Linear time-approximation algorithms for bin packing

Guochuan Zhang\(^a,1\), Xiaoqiang Cai\(^b\), C.K. Wong\(^c,*,2\)

\(^a\) Department of Mathematics, Zhejiang University, Hangzhou 310027, People's Republic of China
\(^b\) Department of Systems Engineering & Engineering Management, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
\(^c\) Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Received 1 April 1999; received in revised form 1 October 1999

Abstract

Simchi-Levi (Naval Res. Logist. 41 (1994) 579–585) proved that the famous bin packing algorithms FF and BF have an absolute worst-case ratio of no more than \(\frac{7}{4}\), and FFD and BFD have an absolute worst-case ratio of \(\frac{3}{2}\), respectively. These algorithms run in time \(O(n \log n)\). In this paper, we provide a linear time constant space (number of bins kept during the execution of the algorithm is constant) off-line approximation algorithms with absolute worst-case ratio \(\frac{3}{2}\). This result is best possible unless \(P=NP\). Furthermore, we present a linear time constant space on-line algorithm and prove that the absolute worst-case ratio is \(\frac{7}{4}\). © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Bin packing; Absolute worst-case ratio; Linear time algorithm

1. Introduction

In the classical one-dimensional bin packing problem, we are given a list of items \(L = (a_1, a_2, \ldots, a_n)\), each with a size \(s(a_i) \in (0, 1]\), and are asked to pack them into a minimum number of unit-capacity bins. This problem is of fundamental theoretical significance, serving as an early example for many of the classical approaches in analyzing the performance ratios, identifying lower bounds on the best-possible online performance, and analyzing the average-case behavior of algorithms. Bin packing also has many potential real-world applications, e.g., in memory allocation in paged computer systems, in assigning newspaper articles to newspaper pages, in loading trucks, in packet routing in communication networks, in assigning commercials to station breaks on televisions and in cutting-stock problems, etc. Since the bin packing problem is well known to be strongly NP-hard \(\cite{3}\), much work has been done in the study of approximation algorithms. A survey of these results is given in \(\cite{1}\).
A bin packing algorithm is called on-line if it packs all items \( a_i \) solely on the basis of the sizes of the items \( a_j \), \( 1 \leq j \leq i \), and without any information on subsequent items. The decisions of the algorithm are irrevocable. Once an item is packed, it is not allowed to be moved again. A bin packing algorithm is called off-line if all items are known before it starts to construct a packing. There are also algorithms which work in situations different from both the on-line and off-line cases, e.g., partial information on future items are known in advance, some items are allowed to be repacked, etc. We call such algorithms semi on-line.

In the case of bin packing, the standard metric for worst-case performance is the asymptotic worst-case ratio. For a given list \( L \) of items and an approximation Algorithm \( A \), let \( A(L) \) denote the number of bins used by Algorithm \( A \) and let \( \text{OPT}(L) \) denote the number of bins in an optimal packing. The absolute worst-case ratio \( R_A \) for Algorithm \( A \) is given by

\[
R_A = \sup_{L} \{ A(L)/\text{OPT}(L) \}.
\]

The asymptotic worst-case ratio \( R_A^\infty \) of Algorithm \( A \) is defined to be

\[
R_A^\infty = \lim_{k \to \infty} \sup_{L} \{ A(L)/\text{OPT}(L)|\text{OPT}(L) \geq k \}.
\]

Note that for any algorithm its asymptotic worst-case ratio is never larger than its absolute worst-case ratio. Most of worst-case results concentrate on asymptotic worst-case ratios, but the question of absolute worst-case ratios is also of interest, especially when we are considering relatively short lists of items.

In addition to the worst-case performance ratio, we also consider the time and the space complexity of bin packing algorithms. The time complexity of Algorithm \( A \) refers to the total time required by \( A \) to pack list \( L \) whose size is \( n \). Algorithm \( A \) is called a linear time algorithm if the time complexity is \( O(n) \). On the other hand, the space complexity means the maximum number of storage locations needed by Algorithm \( A \) during the processing of list \( L \) whose size is \( n \). Using the uniform cost criterion for space, we need one storage location for each open bin. As soon as a bin is declared closed, it is among the output of the algorithm used, and we do not count its storage location in defining the space complexity of the algorithm. A constant space algorithm uses constant storage locations at any time.

In this paper, we present a linear time constant space off-line algorithm with absolute worst-case ratio \( \frac{1}{2} \). This result is best possible unless \( P = \text{NP} \). We point out that this algorithm is also semi on-line. Furthermore, we give a linear time constant space on-line algorithm with absolute worst-case ratio \( \frac{7}{12} \). To the best of our knowledge, this result is the best so far. The rest of this paper is organized as follows. Section 2 introduces previous related results and some definitions. The linear time off-line algorithm and the linear time on-line algorithm are presented in Sections 3 and 4, respectively.

2. Previous results and basic definitions

An item is called a large item if its size is greater than \( \frac{1}{2} \); otherwise it is called a small item. A bin is called open if it has accepted an item and is not closed yet. Once an open bin is declared to be closed, it will never accept items anymore.

Let us first consider on-line algorithms. Perhaps the simplest algorithm for the bin packing problem is Next-Fit (NF), apparently first described under this name by Johnson [4]. In packing item \( a_i \), NF tests if the current open bin has enough room for \( a_i \). If so, it places \( a_i \) in the bin and leaves the bin open. Otherwise, it closes the bin and places \( a_i \) in a new bin which now becomes the open bin. This algorithm can be implemented to run in linear time and use constant space. It is not difficult to show that for any list \( L \), \( \text{NF}(L) \leq 2\text{OPT}(L) - 1 \). Furthermore, there exists a list \( L \) such that \( \text{NF}(L) = 2\text{OPT}(L) - 1 \) holds. Thus we conclude that \( R_{\text{NF}} = 2 \). Moreover, bound 2 still holds even for lists without large items. The famous on-line algorithms First-Fit (FF) and Best-Fit (BF) never close any open bin before all items are packed. FF places item \( a_i \) in the first (lowest indexed) open bin into which it fits. If such a bin does not exist, FF starts a new bin with \( a_i \) as its first item. Note that FF is not a linear time algorithm. Using an appropriate data structure [4], FF runs in time \( O(n \log n) \). Clearly, FF is an \( O(n) \) space algorithm since all non-empty bins remain open until the end of the packing. BF is similar to FF except that it places item \( a_i \) into the fullest bin into which \( a_i \) fits (ties broken in favor of lower index). Like FF, BF can also be implemented to run in time \( O(n \log n) \) and it uses \( O(n) \) space. Johnson
et al. [5] proved $R_{FF}^\infty = R_{BF}^\infty = 1.7$ and Simchi-Levi [10] showed that FF and BF have an absolute worst-case ratio not greater than $\frac{3}{2}$. Lee and Lee [6] presented a linear time on-line algorithm called HAMRONICM and proved that it achieved asymptotic worst-case ratio of less than 1.692, which is better than that of FF. A revised version of HAMRONICM, an $O(n)$ space and linear time algorithm, was given and was shown to have an asymptotic worst-case ratio of less than 1.636. Ramanan et al. [9] provided a linear time on-line algorithm called Modified Harmonic (MH) and showed that it achieves asymptotic worst-case ratio of less than 1.615. All of these harmonic algorithms are based on the harmonic partition of the interval $(0,1]$. They group items into different types depending on their sizes and creates different classes of bins. In general, different type of items will be packed into different class of bins. If the list is short and covers all types of items, it will result in a very large absolute worst-case ratio. In this paper, we give an on-line algorithm with absolute worst-case ratio $\frac{2}{3}$ which runs in linear time and uses only constant space. To the best of our knowledge, in terms of absolute worst-case ratio, our algorithm is the best among linear time constant space on-line algorithms so far.

As to off-line algorithms, it is generally recognized that first-fit decreasing (FFD) and best-fit decreasing (BFD) are two of the best-known algorithms in bin packing. In these algorithms, the items are first sorted in the order of non-increasing size and then FF and BF algorithms are applied, respectively. Johnson [4] proved the asymptotic worst-case ratios of FFD and BFD to be $\frac{11}{7}$. Simchi-Levi [10] showed FFD and BFD have an absolute worst-case ratio $\frac{3}{2}$ which are the best possible unless P = NP. However neither FFD nor BFD is a linear time algorithm. A number of linear time bin-packing algorithms were developed. Fernandez and Leuker [2] showed that for every positive $\epsilon$ there is an Algorithm A such that $A(L) \leq (1 + \epsilon)OPT(L) + C_\epsilon$, and A runs in time $O(n) + D_\epsilon$, where $C_\epsilon$ and $D_\epsilon$ are constants that only depend on $\epsilon$. The authors did not calculate $D_\epsilon$ precisely but remarked that it is “certainly huge”. Clearly $D_\epsilon$ grows exponentially in $1/\epsilon$. Martel [8] presented a linear time bin packing algorithm OffBP and showed that it uses at the most $4OPT(L)/3 + 2$ bins for any list $L$. Simchi-Levi [10] and Lenstra and Shmoys [7] proved that there does not exist any polynomial time Algorithm A for the bin packing problem and a constant $\rho < \frac{1}{3}$ such that $R_A < \rho$, unless P = NP. Thus in terms of the absolute worst-case ratio, our algorithm is the best possible for the bin packing problem, unless P = NP.

3. The off-line case

We give an algorithm as follows:

Algorithm A1.

1. Put large items each into a bin. Index the unfilled bins in arbitrary order. Set all these bins as active bins. Repeatedly arrange small items as follows.
2. If there is an active bin open, put the current item $a_i$ into the open active bin with the lowest index if the bin has enough room for $a_i$, otherwise close this active bin and consider the extra bin as follows.
   (a) If there is an extra bin open and it has enough room for $a_i$, place $a_i$ into it.
   (b) Otherwise close the extra bin open and set this bin as the extra bin.
3. If there is no active bin open, create a new bin for $a_i$ and set this new bin as the active bin.

In fact, in Step 1 of Algorithm A1, we only need to arrange one large item at a time. As soon as the active bin containing a large item is closed, we open a new bin and put a remaining large item into it first and set it as the active bin. At any time we only keep at the most one active bin and at the most one extra bin open. Thus, the algorithm only uses constant space.

The following example shows how Algorithm A1 works. We have nine items $a_1, \ldots, a_9$ where $s(a_1) = 0.8$, $s(a_2) = 0.6$, $s(a_3) = 0.3$, $s(a_4) = 0.4$, $s(a_5) = 0.2$, $s(a_6) = 0.3$, $s(a_7) = 0.4$, $s(a_8) = 0.5$, $s(a_9) = 0.2$. Then the packing is given in Fig. 1.

Theorem 1. Algorithm A1 is a linear time algorithm with absolute worst-case ratio $\frac{3}{2}$. Moreover A1 is the best possible algorithm unless P = NP.

Proof. A1 is clearly a linear time algorithm since $A_1$ packs each item in constant time. The packing by
Algorithm $A_1$ consists of two bin sets $X$ and $Y$, where $X$ contains the active bins and $Y$ is the set of the extra bins. Let $N_a$ and $N_c$ be the numbers of bins of $X$ and $Y$, respectively. Let $N_e$ be the number of closed active bins. Note that an active bin is closed if and only if an item cannot fit in the bin. And this item is packed into an extra bin. Thus the total number of items packed into extra bins is equal to $N_c$. Since all items in extra bins are small items, except the last one all extra bins contain at least two items. It implies that $N_e \leq \lceil N_c/2 \rceil$.

Case 1: $N_e \leq N_a - 1$. In this case, $A_1(L) = N_a + N_e \leq N_a + \lceil N_c/2 \rceil \leq N_a + (N_a - 1)/2 \leq 3N_a/2$.

If $N_e = N_a - 1$,

$$\text{OPT}(L) \geq \sum_{i=1}^{n} s(a_i) = \sum_{a_i \in X} s(a_i) + \sum_{a_i \in Y} s(a_i) > N_c = N_a - 1,$$

i.e., $\text{OPT}(L) \geq N_a$. If $N_e \leq N_a - 2$, there are at least two active bins which are not closed. It happens only when each active bin contains a large item. Hence $\text{OPT}(L) \geq N_a$. Therefore, we get $A_1(L)/\text{OPT}(L) \leq 3/2$.

Case 2: $N_e = N_a$. In this case,

$$A_1(L) = N_a + N_e \leq N_a + \lceil N_c/2 \rceil = N_a + (N_a - 1)/2 \leq 3N_a/2 + 1/2,$$

On the other hand,

$$\text{OPT}(L) \geq \sum_{i=1}^{n} s(a_i) = \sum_{a_i \in X} s(a_i) + \sum_{a_i \in Y} s(a_i) > N_a,$$

i.e., $\text{OPT}(L) \geq N_a + 1$. $A_1(L)/\text{OPT}(L) \leq 1/2$ also holds.

We have proven $R_{A_1} \leq 3/2$.

Simchi-Levi [10] and Lenstra and Shmoys [7] proved that there does not exist any polynomial time Algorithm $A$ for the bin packing problem and a constant $\rho < 3/2$ such that $R_A \leq \rho$, unless $P = NP$. Thus in terms of the absolute worst-case ratio, the linear time Algorithm $A_1$ is the best possible for the bin packing problem, unless $P = NP$.

Observe that Algorithm $A_1$ is a semi on-line algorithm. It only requires the information on large items in advance. If there is no large item, $A_1$ serves as an on-line algorithm.

In the next section, we will study the on-line version of the bin packing problem.

### 4. The on-line case

In the following algorithm, a bin is called an L-bin if the first item it accepts is a large item and has not been set as an active bin. Once such a bin becomes an active bin, it will not be called an L-bin any longer.

**Algorithm A$_2$.**

1. Open a bin for the first item. Set it as the active bin.
2. If there is an active bin $B_j$ open, put the current item $a_i$ into $B_j$ if $B_j$ has enough room for this item. Suppose that $B_j$ does not have enough space for the item.
   (a) If $a_i$ is a large item, then create a new bin for $a_i$. If there are two L-bins then close them.
   (b) If $a_i$ is a small item, then close the active bin.
   Set an open L-bin as the active bin if such a bin exists. If there is an extra bin open and such a bin has enough room for $a_i$, pack $a_i$ into it.
Otherwise open a new bin for \( a_i \) and set the new bin as the extra bin.

3. If there is no active bin open, create a new bin for item \( a_i \) and set it as the active bin.

Clearly, \( A_2 \) is a constant space algorithm since it keeps at most one active bin, at most one extra bin and at most two L-bins open at any time.

We apply \( A_2 \) to the same item sizes as in the example for Algorithm \( A_1 \) except with the following incoming order \( a_3, a_4, a_5, a_2, a_1, a_6, a_7, a_8, a_9 \). The packing is shown in Fig. 2.

For any set \( L \) of items, consider the packing generated by \( A_2 \). Let \( X \) and \( Y \) be the sets of active bins and extra bins, respectively. Let \( Z \) be the set of remaining bins. From Algorithm \( A_2 \), it is easy to see that \( Z \) is the set of L-bins. Denote by \( N_a \), \( N_e \) and \( N_L \) the numbers of bins of sets \( X \), \( Y \) and \( Z \), respectively.

**Lemma 2.**

\[
A_2(L) \leq \begin{cases} 
N_L + 3N_a/2 + 1/2 & \text{if all active bins are closed,} \\
N_L + 3N_a/2 & \text{otherwise.}
\end{cases}
\]

**Proof.** Let \( N_c \) be the number of closed active bins. With the same analysis as that in the proof of Theorem 1, we can prove \( N_c \leq \lfloor N_e/2 \rfloor \). Thus, \( A_2(L) = N_a + N_c + N_L \leq N_a + N_L + \lfloor N_e/2 \rfloor \).

If all active bins are closed then \( N_c = N_a \); otherwise \( N_c = N_a - 1 \). Therefore,

\[
A_2(L) \leq \begin{cases} 
N_L + 3N_a/2 + 1/2 & \text{if all active bins are closed,} \\
N_L + 3N_a/2 & \text{otherwise.}
\end{cases}
\]

**Lemma 3.** \( \text{OPT}(L) \geq N_L \).

**Proof.** In any optimal packing, each bin contains at most one large item. Thus \( \text{OPT}(L) \) is not less than the total number of all large items. Hence the lemma holds.

**Lemma 4.**

\[
\text{OPT}(L) \geq \begin{cases} 
N_a + N_L/2 + 1 & \text{if all active bins are closed,} \\
N_a + N_L/2 & \text{otherwise.}
\end{cases}
\]

**Proof.** We consider two cases below.

*Case 1: All active bins are closed.* In this case, from Algorithm \( A_2 \) the number of L-bins are even. The total size of items of \( X \) bins is greater than \( N_L/2 \). On the other hand, the total size of items of \( Y \) bins and \( Z \) bins is greater than \( N_a \). Thus,

\[
\text{OPT}(L) \geq \sum_{i=1}^w a_i \geq N_a + N_L/2.
\]

Since \( N_L \) is even, we have \( \text{OPT}(L) \geq N_a + N_L/2 + 1 \).

*Case 2: There is an active bin which is not closed.* If the active bin is the last open bin, from Algorithm \( A_2 \) the number of L-bins is even. The total size of items of \( Z \) bins is greater than \( N_L/2 \) and the total size of items of \( X \) bins and \( Y \) bins is greater than \( N_a - 1 \). Analogous to Case 1, we have \( \text{OPT}(L) \geq N_a + N_L/2 \).

If the active bin is not the last open bin, then the last open bin must be an L-bin. The sum of item sizes of the active bin and the L-bin is greater than 1; the total size of items in all closed L-bins is greater than \( (N_L - 1)/2 \); the total size of items in all extra bins and all closed active bins is greater than \( N_a - 1 \). Totally, the sum of all items is greater than \( N_a + (N_L - 1)/2 \). It implies that \( \text{OPT}(L) \geq N_a + N_L/2 \).

**Theorem 5.** Algorithm \( A_2 \) runs in linear time and its absolute worst-case ratio is \( \frac{7}{4} \).

**Proof.** We only prove the case that there is an active bin which is not closed. The other case can be proved in the same way. From Lemmas 2–4, we get \( A_2(L) \leq N_L + 3N_a/2 \) and

\[
\text{OPT}(L) \geq \max \{ N_a + N_L/2, N_L \}.
\]

If \( N_a + N_L/2 > N_L \) then \( N_L < 2N_a \), we have

\[
\frac{A_2(L)}{\text{OPT}(L)} \leq \frac{3N_a/2 + N_L}{(N_a + N_L/2)} = 2 - \frac{N_a}{2N_a + N_L} \leq 2 - \frac{1}{4} = \frac{7}{4}.
\]
On the other hand, if $N_a + N_L/2 \leq N_L$, then $N_a < N_L/2$, we have

$$A_2(L)/OPT(L) \leq (3N_a/2 + N_L)/N_L$$

$$= 1 + (3N_a)/(2N_L) \leq 1 + \frac{3}{4} = \frac{7}{4}.$$

Hence the absolute worst-case ratio of $A_2$ is not greater than $\frac{7}{4}$.

Suppose that $k$ is an arbitrarily large integer and $\varepsilon = 1/(6k)$ is a positive number. Consider an instance where we have three types of items. There are $4k$ items of type $A$ with sizes $1 - \varepsilon$, $2k$ items of type $B$ with sizes $3\varepsilon$ and $4k$ items of type $C$ with sizes $1 + \varepsilon$. The incoming order is given below.

$$L = ((1 - \varepsilon, 3\varepsilon, 1 - \varepsilon), \ldots, (1 - \varepsilon, 3\varepsilon, 1 - \varepsilon), 1 + \varepsilon).$$

Clearly, one item of type $A$ and one item of type $C$ can be packed into a bin. All items of type $B$ can be placed into a bin. All bins are completely full. Thus $OPT = 4k + 1$. In the packing made by $A_2$, there are $7k$ bins where $2k$ bins each contain one item of types $A$ and $B$, $k$ bins each contain two items of type $A$ and $4k$ bins each contain one item of type $C$. Then the ratio of $A_2(L)$ and $OPT(L)$ is arbitrarily close to $\frac{7}{4}$. We have proved that the absolute worst-case ratio of Algorithm $A_2$ is $\frac{7}{4}$. □

**Corollary 6.** If there is no large item, the absolute worst-case ratio of algorithm $A_2$ is $\frac{3}{2}$.

**Proof.** If there is no large item in any list $L$, then Algorithm $A_2$ is the same as $A_1$. We immediately get $A_2(L)/OPT(L) \leq \frac{3}{2}$. On the other hand, the following instance shows that $\frac{3}{2}$ is tight. Given six items with sizes $\frac{2}{3}$, $\frac{2}{3}$, $\frac{3}{10}$, $\frac{3}{10}$, $\frac{3}{10}$ and $\frac{3}{10}$, the optimal packing needs two bins and Algorithm $A_2$ requires three bins. □

**Acknowledgements**

We would like to thank the anonymous referee and the Editor for their comments and suggestions that have improved the presentation.

**References**


