holds that $R_{RH}^\infty \le 373/228$.

General lower bounds for online algorithms [edit]

Yao[14] proved 1980 that there can be no online algorithm with asymptotic competitive ratio smaller than $3/2$. Brown[16] and Liang[17] improved this bound to 1.53635. Afterward, this bound was improved to 1.54014 by Vliet.[18] In 2012, this lower bound was again improved by Békési and Galambos[19] to $248/161 \approx 1.54037$.

Comparison table [edit]

Algorithm	Approximation guarantee	Worst case list L	Time-complexity
Next-fit (NF)	$NF(L) \le 2 \cdot \mathrm{OPT}(L) - 1$ [10]	$NF(L) = 2 \cdot \mathrm{OPT}(L) - 2$ [10]	$\mathcal{O}(\|L\|)$
First-fit (FF)	$FF(L) \le \lfloor 1.7\mathrm{OPT}(L) \rfloor$ [12]	$FF(L) = \lfloor 1.7\mathrm{OPT}(L) \rfloor$ [12]	$\mathcal{O}(\|L\| \log(\|L\|))$ [10]
Best-fit (BF)	$BF(L) \le \lfloor 1.7\mathrm{OPT}(L) \rfloor$ [13]	$BF(L) = \lfloor 1.7\mathrm{OPT}(L) \rfloor$ [13]	$\mathcal{O}(\|L\| \log(\|L\|))$ [10]
Worst-Fit (WF)	$WF(L) \le 2 \cdot \mathrm{OPT}(L) - 1$ [10]	$WF(L) = 2 \cdot \mathrm{OPT}(L) - 2$ [10]	$\mathcal{O}(\|L\| \log(\|L\|))$ [10]
Almost-Worst-Fit (AWF)	$R_{AWF}^\infty \le 17/10$ [10]	$R_{AWF}^\infty = 17/10$ [10]	$\mathcal{O}(\|L\| \log(\|L\|))$ [10]
Refined-First-Fit (RFF)	$RFF(L) \le (5/3) \cdot \mathrm{OPT}(L) + 5$ [14]	$RFF(L) = (5/3)\mathrm{OPT}(L) + 1/3$ (for $\mathrm{OPT}(L) = 6k+1$) [14]	$\mathcal{O}(\|L\| \log(\|L\|))$ [14]
Harmonic-k (Hk)	$R_{Hk}^\infty \le 1.69103$ for $k \to \infty$ [15]	$R_{Hk}^\infty \ge 1.69103$ [15]	$\mathcal{O}(\|L\| \log(\|L\|))$ [15]
Refined Harmonic (RH)	$R_{RH}^\infty \le 373/228 \approx 1.63597$ [15]		$\mathcal{O}(\|L\|)$ [15]
Modified Harmonic (MH)	$R_{MH}^\infty \le 538/33 \approx 1.61562$ [20]		
Modified Harmonic 2 (MH2)	$R_{MH2}^\infty \le 239091/148304 \approx 1.61217$ [20]		
Harmonic + 1 (H+1)		$R_{H+1}^\infty \ge 1.59217$ [21]	
Harmonic ++ (H++)	$R_{H++}^\infty \le 1.58889$ [21]	$R_{H++}^\infty \ge 1.58333$ [21]	

Offline algorithms [edit]

In the offline version of bin packing, the algorithm can see all the items before starting to place them into bins. This allows to attain improved approximation ratios.

Multiplicative approximation [edit]

The simplest technique used by offline algorithms is:

- Ordering the input list by descending size;
- Run an online algorithm on the ordered list.

Johnson[10] proved that any AnyFit algorithm A that runs on a list ordered by descending size has an asymptotic approximation ratio of

$$1.22 \approx \frac{11}{9} \le R_A^\infty \le \frac{5}{4} = 1.25.$$

Some algorithms in this family are (see the linked pages for more information):

- **First-fit-decreasing** (FFD) - orders the items by descending size, then calls First-Fit. Its approximation ratio is $FFD(I) = 11/9\mathrm{OPT}(I) + 6/9$, and this is tight.[22]
- **Next-fit-decreasing** (NFD) - orders the items by descending size, then calls Next-Fit. Its approximate ratio is slightly less than 1.7 in the worst case.[23] It has also been analyzed probabilistically.[24] Next-Fit packs a list and its inverse into the same number of bins. Therefore, Next-Fit-Increasing has the same performance as Next-Fit-Decreasing.[25]
- **Modified first-fit-decreasing** (MFFD)[26] - improves on FFD for items larger than half a bin by classifying items by size into four size classes large, medium, small, and tiny, corresponding to items with size > 1/2 bin, > 1/3 bin, > 1/6 bin, and smaller items respectively. Its approximation guarantee is $MFFD(I) \le (71/60)\mathrm{OPT}(I) + 1$.[27]

Fernandez de la Vega and Lueker[28] presented a PTAS for bin packing. For every $\varepsilon > 0$, their algorithm finds a solution with size at most $(1+\varepsilon)\mathrm{OPT} + 1$ and runs in time $\mathcal{O}(n\log(1/\varepsilon)) + \mathcal{O}_\varepsilon(1)$, where $\mathcal{O}_\varepsilon(1)$ denotes a function only dependent on $1/\varepsilon$. For this algorithm, they invented the method of *adaptive input rounding*: the input numbers are grouped and rounded up to the value of the maximum in each group. This yields an instance with a small number of different sizes, which can be solved exactly by checking all possible configurations.[29]

Additive approximation [edit]

Karmarkar and Karp[30] presented an algorithm with run-time polynomial in n and $1/\varepsilon$. Their algorithm finds a solution with size at most $\mathrm{OPT} + \mathcal{O}(\log^2(OPT))$.

Rothvoss[31] presented an algorithm that generates a solution with size at most $\mathrm{OPT} + O(\log(\mathrm{OPT}) \cdot \log\log(\mathrm{OPT}))$.

Hoberg and Rothvoss[32] improved this algorithm to generate a solution with size at most $\mathrm{OPT} + O(\log(\mathrm{OPT}))$. The algorithm is randomized, and its running-time is polynomial in the total number of items.

Comparison table [edit]

Algorithm	Approximation guarantee	Worst case instance
First-fit-decreasing (FFD)	$FFD(I) \le 11/9\mathrm{OPT}(I) + 6/9$ [22]	$FFD(I) = 11/9\mathrm{OPT}(I) + 6/9$ [22]
Modified-first-fit-decreasing (MFFD)	$MFFD(I) \le (71/60)\mathrm{OPT}(I) + 1$ [27]	$R_{MFFD}^\infty \ge 71/60$ [26]
Karmarkar and Karp[30]	$KK(I) \le OPT(I) + O(\log^2 OPT(I))$	
Rothvoss[31]	$HB(I) \le OPT(I) + O(\log OPT(I) \log\log OPT(I))$	
Hoberg and Rothvoss[32]	$HB(I) \le OPT(I) + O(\log OPT(I))$	

Exact algorithms [edit]

Martello and Toth[33] developed an exact algorithm for the 1-D bin-packing problem, called MTP. A faster alternative is the Bin Completion algorithm proposed by Korf in 2002[34] and later improved.[35]

A further improvement was presented by Schreiber and Korf in 2013.[36] The new Improved Bin Completion algorithm is shown to be up to five orders of magnitude faster than Bin Completion on non-trivial problems with 100 items, and outperforms the BCP (branch-and-cut-and-price) algorithm by Belov and Scheithauer on problems that have fewer than 20 bins as the optimal solution. Which algorithm performs best depends on problem properties like number of items, optimal number of bins, unused space in the optimal solution and value precision.

Goemans and Rothvoss[37] presented an algorithm for bin packing with *d* different item-sizes, where there can be multiple items with each size. Their algorithm is polynomial when *d* is fixed and all numbers are given in binary encoding.

Bin packing with cardinality constraints [edit]

There is a variant of bin packing in which there are cardinality constraints on the bins: each bin can contain at most *k* items, for some fixed integer *k*.

- Krause, Shen and Schwetman[38] introduce this problem as a variant of optimal job scheduling: a computer has some *k* processors. There are some *n* jobs that take unit time (1), but have different memory requirements. Each time-unit is considered a single bin. The goal is to use as few bins (=time units) as possible, while ensuring that in each bin, at most *k* jobs run. They present several heuristic algorithms that find a solution with at most $2OPT$ bins.
- Kellerer and Pferschy[39] present an algorithm with run-time $O(n^2 \log n)$, that finds a solution with at most $\lceil \frac{3}{2}\mathrm{OPT} \rceil$ bins. Their algorithm performs a binary search for OPT. For every searched value *m*, it tries to pack the items into 3*m*/2 bins.

Related problems [edit]

In the bin packing problem, the *size* of the bins is fixed and their *number* can be enlarged (but should be as small as possible).

In contrast, in the **multiway number partitioning** problem, the *number* of bins is fixed and their *size* can be enlarged. The objective is to find a partition in which the bin sizes are as nearly equal is possible (in the variant called **multiprocessor scheduling** problem or **minimum makespan** problem, the goal is specifically to minimize the size of the largest bin).

In the **inverse bin packing** problem,[40] both the number of bins and their sizes are fixed, but the item sizes can be changed. The objective is to achieve the minimum perturbation to the item size vector so that all the items can be packed into the prescribed number of bins.

In the **maximum resource bin packing** problem,[41] the goal is to *maximize* the number of bins used, such that, for some ordering of the bins, no item in a later bin fits in an earlier bin. In a dual problem, the number of bins is fixed, and the goal is to minimize the total number or the total size of items placed into the bins, such that no remaining item fits into an unfilled bin.

In the **bin covering problem**, the bin size is bounded *from below*: the goal is to *maximize* the number of bins used such that the total size in each bin is at least a given threshold.

In the **fair indivisible chore allocation** problem (a variant of **fair item allocation**), the items represents chores, and there are different people each of whom attributes a different difficulty-value to each chore. The goal is to allocate to each person a set of chores with an upper bound on its total difficulty-value (thus, each person corresponds to a bin). Many techniques from bin packing are used in this problem too.[42]

In the **guillotine cutting** problem, both the items and the "bins" are two-dimensional rectangles rather than one-dimensional numbers, and the items have to be cut from the bin using end-to-end cuts.

In the **selfish bin packing** problem, each item is a player who wants to minimize its cost.[43]

There is also a variant of bin packing in which the cost that should be minimized is not the number of bins, but rather a certain concave function of the number of items in each bin.[44]

Other variants are **two-dimensional bin packing**,[45] **three-dimensional bin packing**,[46] **bin packing with delivery**,[47]

Implementations [edit]

- **Online:** visualization of heuristics for 1D and 2D bin packing
- **Python:** the binpacking package contains greedy algorithms for solving two typical bin packing problems: (i) packing items into a constant number of bins, (ii) packing items into a low number of bins of constant size.[48]
- **C++**: The bin-packing package contains various greedy algorithms as well as test data.
- The OR-tools package contains bin packing algorithms in C++, with wrappers in Python, C# and Java.
- **C**: Implementation of 7 classic approximate bin packing algorithms in C with results and images
- **PHP**: PHP Class to pack files without exceeding a given size limit
- **Haskell**: An implementation of several bin packing heuristics, including FFD and MFFD.
- **C**: Fpart : open-source command-line tool to pack files (C, BSD-licensed)
- **C#**: Bin Packing and Cutting Stock Solver
- **Java**: caparf - Cutting And Packing Algorithms Research Framework, including a number of bin packing algorithms and test data.

References [edit]

1. ^ Martello, Silvano; Toth, Paolo (1990), "Bin-packing problem" (PDF), *Knapsack Problems: Algorithms and Computer Implementations* [redacted], Chichester, UK: John Wiley and Sons, ISBN 0471924202
2. ^ Korte, Bernhard; Vygen, Jens (2006). "Bin-Packing". *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics 21. Springer. pp. 426–441. doi:10.1007/3-540-29297-7_18. ISBN 978-3-540-25684-7.
3. ^ Barrington, David Mix (2006). "Bin Packing". Archived from the original on 2019-02-16. Retrieved 2016-02-27.
4. ^ a b c Coffman Jr., Edward G.; Csirik, János; Galambos, Gábor; Martello, Silvano; Vigo, Daniele (2013), Pardalos, Panos M.; Du, Ding-Zhu; Graham, Ronald L. (eds.), "Bin Packing Approximation Algorithms: Survey and Classification", *Handbook of Combinatorial Optimization*, New York, NY: Springer, pp. 455–531, doi:10.1007/978-1-4419-7997-1_35, ISBN 978-1-4419-7997-1, retrieved 2021-08-08
5. ^ Lewis, R. (2009), "A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing" (PDF), *Computers and Operations Research*, **36** (7): 2295–2310, doi:10.1016/j.cor.2008.09.004
6. ^ Sindelar, Michael; Sitaraman, Ramesh; Shenoy, Prashant (2011), "Sharing-Aware Algorithms for Virtual Machine Colocation" (PDF), *Proceedings of 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Jose, CA, June 2011*: 367–378
7. ^ a b c Garey, M. R.; Johnson, D. S. (1979). Victor Klee (ed.). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co. pp. x+338. ISBN 0-7167-1045-5. MR 0519066.
8. ^ Martello & Toth 1990, p. 221
9. ^ Vazirani, Vijay V. (14 March 2013). *Approximation Algorithms*. Springer Berlin Heidelberg. p. 74. ISBN 978-3662045657.
10. ^ a b c d e f g h i j k l m n o p Johnson, David S (1973). "Near-optimal bin packing algorithms" (PDF). *Massachusetts Institute of Technology*.
11. ^ Gonzalez, Teofilo F. (23 May 2018). *Handbook of approximation algorithms and metaheuristics. Volume 2 Contemporary and emerging applications*. ISBN 9781498770156.
12. ^ a b c Dósa, György; Sgall, Jiri (2013). "First Fit bin packing: A tight analysis". *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. **20**: 538–549. doi:10.4230/LIPIcs.STACS.2013.538.
13. ^ a b c György, Dósa; Sgall, Jiri (2014). "Optimal Analysis of Best Fit Bin Packing". *{Automata, Languages, and Programming – 41st International Colloquium (ICALP)}*. Lecture Notes in Computer Science. **8572**: 429–441. doi:10.1007/978-3-662-43948-7_36. ISBN 978-3-662-43947-0.
14. ^ a b c d e Yao, Andrew Chi-Chih (April 1980). "New Algorithms for Bin Packing". *Journal of the ACM*. **27** (2): 207–227. doi:10.1145/322186.322187. S2CID 7903339.
15. ^ a b c d e f g Lee, C. C.; Lee, D. T. (July 1985). "A simple on-line bin-packing algorithm". *Journal of the ACM*. **32** (3): 562–572. doi:10.1145/3828.3833. S2CID 15441740.
16. ^ Donna J, Brown (1979). "A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms" (PDF). *Technical Rept.*
17. ^ Liang, Frank M. (1980). "A lower bound for on-line bin packing". *Information Processing Letters*. **10** (2): 76–79. doi:10.1016/S0020-0190(80)90077-0.
18. ^ van Vliet, André (1992). "An improved lower bound for on-line bin packing algorithms". *Information Processing Letters*. **43** (5): 277–284. doi:10.1016/0020-0190(92)90223-I.
19. ^ Balogh, János; Békési, József; Galambos, Gábor (July 2012). "New lower bounds for certain classes of bin packing algorithms". *Theoretical Computer Science*. 440–441: 1–13. doi:10.1016/j.tcs.2012.04.017 [redacted].
20. ^ a b Ramanan, Prakash; Brown, Donna J; Lee, C.C; Lee, D.T (September 1989). "On-line bin packing in linear time". *Journal of Algorithms*. **10** (3): 305–326. doi:10.1016/0196-6774(89)90031-X. hdl:2142/74206 [redacted].
21. ^ a b c Seiden, Steven S. (2002). "On the online bin packing problem". *Journal of the ACM*. **49** (5): 640–671. doi:10.1145/585265.585269. S2CID 14164016.
22. ^ a b c Dósa, György (2007). "The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9OPT(I) + 6/9". *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. ESCAPE*. doi:10.1007/978-3-540-74450-4_1.
23. ^ Baker, B. S.; Coffman, Jr., E. G. (1981-06-01). "A Tight Asymptotic Bound for Next-Fit-Decreasing Bin-Packing". *SIAM Journal on Algebraic and Discrete Methods*. **2** (2): 147–152. doi:10.1137/0602019. ISSN 0196-5212.
24. ^ Csirik, J.; Galambos, G.; Frenk, J.B.G.; Frieze, A.M.; Rinnooy Kan, A.H.G. (1986-11-01). "A probabilistic analysis of the next fit decreasing bin packing heuristic". *Operations Research Letters*. **5** (5): 233–236. doi:10.1016/0167-6377(86)90013-1. hdl:1765/11645. ISSN 0167-6377.
25. ^ Fisher, David C. (1988-12-01). "Next-fit packs a list and its reverse into the same number of bins". *Operations Research Letters*. **7** (6): 291–293. doi:10.1016/0167-6377(88)90060-0. ISSN 0167-6377.
26. ^ a b Johnson, David S; Garey, Michael R (October 1985). "A 7160 theorem for bin packing". *Journal of Complexity*. **1** (1): 65–106. doi:10.1016/0885-064X(85)90022-6 [redacted].
27. ^ a b Yue, Minyi; Zhang, Lei (July 1995). "A simple proof of the inequality MFFD(L) ≤ 71/60 OPT(L) + 1,L for the MFFD bin-packing algorithm". *Acta Mathematicae Applicatae Sinica*. **11** (3): 318–330. doi:10.1007/BF02011198. S2CID 118263129.
28. ^ Fernandez de la Vega, W.; Lueker, G. S. (1981). "Bin packing can be solved within 1 + ε in linear time". *Combinatorica*. **1** (4): 349–355. doi:10.1007/BF02579456. ISSN 1439-6912. S2CID 10519631.
29. ^ Coursera, "Approximation Algorithms I", week 3.
30. ^ a b Karmarkar, Narendra; Karp, Richard M. (November 1982). "An efficient approximation scheme for the one-dimensional bin-packing problem". *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*: 312–320. doi:10.1109/SFCS.1982.61. S2CID 18583908.
31. ^ a b Rothvoß, T. (2013-10-01). "Approximating Bin Packing within O(log OPT * Log Log OPT) Bins". *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*: 20–29. arXiv:1301.4010 [redacted]. doi:10.1109/FOCS.2013.11. ISBN 978-0-7695-5135-7. S2CID 15905063.
32. ^ a b Hoberg, Rebecca; Rothvoss, Thomas (2017-01-01), "A Logarithmic Additive Integrality Gap for Bin Packing", *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, Society for Industrial and Applied Mathematics, pp. 2616–2625, doi:10.1137/1.9781611974782.172, ISBN 978-1-61197-478-2, S2CID 1647463, retrieved 2021-02-10
33. ^ Martello & Toth 1990, pp. 237–240.
34. ^ Korf, Richard E. (2002). *A new algorithm for optimal bin packing* (PDF). AAAI-02.
35. ^ R. E. Korf (2003), *An improved algorithm for optimal bin packing*. Proceedings of the International Joint Conference on Artificial Intelligence, (pp. 1252–1258)
36. ^ Schreiber, Ethan L.; Korf, Richard E. (2013), "Improved Bin Completion for Optimal Bin Packing and Number Partitioning", *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, Beijing, China: AAAI Press, pp. 651–658, ISBN 978-1-57735-633-2
37. ^ Goemans, Michel X.; Rothvoß, Thomas (2013-12-18), "Polynomiality for Bin Packing with a Constant Number of Item Types", *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, Society for Industrial and Applied Mathematics, pp. 830–839, doi:10.1137/1.9781611973402.61, hdl:1721.1/92865, ISBN 978-1-61197-338-9, S2CID 14006427, retrieved 2021-08-17
38. ^ Krause, K. L.; Shen, V. Y.; Schwetman, H. D. (1975-10-01). "Analysis of Several Task-Scheduling Algorithms for a Model of Multiprogramming Computer Systems". *Journal of the ACM*. **22** (4): 522–550. doi:10.1145/321906.321917. ISSN 0004-5411. S2CID 10214857.
39. ^ Kellerer, H.; Pferschy, U. (1999-01-01). "Cardinality constrained bin-packing problems". *Annals of Operations Research*. **92**: 335–348. doi:10.1023/A:1018947117526. ISSN 1572-9338. S2CID 28963291.
40. ^ Chung, Yerim; Park, Myoung-Ju (2015-01-01). "Notes on inverse bin-packing problems". *Information Processing Letters*. **115** (1): 60–68. doi:10.1016/j.ipl.2014.09.005. ISSN 0020-0190.
41. ^ Boyar, Joan; Epstein, Leah; Favrholdt, Lene M.; Kohrt, Jens S.; Larsen, Kim S.; Pedersen, Morten M.; Wøhlk, Sanne (2006-10-11). "The maximum resource bin packing problem". *Theoretical Computer Science*. **362** (1): 127–139. doi:10.1016/j.tcs.2006.06.001. ISSN 0304-3975.
42. ^ Huang, Xin; Lu, Pinyan (2020-11-10). "An Algorithmic Framework for Approximating Maximin Share Allocation of Chores". arXiv:1907.04505 [redacted] [cs.GT].
43. ^ Ma, Ruixin; Dósa, György; Han, Xin; Ting, Hing-Fung; Ye, Deshi; Zhang, Yong (2013-08-01). "A note on a selfish bin packing problem". *Journal of Global Optimization*. **56** (4): 1457–1462. doi:10.1007/s10898-012-9856-9. ISSN 0925-5001. S2CID 3082040.
44. ^ Anily, Shoshana; Bramel, Julien; Simchi-Levi, David (1994-04-01). "Worst-Case Analysis of Heuristics for the Bin Packing Problem with General Cost Structures". *Operations Research*. **42** (2): 287–298. doi:10.1287/opre.42.2.287. ISSN 0030-364X.
45. ^ Lodi A., Martello S., Monaci, M., Vigo, D. (2010) "Two-Dimensional Bin Packing Problems". In V.Th. Paschos (Ed.), *Paradigms of Combinatorial Optimization*, Wiley/ISTE, pp. 107–129
46. ^ Optimizing Three-Dimensional Bin Packing Through Simulation
47. ^ Benkő A., Dósa G., Tuza Z. (2010) "Bin Packing/Covering with Delivery, Solved with the Evolution of Algorithms," *Proceedings 2010 IEEE 5th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010*, art. no. 5645312, pp. 298–302.
48. ^ Vaccaro, Alessio (2020-11-13). "4 Steps to Easily Allocate Resources with Python & Bin Packing". *Medium*. Retrieved 2021-03-21.