THE ORDERED OPEN-ENDED BIN-PACKING PROBLEM

JIAN YANG
Department of Industrial and Manufacturing Engineering, New Jersey Institute of Technology, Newark, New Jersey 07102, yang@adm.njit.edu

JOSEPH Y.-T. LEUNG
Department of Computer Science, New Jersey Institute of Technology, Newark, New Jersey 07102, leung@cis.njit.edu

We study a variant of the classical bin-packing problem, the ordered open-ended bin-packing problem, where first a bin can be filled to a level above 1 as long as the removal of the last piece brings the bin’s level back to below 1 and second, the last piece is the largest-indexed piece among all pieces in the bin. We conduct both worst-case and average-case analyses for the problem. In the worst-case analysis, pieces of size 1 play distinct roles and render the analysis more difficult with their presence. We give lower bounds for the performance ratio of any online algorithm for cases both with and without the 1-pieces, and in the case without the 1-pieces, identify an online algorithm whose worst-case performance ratio is less than 2 and an offline algorithm with good worst-case performance. In the average-case analysis, assuming that pieces are independently and uniformly drawn from [0, 1], we find the optimal asymptotic average ratio of the number of occupied bins over the number of pieces. We also introduce other online algorithms and conduct simulation study on the average-case performances of all the proposed algorithms.

Received April 2001; revisions received July 2001, December 2001, July 2002, October 2002; accepted November 2002.


Area of review: Optimization.

1. INTRODUCTION

In this paper, we consider a variant of the classical bin-packing (CBP) problem, which we call the ordered open-ended bin-packing (OOBP) problem. Like CBP, we are given a list $L = (p_1, p_2, \ldots, p_n)$ of $n$ pieces with each $p_i$ being a real number in $(0, 1]$ or $(0, 1)$ (as will be shown later, pieces of size 1, hereafter to be abbreviated as 1-pieces, play distinct roles in the worst-case analysis) and an infinite collection of unit-capacity bins, and our goal is to pack the pieces into the minimum number of bins. However, unlike CBP, a bin can be filled to a level exceeding 1 given that there exists a designated last piece in the bin such that the removal of this piece brings the bin’s level back to be less than 1. In addition, we require that the designated last piece in a bin be the largest-indexed piece among all pieces in that bin.

OOBP models an optimization problem in fare payment in the subway stations in Hong Kong. There, a passenger can purchase a ticket with a standard denomination, say $20, and the amount is magnetically recorded on the ticket. Every time the passenger reaches his/her destination of a subway trip, the ticket will be presented to a machine which automatically deducts the fare from the ticket. If the remaining balance is still positive, the ticket will be returned to the passenger; otherwise, it will not. Thus, the passenger can gain if the fare of a trip is more than the balance in the ticket at the start of the trip. In this situation, the bins correspond to the tickets and the pieces correspond to the fares. For a traveler who makes several trips, his/her goal is to minimize the number of tickets that he/she needs to purchase. The additional “ordered” requirement is imposed on the problem to reflect that the trips that are charged to each ticket follow the order of the passenger’s overall itinerary.

CBP, the problem from which the current OOBP stems, has been intensively studied since the 1970s both for its wide applicability and its theoretical richness. The readers are referred to the survey by Coffman et al. (1996) and the references therein for a firmer grasp of the problem.

An even more related problem is the plain open-end bin-packing (POBP) problem where the only departure from the current problem is the former’s lack of the “ordered” requirement. This problem was studied by Leung et al. (2001). There, the problem was found to be NP-hard and the next-fit (NF) heuristic was found to be asymptotically the best online algorithm with a worst-case performance ratio of 2. Also, a polynomial time optimization scheme was found.

We shall conduct both worst-case and average-case analyses to algorithms for OOBP. Let us first briefly introduce the measures for gauging the performances of the algorithms. For a given list $L$ and an algorithm $A$, let $A(L)$ be the number of bins used when algorithm $A$ is applied to list $L$ and let $OPT(L)$ denote the optimum number of bins for a packing of $L$. Define $R_A(L) = A(L)/OPT(L)$. The absolute worst-case ratio $R_A$ for algorithm $A$ is defined as

$$R_A = \inf \{ r \mid R_A(L) \leq r \text{ for all lists } L \}.$$ 

The asymptotic worst-case ratio $R_A^*$ is defined as

$$R_A^* = \inf \{ r \mid \text{For some } N > 0, R_A(L) \leq r \text{ for any } L \text{ with } OPT(L) \geq N \}.$$ 

When the list $L_n$ consists of $n$ pieces independently drawn from a random population with distribution $F$, the
average-case ratio \( \overline{R}_A^\infty(F) \) for lists over distribution \( F \) of length \( n \) is defined as
\[
\overline{R}_A^\infty(F) = E[R_A(L_n)] = E\left[ \frac{A(L_n)}{\text{OPT}(L_n)} \right].
\]

The asymptotic average-case ratio \( \overline{R}_A^\infty(F) \) is defined as
\[
\overline{R}_A^\infty(F) = \limsup_{n \to \infty} \overline{R}_A^n(F).
\]

Here, for any algorithm \( A \), we are only interested in \( \overline{R}_A^\infty \), its asymptotic worst-case ratio, and \( \overline{R}_A^\infty(U[0,1]) \), its asymptotic average-case ratio when the distribution is \( U[0,1] \). Because in this paper \( U[0,1] \) is the only distribution and the asymptotic performances are the only performances that we care about, we shall omit mentioning both \( U[0,1] \) and “asymptotic” later on.

For average-case analysis, we use \( \overline{A}^\infty \) to denote \( E[A(L_n)]/n \) and \( \overline{\text{OPT}}^\infty \) to denote \( \limsup_{n \to \infty} \overline{A}^\infty \). Also, we use \( \text{OPT}^\infty \) to denote \( E[\text{OPT}(L_n)]/n \) and \( \overline{\text{OPT}}^\infty \) to denote \( \limsup_{n \to \infty} \overline{\text{OPT}}^\infty \). For OOBP, as for many other problems, obtaining \( \overline{R}_A^\infty \) by its definition seems to be more difficult than obtaining \( \overline{A}^\infty \) and \( \overline{\text{OPT}}^\infty \) individually and applying the identity
\[
\overline{R}_A^\infty = \frac{\overline{A}^\infty}{\overline{\text{OPT}}^\infty}.
\]

Later, we will verify that (1) indeed works for OOBP, which justifies finding \( \overline{A}^\infty \) and \( \overline{\text{OPT}}^\infty \) separately.

We will propose three online algorithms. An online algorithm must pack pieces in the order they arrive without later repacking. A key property of an online algorithm is its nonanticipativity: When packing pieces for lists \( LL_1 \) and \( LL_2 \) separately, it should pack pieces in the \( L \) portions of both lists in the same manner. The three online algorithms are mixed fit (MXF), next fit (NF), and modified best fit (MBF), which we will categorize all the packing patterns into \( K \) types: Those with \( a_1 \geq 1, \) which we let belong to a set \( M_1; \) those with \( a_1 = 0, a_2 \geq 1, \) which we let belong to a set \( M_2; \) those with \( a_1 = 0, a_2 = 0, a_3 \geq 1, \) which we let belong to a set \( M_3; \) and so forth. We can name the patterns 1, 2, \ldots, \( K \), and \( q \) for convenience, denote \( \sum_{k=1}^{K} |M_k| \) by \( Q_k \). For any \( q \) and \( k \), let pattern \( q \) contain \( a_{q,k} \) pieces.

For an online algorithm \( A \), for any \( q \), suppose \( A \) produces \( x_k n + O(n) \) pattern-\( q \) bins when being applied to the long list \( L_1^n L_2^n \cdots L_k^n \). By the conservation of the number of pieces, we must have
\[
\sum_{q=1}^{Q_k} a_{q,k} x_k \geq c_k \quad \forall k = 1, 2, \ldots, K.
\]

By the nonanticipativity of \( A \), we have
\[
A(L_1^n L_2^n \cdots L_k^n) = \sum_{q=1}^{Q_k} x_q n + O(n).
\]

For any \( k \), suppose there is a \( z_k \) such that
\[
\text{OPT}(L_1^n L_2^n \cdots L_k^n) = z_k n + O(n).
\]

Then,
\[
r_k = \limsup_{n \to \infty} A(L_1^n L_2^n \cdots L_k^n)/\text{OPT}(L_1^n L_2^n \cdots L_k^n)
\]
must be equal to \( \sum_{q=1}^{Q_1} x_q / z_k \). For \( r = \max_{k=1}^K r_k \), we then have
\[
z_k r \geq \sum_{q=1}^{Q_1} x_q \quad \forall k = 1, 2, \ldots, K.
\]
Therefore, the linear program \( \min r = (2) \), \( (3) \), \( x_q \geq 0 \) for \( q = 1, 2, \ldots, Q_1 \) provides a lower bound for \( R_{\alpha}^* \). On the other hand, we say that pattern \( q \) is totally dominated by another pattern if they belong to the same \( M_k \) for some \( k \), and for every \( k' = k, \ldots, K \), \( a_{g_{k'}} \leq a_{g_{k}} \). In the above linear program, we apparently do not have to consider any pattern that is totally dominated by another pattern.

Consider the following sequence of numbers: \( b_1 = 1 \), \( b_2 = 2, b_3 = b_1 b_2 + 1 = 3 \), \( b_4 = b_1 b_3 + 1 = 7 \), \( \ldots \), \( b_k = b_1 b_2 \cdots b_{k-1} + 1 \). For a given \( K \), we let \( p_1 = 1/b_1 \), \( p_2 = 1/b_{k-1} \), \( \ldots \), \( p_{k-1} = 1/b_2 = 1/2 \), \( p_k = 1/b_1 = 1 \); and \( c_1 = c_2 = \cdots = c_k = 1 \). Using a C program, we can identify all undominated patterns. We can then find the \( z_k \)s by solving certain simple linear programs using CPLEX. Finally, we can solve the linear program mentioned in the preceding paragraph by again using CPLEX. So far, we have the results for \( K \) upwards to 6. When \( K = 7, b_7 \) is already as large as 3,263,443 and the number of undominated patterns is simply unmanageable by our personal computer. Table 1 shows the results, where \( Q \) is the number of undominated patterns. Hence, we obtain a lower bound of 1.630297 for \( R_{\alpha}^* \).

Without the presence of the 1-pieces, we have to settle with a slightly lower lower bound.

**Theorem 2.** *Without the 1-pieces, \( R_{\alpha}^* \geq 1.415715 \) for any online algorithm \( A \) for OOBP.*

**Proof.** We use the same idea and notation here as in the proof of Theorem 1. The only difference here is that, for a given \( K \), we let \( p_1 = 1/b_{k+1} \), \( p_2 = 1/b_k \), \( \ldots \), \( p_{k-1} = 1/b_3 = 1/3 \), \( p_k = 1/b_2 = 1/2 \); and \( c_1 = c_2 = \cdots = c_{k-1} = 1 \) and \( c_k = 2 \). Table 2 shows the achievable results for the current setting. Hence, we obtain a lower bound of 1.415715 for \( R_{\alpha}^* \).

### Table 1. Results for Theorem 1.

<table>
<thead>
<tr>
<th>( K )</th>
<th>( b_k )</th>
<th>( Q )</th>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( z_3 )</th>
<th>( z_4 )</th>
<th>( z_5 )</th>
<th>( z_6 )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1.333</td>
<td>1</td>
<td>1.333</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>8</td>
<td>0.3333</td>
<td>0.6667</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>22</td>
<td>0.142857</td>
<td>0.375</td>
<td>0.6667</td>
<td>1</td>
<td>1</td>
<td>1.62018</td>
<td>1.62018</td>
</tr>
<tr>
<td>5</td>
<td>43</td>
<td>87</td>
<td>0.023256</td>
<td>0.145833</td>
<td>0.375</td>
<td>0.6667</td>
<td>1</td>
<td>1.629434</td>
<td>1.629434</td>
</tr>
<tr>
<td>6</td>
<td>1,807</td>
<td>1,291</td>
<td>0.000553</td>
<td>0.023268</td>
<td>0.145833</td>
<td>0.375</td>
<td>0.6667</td>
<td>1</td>
<td>1.630297</td>
</tr>
</tbody>
</table>

### Table 2. Results for Theorem 2.

<table>
<thead>
<tr>
<th>( K )</th>
<th>( b_{k+1} )</th>
<th>( Q )</th>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( z_3 )</th>
<th>( z_4 )</th>
<th>( z_5 )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0.3333</td>
<td>1</td>
<td>1</td>
<td>1.2857</td>
<td>1</td>
<td>1.2857</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>13</td>
<td>0.142857</td>
<td>0.375</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.39007</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td>57</td>
<td>0.023256</td>
<td>0.145833</td>
<td>0.375</td>
<td>1</td>
<td>1</td>
<td>1.41496</td>
</tr>
<tr>
<td>5</td>
<td>1,807</td>
<td>912</td>
<td>0.000553</td>
<td>0.023268</td>
<td>0.145833</td>
<td>0.375</td>
<td>1</td>
<td>1.415715</td>
</tr>
</tbody>
</table>

### 2.2. The Mixed-Fit (MXF) Algorithm

A reasonable algorithm will have the majority of bins filled to levels no less than 1; while under any algorithm, no bin can be filled to a level more than 2. Hence, any reasonable algorithm will have a worst-case performance ratio of no more than 2. Nevertheless, it is not an easy task to find an online algorithm with a provable less-than-2 worst-case performance ratio. Even without the 1-pieces, the mixed-fit (MXF) algorithm is the only such algorithm we have found so far.

The mixed-fit (MXF) algorithm: Bins consisting of small pieces are only closed by the arrivals of large pieces. This prevents the premature closing of bins. In this algorithm, we divide the pieces into four types according to their sizes \( p \), with a type-1 piece having \( 0 < p < 1/3 \), a type-2 piece having \( 1/3 < p < 1/2 \), a type-3 piece having \( 1/2 < p < 1 \), and a type-4 piece having \( p = 1 \). Accordingly, we define a type-1 open bin as an open bin containing a number of type-1 pieces, a type-2 open bin as a bin containing one or two type-2 pieces, and a type-3 open bin as a bin containing one type-3 piece. We pack pieces in the order of their arrivals. When the current piece is of type-1, we pack it into the type-1 open bins in the first-fit (FF) manner for CBP, without closing any bins. When the current piece is of type-2, we pack it into type-2 open bins, again in FF manner for CBP, without closing any bins. Suppose the current piece is of type-3 or type-4. We pack this piece into the first type-1 or type-2 open bin whose level is at least \( 1/3 \) and close it. If there is no such a bin, we pack the piece into a type-3 open bin and close it. If again there is no such a bin, we pack the piece into a new bin, and close it if the piece is a type-4 piece. The running time for MXF is \( O(n \log n) \).

The following theorem, whose proof is in the Appendix for its lengthiness, offers an upper bound for \( R_{\alpha}^* \) when there is no 1-piece.

**Theorem 3.** *Without the 1-pieces, \( R_{\alpha}^* \leq 35/18 \approx 1.9444 \).*
The following result, however, does not involve the 1-pieces.

**Theorem 4.** Regardless of the presence of the 1-pieces, $R_{\text{MXF}}^c \geq 25/13 \simeq 1.9231$.

**Proof.** Consider the following sequence of numbers: $b_1 = 3, b_2 = 4, b_3 = b_1 b_2 + 1 = 13, \ldots, b_k = b_1 b_2 \cdots b_{k-1} + 1, \ldots$. For a given $K$, let $L_K$ be a list with $K(K+2)$ pieces, which consists of first $K(b_k - 1)$ pieces of size $1/b_k$, then $K(b_k - 1)$ pieces of size $1/b_{k-1}$, $\ldots$, then $K(b_k - 1)$ pieces of size $1/b_2 = 1/4$, and finally $3K(b_k - 1)$ pieces of size $1/b_1 = 1/3$.

Apply MXF to $L_K$ and we will get $K[3(b_k - 1)/2 + \sum_{k=2}^K \Pi_{i=k}^{k-1} b_k]$ bins: $K$ bins each containing $(b_k - 1)$ pieces of size $1/b_k$, $Kb_{k-1}$ bins each containing $(b_{k-1} - 1)$ pieces of size $1/b_{k-1}$, $Kb_{k-2}$ bins each containing $(b_{k-2} - 1)$ pieces of size $1/b_{k-2}$, $\ldots$, $Kb_1$ bins each containing $(b_1 - 1)$ pieces of size $1/b_1 = 1/4$, and $3K(b_k - 1)/2$ bins each containing $(b_1 - 1) = 2$ pieces of size $1/b_1 = 1/3$. On the other hand, the optimal solution occupies only $K(b_k - 1)$ bins, each containing one piece of size $1/b_k$, one piece of size $1/b_{k-1}$, $\ldots$, one piece of size $1/b_2 = 1/4$, and three pieces of size $1/b_1 = 1/3$. Hence,

$$R_{\text{MXF}}^c \geq \frac{3}{2} + \frac{3}{b_1 - 1} + \frac{1}{b_2 - 1} + \frac{1}{b_3 - 1} + \cdots + \frac{1}{b_k - 1} \geq \frac{3}{2} + \frac{1}{3} + \frac{1}{12} + \frac{1}{156} = \frac{25}{13} \simeq 1.9231.$$

Because $b_k$ is already 24,493 and the sequence grows faster than exponentially, the improvement we can gain from going to $K$s higher than 4 is negligible. \qed

### 2.3. The Greedy Look- Ahead Next-Fit (GLANF) Algorithm

The greedy look-ahead next-fit (GLANF) algorithm: There is one open bin at any moment. GLANF keeps on filling the current bin with pieces in their original order unless the first piece is 1 or the addition of the next piece will bring the bin’s level to be at least 1. For the latter situation, GLANF makes some greedy effort in filling the current open bin to the highest-possible level. We describe this effort in more detail in the following.

When an empty bin is just opened, we can always say that $p(i_1), \ldots, p(i_w)$ are the pieces in the targeted list $L$ that are not packed yet, where $i(1) < \cdots < i(w)$. To ease our notational burden, we let each $i(k)$ be $k$. There must be a nonnegative integer $r$ and $2r+2$ values $s_0, t_0, \ldots, s_r, t_r$ with $1 = s_0 \leq t_0 < s_1 < t_1 < \cdots < s_{r-1} < t_{r-1} < s_r \leq t_r = w+1$ such that:

1. for any $m$ between 0 and $r$,

$$\sum_{m=0}^m \sum_{j=m}^{m-1} p_j < 1;$$

2. for any $m$ between 0 and $r-1$ and any $u$ between $t_m$ and $s_{m+1} - 1$,

$$\sum_{m'=0}^m \sum_{j=m'}^{m-1} p_j + p_u \geq 1.$$

Note that the above $s_0, \ldots, t_r$ are unique. For any $m$ between 0 and $r-1$, let $l(m) = \arg\max\{p_j \mid t_m \leq j \leq s_{m+1} - 1\}$. The current open bin will be filled to the level $\max_{m'=0}^{m} \sum_{j=m'+1}^{m} p_j + p_{l(m)}$ with the corresponding pieces.

When running GLANF, every time a new bin is to be packed the selection of the remaining pieces to be packed into it can be done by one sweep of all the pieces. Thus, every bin takes $O(n)$ time to be packed, and hence the running time of GLANF is $O(n^2)$.

We can prove an upper bound for $R_{\text{GLANF}}^c$ when there is no 1-piece.

**Theorem 5.** Without the 1-pieces, $R_{\text{GLANF}}^c \leq 3/2$.

**Proof.** We assign weights $W(\cdot)$ to pieces. For a small piece $p \in (0, 1/2]$, we let $W(p) = p$; and for a large piece $p \in (1/2, 1)$, we let $W(p) = 1/2$. For any bin $B$, we let the weight of the bin $W(B)$ be the total weight of all pieces in it. It is easy to see that $W(B) < 3/2$ for any bin in any packing. Thus, $W(L) < 3 \cdot OPT(L)/2$, where $W(L)$ is the total weight of all pieces in $L$. We shall show that $W(L) \geq GLANF(L) - O(1)$. Combining the two, we get the desired result.

It is clear that GLANF fills every bin to a level of at least 1, except the last one. From now on, let us ignore this last bin. If a bin in the GLANF packing either has no large piece or has two large pieces, then the bin has a weight of at least 1. Thus, if a bin has a weight less than 1, it must have exactly one large piece. We now show that after deleting $O(1)$ bins, the average weight of bins in the GLANF packing is at least 1.

Let $B_k$ be the first bin in the GLANF packing with a weight less than 1. Let the total size of all small pieces in $B_k$ be $x$ and the size of the large piece be $a$. Because $B_k$ has a weight less than 1, we have $x < 1/2$.

We first assume that the large piece in $B_k$ is not the top piece in the bin. In this case we assert that there is no large piece appearing in any bins following $B_k$, and hence each of the subsequent bins has a weight of at least 1 and the theorem is proved. Suppose not. Let $b$ be a large piece appearing in bin $B_{k+j}$. It is easy to see that $b$ must appear in the list before the top piece in $B_k$; otherwise, $b$ would have been packed into $B_k$ by the GLANF algorithm because $b$ is a large piece and the top piece in $B_k$ is a small piece. However, this means that $b$ was skipped over when the GLANF algorithm considered packing $b$ into $B_k$, which implies that the total size of all small pieces after $b$ in the list that were packed into $B_k$ is larger than or equal to the size of $b$. Because the size of $b$ is larger than 1/2, the total size of all small pieces in $B_k$ is larger than 1/2, contradicting our assumption that $x < 1/2$. 

From the above argument, we may assume that the large piece in $B_k$ is the top piece in the bin. Let $B_{k+j}$ be the first bin after $B_k$ such that (1) $B_{k+j}$ has a weight less than 1, and (2) the sum of the weights of the bins $B_k, B_{k+1}, \ldots, B_{k+j-1}$ is less than $j$. By the previous argument, we may assume that the large piece in $B_{k+j}$ is the top piece in the bin. Let the size of the large piece in $B_{k+j}$ be $b$ and the total size of all small pieces in $B_{k+j}$ be $y$. By our assumption, we have $y < 1/2$.

We first show that $j$ cannot be 1. Suppose $j = 1$. The small pieces in $B_{k+1}$ must appear in the list after $a$; otherwise, they would have been packed into $B_k$ by the GLANF algorithm. Because $y < 1/2$ and the total size of all the pieces in $B_{k+1}(b + y) > 1$ is larger than the size of $a$, the GLANF algorithm would have packed all the pieces in $B_{k+1}$, instead of $a$, into $B_k$. This is the contradiction we sought. Thus, $j \geq 2$.

Assume for the moment that there is no bin among $B_{k+1}, B_{k+2}, \ldots, B_{k+j-1}$, that contains only small pieces. In other words, each bin has one or two large pieces. First, assume that every bin has two large pieces. In this case the total size of all the small pieces in $B_k, B_{k+1}, \ldots, B_{k+j}$ is less than 1, by definition of $B_{k+j}$. The GLANF algorithm would have packed all these small pieces along with $b$, instead of $a$, into $B_k$, because the total size of all the former pieces is at least 1 and $a$ is less than 1. Thus, there must be a bin $B_{k+i}$ such that $B_{k+i}$ contains one large piece, and each of the bins $B_{k+i+1}, B_{k+i+2}, \ldots, B_{k+j-1}$ contains two large pieces.

Suppose the large piece in $B_{k+i}$ is the top piece. Then, the total size of all the small pieces in $B_k, B_{k+1}, \ldots, B_{k+i}$ is less than 1, by definition of $B_{k+i}$. The GLANF algorithm would have packed all the small pieces in $B_{k+1}, \ldots, B_{k+i-1}$, and all the pieces in $B_{k+i}$ into $B_k$, because the total size of all these pieces is larger than $a$. Thus, the large piece in $B_{k+i}$ must not be the top piece. In other words, the top piece in $B_{k+i}$ is a small piece. As before, $b$ must appear in the list before the top piece in $B_{k+i}$. Let $z_j$ be the total size of all small pieces in $B_{k+i}$ that appear before $b$ in the list, and let $z_2$ be the total size of all small pieces in $B_{k+i}$ that appear after $b$. By the nature of the GLANF algorithm, we have $z_2 \geq b > 1/2$. A moment of reflection shows that the GLANF algorithm would have packed all the small pieces in $B_{k+i}, B_{k+i+1}, \ldots, B_{k+i-1}$, all the small pieces in $B_{k+i}$ that appear before $b$ in the list, and all the pieces in $B_{k+i}$, into $B_k$. This is because the total size of all small pieces in $B_k, B_{k+1}, \ldots, B_{k+i}$ is less than 1. Because $z_2 > 1/2$ and $y < 1/2$, the small pieces can all be packed into $B_k$ without causing it to be closed. Thus, this possibility does not exist either.

Finally, we consider the possibility where there is a bin, say $B_{k+l}$, among $B_{k+l}, B_{k+l+1}, \ldots, B_{k+j-1}$, such that $B_{k+l}$ has only small pieces. We want to show that the weight of $B_{k+l}$ plus the weight of $B_{k+l}$ is greater than 2. (In this case the argument continues for subsequent bins, looking for the first bin $B_{k+l}$, $l > j$, such that $B_{k+l}$ has a weight less than 1.) Observe that $b$ appears in the list before the top piece in $B_{k+l}$, otherwise, $b$ would have been packed into $B_{k+l}$ as the top piece. The small pieces in $B_{k+l}$ all appear in the list before $b$. Let $z_i$ be the total size of all pieces in $B_{k+i}$ that appear before $b$ in the list, and $z_3$ be the total size of all pieces that appear after $b$ in the list. From the nature of the GLANF algorithm, we have $z_3 \geq b > 1/2$ and $z_2 + y > 1$. Thus, the sum of the weights of the two bins, $B_{k+l}$ and $B_{k+j}$, is greater than 2.

Borrowing ideas from the worst-case example for the first fit (FF) algorithm for CBP (Johnson et al. 1974), a lower bound for $R_{\text{GLANF}}^\infty$ when there is no 1-piece can be found in the following theorem, whose proof is in the Appendix.

**Theorem 6. Without the 1-pieces, $R_{\text{GLANF}}^\infty \geq 27/20 = 1.35$.**

However, when 1-pieces are allowed in the list, we can “improve” the lower bound for $R_{\text{GLANF}}^\infty$ to 3/2.

**Theorem 7. With the 1-pieces, $R_{\text{GLANF}}^\infty \geq 3/2$.**

**Proof.** Given an arbitrary positive integer $K$, consider a list $L_K$ with $K(4K + 2)$ pieces, where the first $K$ pieces have the same size of $1 − 1/(2K)$, the next $K$ pieces have the same size of 1, the next $K(4K − 1)$ pieces have the same size of $1/(4K)$, and the last $K$ pieces have the same size of 1.

Apply GLANF to $L_K$ and we will get 3 bins: $K$ bins each containing one piece of size $1 − 1/(2K)$, one piece of size $1/(4K)$, and one piece of size 1; $K$ bins each containing one piece of size 1; $K − 1$ bins each containing $4K$ pieces of size $1/(4K)$, and 1 last bin containing 2 pieces of size $1/(4K)$. On the other hand, the optimal solution occupies only 2 bins: $K$ bins each containing one piece of size $1 − 1/(2K)$ and one piece of size 1, and $K$ bins each containing $4K − 1$ pieces of size $1/(4K)$ and one piece of size 1.

### 3. Average-Case Analysis

For OOBP, we can easily verify that we are able to use (1) in the Introduction. First, $OPT(\cdot)$ is a subadditive function. (Unlike those for CBP and POBP, the $OPT(\cdot)$ is a function of lists rather than a function of sets in that not only what the pieces are, but also in what order the pieces appear, affects its value. However, our arguments follow with this fact.) That is, given two lists $L^1$ and $L^2$, we have

$$OPT(L^1L^2) \leq OPT(L^1) + OPT(L^2).$$

The reason is simple: Combining the optimal solutions for $L^1$ and $L^2$, we get at least a feasible solution for $L^1L^2$. Next, when two lists $L$ and $L'$ differ only in one piece, the resulting $OPT(L)$ and $OPT(L')$ apparently differ by at most 1. So, using the standard techniques revolving around Azuma’s Inequality (Steele 1997), we have

$$Pr(|OPT(L_n) − E[OPT(L_n)]| ≥ t) \leq 2e^{−t^2/2n}.$$

Finally, it is also obvious that for every algorithm $A$ we have considered,

$$A(L_n) < n \times OPT(L_n).$$
As was pointed out by Coffman et al. (1996), these three properties guarantee the validity of (1) in the Introduction.

3.1. A Lower Bound for Arbitrary Algorithms

Obviously, we first need to find \( \text{OPT}^\infty \) for OOBP. A trivial lower bound for \( \text{OPT}^\infty \) is \( 1/4 \), due to the facts that no bin can have a level higher than 2 and that the average size of pieces is \( 1/2 \). On the other hand, we do find a lower bound for \( \text{OPT}^\infty \) that is significantly higher than the trivial one.

**Theorem 8.**
\[ \text{OPT}^\infty \geq 2 - \sqrt{3}. \]

**Proof.** Let \( \{ (1), \ldots, (n) \} \) be a permutation of \( \{ 1, \ldots, n \} \) such that \( p_1 \leq \cdots \leq p_n \). Then, we may first prove that
\[ \text{OPT}(L_n) \geq K_0(L_n) + 1 \]
\[ \equiv \max \{ k = 0, 1, \ldots, n \mid p_1 + \cdots + p_{n-k} \geq k \} + 1. \]

Let \( \text{OPT}'(L_n) \) be the number of bins used by POBP when being applied to \( L_n \). Because POBP is a relaxation of OOBP, it follows that \( \text{OPT}(L_n) \geq \text{OPT}'(L_n) \). For any optimal solution of POBP, if there are two pieces \( p_i \) and \( p_j \) such that \( p_i < p_j \) and \( p_i \) is the last piece of a bin, then swapping the places of these two pieces will produce another solution with no more bins. Keep on swapping and we will get an optimal solution for POBP in which pieces on top of the bins are those largest pieces. Hence, the total size of the remaining smaller pieces does not exceed the optimal number of bins:
\[ p_1 + \cdots + p_{\text{n-OPT}(L_n)} < \text{OPT}'(L_n). \]

Now, \( p_1 + \cdots + p_{n-k} \) is a decreasing function of \( k \). So, for any \( k \geq \text{OPT}'(L_n) \), it follows that
\[ p_1 + \cdots + p_{n-k} < k. \]

Therefore,
\[ \text{OPT}'(L_n) \geq K_0(L_n) + 1 \]
\[ \equiv \max \{ k = 0, 1, \ldots, n \mid p_1 + \cdots + p_{n-k} \geq k \} + 1. \]

Given two lists \( L_1 \) and \( L_2 \), by the definition of \( K_0(\cdot) \), the smallest \( |L_1| - K_0(L_1) - 1 \) pieces in \( L_1 \) add up to less than \( K_0(L_1) + 1 \) and the smallest \( |L_2| - K_0(L_2) - 1 \) pieces add up to less than \( K_0(L_2) + 1 \). Therefore, the smallest \( |L_1| + |L_2| - K_0(L_1) - K_0(L_2) - 2 \) pieces in \( L_1 \cup L_2 \) add up to less than \( K_0(L_1) + K_0(L_2) + 2 \). Hence,
\[ K_0(L_1L_2) < K_0(L_1) + K_0(L_2) + 2. \]

That is, \( K_0(\cdot) \) is a subadditive function. By Fekete’s Lemma (Steele 1997), \( \lim_{n \to \infty} \frac{E[K_0(L_n)]}{n} \) exists and we denote it by \( \overline{K}_0 \). Also, by definition,
\[ p_1 + \cdots + p_{n-K_0(L_n)-1} + p_{n-K_0(L_n)} \geq K_0(L_n) \]
and
\[ p_1 + \cdots + p_{n-K_0(L_n)-1} < K_0(L_n) + 1. \]

So, by the boundedness of the \( E[p] \)s, we have
\[ \lim_{n \to \infty} \frac{E[p_1 + \cdots + p_{n-K_0(L_n)}]}{n} = \lim_{n \to \infty} \frac{E[K_0(L_n)]}{n} = \overline{K}_0. \]

However, due to the nature of the order statistics, we have
\[ \lim_{n \to \infty} \frac{E[p_1 + \cdots + p_{n-K_0(L_n)}]}{n} \]
\[ = \lim_{n \to \infty} \left[ \frac{1}{n+1} + \cdots + \frac{n - K_0(L_n)}{n+1} \right] = \frac{(1 - \overline{K}_0)^2}{2}. \]

Hence,
\[ \overline{K}_0 = \frac{(1 - \overline{K}_0)^2}{2}. \]

Therefore, \( \overline{K}_0 = 2 - \sqrt{3} \). At last, we get
\[ \lim_{n \to \infty} \frac{E[\text{OPT}(L_n)]}{n} \geq \lim_{n \to \infty} \frac{E[\text{OPT}'(L_n)]}{n} \]
\[ \geq \lim_{n \to \infty} \frac{E[K_0(L_n) + 1]}{n} = \overline{K}_0 \]
\[ = 2 - \sqrt{3}. \]

3.2. The Divide-and-Pack (DP) Algorithm

The **divide-and-pack** (DP) algorithm: For convenience, denote \( \lfloor n^{2/3} \rfloor \) by \( m_1 \), the quotient of \( n \) being divided by \( m_1 \), by \( m_2 \), and the remainder of \( n \) being divided by \( m_1 \) by \( r \). Given a list \( L_n \), DP first divides it into \( m_2 + 1 \) smaller lists \( L_n^1, L_n^2, \ldots, L_n^m \), and \( L_n^{m+1} \), with \( L_n^1 \) containing the first batch of \( m_1 \) pieces in \( L_n \), \( L_n^2 \) containing the second batch of \( m_1 \) pieces in \( L_n \), and so forth, and \( L_n^{m+1} \) containing the remaining pieces in \( L_n \). It will not matter whether \( L_n^{m+1} \) is empty or not. Now, for \( i = 1, \ldots, m_2 \), let
\[ L_n^i = \{ p \in L_n \mid p \leq \sqrt{3} - 1 \} \]
and \( L_n^{i+1} = L_n^i \setminus L_n^i \). Then, for \( i = 1, \ldots, m_2 - 1 \), DP applies algorithm DP’ for POBP to \( L_n^i \cup L_n^{i+1} \). Finally, DP packs the remaining pieces in \( L_n^{m+1} \) into \( \text{OPT}^C(L_n^1) \) in the next-fit (NF) fashion, on whose description we feel no need to elaborate right here. The tricks we played in the above guarantee that the “ordered” requirement is satisfied even when the pieces are meddling by DP’.

The description of DP’ for POBP is as follows: For a given list \( L_n \), let
\[ L_n^i = \{ p \in L_n \mid p \leq \sqrt{3} - 1 \} \]
and \( L_n^{i+1} = L_n \setminus L_n^i \). Use the asymptotically average-sense optimal algorithm for CBP (there exists such a perfect packing) to pack the pieces in \( L_n^i \) into \( \text{OPT}^C(L_n^i) \) bins. For convenience, denote \( |L_n^i| \) by \( T_i \), \( |L_n| - T_i \) by \( r_i \), and \( \text{OPT}^C(L_n^i) \)
by $Q$. Then, put $\min\{R, Q\}$ pieces in $L^k$ onto the existing bins which are not of exactly size 1 (almost surely there are $Q$ of them), one on top of every existing bin. Finally, pack the remaining $(R - Q)^+$ pieces in $L^k$ into $(R - Q)^+$ individual bins. In total, algorithm DP consumes $DP'(L) = \max\{Q, R\}$ bins. Note that, for those $(Q - R)^+$ bins consisting of pieces solely from $L^3$, because their levels are all below 1, one can always rearrange pieces in them so that the pieces on top are always those largest-indexed.

Strictly speaking, the asymptotically average-sense algorithm for CBP is actually not one algorithm, but a sequence of algorithms. In real implementation, we will have to select the most appropriate one from the sequence, depending on the lists to be faced. We will discuss the implementation of DP later on.

**Theorem 9.**

\[ DP^* \leq 2 - \sqrt{3}. \]

**Proof.** For $i = 1, \ldots, m_1 - 1$, we denote $|L^{is}_n|$ by $T_i$, $OPT^*(L^{is}_n)$ by $Q_i$, $|L^{is+1}_n|$ by $R_i$, and $\sum_{p \in L^k} b$ by $\Sigma_i$. Also, for any arbitrary $\epsilon > 0$, we define event sets

\[ C^{es}_i = \{ T = 0, 1, \ldots, m_1 \mid |T_1 - (\sqrt{3} - 1)m_1| \leq m_1^{1/2+\epsilon} \}, \]

\[ C^{el}_i = \{ R = 0, 1, \ldots, m_1 \mid |R_1 - (\sqrt{3} - 1)m_1| \leq m_1^{1/2+\epsilon} \}, \]

and also define $C^{es}$ and $C^{el}$ as the complementary events of $C^{es}_i$ and $C^{el}_i$, respectively. $T_i$ is a binomial random variable with parameters $m_1$ and $\sqrt{3} - 1$ and $R_i$ is a binomial random variable with parameters $m_1$ and $2 - \sqrt{3}$. Random variable $R_i$ is independent of random variables $T_i$ and $\Sigma_i$. According to Coffman and Lueker (1991, p. 14), we have

\[ Pr[T_i \in C^{es}_i] = O\left(\frac{1}{m_1^2}\right) \text{ and } Pr[R_i \in C^{el}_i] = O\left(\frac{1}{m_1^2}\right). \]

So,

\[ E[DP'(L^{is}_n, L^{is+1}_n)] \]

\[ \leq E[\max\{Q_i, R_i\}] \]

\[ \leq E[\max\{Q_i, R_i\} \mid T_i \in C^{es}_i \text{ and } R_i \in C^{el}_i] \]

\[ + E[\max\{Q_i, R_i\} \mid T_i \in C^{es}_i \text{ or } C^{el}_i] \]

\[ \times Pr[T_i \in C^{es}_i \text{ or } C^{el}_i] \]

\[ \leq E[\max\{Q_i, R_i\} \mid T_i \in C^{es}_i \text{ and } R_i \in C^{el}_i] + O\left(\frac{1}{m_1}\right) \]

\[ \leq E[\max\{Q_i, (2 - \sqrt{3})m_1 + m_1^{1/2+\epsilon}\} | T_i \in C^{es}_i] + O\left(\frac{1}{m_1}\right). \]

Our remaining job is to show

\[ E[\max\{Q_i, (2 - \sqrt{3})m_1 + m_1^{1/2+\epsilon}\} | T_i \in C^{es}_i] \]

\[ < (2 - \sqrt{3})m_1 + O(m_1^{1/2+\epsilon}), \]

and therefore $E[DP'(L^{is}_n, L^{is+1}_n)]$ has the same bound.

By the central limit theorem,

\[ 12((\Sigma_1 | T_1 = t) - (\sqrt{3} - 1)t/2)/((3 - \sqrt{3})\sqrt{t}) \]

approaches the standard normal distribution $N(0, 1)$ as $t$ tends to $+\infty$. The convergence rate can be bounded by the Berry-Esseen Theorem (Patel and Read 1982), from which we can derive

\[ Pr\left[ \Sigma_1 < \frac{\sqrt{3} - 1}{2}t \right] \]

\[ < \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\sqrt{3} - 1} e^{-x^2/2} dx + O\left(\frac{1}{\sqrt{t}}\right) \]

\[ = \frac{1}{2\sqrt{2\pi}} \int_{-\sqrt{3} + 1}^{\sqrt{3} - 1} \frac{dx}{x^{1/2}} + O\left(\frac{1}{\sqrt{t}}\right) = O\left(\frac{1}{\sqrt{t}}\right). \]

Hence, by the definition of $C^{is}_e$, we have

\[ Pr[Q_i < (2 - \sqrt{3})m_1 - m_1^{1/2+\epsilon} | T_i \in C^{is}_e] = O\left(\frac{1}{m_1}\right). \]

Because $Q_1 \geq \Sigma_1$ for sure, we get

\[ Pr[Q_i < (2 - \sqrt{3})m_1 - m_1^{1/2+\epsilon} | T_i \in C^{is}_e] = O\left(\frac{1}{m_1}\right). \quad (4) \]

When $T_i = t$, the $t$ pieces in $L^{is}_n$ are i.i.d. random variables uniformly distributed in $[0, \sqrt{3} - 1]$. Hence, the distribution function for each of these pieces is nonincreasing. According to Karmarkar (1982), Knodel (1981), and Loulou (1984), perfect packing in the CBP sense exists for these pieces:

\[ E[Q_i | T_i = t] = E[\Sigma_i | T_i = t] + O(\sqrt{t}) = \frac{\sqrt{3} - 1}{2}t + O(\sqrt{t}). \]

Then, again by the definition of $C^{is}_e$, we have

\[ E[Q_i | T_i \in C^{is}_e] \leq (2 - \sqrt{3})m_1 + O(m_1^{1/2+\epsilon}). \quad (5) \]

For nonnegative random variable $a$, and nonnegative constants $A$ and $B$ where $A \leq B$, because

\[ \max\{a, B\} \leq B \times 1[a < A] + (a + B - A) \times 1[a \geq A], \]

we have

\[ E[\max\{a, B\}] \leq B \times Pr[a < A] + E[a] + B - A. \]

Treating $(Q_i | T_i \in C^{is}_e)$ as $a$, $(2 - \sqrt{3})m_1 - m_1^{1/2+\epsilon}$ as $A$, and $(2 - \sqrt{3})m_1 + m_1^{1/2+\epsilon}$ as $B$, and combining inequalities $(4)$ and $(5)$, we get

\[ E[\max\{Q_i, (2 - \sqrt{3})m_1 + m_1^{1/2+\epsilon}\} | T_i \in C^{is}_e] \]

\[ \leq E[Q_i, T_i \in C^{is}_e] + 2m_1^{1/2+\epsilon} + (2 - \sqrt{3})m_1 + m_1^{1/2+\epsilon} \]

\[ \times Pr[Q_i < (2 - \sqrt{3})m_1 - m_1^{1/2+\epsilon} | T_i \in C^{is}_e] \]

\[ \leq (2 - \sqrt{3})m_1 + O(m_1^{1/2+\epsilon}). \]
Now, we have

\[
E[DP(L_{n})] = \sum_{i=1}^{m_{2}} E[DP'((L_{2i}^{i+1})_{n})] + E[\text{NF}(L_{1_{0}}L_{n}^{m_{2}}L_{n+1_{0}})]
\]

\[
= (m_{2} - 1)E[\text{DP}'((L_{2i}^{i+1})_{n})] + O(n^{2/3})
\]

\[
\leq (m_{2} - 1)((2 - \sqrt{3})n + O(m_{1}^{2/3}e)) + O(n^{2/3})
\]

\[
\leq (2 - \sqrt{3})n + O(n^{2(1+e)/3}). \quad \square
\]

Indeed, we have succeeded in finding the optimal average-case performances for both OOBP and POBP. Namely, we have found that \(OPT' = OPT'' = 2 - \sqrt{3} \cong 0.268\).

### 3.3. Implementation Issues of the DP Algorithm

Algorithm DP involves the perfect packing in the CBP sense for pieces independently drawn from the distribution \(U[0, \sqrt{3} - 1] \). As described in Coffman and Lueker (1991, pp. 105–106), the perfect packing in turn involves partitioning the above distribution into an infinite sequence of symmetric uniform distributions centered around \(2^{-k}\)s for \(k = 1, 2, \ldots\) In real implementations, we have to make a truncation at some \(K\) and treat pieces that fall in the intervals of the first \(K\) distributions and pieces that do not differently. For convenience, we further let \(DP_{K}\) refer to the truncated version of \(DP\) at a particular \(K\).

We implement \(DP_{K}\) as in the following. Given a list \(L_{n}\), we first partition it into \(L_{n}^{1_{0}}, L_{n}^{1}, \ldots, L_{n}^{m_{2}}, L_{n}^{m_{1}}, \text{and } L_{n}^{m_{0}+1}\) as in the description of \(DP\). For each \(L_{n}^{i}\) for \(i = 1, \ldots, m_{2} - 1\), we apply to it a truncated version of the perfect packing method. To do so, we first partition \(U[0, \sqrt{3} - 1] \) as

\[
U[0, \sqrt{3} - 1] = \sum_{k=1}^{K} \frac{b_{k} - a_{k}}{\sqrt{3} - 1} \times U[a_{k}, b_{k}] + \frac{b_{K+1} - \sqrt{3}}{\sqrt{3} - 1} \times U[0, b_{K+1}],
\]

where \(b_{1} = \sqrt{3} - 1\), \(a_{1} = 2 - \sqrt{3}\); for \(k = 2, \ldots, K\), \(b_{k} = a_{k-1}\), \(a_{k} = 2^{1-k} - b_{k}\) when \(2^{1-k} < a_{k-1}\) and \(a_{k} = b_{k}\) otherwise; and \(b_{K+1} = a_{K}\). Furthermore, we partition \(L_{n}^{i}\) into \(L_{n}^{i, 1}, \ldots, L_{n}^{i, K+1}\) so that each \(L_{n}^{i, k}\) contains all pieces that fall in the interval of the \(k\)th uniform distribution in the above. Then, we apply the MATCH algorithm described in Coffman and Lueker (1991, p. 100) to each \(L_{n}^{i, k}\) for \(k = 1, \ldots, K\) so that the pieces there are packed into subbins with size \(2^{1-k}\). MATCH keeps on putting the currently largest unpowered piece into a new subbin and trying to add to the bin, if possible, another currently unpowered piece that is the largest possible. Afterwards, we pack the resulting subbins along with pieces in \(L_{n}^{i, K+1}\) into bins of size 1 using the next fit decreasing (NFD) algorithm for CBP. After all pieces in every \(L_{n}^{i}\) for \(i = 1, \ldots, m_{2} - 1\) have been packed, the rest of the implementation can just follow the description of \(DP\).

Each \(|L_{n}^{i}|\) specified in \(DP_{K}\) is in the order of \(n^{2/3}\). The chance of there being at least one piece that does not fall in the intervals of the first \((2 \log_{2} n) / 3\) distributions is significantly less than 1. These pieces thus have only an \(O(1)\) contribution to the packing result of \(L_{n}^{i}\), and hence we can treat them arbitrarily in our implementation of \(DP_{K}\). So, it will be a good choice to let \(K\) be around \(K(n) \equiv \lceil(2 \log_{2} n) / 3\rceil\). The running time for \(DP_{K(n)}\) is \(O(n \log n)\).

We tested the \(DP_{K(1,000,000)}\) that is, \(DP_{14}\), using 20 independent 1,000,000-long lists and obtained an estimate \(\overline{R}_{DP_{14}} \approx 0.354\). Hence, we obtain the estimate \(\overline{R}_{DP_{1}} \approx 1.32\). This result is far away from the ideal ratio 1. This is what the \(O(n^{2/3})\) bound would foretell.

### 4. OTHER ALGORITHMS AND SIMULATION RESULTS

In this section, we introduce two more online algorithms, NF and MBF\([a]\). Unlike MXF, GLANF, and DP, which we introduced earlier, these algorithms do not possess good worst-case or provably good average-case performance ratios. However, NF is very simple and MBF\([a]\) empirically shows very promising average-case performances at suitable \(a\).

Also in this section, we present simulation results on the average-case performances of the various algorithms. Every time the average-case performance for any algorithm \(A\) is estimated empirically, we apply \(A\) to 20 independent random lists of 1,000,000 i.i.d. uniformly distributed pieces. These 1,000,000-long lists result in sample means less than 0.1%.

The next-fit (NF) algorithm: It is a natural online algorithm for OOBP which operates by keeping only one bin open and packing the pieces in the order they arrive. When the open bin is filled to level 1 or above, the bin will be closed and a new open bin will be started. The running time for NF is clearly \(O(n)\).

It is easy to obtain that \(R_{NF}^{\infty} = 2\). The argument for \(R_{NF}^{\infty} < 2\) is already stated in \$2\). To show \(R_{NF}^{\infty} \geq 2\), we construct a bad list of 4\(K\) pieces with \((1 - e)\)-pieces and 2\(e\)-pieces appearing alternately. NF will produce \(2\)\(K\) bins, while the optimal solution will only produce \(K + 1\) bins.

The analysis for the average-case performance of NF has already been conducted in the context of the bin-covering problem. Using the result of Csirik et al. (1991), we can easily obtain \(NF^{\infty} = 1/e \leq 0.368\). The proof idea is basically as follows. Let \(\{p_{i} \mid i = 1, 2, \ldots\}\) be an infinitely long list independently drawn from \(U[0,1]\). Let \(M\) denote the number of pieces contained in an arbitrary closed bin produced by NF when being applied to this list. For any \(k = 1, 2, \ldots\) and any \(y \in [0, 1]\), we have

\[
Pr[p_{1} + p_{2} + \cdots + p_{k} \leq y] = \frac{\gamma^{k}}{k!}.
\]

Hence,

\[
Pr[M = 0] = 0, \quad Pr[M = 1] = Pr[p_{1} = 1] = 0,
\]
and for $m = 2, 3, \ldots$,

$$
Pr[M = m] = \int_0^1 Pr[x \leq p_1 + \cdots + p_{m-1} \leq x + dx] Pr[p_m \geq 1 - x] = \int_0^1 \frac{x^{m-1}}{(m-2)!} dx = \frac{1}{m(m-2)!}.
$$

So,

$$
E[M] = \sum_{m=2}^{\infty} m \times \frac{1}{m(m-2)!} = \sum_{m=0}^{\infty} \frac{1}{m!} = e.
$$

Our simulation confirms that $\overline{NP} \approx 0.368$.

The modified best-fit (MBF) algorithm parameterized by $a$ where $0 < a < 1$: By its name, it is modified from the best-fit (BF) algorithm for CBP. Let the open bins at any moment be $1, 2, \ldots, k$, with their levels being $x_1, \ldots, x_k$. Assume without loss of generality that $x_1 \geq \cdots \geq x_k$. Let $y$ be the new piece to be considered. The destiny of $y$ is determined as follows:

1. If $x_1 + y \geq 1 + a$, put $y$ into bin 1 and close that bin;
2. If $1 \leq x_1 + y < 1 + a$ and $x_k + y \geq 1$, open a new bin and put $y$ into it;
3. If there is an $i$ between 1 and $k$ such that $1 + a > x_i + y \geq \cdots \geq x_{i-1} + y \geq 1 > x_i + y$, put $y$ into bin $i$.

If $a$ can be 0, MBF$[a]$ at that $a$ is just NF. If we define a piece $y$ being fit into a bin with level $x$ to be that $x + y \geq 1 + a$ or $x + y < 1$, MBF$[a]$ can be described as operating in exactly the same manner as BF for CBP. The only difference is that in the latter, piece $y$ being fit into a bin with level $x$ is that $x + y \leq 1$, and no bin will ever be closed. MBF$[a]$ runs in $O(n \log n)$ time.

We have only been able to show that $R_{MBF[a]}^\infty \geq 2$ for any $a$: For any positive $K$, let $L_K$ be a list consisting of $2K$ pieces with each being $(2 + a)/4$. Then, running MBF$[a]$ results in $2K$ bins with every bin containing one piece, while running the optimal offline algorithm results in $K$ bins with every bin containing two pieces.

Based on the simulation results, we may draw a plot of $MBF[a]^{\infty}$ vs. $a$. The plot is presented in Figure 1. From the

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MXF</th>
<th>NF</th>
<th>MBF[0.71]</th>
<th>DP$_{a4}$</th>
<th>GLANF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio $R^\infty$</td>
<td>1.22</td>
<td>1.37</td>
<td>1.01</td>
<td>1.32</td>
<td>1.04</td>
</tr>
</tbody>
</table>

**Figure 1.** Average-case performances of MBF$[a]$ vs. $a$.

5. **CONCLUSIONS**

We introduced the NP-hard OOBP and proposed for it five algorithms: MXF, NF, MBF$[a]$, DP, and GLANF. Our worst-case analyses showed that no online algorithm could do better than 1.630297 or 1.415715 in performance ratio, depending on whether or not the 1-pieces are allowed in the list. Of all the online algorithms, only MXF has a provable worst-case performance ratio of less than 2 even when there is no 1-piece. For the offline algorithm GLANF, we showed that its worst-case performance is very good when there is no 1-piece. In addition, we were able to show that $2 - \sqrt{3}$ is the optimal asymptotic average ratio of number of bins used over number of pieces, that DP achieves this performance in the asymptotic sense, and that this ratio is $1/e$ for NF. Our simulation results over the uniform distribution demonstrated that MBF$[a]$ at certain $a$s, and GLANF, possess performance ratios lower than 1.05, and they are much better than DP when the list length is in the millions.

For future research, it will be worthwhile to investigate the upper bounds for $R_{MXF}^\infty$ and $R_{GLANF}^\infty$ when the 1-pieces are present. Indeed, we conjecture that the bounds should be as tight as what we have achieved for when the 1-pieces are not present. Also, tighter bounds for the worst-case performances of arbitrary online algorithms, MXF, and GLANF in both cases, are certainly needed. Moreover, better online and offline algorithms are still needed. In terms of average-case analysis, just changing the distribution from $U[0, 1]$ to $U[0, b]$ from some $b \in (0, 1)$ will render most of our techniques ineffective. Therefore, more work is needed on the average-case analysis of algorithms for more general distributions.

**APPENDIX**

**Proof of Theorem 3.** Hoping that some of the ideas here might be used to tackle the case with the 1-pieces, we build up the framework of the proof with 1-pieces being considered as well. Let $\{L_n| n = 1, 2, \ldots \}$ be a series of lists. For any $n$, $L_n$ is just an arbitrary list with $OPT(L_n) = n$. Using the last type-3 or type-4 piece in $L_n$ as a dividing point, we call the type-1 and type-2 pieces that appear before this piece in the list the early pieces, and the type-1 and type-2 pieces that appear after this piece in the list the late pieces.
After applying MXF to the list, all type-1 and type-2 pieces in closed bins are early; some type-1 open bins have some early pieces and some have purely late pieces; and some type-2 open bins have two early pieces, some have two late pieces, and only \( O(1) \) type-2 open bins are of different kinds. We let:

- \( x^n_1 \) be the number of closed bins containing two type-3 pieces;
- \( x^n_2 \) be the number of closed bins containing one type-3 and one type-4 pieces;
- \( x^n_3 \) be the number of closed bins containing just one type-4 piece;
- \( x^n_4 \) be the number of closed bins containing a number of early type-1 pieces and one type-3 piece;
- \( x^n_5 \) be the number of closed bins containing two early type-2 pieces and one type-3 piece;
- \( x^n_6 \) be the number of closed bins containing a number of early type-1 pieces and one type-4 piece;
- \( x^n_7 \) be the number of closed bins containing two early type-2 pieces and one type-4 piece;
- \( x^n_8 \) be the number of type-1 open bins having some early pieces;
- \( x^n_9 \) be the number of type-1 open bins having purely late pieces;
- \( x^n_{10} \) be the number of type-2 open bins having two early pieces;
- and \( x^n_{11} \) be the number of type-2 open bins having two late pieces.

For \( i = 1, 2, \ldots, 11 \) and \( j = 1, 2 \), let \( a^{ij}_{E(L)} \) be the average total size of early (late) type-\( j \) pieces in the above \( i \)-th type of bins.

When we apply OPT to \( L_n \), there can be 28 major types of closed bins and \( O(1) \) other bins. We may describe each type using a 6-tuple \((n_{E1}, n_{E2}, n_3, n_{E4}, n_{L1}, n_{L2})\), where \( n_{E1} \), always taking the value \( A \), which means a number of and possibly none, describes the number of early type-1 pieces in this type of bins; \( n_{E4} \), taking the values of \( A \) and 0, describes the number of late type-1 pieces in this type of bins; and \( n_{E2}, n_3, n_{E4}, \) and \( n_{L2} \), taking integer values, describe the numbers of early type-2, type-3, type-4, and late type-2 pieces in this type of bins, respectively. The following list the 6-tuples of all the types:

**Type 1:** \((A, 0, 0, 0, 0, 0)\); **Type 2:** \((A, 1, 2, 0, 0, 0)\); **Type 3:** \((A, 0, 1, 1, 0, 0)\); **Type 4:** \((A, 1, 1, 1, 0, 0)\); **Type 5:** \((A, 0, 1, 0, 0, 0)\); **Type 6:** \((A, 1, 1, 0, 0, 0)\); **Type 7:** \((A, 2, 1, 0, 0, 0)\); **Type 8:** \((A, 0, 0, 1, 0, 0)\); **Type 9:** \((A, 0, 1, 1, 0, 0)\); **Type 10:** \((A, 2, 0, 1, 0, 0)\); **Type 11:** \((A, 0, 0, 0, 0, 0)\); **Type 12:** \((A, 1, 0, 1, 0, 0)\); **Type 13:** \((A, 2, 0, 0, 0, 0)\); **Type 14:** \((A, 3, 0, 0, 0, 0)\); **Type 15:** \((A, 0, 1, 0, A, 0)\); **Type 16:** \((A, 1, 1, 0, A, 0)\); **Type 17:** \((A, 0, 1, 0, A, 1)\); **Type 18:** \((A, 1, 1, 0, A, 1)\); **Type 19:** \((A, 0, 1, 0, A, 2)\); **Type 20:** \((A, 0, 0, 0, A, 0)\); **Type 21:** \((A, 1, 0, 0, A, 0)\); **Type 22:** \((A, 2, 0, 0, A, 0)\); **Type 23:** \((A, 0, 0, 0, A, 1)\); **Type 24:** \((A, 1, 0, 0, A, 1)\); **Type 25:** \((A, 2, 0, 0, A, 1)\); **Type 26:** \((A, 0, 0, 0, A, 2)\); **Type 27:** \((A, 1, 0, 0, A, 2)\); **Type 28:** \((A, 0, 0, 0, A, 3)\).

For \( i = 1, 2, \ldots, 28 \), let \( y^n_i \) be the number of the above \( i \)-th type of bins and for \( j = 1, 2 \), let \( b^{ij}_{E(L)} \) be the average total size of early (late) type-\( j \) pieces in the above \( i \)-th type of bins.

By the conservation of the number of early type-2 pieces, we have

\[
2x^n_1 + 2x^n_3 + 2x^n_{10} = y^n_1 + y^n_3 + y^n_{10} + 2y^n_2 + y^n_6 + 2y^n_{12} + 2y^n_{13} + 3y^n_{14} + y^n_{16} + y^n_{18} + y^n_{21} + 2y^n_{22} + y^n_{23} + 2y^n_{25} + y^n_{27} + O(1). \quad (A1)
\]

By the conservation of the number of type-3 pieces, we have

\[
2x^n_5 + x^n_2 + x^n_4 + x^n_5 = y^n_5 + y^n_2 + y^n_3 + y^n_5 + y^n_6 + y^n_9 + y^n_{15} + y^n_{17} + y^n_{19} + y^n_{23} + O(1). \quad (A2)
\]

By the conservation of the number of type-4 pieces, we have

\[
x^n_3 + x^n_4 + x^n_7 = y^n_4 + y^n_3 + y^n_6 + y^n_7 + y^n_{10} + O(1). \quad (A3)
\]

By the conservation of the number of late type-2 pieces, we have

\[
2x^n_{11} = y^n_{17} + y^n_{18} + 2y^n_{20} + 2y^n_{21} + 2y^n_{23} + 2y^n_{26} + 2y^n_{27} + 3y^n_{28} + O(1). \quad (A4)
\]

By the various conservations of piece sizes, we have

\[
\sum_{i=1}^{11} a^{11}_{E(L)} x^n_i = \sum_{i=1}^{28} b^{11}_{E(L)} y^n_i + O(1), \quad \sum_{i=1}^{11} (a^{11}_{E(L)} + a^{11}_{E(L)}) x^n_i = \sum_{i=1}^{28} (b^{11}_{E(L)} + b^{11}_{E(L)}) y^n_i + O(1), \quad \sum_{i=1}^{11} (a^{11}_{E(L)} + a^{11}_{L}) x^n_i = \sum_{i=1}^{28} (b^{11}_{E(L)} + b^{11}_{L}) y^n_i + O(1), \quad \sum_{i=1}^{11} (a^{11}_{E(L)} + a^{11}_{L}) x^n_i = \sum_{i=1}^{28} (b^{11}_{E(L)} + b^{11}_{L}) y^n_i + O(1), \quad \sum_{i=1}^{11} (a^{11}_{E(L)} + a^{11}_{E}) x^n_i = \sum_{i=1}^{28} (b^{11}_{E(L)} + b^{11}_{E}) y^n_i + O(1). \quad \sum_{i=1}^{11} (a^{11}_{E(L)} + a^{11}_{E}) x^n_i = \sum_{i=1}^{28} (b^{11}_{E(L)} + b^{11}_{E}) y^n_i + O(1). \quad (A5) \quad (A6) \quad (A7) \quad (A8) \quad (A9)
\]

From the description of the algorithm, we know \( a^{11}_{E(L)} \geq 2/3, a^{11}_{E(L)} \geq 2/3, a^{11}_{E(L)} \geq 2/3, a^{11}_{E(L)} \geq 0, a^{11}_{L} \geq 0, a^{11}_{E(L)} \geq 2/3, \) and \( a^{11}_{E(L)} \geq 2/3. \) From the well-known result of FF for CBP with piece sizes in \( (0, 1/3) \), we know that \( a^{11}_{E(L)} \geq 3x^n_i/4 \) \( + O(1) \) and \( a^{11}_{L} \geq 3x^n_i/4 + O(1) \) (Johnson et al. 1974). All other unmentioned \( a^{11}_{E(L)} \) are 0.

We can also easily obtain that

\[
b^{11}_{E(L)} < 1/2, \quad b^{11}_{E} = 0, \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E(L)} < 1/2, \quad b^{11}_{E(L)} < 1/6, \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E(L)} = 1/2, \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E(L)} = 0; \quad b^{11}_{E(L)} = 0, \quad b^{11}_{E} = 0, \quad b^{11}_{E(L)} = 0; \quad b^{11}_{E(L)} = 0,
\]
Combining these with Equations (A5)–(A9), we obtain

\[2x_i^u/3 + 2x_j^u/3 + 3x_k^u/4\]

\[\leq y_i^u/2 + y_j^u/2 + y_k^u/2 + y_r^u/2 + y_s^u + y_t^u + y_v^u + y_w^u + 3y^u + 4y_{10}^u + 3y_{12}^u/2 + 3y_{13}^u/2 + 2y_{14}^u/3 + y_{15}^u/2\]

\[+ y_{16}^u/2 + y_{17}^u/2 + y_{18}^u/2 + y_{19}^u/6 + y_{20}^u + y_{21}^u + y_{22}^u + y_{23}^u + y_{24}^u + 2y_{25}^u + 2y_{27}^u/3 + 2y_{28}^u/3 + O(1).\]

(11)

\[2x_i^u/3 + 2x_j^u/3 + 3x_k^u/4 + 3x_l^u/4 + 2x_m^u/3 + 2x_n^u/3 + 2x_o^u/3 \leq y_i^u/2 + y_j^u/2 + y_k^u/2 + y_r^u/2 + y_s^u + y_t^u + y_v^u + y_w^u + 3y^u + 4y_{10}^u + 3y_{12}^u/2 + 3y_{13}^u/2 + 2y_{14}^u/3 + 5y_{15}^u/6 + 5y_{16}^u/6 + y_{17}^u/2 + y_{18}^u/2 + 2y_{19}^u/6 + 4y_{20}^u/3 + 4y_{21}^u/3\]

\[+ 4y_{22}^u/3 + y_{23}^u + y_{24}^u + 2y_{25}^u/3 + 2y_{27}^u/3 + y_{28}^u/3 + O(1).\]

(12)

\[2x_i^u/3 + 2x_j^u/3 + 3x_k^u/4 + 3x_l^u/4 + 2x_m^u/3 + 2x_n^u/3 + 2x_o^u/3 \leq y_i^u/2 + y_j^u/2 + y_k^u/2 + y_r^u/2 + y_s^u + y_t^u + y_v^u + y_w^u + 3y^u + 4y_{10}^u + 3y_{12}^u/2 + 3y_{13}^u/2 + 2y_{14}^u/3 + 5y_{15}^u/6 + 5y_{16}^u/6 + y_{17}^u/2 + y_{18}^u/2 + 2y_{19}^u/6 + 4y_{20}^u/3 + 4y_{21}^u/3\]

\[+ 4y_{22}^u/3 + y_{23}^u + y_{24}^u + 2y_{25}^u/3 + 2y_{27}^u/3 + y_{28}^u/3 + O(1).\]

(13)

By the definition in the beginning, we also have

\[\sum_{i=1}^{28} y_i^* = n + O(1).\]

(15)

When the 1-pieces are not present, we further have

\[x_3^u + x_5^u + x_6^u + x_7^u + y_3^u + y_4^u + y_5^u + y_6^u + y_{10}^u = 0.\]

(16)

Hence,

\[\sum_{i=1}^{11} x_i^u\]

is upper-bounded by \(\{\sum_{i=1}^{11} x_i^u \mid x_i^u \geq 0, i = 1, \ldots, 11, y_i^u \geq 0, i = 1, \ldots, 28, (A1)–(A4), (A10)–(A11), (A13)–(A16)\}/n. The limit of this upper bound is another linear program of the apparent form. By the simplex method, we find its solution value to be 35/18. □

We believe \(R_{MAX}^\infty \leq 35/18\) even when there are 1-pieces in the list. However, without being enforced by (A16), the
linear program gives the result of $22/9 \approx 2.4444$ and a solution vector indicating that the valid constraints we have come up with so far have not captured the delicate timing of the appearances of the 1-pieces in both the MXF and OPT packings. Therefore, it calls for more subtle arguments than our existing ones to prove the result for the case with 1-pieces. Also, letting the underlying CBP algorithm be FF instead of NP is important because only the former allows the lower bound of $3/4$ relevant to $a_{i1}^{k+1}$ and $a_{i1}^{k-1}$. When FF is replaced by NP, we can only achieve a lower bound of 2 using similar arguments.

**Proof of Theorem 6.** The idea here is to have $20K$ pieces around 1/6, $20K$ pieces around 1/3, $20K$ pieces around 1/2, and $20K$ pieces of size $1 - \epsilon$. GLANF will result in $27K$ bins where there are 4K bins each containing five 1/6-pieces and one $(1 - \epsilon)$-piece, 10K bins each containing two 1/3-pieces and one $(1 - \epsilon)$-piece, 6K bins each containing one 1/2-piece and one $(1 - \epsilon)$-piece, and 7K bins each containing two 1/2-pieces. On the other hand, the optimal algorithm will result in $20K + 1$ bins where virtually every bin contains one 1/6-piece, one 1/3-piece, one 1/2-piece, and one $(1 - \epsilon)$-piece. Our choice of the fractional pieces following Johnson et al. (1974) guarantees these outcomes. Let $L_K$ be a list consisting of 80K pieces. For $k = 1, \ldots, 2K$,

$$p_{30k-29} = 1/6 + 33/18^{k+1}, \quad p_{30k-28} = 1/6 - 3/18^{k+1},$$

$$p_{30k-27} = p_{30k-26} = 1/6 - 7/18^{k+1},$$

$$p_{30k-25} = 1/6 - 13/18^{k+1}, p_{30k-24} = 1/6 + 9/18^{k+1},$$

$$p_{30k-23} = p_{30k-22} = p_{30k-21} = p_{30k-20} = 1/6 - 2/18^{k+1},$$

$$p_{30k-19} = 1/3 + 46/18^{k+1}, p_{30k-18} = 1/3 - 34/18^{k+1},$$

$$p_{30k-17} = p_{30k-16} = 1/3 + 6/18^{k+1},$$

$$p_{30k-15} = 1/3 + 12/18^{k+1}, p_{30k-14} = 1/3 - 10/18^{k+1},$$

$$p_{30k-13} = p_{30k-12} = p_{30k-11} = p_{30k-10} = 1/3 + 1/18^{k+1},$$

and for $i = 1, \ldots, 10$,

$$p_{30k-10+i} = 1/2 + 1/18^{11k}.$$ 

For $i = 60K + 1, \ldots, 80K$, $p_j = 1 - \epsilon$. Applying GLANF to $L_K$ results in 27K bins. For $k = 1, \ldots, 2K$, bin 10K - 9 contains $p_{30k-29}$, $p_{30k-28}$, $p_{30k-27}$, $p_{30k-26}$, $p_{30k-25}$, and $p_{60k+10k-9}$; bin 10K - 8 contains $p_{30k-24}$, $p_{30k-23}$, $p_{30k-22}$, $p_{30k-21}$, $p_{30k-20}$, $p_{60k+10k-8}$; for $i = 1, \ldots, 5$, bin 10K - 8 + $i$ contains $p_{30k-21+i}$, $p_{30k-20+i}$, and $p_{60k+10k-8+i}$; and for $i = 1, 2, 3$, bin 10K - 3 + $i$ contains $p_{30k+10+i}$, and $p_{60k+10k-3+i}$. And, for $k = 1, \ldots, 7K$, bin 20K + $k$ contains $p_{30k+3}$ and $p_{30k+4}$. On the other hand, the optimal algorithm results in only 20K + 1 bins. In this solution, for $k = 1, \ldots, 2K$, bin 10K - 9 contains $p_{30k-29}$, $p_{30k-18}$, $p_{30k-9}$, and $p_{60k+10k-9}$; for $k = 1, \ldots, 2K$ and $i = 2, \ldots, 9$, bin 10K - 10 + $i$ contains $p_{30k-30+i}$, $p_{30k-20+i}$, $p_{30k-10+i}$, and $p_{60k+10k-10+i}$; for $k = 1, \ldots, 2K - 1$, bin 10K contains $p_{30k-28}$, $p_{30k+11}$, $p_{30k+12}$, and $p_{60k+10k}$; bin 20K contains $p_{11}, p_{60k-28}$, and $p_{60k}$; and bin 20K + 1 contains $p_{60k}$. Therefore, $R_{GLANF}^7 \geq 27/20$. □

**Acknowledgments**

The authors thank Area Editor Rolf Moehring, the associate editor, and two anonymous referees for their comments, which greatly improve the paper. They also acknowledge two referees for pointing out directions for further investigation and errors in early versions of the paper. The first author is supported, in part, by New Jersey Institute of Technology under Grant No. 4-21830 and the National Center for Transportation and Industrial Productivity under Grant No. 9-92518.

**References**


