Multidimensional on-line bin-packing: An algorithm and its average-case analysis

Ee-Chien Chang Weiguo Wang Mohan S. Kankanhalli
Institute of Systems Science
National University of Singapore
Kent Ridge, Singapore 0511
Email : wwang@iss.nus.sg

Abstract

We describe an algorithm for on-line d-dimensional bin packing which achieves $O(n^{\frac{d+1}{d+2}})$ average wasted space. This algorithm runs in linear time in the number of items packed.

Keywords: Analysis of algorithms, average-case analysis, bin-packing, on-line algorithms.

1 Introduction

We consider the d-dimensional bin-packing problem. Given a list of items, the problem is to pack them into the least number of bins with certain constraints. Each item can be represented by a point in $(0,1]^d$. The items can be packed in several ways. We only consider box-packing [2] (also known as rectangle-packing in the literature [5]) where an item, $(x^1, x^2, .., x^d)$, is a d-dimensional box whose sides are $x^1, x^2, .., x^d$. Each bin is an unit d-dimensional cube. The items assigned to a bin can be packed if they can fit in it without intersecting one another. This is an *NP-hard* problem . There are variations of this problem which allow rotation of items for packing. We do not consider them here.

In on-line packing, items are given to the algorithm sequentially and each item must be packed before the next item is known. There are two types of on-line packing, *open-end* and *closed-end* packing. For closed-end packing, the number of items to be packed is known before the packing, while for the open-end packing, the number of items is not known until a "stop" signal is activated [6].

For average-case analysis, the inputs are assumed to follow a known distribution. The performance of an algorithm is measured by the wasted space of the packing produced by the algorithm. The wasted space is the total amount of empty space in the bins (formal definitions will be given in the next section).

Much work has been done in the worst-case analysis of on-line algorithms [2] and average-case analysis of one-dimensional on-line bin packing algorithms[4, 6]. It has been shown that for one-dimensional open-end on-line bin-packing, the expected wasted space is bounded below by $\Omega(\sqrt{n \log n})$ for all algorithms and the Best Fit produces an expected wasted space of order $\theta(\sqrt{n}(\log n)^{\frac{3}{4}})$ [6].

For the off-line version of the problem, Karp et. al. [5] gave an algorithm which achieves expected wasted space of $O(G(n,d))$ for d-dimensional bin-packing where $G(n,1) = \sqrt{n}$, $G(n,2) = \sqrt{n}\log n$ and $G(n,d) = n^{\frac{d-1}{d}}$ for $d \geq 3$.

We give an algorithm `HashPacking`, for on-line (both open-end and closed-end) d-dimensional bin packing which achieves average wasted space of order $O(n^{\frac{d+1}{d+2}})$ and runs in linear time in the number of item packed.

2 Definition

The *volume* of an item $x_i = (x_i^1, x_i^2, .., x_i^d)$ is the product $\prod_{j=1}^{d} x_i^j$. Given a list l_n of items $x_1, x_2, .., x_n$, and a packing algorithm `A`, let $\mathtt{A}[l_n]$ be the number of bins used by `A` to pack the items in l_n. The wasted space produced by algorithm `A`, denoted by $\mathtt{A}^W[l_n]$, is the total amount of empty space in all the bins used by `A`. That is, (recall that all bins have unit volume)

$$\mathtt{A}^W[l_n] \quad = \quad \mathtt{A}[l_n] - \sum_{i=1}^{n}\prod_{j=1}^{d} x_i^j. \tag{1}$$

For the average-case analysis, we assume that the input follows a known distribution. Let $X_i = (X_i^1, X_i^2, .., X_i^d)$ be a random variable formed by $X_i^1, X_i^2, .., X_i^d$ which are independently and identical distributed random variables each following the uniform distribution. That is, $X_i^j \sim U(0,1]$. Let L_n be a sequence of n such random variables $X_1, X_2, .., X_n$.

In the closed-end on-line bin packing setting, the number of items, n, to be packed is known before the packing. We treat L_n as the random variable of the input. Random variables $\mathtt{A}^W[L_n]$ and $\mathtt{A}[L_n]$ are similarly defined. From (1),

$$E[\mathtt{A}^W[L_n]] \quad = \quad E[\mathtt{A}[L_n]] - \frac{n}{2^d}. \tag{2}$$

For open-end on-line bin packing, the number of items to be packed is not known until the last item is seen. We follow the model of open-end on-line bin packing used in [6] where the inputs are drawn as follows:

For a given n,

1. Choose k from 1 to n at random.
2. Choose k items uniformly from $(0,1]^d$.
3. Feed these items to the algorithm one-by-one followed by a "stop" signal after the *k-th* item.

3 One-dimensional bin-packing

This section describes an algorithm, `HashPacking`, for one-dimensional on-line bin-packing. This algorithm works in both closed-end and open-end settings except that in the former, the computation halts after the *n-th* item is encountered, whereas in the latter, the computation halts after the "stop" signal is received.

The idea of $\texttt{HashPacking}_m$ is: for each incoming item, it is packed with an item such that, the wasted space does not exceed a pre-defined constant $\frac{1}{m}$ where m is an integer parameter of the algorithm.

Each item is classified into two types, $+1$ (*big*) if it is greater than $\frac{1}{2}$ and -1 (*small*) otherwise. Define a function h depending on m as

$$h_m(x) = \begin{cases} \lfloor 2(1-x)m \rfloor, & x > \frac{1}{2}, \\ \lceil 2xm \rceil, & otherwise. \end{cases} \tag{3}$$

Note that for a big item x and a small item y, if $h_m(x) = h_m(y)$, then they can be packed in a bin with wasted space less than $\frac{1}{m}$.

The algorithm, $\texttt{HashPacking}_m$, is described as follows:

1. Create $m+1$ empty buckets labeled as $b_0, b_1, .., b_m$. Each bucket will contain a list of bins.
2. For each incoming item x, compute $h_m(x)$ and its type. If there is a bin in the bucket $b_{h_m(x)}$, which contains an item of type different from that of item x, then pack x in this bin and remove the bin from $b_{h_m(x)}$. If no such bin exists, then pack x into a new bin and add the bin to $b_{h_m(x)}$.
3. Repeat step 2 until either the last item is encountered (closed-end) or the "stop" signal is received (open-end).

The function h_m can be viewed as a hashing function which hashes an item to a bucket. Note that only the big items can be hashed into bucket b_0 and only the small items into bucket b_m. We call these two buckets *boundary buckets* and others *non-boundary buckets.*
Note that items which are hashed into the boundary buckets will not be packed with other items. (In order to simplify the proof and the description of the generalization to multi-dimension, although any two items hashed to the bucket b_m can be packed in a bin, we do not take this advantage. It's easy to see that whether to take this advantage, the expected wasted space remains in the same order of magnitude.) Furthermore, the value of m affects the performance of the algorithm. The larger the value of m, the less items will be hashed to the boundary buckets. But then the number of bins containing only one item in the non-boundary buckets will be increased as it is more difficult to satisfy the requirement on wasted space. This observation is reflected in the following lemma.

Lemma 1 *For every $\alpha < 1$, $m = n^\alpha$, when n is large enough,*

$$E^W[\texttt{HashPacking}_m[L_n]] = \frac{n}{m} + \theta(\sqrt{nm}).$$

Proof

Let B be the number of bins containing only one item after the packing. Note that $\texttt{HashPacking}_m[L_n] = \frac{n-B}{2} + B$, thus $E[\texttt{HashPacking}_m[L_n]] = \frac{n}{2} + \frac{1}{2}E[B]$. By (2),

$$E^W[\texttt{HashPacking}_m[L_n]] = \frac{1}{2}E[B].$$

That is, the expected wasted space is of the same order as the expected value of B. We find the expected value of B to estimate the expected wasted space.

All the bins in the boundary buckets will contain only one item. The expected number of bins in the boundary buckets is $2\frac{n}{2m} = \frac{n}{m}$.

$$E[\text{number of bins in boundary buckets }] = \frac{n}{m}. \tag{4}$$

For each non-boundary bucket b_j, consider the items been hashed to it. The difference between the number of big items and that of small items is the number of bins left in b_j after the computation halts.

Consider a non-boundary bucket, b_j. The process of hashing n items can be viewed as a sequence of n Bernoulli trials with probability $\frac{1}{m}$ of success (being hashed to b_j). The expected number of items hashed to b_j is $\frac{n}{m}$. Thus the probability that more than $2\frac{n}{m}$ items are hashed to b_j is smaller than $e^{-\frac{n}{m}}$ [3].

$$\begin{aligned} Pr(\text{more than } \quad 2\tfrac{n}{m} & \quad \text{items are hashed to a particular bucket }) \\ &= \sum_{k=2\frac{n}{m}}^{n} \binom{n}{k} \left(\frac{1}{m}\right)^k \left(\frac{m-1}{m}\right)^{n-k} \\ &< e^{-\frac{n}{m}}, \quad \text{(for large } n). \end{aligned}$$

By *Boole's inequality* (for any finite or countably infinite sequence of events $A_1, A_2, ..., Pr(A_1 \cup A_2 \cup ...) \leq Pr(A_1) + Pr(A_2) + ...$), the probability that at least one bucket has more than $2\frac{n}{m}$ items hashed to it is less than $me^{-\frac{n}{m}}$ which is smaller than $\frac{1}{n}$ for large n and when $m = n^\alpha, \alpha < 1$. In other words, with high probability, every bucket has at most $2\frac{n}{m}$ items been hashed to it. (To pack n items, at most n bins are needed. Thus any event with probability lower than $\frac{1}{n}$ can be ignored. This is because if $f(n)$ bins are needed when the event does not occurred, then the expected number of bins required is of the order $(1-\frac{1}{n})O(f(n)) + \frac{1}{n}O(n) = O(f(n))$.)

Suppose that every non-boundary bucket has up to $2\frac{n}{m}$ items been hashed to it. Consider the items hashed to a non-boundary bucket. Each item has equal probability of being big or small independent of the types of other items in the bin. Thus, the expected difference between the number of big items and small items is of the order $\theta(\sqrt{\frac{n}{m}})$. Therefore the expected total number of bins left in all the non-boundary buckets is of the order $m\theta(\sqrt{\frac{n}{m}}) = \theta(\sqrt{nm})$.

$$E[\text{number of bins in non-boundary buckets }] = \theta(\sqrt{nm}). \tag{5}$$

From (4) and (5), the lemma follows. □

To minimize the wasted space, choose $\alpha = \frac{1}{3}$, and the expected wasted space is of the order $\theta(n^{\frac{2}{3}})$.

Lemma 2 *The expected wasted space for* $\texttt{HashPacking}_{n^{\frac{1}{3}}}$ *as a closed-end on-line bin packing algorithm is of the order* $\theta(n^{\frac{2}{3}})$.

Consider the open-end on-line bin packing. If the number of items to be packed is given to be k, then the expected wasted space produced by $\texttt{HashPacking}_m$ in the open-end setting is same as

$E[\texttt{HashPacking}_m^W[L_k]]$. In fact, the expected wasted space produced by $\texttt{HashPacking}_m$ in the open-end setting is:

$$\sum_{k=1}^{n} Pr(N=k)E[\texttt{HashPacking}_m^W[L_k]]$$

During the packing, for each incoming item, the probability that the addition of a new item will reduce the number of bins containing one item by one is less than $\frac{1}{2}$. On the other hand, the probability that it will increase such number by one is more than $\frac{1}{2}$. Thus $E[\texttt{HashPacking}_m^W[L_k]] < E[\texttt{HashPacking}_m^W[L_{k+1}]]$. Therefore, in the open-end on-line setting, the expected wasted space is bounded by $E[\texttt{HashPacking}_m^W[L_n]]$. Choosing $m = n^{\frac{1}{3}}$, the expected wasted space is bounded by $O(n^{\frac{2}{3}})$. In particular, if the number of items to be packed is chosen uniformly from 1 to n, then the expected wasted space is $\theta(n^{\frac{2}{3}})$.

Lemma 3 *The expected wasted space for* $\texttt{HashPacking}_{n^{\frac{1}{3}}}$ *as an open-end on-line bin packing algorithm is bounded by* $O(n^{\frac{2}{3}})$

4 Multi-dimensional bin-packing

This is the generalization of the one-dimensional case to d-dimensional for $d \geq 2$. Each item is classified into 2^d types instead of two types in the one-dimensional case. For an item represented by $x_i = (x_i^1, ..., x_i^d)$, its type is $c_i = (c_i^1, .., c_i^d)$ where c_i^j is equal to $+1$ if $x_i^j > \frac{1}{2}$, -1 otherwise. Define the function h_m^d as

$$h_m^d((x^1, x^2, .., x^d)) = (h_m(x^1), h_m(x^2), .., h_m(x^d))$$

where h_m is defined as in (3). Note that any 2^d items of distinct types which are hashed to the same value by h_m^d can be packed into one bin.

The algorithm, $\texttt{HashPacking}_m^d$, is described as follows:

1. Create $(m+1)^d$ empty buckets with label $b_{(i_1,i_2,..i_d)}$, where $0 \leq i_1, i_2, .., i_d \leq m$. Each bucket will contain a list of bins.

2. For each incoming item x, compute $h_m^d(x)$ and its type. Find the first bin in $b_{h_m^d(x)}$ such that it does not contain an item of the same type as x. Pack x into this bin. If the bin contains 2^d items, then remove it from $b_{h_m^d(x)}$. If no such bin exists, then pack x in an empty bin and add the bin to the end of $b_{h_m^d(x)}$.

3. Repeat step 2 until either the last item is encountered (closed-end) or the "stop" signal is received (open-end).

As in the one-dimensional case, there are buckets in which it is impossible to have items of all types hashed to it. Call these buckets as *boundary buckets*, others as *non-boundary buckets*. The buckets $b_{(i_1,i_2,..,i_d)}$, where $0 < i_1, i_2, .., i_d < m$, are all the non-boundary buckets. Thus, the number of boundary buckets is of the order $m^d - (m-1)^d = \theta(m^{d-1})$. Therefore, the expected total number of items hashed to all the boundary buckets is $\frac{n}{m^d}\theta(m^{d-1}) = \theta(\frac{n}{m})$.

Applying similar arguments as in the one-dimensional case, we can assume that every bucket b_j has up to $k\frac{n}{m^d}$ items hashed to it, where k is a constant independent of n. Let n_c be the number of items of type c in a bucket. The number of bins remaining in this bucket when the computation halts is the difference between $\max_{c\in\{-1,1\}^d}\{n_c\}$ and $\min_{c\in\{-1,1\}^d}\{n_c\}$. It is easy to prove by induction on d that the expectation of this difference is of the order $\theta(\sqrt{\frac{n}{m^d}})$.

Thus, the expected wasted space is of the order $\theta(\frac{n}{m}) + m^d\theta(\sqrt{\frac{n}{m^d}})$. To minimize the wasted space, choose $m = n^{\frac{1}{d+2}}$.

By similar argument as in the one-dimensional case, the expected wasted space produced by $\texttt{HashPacking}^d_{n^{\frac{1}{d+2}}}$ as an open-end on-line bin packing algorithm is of the order $O(n^{\frac{d+1}{d+2}})$.

Theorem $\texttt{HashPacking}^d_{n^{\frac{1}{d+2}}}$ *produces expected wasted space of order* $O(n^{\frac{d+1}{d+2}})$ *as an on-line (both open-end and closed-end) d-dimensional bin packing algorithm.*

5 Remarks

An off-line version of bin-packing algorithm can be obtained from the `HashPacking` algorithm by further packing the remaining items after the last item is seen. These items were not packed by `HashPacking` because by packing them, the wasted space would exceed the predefined requirement ($\frac{1}{m}$ for the one-dimensional case). Thus if `HashPacking` is repeatedly applied on the remaining items by relaxing the requirement, more items can be packed. In fact, for one-dimensional case, by choosing the initial value of m to be n^3, decreasing m by half and running of `HashPacking` again on the remaining items, the expected wasted space is of order $\theta(\sqrt{n})$. Unfortunately, the generalization of this method to higher dimension does not seem to achieve a good bound. One advantage of this off-line algorithm is that it can be easily transformed into a parallel algorithm [1].

References

[1] E. C. Chang. Probabilistic analysis and parallelization of bin-packing algorithms. Master's thesis, Institute of Systems Science, National University of Singapore, 1992. (in preparation).

[2] D. Coppersmith and P. Raghavan. Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Operation Research Letters*, 8:17–20, 1989.

[3] W. Feller. *Probability theory and its applications.* John Wiley & Sons, inc, New York, 1952.

[4] E. G. Coffman Jr. and G. S. Lueker. *Probabilistic analysis of packing and partitioning algorithms.* A Wiley-Interscience Publication, 1991.

[5] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. *STOC 84*, pages 289–298, 1984.

[6] P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986.