Lower Bounds for Online Bin Stretching with Several Bins

Martin Böhm

Computer Science Institute of Charles University,
Faculty of Mathematics and Physics, Malostranské nám. 25,
118 00 Prague 1, Czech Republic
bohm@iuuk.mff.cuni.cz.

Abstract. Online Bin Stretching is a semi-online variant of Bin Packing with a set number of m bins, where all bins can be overpacked to capacity $S \geq 1$, which is to be minimized. There is also a guarantee that an offline algorithm can pack the input to m bins of unit size.
We focus on the problem of Online Bin Stretching for small m, namely $3 \leq m \leq 5$. Recent progress on this problem has led into a lower bound of $19/14 \approx 1.357$ for $m = 3$ and an upper bound of $11/8 = 1.375$ for the same. For $m = 4$ and $m = 5$, only a trivial lower bound of $4/3$ was known.
We improve the techniques used in the previous lower bound for $m = 3$ to reach a lower bound of $45/33 = 1.\overline{36}$ for $m = 3$ and a new lower bound of $19/14 \approx 1.357$ for $m = 4$ and $m = 5$.

1 Introduction

Online Bin Stretching is a semi-online generalization of Online Bin Packing, also related to semi-online scheduling. It has been introduced by Azar and Regev in 1998 ([2], journal version [3]).

As in Online Bin Packing, items of size between 0 and 1 arrive in a sequence, and the algorithm needs to pack them as soon as each item arrives. In Online Bin Stretching, there are three main differences:

1. The number of bins m is fixed at the start of the input.
2. There is a guarantee that an offline algorithm is able to pack any input into m bins of capacity 1.
3. The online algorithm for Online Bin Stretching can use bins of capacity S for some $S \geq 1$. The goal is to minimize the parameter S, known as the *stretching factor*.

Even though Online Bin Stretching is described using bin packing terminology, it is clear from the description that it is equivalent to the problem of semi-online makespan minimization on m identical machines with a guarantee that there exists a schedule with unit makespan. Therefore, Online Bin Stretching is related to other semi-online scheduling problems, such as online makespan minimization with a known sum of processing times [4].

Previous work. Online Bin Stretching has been proposed by Azar and Regev [2]. The original lower bound of 4/3 for three bins has appeared even before that, in [1]. Azar and Regev extended the same lower bound to any number of bins and gave an online algorithm with a stretching factor 1.625 for any number of bins.

The problem has been revisited recently, with algorithmic results from Kellerer and Kotov [5], Gabay, Kotov and Brauner [6], and most recently Böhm, Sgall, van Stee, Veselý [9], with the best algorithm [9] having a stretching factor 1.5 for any number of bins. The only known lower bound for the general case is still a simple bound of 4/3.

We focus on the setting where the number of bins m is fixed and small. For $m = 2$, a lower and upper bound of 4/3 is easy and well-known.

The case $m = 3$ is the smallest case where the optimal stretching factor is not yet known. The best current online algorithm is from [9] with stretching factor $11/8 = 1.375$. Furthermore, a recent paper of Gabay, Brauner and Kotov [7] shows an improved lower bound of $19/14 \approx 1.357$, a first improvement above the classical bound of 4/3.

The lower bound of [7] was found using computer search, namely a minimax algorithm implemented in Python. In [7], the input space is discretised by considering only inputs consisting of multiples of $1/K$ for a fixed integer K. For instance, in the lower bound of 19/14, all items have size $L/14$ for some $L \in \{1, 2, \ldots, 14\}$. The authors of [7] were able to check inputs where the maximum denominator was at most 20.

For the case $m = 4$ and $m = 5$, the best online algorithm is from the original paper by Azar and Regev [3], achieving the stretching factor $\frac{5m-1}{3m+1}$, which is approximately 1.461 for $m = 4$ and exactly 1.5 for $m = 5$. Before our work, the best lower bound for $m = 4$ and $m = 5$ was the previously mentioned 4/3.

Very recently and independently on our work, the authors of [7] have updated their preprinted paper from 2013 with a new version [8], showing a lower bound of 19/14 for $m = 4$. This lower bound is presented in our paper as well.

Our contributions. We build on the paper of Gabay, Brauner and Kotov [7] but significantly change the algorithm, both conceptually and technically.

On the conceptual side, we propose a different algorithm for computing the offline optimum packing, suggest new ways of pruning the game tree and show how the alpha-beta pruning of [7] can be skipped entirely.

On the technical side, we reimplement the algorithm of [7], gaining significant speedup from the reimplementation alone. While the lower bound search program of [7] was written in Python, employed CSP solvers and had unrestricted caching, our program is written in C, is purely combinatorial and it sets limits on the cache size, making time the only exponentially-increasing factor.

With these improvements, we were able to find an improved lower bound for Online Bin Stretching for three bins, namely $45/33 = 1.\overline{36}$.

We also present the first non-trivial lower bound of $19/14 \approx 1.357$ for $m = 4$ and $m = 5$. (A lower bound for $m = 4$ is also independently presented in the recently updated preprint [8].)

To see the strength of our improvements, consider the granularity of the items defined above as K. It is easy to see that a general game tree search is exponential in the running time with respect to K. The algorithm of [7] is able to check all $K \leq 20$ before claiming that "even with many efficient cuts, we cannot tackle much larger problems."

In contrast, our proposed algorithm is able to check all $K \leq 41$ and is fast enough to produce results for $m = 4$ and $m = 5$.

The source code for the computer search as well as the complete results can be found at http://iuuk.mff.cuni.cz/ bohm/p/bs/lowerbound-sofsem.zip.

Definitions and scaling. We now present a formal definition of the Online Bin Stretching problem on three bins. Throughout the text, we use the standard bin packing notions of *size*, *load* and *capacity*. Each item i has an associated size $s(i) \in \mathbb{R}$, usually integral or fractional. A load of a bin is the sum of the items currently packed into it. A capacity of a bin is the maximum load that it allows.

To make the problem easier for computer and human alike, we scale the bins so that the optimal bins have capacity $T \in \mathbb{N}$ (instead of unit capacity) and the stretched bins have capacity $S \in \mathbb{N}$. The resulting stretching factor is then S/T.

Definition 1. *The problem* Online Bin Stretching *on three bins is defined as follows:*

Parameters: *The limit of stretched bins S and the limit of the offline optimum bins T.*

Input: *A sequence of items $I = i_1, i_2, \ldots$ given online one by one. Each item has a size $s(i) \in (0, T]$ and must be packed immediately and irrevocably.*

Output: *Partitioning (packing) of I into the three bins B_1, B_2, B_3 so that for all three bins it holds that $\sum_{i \in B_j} s(i) \leq S$.*

Guarantee: *there exists a packing of all items in I into 3 bins of capacity T.*

Goal: *Design an online algorithm with the stretching factor S/T as small as possible which packs all input sequences satisfying the guarantee.*

2 Lower Bound Technique

In Section 2 and 3, we describe our lower bound technique. To simplify our arguments, we describe the technique only for $m = 3$. We discuss the pecularities of the generalization to any fixed m in Section 5.

As with many other online algorithms, we can think of Online Bin Stretching as a two player game. The first player (Algorithm) is presented with an item i. Algorithm's goal is to pack it into m bins of capacity S. This mimics the task of any algorithm for Online Bin Stretching. The other player (Adversary) decides which item to present to the Algorithm in the next step. The goal of the Adversary is to force Algorithm to overpack at least one bin.

It is clear that knowing the game tree for a parameter S of the aforementioned game is equal to knowing whether there is an algorithm for Online Bin Stretching with stretching factor S.

We are interested primarily in the lower bound. Therefore, it makes sense to slightly reformulate the previous game:

- The player Algorithm wins if it can pack all items into bins with capacity strictly less than S.
- The player Adversary wins if it can force Algorithm to pack a bin with load $\geq S$ while making sure that the Online Bin Stretching guarantee is satisfied.

This way, a victory for the player Adversary immediately implies that no online algorithm for Online Bin Stretching with stretching factor less than S exists.

The two main obstacles to implementing a search of the described two player game are the following:

1. Adversary can send an item of arbitrary small size;
2. Adversary needs to make sure that at any time of the game, an offline optimum can pack the items arrived so far into three bins of size T.

To overcome the first problem, it makes sense to create a sequence of games based on the granularity of the items that can be packed. A natural granularity for the scaled game are integral items, which correspond to multiples of $1/T$ in the non-scaled problem.

The second problem increases the complexity of every game turn of the Adversary, as it needs to run a subroutine to verify the guarantee for the next item it wishes to place.

Note that the ideas described above have been described previously in [7].

To precisely formulate our setting, we first define one state of a game:

Definition 2. *For given parameters $S \in \mathbb{N}, T \in \mathbb{N}$, a **bin configuration** is a tuple $(a, b, c, \mathcal{I})$, where*

- *$a, b, c \in \{0, 1, \ldots, S\}$ denote the current sorted loads of the bins,*
- *$\mathcal{I}$ is a multiset with ground set $\{1, 2, \ldots, T\}$ which lists the items used in the bins.*

Additionally, in a bin configuration, it must hold:

- *that there exists a packing of items from $\mathcal{I}$ into three bins with loads exactly a, b, c,*
- *that there exists a packing of items from $\mathcal{I}$ into three bins that does not exceed T in any bin.*

It is clear that every bin configuration is a valid state of the game with Adversary as the next player. We may also observe that the existence of an online algorithm for Online Bin Stretching implies an existence of an oblivious algorithm wit the same stretching factor that has access only to the current bin configuration B and the incoming item i.

Using the concept of bin configuration and the previous two facts, we may formally define the game we investigate:

Definition 3. *For a given $S \in \mathbb{N}, T \in \mathbb{N}$, the **bin stretching game** $\mathrm{BSG}(S,T)$ is the following two player game:*

1. *There are two players named* Adversary *and* Algorithm. *The player* Adversary *starts.*
2. *Each turn of the player* Adversary *is associated with a bin configuration $B = (a, b, c, \mathcal{I})$. The start of the game is associated with the bin configuration $(0, 0, 0, \emptyset)$.*
3. *The player* Adversary *receives a bin configuration B. Then,* Adversary *selects a number i such that the multiset $\mathcal{I} \cup \{i\}$ can be packed by an offline optimum into three bins of capacity T. The pair (B, i) is then sent to the player* Algorithm.
4. *The player* Algorithm *receives a pair (B, i). The player* Algorithm *has to pack the item i into the three bins as described in B so that each bin has load strictly less than S.* Algorithm *then updates the configuration B into a new bin configuration, denoted B'.* Algorithm *then sends B' to the player* Adversary.
5. *If the player* Algorithm *receives a pair (B, i) such that it cannot pack the item according to the rules, the bin configuration B is won for player* Adversary.
6. *If the player* Adversary *has no more items i that it can send from a configuration B, the bin configuration B is lost for player* Adversary.
7. *For any bin configuration B where the player* Adversary *has a possible move, the configuration is won for player* Adversary *if and only if the game ends in a bin configuration C that is won for the player* Adversary *no matter which decision is made by the player* Algorithm *at any point.*

Definition 4. *We say that a game $\mathrm{BSG}(S,T)$ is a **lower bound** if and only if the bin configuration $(0, 0, 0, \emptyset)$ is won for the player* Adversary.

3 The Minimax Algorithm

Our implemented algorithm is a fairly standard implementation of the minimax game search algorithm. The pecularities of our algorithm (caching, pruning, and other details) are described in the following sections.

One of the differences between our algorithm and the algorithm of Gabay et al. [7] is that our algorithm makes no use of alpha-beta pruning – indeed, as every bin configuration is either won for Algorithm or won for Adversary, there is no need to use this type of pruning.

Procedure EvaluateAdversary:
Input is a bin configuration $B = (a, b, c, \mathcal{I})$.
(1) Check if the bin configuration is cached (Section 3.2); if so, output the value found in cache and return.
(2) Create a list L of items which can be sent as the next step of the player Adversary (Section 3.1).
(3) For every item size i in the list L:
(4) Recurse by running EvaluateAlgorithm(B, i).
(5) If EvaluateAlgorithm(B, i) returns 0 (the configuration is won for player Adversary), stop the cycle and end EvaluateAdversary with value 0.
(6) Otherwise, continue with the next item size.
(7) If the evaluation reaches this step, store the configuration in the cache and return value 1 (player Algorithm wins).

Procedure EvaluateAlgorithm:
Input is a bin configuration $B = (a, b, c, \mathcal{I})$ and item i.
(1) If applicable, prune the tree using known algorithms (Section 3.3).
(2) For any one of the three bins:
(3) If i can be packed into the bin so that its load is less than T:
(4) Create a configuration B' that corresponds to this packing.
(5) Run EvaluateAdversary(B'). if EvaluateAdversary(B') returns 1, exit the procedure with value 1 as well.
(6) Otherwise, continue with another bin.
(7) If we reach this step, no placement of i results in victory of Algorithm. We return 0 and exit.

Procedure Main:
Input is a bin configuration $B = (a, b, c, \mathcal{I})$.
(1) Fix parameters S, T.
(2) Run EvaluateAdversary(B).
(3) If EvaluateAdversary(B) returns 1 (the game is won for player Algorithm), report failure.
(4) Otherwise:
(5) Print the tree currently in memory, erase the bin configuration cache.
(6) For any cached bin configuration C that is linked in the tree, run Main(C) with a blank cache.
(7) Output the game tree.

3.1 Verifying the Offline Optimum Guarantee

When we evaluate a turn of the Adversary, we need to create the list $L = \{0, 1, \ldots, y\} \subseteq \{0, 1, \ldots, T\}$ of items that Adversary can actually send while satifying the Online Bin Stretching guarantee. We employ the following steps:

1. First, we calculate a lower and upper bound $LB \leq UB$ of the maximal value y of L.
2. Then, we do a linear search on the interval $\{UB, UB-1, \ldots, LB\}$ using a procedure TEST that checks a single multiset $\mathcal{I}'$, where $\mathcal{I}'$ is $\mathcal{I}$ plus the item in question.
3. The first feasible item size is the desired value of y.

Upper and lower bounds. The running time of procedure TEST will be cubic in terms of T in the worst case. We therefore reduce the number of calls to TEST by creating good lower and upper bounds on the maximal item y which ADVERSARY can send.

To find a good lower bound, we employ a standard bin packing algorithm called BEST FIT DECREASING. BEST FIT DECREASING packs items from $\mathcal{I}$ into three bins of capacity T with items in decreasing order, packing an item into a bin where it "fits best" – where it minimizes the empty space of a bin. BEST FIT DECREASING is a linear-time algorithm (it does not need to sort items in $\mathcal{I}$, as the implementation of $\mathcal{I}$ stores them in sorted order).

Our desired lower bound LB will be the maximum empty space over all three bins, after BEST FIT DECREASING has ended packing. Such an item can always be sent without invalidating the ONLINE BIN STRETCHING guarantee.

Our upper bound UB is comparatively simpler; for a bin configuration $(a, b, c, \mathcal{I})$, it will be set to $\min(T, 3T - a - b - c)$. Clearly, no larger item can be sent without raising the total size of all items above $3T$.

Procedure TEST. Procedure TEST is a sparse modification of the standard dynamic programming algorithm for KNAPSACK. Given a multiset $\mathcal{I}, |\mathcal{I}| = n$ on input, our task is to check whether it can be packed into three bins (knapsacks) of capacity T each.

We use a queue-based algorithm that generates a queue Q_i of all valid triples (a, b, c) that can arise by packing the first i items.

To generate a queue Q_{i+1}, we traverse the old queue Q_i and add the new item $\mathcal{I}[i+1]$ to the first, second and third bin, creating up to three triples that need to be added to Q_{i+1}.

We make sure that we do not add a triple several times during one step, we mark its addition into a auxilliary $\{0, 1\}$ array F. Note that the queue Q_{i+1} needs only Q_i and the item $\mathcal{I}[i]$ for its construction, and so we can save space by switching between queues Q_1 and Q_2, where $Q_{2i+1} = Q_1$ and $Q_{2i} = Q_2$.

The time complexity of the procedure TEST is $\mathcal{O}(|\mathcal{I}| \cdot T^3)$ in the worst case. However, when a bin configuration contains large items, the size of the queue is substantially limited and the actual running time is much better.

Procedure TEST:
Input is a multiset of items $\mathcal{I}$.
(1) Create two queues Q_1, Q_2.
(2) Add the triple $(\mathcal{I}[1], 0, 0)$ to Q_1.
(3) For each item i in the multiset $\mathcal{I}$, starting with the second item:
(4) For each triple $(a, b, c) \in Q_1$:
(5) Add the triple $(a + s(i), b, c)$ to Q_2 unless $F[a + s(i), b, c] = 1$.
(6) Set $F[a + s(i), b, c] = 1$.
(7) Do the same for triples $(a, b + s(i), c)$ and $(a, b, c + s(i))$.
(8) Swap the queues Q_1 and Q_2.
(9) Return True if the queue Q_1 is non-empty, False otherwise.

Notes: We employ two small optimizations that were not yet mentioned. First, we sort the numbers (a, b, c) in each triple to ensure $a \geq b \geq c$, saving a small amount of space and time. Second, we use one global array F in order to avoid initializing it with every call of the procedure TEST.

It is also worth noting that we could alternatively implement the procedure TEST using integer linear programming or using a CSP solver (which has been done in [7]). However, we believe our sparse dynamic programming solution carries little overhead and for large instances it is much faster than the CSP/ILP solvers.

3.2 Caching

Our minimax algorithm employs extensive use of caching. We cache any solved instance of procedure TEST as well as any evaluated bin configuration B with its value.

Hash table limitation. We store a large hash table of fixed size, with each entry being a separate chain. With each entry we store the number of accesses. When a chain is to be filled over a fixed limit, we eliminate an entry with the least number of accesses.

To allow hash tables of variable size, our hash function returns a 64-bit number, which we trim to the desired size of our hash table.

In our definition of a bin configuration $(a, b, c, \mathcal{I})$, we do not require the loads a, b, c to be sorted. However, configurations which differ only by a permutation of the values a, b, c are equivalent, and so we sort these numbers when inserting a bin configuration into the hash table.

Hash function. Our hash function is based on Zobrist hashing [10], which we now describe.

For each bin configuration, we count occurences of items, creating pairs $(i, f) \in \{1, \ldots, T\} \times \{0, 1 \ldots, 3T\}$, where i is the item type and f its frequency. As an example, a bin configuration $(3, 2, 3, \{1, 1, 1, 1, 2, 3\})$ forms pairs $(1, 4), (2, 1), (3, 1), (4, 0), (5, 0)$ and so on.

At the start of our program, we associate a 64-bit number with each pair (i, f). We also associate a 64-bit number for each possible load of bin A, bin B and bin C.

The Zobrist hash function is then simply a XOR of all associated numbers for a particular bin configuration.

The main advantage of this approach is fast computation of new hash values. Suppose that we have a bin configuration B with hash H. After one round of the player ADVERSARY and one round of the player ALGORITHM, a new bin configuration B' is formed, with one new item placed. Calculating the hash H' of B' can be done in time $\mathcal{O}(1)$, provided we remember the hash H – the new hash is calculated by applying XOR to H, the new associated values, and the previous associated values which have changed.

Caching of the procedure TEST. So far, we have described cashing of the bin configurations. We also use the same approach for caching the values of the procedure TEST. To see the usefulness, note that the procedure TEST does not use the entire bin configuration $B = (a, b, c, \mathcal{I})$ as input, but only the multiset $\mathcal{I}$. Therefore, we aim to eliminate overhead that is caused by calling TEST on a different bin configuration, but with the same multiset $\mathcal{I}$.

Our hash function and hash table approaches are the same in both cases.

3.3 Tree Pruning

Alongside the extensive caching described in Subsection 3.2, we also prune some bin configurations where it is possible to prove that a simple online algorithm is able to finalize the packing. Such a bin configuration is then clearly won for player ALGORITHM, as it can follow the output of the online algorithm.

Such situations are called *good situations* and have been introduced in the paper of Böhm, Sgall, van Stee and Veselý [9].

Recall that in our game $\mathrm{BSG}(S, T)$, the player ALGORITHM is trying to pack all three bins with capacity $S - 1$. Therefore, we define $S' = S - 1$ and use S' in our definitions.

We restate the good situations for an instance of $\mathrm{BSG}(S', T)$ for general S', T with $\alpha = S' - T$ satisfying $\alpha \geq T/3$, while the authors of [9] formulate the good situations only for $\mathrm{BSG}(22, 18)$. The proofs are however equivalent and we omit them.

Good Situation 1. *[9] Given a bin configuration $(a, b, c, \mathcal{I})$ such that $a + b \geq 2T - \alpha$ and c is arbitrary, there exists an online algorithm that packs all remaining items into three bins of capacity S'.* □

Good Situation 2. *[9] Given a bin configuration $(a, b, c, \mathcal{I})$ such that $a \in [T - 2\alpha, \alpha]$ and b and c are arbitrary, there exists an online algorithm that packs all remaining items into three bins of capacity S'.* □

Good Situation 3. *[9] Given a bin configuration $(a, b, c, \mathcal{I})$ such that $a \in [\frac{3}{2}(T - \alpha), S']$ and either* (i) *$c \leq \alpha$ and b is arbitrary or* (ii) *$b + c \geq S'$, there exists an online algorithm that packs all remaining items into three bins of capacity S'.* □

Good Situation 4. *[9] Given a bin configuration $(a, b, c, \mathcal{I})$ such that $a + b \geq \frac{3}{2}(T - \alpha) + c/2$, $b < T - 2\alpha$, and $c < T - 2\alpha$, there exists an online algorithm that packs all remaining items into three bins of capacity S'.* □

Good Situation 5. *[9] Suppose that we are given a bin configuration $(a, b, c, \mathcal{I})$ such that an item i with $s(i) > \alpha$ is present in the multiset $\mathcal{I}$ and the following holds: $a \geq s(i), b \geq (3T - 7\alpha)/2, b \leq \alpha, c = 0$. Then there exists an algorithm that packs all remaining items into three bins of capacity S'.* □

4 Results

Table 1 summarizes our results. The paper of Gabay, Brauner and Kotov [7] contains results up to the denominator 20; we include them in the table for completeness. Results after the denominator 20 are new. Note that there may be a lower bound of size 56/41 even though none was found with this denominator; for instance, some lower bound may reach 56/41 using item sizes that are not multiples of 1/41.

Table 1. Results produced by our minimax algorithm, along with elapsed time. The column *L. b. found* indicates whether a lower bound was found when starting with the given granularity. Fractions lower than 19/14 and higher than 11/8 are omitted. Results were computed on a server with an AMD Opteron 6134 CPU and 64496 MB RAM. The size of the hash table was set to 2^{25} with chain length 4. Output of the program was disabled during the computation

Target fraction	*Decimal form*	*L. b. found*	*Elapsed time*
19/14	1.3571	Yes	2s.
22/16	1.375	No	2s.
26/19	1.3684	No	3s.
30/22	$1.\overline{36}$	No	6s.
33/24	1.375	No	5s.
34/25	1.36	**Yes**	15s.
37/27	$1.\overline{370}$	No	10s.
41/30	$1.3\overline{6}$	No	32s.
44/32	1.375	No	34s.
45/33	$1.\overline{36}$	**Yes**	1min. 48s.
48/35	1.3714	No	2min. 8s.
52/38	1.3684	No	6min. 14s.
55/40	1.375	No	3min. 6s.
56/41	1.3659	No	30min.

5 Lower Bound for Four and Five Bins

The notion of bin configuration (Definition 2) as well as most of the minimax algorithm can be straightforwardly generalized for $m > 3$. When generalizing

the algorithm for larger m, one must expect a slowdown, as the complexity of the sparse dynamic programming from Section 3.1 is now $\mathcal{O}(|\mathcal{I}| \cdot T^m)$.

One notion that does not generalize very well are the good situations of Section 3.3. For instance, the formula $a + b \geq mT - \alpha$ in the statement of Good Situation 1 will be much less useful as m grows. Some good situations, like Good Situation 2, have no clear generalization for growing m.

Therefore, we disable the pruning using good situations whenever computing a lower bound for $m > 3$.

Despite a significant increase in time complexity, we were able to produce results for $m = 4$ and $m = 5$. See Table 2 for our results on four and five bins.

Table 2. Results produced by our minimax algorithm in the case of 4 and 5 bins. Tested on the same machine and with the same parameters as in Table 1 Output of the program was disabled during the computation

Number of bins	*Target fraction*	*Decimal form*	*L. b. found*	*Elapsed time*
4 bins	19/14	1.3571	**Yes**	18s.
5 bins	19/14	1.3571	**Yes**	25min.

6 Verification of the Results

We include the lower bound along with the implementations, publishing it online at http://iuuk.mff.cuni.cz/ bohm/p/bs/lowerbound-sofsem.zip.

We have implemented a simple independent C++ program which verifies that a given game tree is valid and accurate. While verifying our lower bound manually may be laborious, verifying the correctness of the C++ program should be manageable. The verifier is available along with the rest of the programs and data.

7 Conclusion

We have computed new lower bounds for the problem Online Bin Stretching on 3,4 and 5 bins. A natural next step is to produce a better online algorithm for the case $m = 3$ and tighten the lower bound/upper bound gap. Still, we are not particularly convinced that 45/33 is the right lower bound for this problem.

We also hope that insight generated from our lower bounds for Online Bin Stretching on a small number of bins may yield a new lower bound for Online Bin Stretching on any number of bins, a long-standing open problem in the area.

We thank Jiří Sgall, Rob van Stee and Pavel Veselý for their suggestions, comments and proofreading.

References

1. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Y.: Semi on-line algorithms for the partition problem. Operations Research Letters, vol. 21, pp. 235–242 (1997)
2. Azar, Y., Regev, O.: On-line bin-stretching. In: Randomization and Approximation Techniques in Computer Science, pp. 71–81, Springer (1998)
3. Azar, Y., Regev, O.: On-line bin-stretching. Theoretical Computer Science, vol. 268(1), pp. 17–41 (2001)
4. Angelelli, E., Nagy, A.B., Speranza, M.G., Tuza, Z.: The on-line multiprocessor scheduling problem with known sum of the tasks. Journal of Scheduling, vol. 7(6), pp. 421-428, (2004)
5. Kellerer, H., Kotov, V.: An efficient algorithm for bin stretching. Operations Research Letters, vol. 41(4), pp. 343–346 (2013)
6. Gabay, M., Kotov, V., Brauner, N.: Semi-online bin stretching with bunch techniques. HAL preprint hal-00869858 (2013)
7. Gabay, M., Brauner, N., Kotov, V.: Computing lower bounds for semi-online optimization problems: Application to the bin stretching problem. HAL preprint hal-00921663 version 2 (2013)
8. Gabay, M., Brauner, N., Kotov, V.: Computing lower bounds for semi-online optimization problems: Application to the bin stretching problem. HAL preprint hal-00921663 version 3 (2015)
9. Böhm, M., Sgall, J., van Stee, R., Veselý, P.: Better Algorithms for Online Bin Stretching. Approximation and Online Algorithms, 12th International Workshop, WAOA 2014, Lecture Notes in Computer Science, Springer (2015)
10. Zobrist, A.L.: A new hashing method with application for game playing. ICCA journal, vol. 13(2), pp. 69–73 (1970)