Better Algorithms for Online Bin Stretching via Computer Search

Matej Lieskovský(✉)

Charles University, Ovocný Trh 560/5, Praha 1 116 36, Czechia
ml@iuuk.mff.cuni.cz

Abstract. Online Bin Stretching is a problem closely related to Online Bin Packing and various scheduling problems. There is extensive history of computer search being used to establish lower bounds for this problem by identifying difficult sets of inputs. We demonstrate a novel approach enabling the use of computer search for finding new algorithms and therefore upper bounds for this problem. This not only leads to improved results for Online Bin Stretching, but also shows that computer search can be used to find new algorithms, even for problems that might not appear suitable for this approach.

Keywords: Bin stretching · Multiprocessor scheduling · Bin packing · Online algorithms · Computer search

1 Introduction

Online Bin Stretching, introduced by Azar and Regev [1], is a problem somewhat similar to Online Bin Packing. We are given the number of bins m and then a sequence of items with sizes between 0 and 1 arrives. Each item must be assigned to one of the m bins before processing the next item. The goal is to minimize the total load packed into the largest bin. Importantly, we are assured that the entire input does fit into m bins of size at most 1. We measure the performance of an algorithm by the worst-case load of the largest bin which is known as the stretching factor.

Example: Let us consider the case $m = 2$. The first item that arrives can be packed into the first bin without loss of generality. Suppose that first item is of size $1/3$ and the second item is also of size $1/3$. If we pack the second item into the first bin, two items of size $2/3$ may then arrive, forcing the total load of the largest bin to be at least $4/3$. If we pack the second item into the second bin, an item of size 1 may then arrive, forcing the total load of the largest bin to be at least $4/3$.

In the language of scheduling problems, Online Bin Stretching is a variant of $Pm||C_{max}$ where jobs must be assigned in an online manner but there are

Partially supported by GAUK project 234723, and GA ČR project 19-27871X.

no release times and we know the optimal makespan of the instance in advance. We will, however, stick to the original terminology of bins and items.

Since lower bounds for these problems typically consist of sets of inputs utilizing a finite number of item sizes, computer search for Online Bin Stretching lower bounds has been successful via constraining the adversary to a finite set of possible item sizes. This approach cannot be directly used for computing upper bounds as the resulting algorithm must be able to pack items of arbitrary size, which prevents both an upper bound on the number of distinct item sizes and on the number of items. We demonstrate a novel approach that where we instead limit the decision abilities of the potential algorithms and thus enable computer search for Online Bin Stretching upper bounds.

1.1 Previous Results

See Fig. 1 for a graphical representation of the best known results.

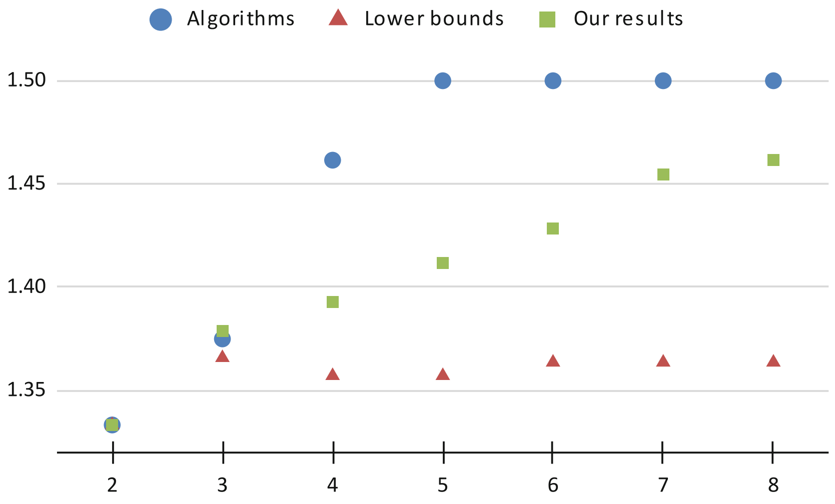

Fig. 1. Best known results

Let us denote the optimal stretching factor for m bins as α_m. Trivially, $\alpha_1 = 1$. Kellerer et al. [7] showed that $\alpha_2 = 4/3 < 1.334$ by demonstrating an algorithm using two bins of size 4/3 and a set of inputs such that any deterministic algorithm packs at least 4/3 load into one of the bins for one of the inputs. (See the example earlier.) They also proved that a similar set of inputs exists for three bins and thus $\alpha_3 \geq 4/3 > 1.333$. Azar and Regev [1] generalised the lower bound further, proving that $\alpha_m \geq 4/3 > 1.333$ for all $m \geq 2$. They also presented an algorithm proving $\alpha_m \leq \frac{5m-1}{3m+1}$ for $m \in \{3, 4, \ldots, 21\}$ and $\alpha_m \leq 13/8 = 1.625$ for all m.

The upper bounds have since been improved by multiple algorithms. Kellerer and Kotov [8] proved $\alpha_m \leq 11/7 < 1.572$ for all m, Gabay et al. [5] improved that to $\alpha_m \leq 26/17 < 1.530$ for all m, and the latest results are by Böhm et al. [2] achieving $\alpha_3 \leq 11/8 = 1.375$ and $\alpha_m = 3/2 = 1.5$ for all $m \geq 5$.

Although no better lower bound for general m was found, there are results for small values of m. Gabay et al. [4] introduced the idea of using computer search to find new lower bounds, leveraging the fact that online problems can be viewed as two-player games. This approach was then improved by Böhm and Simon [3] who proved $\alpha_3 \geq 56/41 > 1.365$ and $\alpha_m \geq 19/14 > 1.357$ for $m \in \{4, 5, 6, 7, 8\}$. Ongoing efforts to further improve these results have so far yielded $\alpha_m \geq 15/11 > 1.363$ for $m \in \{6, 7, 8\}$ [9] with more results known to be in preparation.

So far, computer search approaches for lower bounds constrain the possible item sizes to the equally-spaced values $1/k, 2/k, \ldots, 1$ for some granularity k, and then view the problem as a two player game where one player generates items and the other assigns them to the m bins. This game has a finite branching factor $\max(m, k)$ and length $O(mk)$. Unfortunately, placing additional constraints on the item-generating adversary is not permitted in the search for new algorithms.

1.2 Our Results

We modify the computer search approach so that it can be used for finding new algorithms for small values of m. This yields new algorithms, improving upper bounds on α_m for $m \in \{4, 5, 6, 7, 8\}$ significantly.

Number of bins	4	5	6	7	8
Previous upper bound	$19/13 < 1.462$	$3/2 = 1.5$	$3/2 = 1.5$	$3/2 = 1.5$	$3/2 = 1.5$
Our upper bound (algorithm)	$39/28 < 1.393$	$31/22 < 1.410$	$20/14 < 1.429$	$16/11 < 1.455$	$19/13 < 1.462$
Time needed (hours)	91	468	23	3.6	235

The computer search was done using a server with an Intel Xeon E5-2630 v3 CPU and 126 GB of RAM. The time complexity grows rapidly with increasing m and granularity, but we do not believe that we have reached any fundamental limit of this method.

For three bins, we found an upper bound of $91/66 < 1.379$ with a run time of 46 h. This is worse than the previous upper bound of $11/8 = 1.375$ by Böhm et al. [2] For $m \geq 9$ we did not find any upper bound better than $3/2 = 1.5$.

2 Reductions Between Games

We view Online Bin Stretching as a game. The game is:

– deterministic
– 2-player – we call the players $\mathcal{A}$ (the adversary) and $\mathcal{B}$ (the algorithm)
– sequential – players alternate in taking turns with $\mathcal{A}$ going first
– partisan – $\mathcal{A}$ generates items and $\mathcal{B}$ assigns them to bins
– perfect information
– zero-sum – when the game ends, one player is the winner and the other is the loser

When we talk about winning or losing the game, it will be from the point of view of $\mathcal{B}$. The following definitions are worded carefully since we shall need to apply these methods to games with both unbounded branching and unbounded number of turns.

We shall define a pair of functions, $M_{\mathcal{A}}^{G}$ and $M_{\mathcal{B}}^{G}$, which will indicate the possible moves. For example, $M_{\mathcal{B}}^{G}(p)$ will return the set of positions to which $\mathcal{B}$ can move from position p, always including $lose^{G}$ (letting $\mathcal{B}$ resign) and possibly including win^{G} (if $\mathcal{B}$ can claim victory).

We formally define the type of games we are interested in as follows.

Definition 2.1. *Game G is a 7-tuple $(P_{\mathcal{A}}^{G}, P_{\mathcal{B}}^{G}, init^{G}, win^{G}, lose^{G}, M_{\mathcal{A}}^{G}, M_{\mathcal{B}}^{G})$ where:*

– *$P_{\mathcal{A}}^{G}$ is the set of all possible game positions where $\mathcal{A}$ is the next player to move*
– *$P_{\mathcal{B}}^{G}$ is the set of all possible game positions where $\mathcal{B}$ is the next player to move*
– *$init^{G} \in P_{\mathcal{A}}^{G}$ is the initial position*
– *win^{G} is the winning position where the game ends and $\mathcal{B}$ wins*
– *$lose^{G}$ is the losing position where the game ends and $\mathcal{B}$ loses*
– *$M_{\mathcal{A}}^{G} : P_{\mathcal{A}}^{G} \to \mathcal{P}\left(P_{\mathcal{B}}^{G} \cup \{win^{G}, lose^{G}\}\right)$ indicates all possible moves of $\mathcal{A}$*
– *$M_{\mathcal{B}}^{G} : P_{\mathcal{B}}^{G} \to \mathcal{P}\left(P_{\mathcal{A}}^{G} \cup \{win^{G}, lose^{G}\}\right)$ indicates all possible moves of $\mathcal{B}$*

We further require that the following constraints hold:

– *$P_{\mathcal{A}}^{G}$, $P_{\mathcal{B}}^{G}$, $\{win^{G}\}$ and $\{lose^{G}\}$ are disjoint sets*
– *$\forall p \in P_{\mathcal{A}}^{G} : win^{G} \in M_{\mathcal{A}}^{G}(p)$ – $\mathcal{A}$ can resign whenever it is their turn*
– *$\forall p \in P_{\mathcal{B}}^{G} : lose^{G} \in M_{\mathcal{B}}^{G}(p)$ – $\mathcal{B}$ can resign whenever it is their turn*

When talking about positions, we are mostly interested in the case where the next player to move is $\mathcal{A}$. Since $\mathcal{A}$ can make the game of Online Bin Stretching last for an unbounded number of turns, and an algorithm that can always pack the next item is satisfactory, we are interested in non-losing rather than winning positions. Therefore, we define the concept of losing positions as follows by building up the set of losing positions iteratively.

Definition 2.2. *Let G be a game. We define*

– *$L_{0}^{G} = \{lose^{G}\}$*

- $L^G_{n+1} = L^G_n \cup \{a \in P^G_{\mathcal{A}} \mid (\exists b \in M^G_{\mathcal{A}}(a))(\forall c \in M^G_{\mathcal{B}}(b))(c \in L^G_n)\}$
- $L^G = \bigcup_{i=0}^{\infty} L^G_i$. *We then call* L^G *the set of losing positions.*

A position is non-losing if it is not losing and game G *is non-losing if* $init^G$ *is a non-losing position.*

Our paper relies on the concept of reductions among games. We say that a game G reduces to game H if we can translate the moves of the adversary from G to H and then translate the algorithms response back from H to G in such a way, that if the algorithm avoids losing H, then the same result is achieved in G.

Definition 2.3 (Reduction). *For any two games* G *and* H*, we say that* $f : P^G_{\mathcal{A}} \cup P^G_{\mathcal{B}} \cup \{win^G, lose^G\} \to P^H_{\mathcal{A}} \cup P^H_{\mathcal{B}} \cup \{win^H, lose^H\}$ *is a reduction from game* G *to game* H *if it satisfies the following four conditions:*

1. $f(init^G) = init^H$, $f(win^G) = win^H$, *and* $f(lose^G) = lose^H$
2. $f(P^G_{\mathcal{A}}) \subseteq P^H_{\mathcal{A}}$ *and* $f(P^G_{\mathcal{B}}) \subseteq P^H_{\mathcal{B}}$
3. $(\forall a \in P^G_{\mathcal{A}})(\forall b \in M^G_{\mathcal{A}}(a))(f(b) \in M^H_{\mathcal{A}}(f(a)))$
4. $(\forall b \in P^G_{\mathcal{B}})(\forall c' \in M^H_{\mathcal{B}}(f(b)))(\exists c \in M^G_{\mathcal{B}}(b))(f(c) = c')$

We also say that a reduction is computable if there exists an algorithm that can find

- $f(b)$ *for any* $b \in P^G_{\mathcal{B}}$
- $c \in M^G_{\mathcal{B}}(b)$ *such that* $f(c) = c'$ *for any* $b \in P^G_{\mathcal{B}}$ *and any* $c' \in P^H_{\mathcal{A}}(f(b))$

Theorem 2.4. *If* f *is a reduction from game* G *to game* H*, and* H *is non-losing, then* G *is non-losing. Furthermore, if the reduction is computable and we are given an algorithm for* H*, this gives an algorithm for* G*.*

Proof. In order to prove that $init^G \in L^G$ implies $init^H \in L^H$, we prove by induction on n that $(\forall a \in P^G_{\mathcal{A}})(\forall n \in \mathbb{N})(a \in L^G_n \implies f(a) \in L^H_n)$.

By Definition 2.2, $L^G_0 = \{lose^G\}$ and $L^H_0 = \{lose^H\}$. By Condition 1, $f(lose^G) = lose^H$. The basis of the induction thus holds.

For the induction step, suppose that $a \in L^G_{n+1}$. We now need to prove that $f(a) \in L^H_{n+1}$. By definition of L^G_{n+1}, either position a was already contained in L^G_n or there exists a position $b \in M^G_{\mathcal{A}}(a)$ such that $(\forall c \in M^G_{\mathcal{B}}(b))(c \in L^G_n)$. In the first case, $f(a) \in L^H_n$ by induction and $L^H_n \subseteq L^H_{n+1}$ by definition of L^H_{n+1}.

In the latter case, let us take an arbitrary $b \in M^G_{\mathcal{A}}(a)$ such that $(\forall c \in M^G_{\mathcal{B}}(b))(c \in L^G_n)$. We shall now prove that position $f(b)$ in game H has properties similar to those of position b in game G.

Condition 3 gives us $f(b) \in M^H_{\mathcal{A}}(f(a))$. Condition 4 ensures that $(\forall c' \in M^H_{\mathcal{B}}(f(b)))(\exists c \in M^G_{\mathcal{B}}(b))(f(c) = c')$. By induction we already know that $(\forall c \in M^G_{\mathcal{B}}(b))(f(c) \in L^H_n)$, giving us $(\forall c' \in M^H_{\mathcal{B}}(f(b)))(c' \in L^H_n)$ and thus $f(a) \in L^H_{n+1}$.

Now that we know that $a \in L^G_{n+1}$ implies $f(a) \in L^H_{n+1}$, we simply observe that if G is losing, then $init^G$ is in some L^G_n which implies that $init^H$ is in L^H_n and thus H is losing.

The resulting algorithm will, when in position $b \in P^G_{\mathcal{B}}$, find $f(b)$, observe to which position $c' \in P^H_{\mathcal{A}}(f(b))$ the algorithm for H moves from $f(b)$, and move to a position $c \in M^G_{\mathcal{B}}(b)$ such that $f(c) = c'$.

3 Defining the Games Formally

We view Online Bin Stretching as a game between two players, $\mathcal{A}$ and $\mathcal{B}$. Let us first introduce a version that represents Online Bin Stretching the most directly. For a given number of bins m and target stretching factor α, we call this game Real Game(m, α) and it proceeds as follows:

In every round, $\mathcal{A}$ either generates an item or resigns. $\mathcal{B}$ must then place into one of the m bins or resign. We keep track of the sequence s of items generated by $\mathcal{A}$, and their packing t into the bins by $\mathcal{B}$. We restrict $\mathcal{A}$ to sequences that can be packed into m bins of size 1 and $\mathcal{B}$ to packings where no bin exceeds load α, forcing them to resign if they have no other choice.

Definition 3.1 (Real Game). *We define* Real Game(m, α) *as the game G with the following components:*

- *each position in $P^G_{\mathcal{A}} \setminus \{init^G\}$ is an ordered triple (n, s, t) where*
  - *$n \in \mathbb{N}$ is the number of already packed items*
  - *$s \in (0, 1]^n$ is a sequence of n items that can be packed into m bins of size at most 1*
  - *$t \in \{1, \ldots, m\}^n$ such that $(\forall i)(\sum_{j:t_j=\alpha} s_j \leq 1)$, which is a packing of those n items into m bins of size at most α*
- *each position in $P^G_{\mathcal{B}}$ is an ordered triple (n, s, t) where*
  - *$n \in \mathbb{N}$ is the number of already packed items*
  - *$s \in (0, 1]^{n+1}$ is a sequence of $n + 1$ items that can be packed into m bins of size at most 1*
  - *$t \in \{1, \ldots, m\}^n$ is a packing of the first n items into m bins of size at most α*
- *For any position $(n, s, t) \in P^G_{\mathcal{A}} \setminus \{init^G\}$ the set $M^G_{\mathcal{A}}((n, s, t))$ of possible moves by $\mathcal{A}$ contains:*
  - *positions $(n, (s_1, \ldots, s_n, s'), t)$ for all $s' \in (0, 1]$ such that the position is in $P^G_{\mathcal{B}}$*
  - *win^G*
- *The set $M^G_{\mathcal{A}}(init^G)$ of possible moves by $\mathcal{A}$ contains:*
  - *positions $(0, (s_1), \emptyset)$ for all $s_1 \in (0, 1]$*
  - *win^G*
- *For any position $(n, s, t) \in P^G_{\mathcal{B}}$ the set $M^G_{\mathcal{B}}((n, s, t))$ of possible moves by $\mathcal{B}$ contains:*
  - *positions $(n + 1, s, (t_0, \ldots, t_n, t'))$ for all $t' \in \{1, \ldots, m\}$ such that the position is in $P^G_{\mathcal{A}}$*
  - *$lose^G$*

Theorem 3.2. *An algorithm for* Online Bin Stretching *with m bins and stretching factor α exists if a non-losing* Real Game *strategy exists for $\mathcal{B}$.*

Proof. Consider any finite input for Online Bin Stretching. If we have $\mathcal{A}$ play according to the input and the algorithm plays according to $\mathcal{B}$, then at the end of input no bin exceeds size α.

There are several main obstacles to implementing a computer search of REAL GAME:

1. $\mathcal{A}$ has an infinite selection of item sizes to pick from.
2. By sending arbitrarily small items, $\mathcal{A}$ can make REAL GAME last arbitrarily many rounds.
3. Computing the maximum size of item $\mathcal{A}$ can generate in a given position requires solving BIN PACKING PROBLEM, which is NP-hard.

In order to avoid these problems, we analyse a simplified version of REAL GAME that we shall call ROUNDED GAME. We then construct a computable reduction from REAL GAME to ROUNDED GAME, proving that a winning strategy for $\mathcal{B}$ in ROUNDED GAME implies an algorithm for ONLINE BIN STRETCHING. We do this by defining ROUNDED GAME such that it corresponds to ONLINE BIN STRETCHING where $\mathcal{B}$ is restricted in its decision-making process and $\mathcal{A}$ is permitted to cheat to a limited extent.

In essence, $\mathcal{A}$ only informs $\mathcal{B}$ about the item sizes and bin loads after rounding them. $\mathcal{A}$ is then allowed to modify the real values afterwards as long as they remain consistent with what $\mathcal{B}$ was told. The branching factor of the game is thus reduced to the granularity of the rounding. This makes it possible to analyse ROUNDED GAME using computer search at the price of a lack of a winning strategy for $\mathcal{B}$ no longer resulting in new lower bounds. Perhaps surprisingly, these modifications do not prevent us from finding new algorithms for ONLINE BIN STRETCHING. We shall now first describe and define ROUNDED GAME, then explain the connection to REAL GAME and finally describe the reduction between them.

An instance of ROUNDED GAME is parameterised by three values—the number of bins m, the granularity level k and the target bin size z. In every round of ROUNDED GAME(m, k, z), $\mathcal{A}$ generates an item of integer size x which $\mathcal{B}$ must then place into one of the m bins. Finally, $\mathcal{A}$ chooses whether the load of that bin increased by x or $x - 1$. We call the latter case an underflow. We keep track of the current loads of the individual bins, not letting any exceed z, and of the multiset of item sizes we have seen so far. $\mathcal{B}$ may claim victory when the total load of the bins exceeds $m(k + 1) - 1$ or, after decreasing the size of each item by 1, the items do not fit into m bins of size $k - 1$. We define these promises formally below as (A) and (B) and will refer to them later. Finally, $\mathcal{B}$ must resign if it cannot claim victory and an item cannot be packed without a bin exceeding load z.

Definition 3.3 (Rounded Game). *We define* ROUNDED GAME(m, k, z) *as the game H with components as follows:*

- *Each position in $P_{\mathcal{A}}^{H} \setminus \{init^{H}\}$ is an ordered triple (v, w, u) where*
  - *v is a multiset of integers from $\{1, \ldots, k\}$ representing item sizes*
  - *$w \in \{0, \ldots, z\}^{m}$ is an m-tuple representing the current loads of the individual bins*
  - *$u \in \{1, \ldots, m\}$ represents the index of the bin the last item was placed into*

- *Each position in $P^H_\mathcal{B}$ is an ordered triple (v, w, x) where*
  - *v and w are the same as for $P^H_\mathcal{A}$*
  - *$x \in v$ is the size of the last generated (and not yet packed) item*
- *For any position $(v, w, u) \in P^H_\mathcal{A} \setminus \{init^H\}$ the set $M^H_\mathcal{A}((v, w, u))$ of possible moves by $\mathcal{A}$ contains:*
  - *positions $(v \cup \{x\}, w', x)$ for all $x \in \{1, \ldots, k\}$ such that $w'_u \in \{w_u, w_u - 1\}$ and $w'_i = w_i$ for all $i \neq u$, and the position is in $P^H_\mathcal{B}$*
  - *win^H*
- *The set $M^H_\mathcal{A}(init^H)$ of possible moves by $\mathcal{A}$ contains:*
  - *positions $(\{x\}, (0, \ldots, 0), x)$ for all $x \in \{1, \ldots, k\}$ such that the position is in $P^H_\mathcal{B}$*
  - *win^H*
- *For any position $(v, w, x) \in P^H_\mathcal{B}$ the set $M^H_\mathcal{B}((v, w, x))$ of possible moves by $\mathcal{B}$ contains:*
  - *positions (v, w', u) for all $u \in \{1, \ldots, m\}$ such that $w'_u = w_u + x$ and $w'_i = w_i$ for all $i \neq u$, and the position is in $P^H_\mathcal{A}$*
  - *$lose^H$*
  - *win^H if either of the following two promises is broken:*

    (A) the total load of the bins $\sum_{i \in \{1,\ldots,m\}} w_i$ is at most $m(k + 1) - 1$

    (B) after decreasing the size of each item by 1, the items fit into m bins of size $k - 1$

We now need to find a reduction from Real Game to Rounded Game. Consider how we could modify Real Game to make it possible to solve via computer search. Since we cannot restrict the items $\mathcal{A}$ has at their disposal to a finite number of types, we instead restrict information $\mathcal{B}$ has about the game state. We scale the problem to avoid working with fractional values, selecting a granularity $k \in \mathbb{N}$ and target bin size $z \in \mathbb{N}$, which will allow us to analyse Real Game with stretching factor $\alpha = z/k$.

We now provide $\mathcal{B}$ with item sizes and bin loads which have been rounded by function $round(x) = \lceil kx \rceil$ instead of their real loads. Due to the uncertainty about the real sizes, placing an item of rounded size x into a bin of rounded load y might result in the new rounded load being merely $x + y - 1$ instead of $x + y$. We call this effect an *underflow*. In order to ensure we do not restrict $\mathcal{A}$, we let $\mathcal{A}$ decide whether that has happened on their following turn.

The game ends with $\mathcal{B}$ losing whenever the rounded load of any bin exceeds z as this corresponds to that bin having a real load higher than z/k. On the other hand, $\mathcal{B}$ gets the option to claim victory when one of the promises defined above is broken, as this implies that $\mathcal{A}$ is cheating in Real Game. Breaking promise (A) means that the real total size of all items exceeds m, which is not possible in Real Game. Since $round(x) - 1 < kx$, breaking promise (B) means that the real items cannot be packed into m bins of size 1, which is also not possible in Real Game.

Definition 3.4. *With the above in mind, we define $round(x) = \lceil kx \rceil$ and construct the reduction f from G to H where G is* Real Game$(m, z/k)$ *and H is* Rounded Game(m, k, z) *as follows:*

– *$f(init^G) = init^H$, $f(win^G) = win^H$, and $f(lose^G) = lose^H$*
– *For any $(n, s, t) \in P^G_{\mathcal{A}} \setminus \{init^G\}$ we define $f((n, s, t)) = (v, w, u)$ where:*
  - *v is the multiset $\{round(s_1), \dots, round(s_n)\}$*
  - *$u = t_n$*
  - *$w_u = min(z, round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n))$*
  - *$w_i = round(\sum_{j:t_j=i} s_j)$ for $i \neq u$*
– *For any $(n, s, t) \in P^G_{\mathcal{B}}$ we define $f((n, s, t)) = (v, w, x)$ where:*
  - *v is the multiset $\{round(s_1), \dots, round(s_n)\}$*
  - *$x = round(s_{n+1})$*
  - *$w_i = round(\sum_{j:t_j=i} s_j)$ for all i*

Lemma 3.5. *For any position $a = (n, (s_1, \dots, s_n), t) \in P^G_{\mathcal{A}} \setminus \{init^G\}$ and any successor position $(n, (s_1, \dots, s_n, s'), t) \in M^G_{\mathcal{A}}(a) \setminus \{win^G\}$, let $u = t_n$, $w_u = min(z, round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n))$, and $w'_u = round(\sum_{j:t_j=u} s_j)$. Then $w'_u \in \{w_u, w_u - 1\}$.*

Proof. First, observe that $round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n)$ is equal to either $round(\sum_{j:t_j=u} s_j)$ or $round(\sum_{j:t_j=u} s_j) + 1$. This proves the lemma for all cases where $round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n) \leq z$. Since $\sum_{j:t_j=u} s_j$ is at most z/k, $round(\sum_{j:t_j=u} s_j)$ must be at most z, and therefore $round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n)$ is $z + 1$ only if $round(\sum_{j:t_j=u} s_j)$ is z.

Theorem 3.6. *The function f, as defined by Definition 3.4, is indeed a reduction from* Real Game*$(m, z/k)$ to* Rounded Game*(m, k, z). Furthermore, this reduction is computable and any algorithm for* Rounded Game*(m, k, z) thus gives an algorithm for* Real Game*$(m, z/k)$.*

Proof. Let us go over the conditions from Definition 2.3.
Condition 1 obviously holds.
Condition 2:

– All $round(s_i)$ are in $\{1, \dots, k\}$ as all items in Real Game are of size ≤ 1. Thus $\{round(s_1), \dots, round(s_n)\}$ is a multiset of integers from $\{1, \dots, k\}$.
– Since $\sum_{j:t_j=u} s_j$ is at most z/k, $min(z, round(\sum_{j:t_j=u \wedge j \neq n} s_j) + round(s_n))$ and $round(\sum_{j:t_j=i} s_j)$ are always in $\{0, \dots, z\}$.
– t_n is always in $\{1, \dots, m\}$.

Condition 3:

– Let us consider a move from $a \in P^G_{\mathcal{A}}$ to $b \in M^G_{\mathcal{A}}(a)$.
– The condition obviously holds whenever $b = win^G$.
– If $a = init^G$, then:
  - $f(a)$ is $init^H$.
  - Any $b \in M^G_{\mathcal{A}}(init^G) \setminus \{win^G\}$ is $(0, (s_1), \emptyset)$ for some $s_1 \in (0, 1]$.
  - $f(b)$ is $(\{round(s_1)\}, (0, \dots, 0), round(s_1))$.
  - Since $round(s_1) \in \{1, \dots, k\}$, we get $f(b) \in M^H_{\mathcal{A}}(f(a))$.
– If a is some $(n, (s_1, \dots, s_n), t)$ from the set $P^G_{\mathcal{A}} \setminus \{init^G\}$, then:

- $f(a)$ is $(\{round(s_1), \dots, round(s_n)\}, w, u)$
- Any position b from $M^G_{\mathcal{A}}(a) \setminus \{win^G\}$ is $(n+1, (s_1, \dots, s_n, s'), t)$ for some $s' \in (0, 1]$.
- $f(b)$ is $(\{round(s_1), \dots, round(s_n), round(s')\}, w', round(s'))$ where $w'_i = w_i$ for all $i \neq u$ and, by Lemma 3.5, $w'_u \in \{w_u, w_u - 1\}$.
- Since $round(s') \in \{1, \dots, k\}$, we get $f(b) \in M^H_{\mathcal{A}}(f(a))$.

Condition 4:

- Let us consider some $b \in P^G_{\mathcal{B}}$ and $c' \in M^H_{\mathcal{B}}(f(b))$.
- The condition obviously holds whenever $c' = lose^G$.
- Since neither of the two promises is ever broken in state $f(b)$, there are no moves from any $f(b)$ to win^G.
- If $b = (n, (s_1, \dots, s_n, s_{n+1}), ())$, then
  - $f(b) = (\{round(s_1), \dots, round(s_{n+1})\}, w, round(s_{n+1}))$.
  - Any position $c' \in M^H_{\mathcal{B}}(f(b))$ is either contained in $\{win^G, lose^G\}$ or is $(\{round(s_1), \dots, round(s_{n+1})\}, w', u)$ for some $u \in \{1, \dots, m\}$, where:
    - By definition of $f(b)$, $w_i = round(\sum_{j:t_j=i} s_j) \leq z$ for all i.
    - By definition of $M^H_{\mathcal{B}}(f(b))$, $w'_i = w_i \leq z$ for all $i \neq u$ and $w'_u = w_u + round(s_{n+1}) \leq z$.
  - For any $c' \in M^H_{\mathcal{B}}(f(b))$, since $w'_i \leq z$, we know that $\sum_{j:t_j=i} s_j \leq z/k$ for all i and thus $c = (n+1, (s_1, \dots, s_n, s_{n+1}), (, u))$ is in $M^G_{\mathcal{B}}(b)$.

We also observe that $f(a)$ is easily computable for any position a and $M^G_{\mathcal{B}}(b)$ is easily computable for any $b \in P^G_{\mathcal{B}}$, making the entire reduction computable.

In order to analyse the game using computer search, we employ some further modifications of the game, which will be described in the following section. However, those modifications are much simpler.

4 Computer Search

Let us first describe the simple modifications of the game that we use to make the computer search easier.

Rounded Game still has an infinite number of states due to not placing any limit on the number of items of rounded size 1 as long as almost all of them underflow. To avoid this problem, we do not include items of size 1 in v, thus ensuring that the number of positions is finite. This is obviously a reduction as items of size 1 never affect the set of possible moves. However, this reduction causes the game to include cycles among the states, caused by adding an item of rounded size 1 and then having it underflow.

In order to prevent these cycles, we forbid $\mathcal{A}$ from making the move from any (v, w, u) to $(v, w', 1)$ where $w'_u = w_u - 1$. While this is not a reduction, we can observe that if $\mathcal{A}$ has a winning strategy, then $\mathcal{A}$ also has a winning strategy that never makes such a move. This is because $\mathcal{B}$ can pack the new item of size 1 into bin u, returning the game to position (v, w, u). We are now guaranteed that the sum of w_i is strictly increasing with every pair of successive moves, ensuring that the game takes at most $O(mk)$ moves.

The computer search uses special game positions which occur after $\mathcal{A}$ chooses whether an underflow occurred and before they select the size of the next item. This is because such positions are completely described by the multiset of items and the m-tuple of current loads, minimizing the number of positions to analyse. Instead of forbidding, as per the previous paragraph, the generation of an item of rounded size 1 after an underflow, we can now forbid the underflowing of items of size 1 with very similar reasoning.

We can now describe the basic version of our computer program, which searches for a strategy that would allow $\mathcal{B}$ to win Rounded Game(m, k, z) where m, k and z are, for the purposes of the program, global constants. We have two helper functions:

- Check takes a multiset of item sizes and checks whether promise (B) was broken
- Add takes (w, x, u) and returns a copy of w with w_u increased by x

The program is then a straightforward application of the MiniMax algorithm first described by von Neumann [11]. Ultimately, if our search returns *True* for the initial game state, the whole Rounded Game is winnable for $\mathcal{B}$, an algorithm for Online Bin Stretching corresponding to the winning strategy exists and therefore $\alpha_m \leq s/k$.

5 Optimizations

We observe that permuting the bins does not affect the game significantly. We thus have the helper function Add sort the bins by their load, decreasing the number of states. This is trivially a reduction. By sorting the bins in decreasing order, the search tries BestFit first, further improving overall computation time.

We implemented custom caching of results for Solve. Since caching for all possible game states used too much memory, we made use of the fact that, for a given game state, we can often find either a less favourable game state that we managed to win previously or a more favourable game state that we could not win. LRU caching was tried and was clearly inferior when compared to the following custom caching approach.

Definition 5.1 (Order on v). *We define $v_1 \preceq v_2$ to be true if the items from v_1 can fit into bins with sizes corresponding to the items in v_2 after decreasing all rounded sizes in both multisets by 1.*

Theorem 5.2. *If $v_1 \preceq v_2$, then* Solve(v_1, w) *implies* Solve(v_2, w)

Proof. If $v_1 \preceq v_2$, then $v_1 \cup v'$ breaking promise (B) implies $v_2 \cup v'$ breaking promise (B) for any multiset of items v'. Otherwise we could pack items of v_1 into the spaces occupied by the items of v_2, showing that $v_1 \cup v'$ does not break promise (B) after all.

Algorithm 1. Computer search for a winning strategy for Rounded Game

```
 1: function Solve($v, w$)                                   ▷ Return $True$ iff position is winnable.
 2:     $remaining = m(k+1) - \sum(w_i) - 1$                 ▷ This ensures promise (A).
 3:     if $remaining + \min(w) < z$ then return $True$      ▷ Put all into emptiest bin.
 4:     end if
 5:
 6:     $limit = \min(remaining, k)$ ▷ Upper bound on largest item that does not let $\mathcal{B}$ win.
 7:
 8:     for $item \in \{1, \ldots, limit\}$ do                ▷ If there exists an item size...
 9:         $v' = v \cup \{item\}$
10:         $result = False$
11:         for $bin \in \{1, \ldots, m\}$ do                 ▷ ...which cannot be packed into any bin...
12:             $w' = $ Add($w, item, bin$)
13:             $w'' = $ Add($w, item - 1, bin$)
14:             if Solve($v', w'$) $\wedge$ ($item == 1 \vee$ Solve($v', w''$)) then
15:                 $result = True$
16:                 break
17:             end if
18:         end for
19:         if not $result$ and not Check($v$) then        ▷ ...and $\mathcal{B}$ cannot claim victory...
20:             return $False$                              ▷ ...then declare this position lost.
21:         end if
22:     end for
23:     return $True$
24: end function
```

For any given w, we now store only the set of minimal v for which Solve(v, w) is True and the set of maximal v for which Solve(v, w) is False. When evaluating Solve(v, w), we query the cache for that w, checking if there is any $v' \preceq v$ for which we know Solve(v', w) to be True, or there is any $v'' \succeq v$ for which we know Solve(v', w) to be False. This is done after line 4 of the algorithm. If the position needs to be evaluated further, we add v to the relevant cache before returning the result and remove any v' made redundant by it.

The comparisons between v are implemented by generalising the function Check. Indeed, the original use of Check corresponds to comparing v to m items of rounded size k.

The (now generalised) helper function Check is relatively computationally intensive. We preprocess the input to remove some easy cases and then use dynamic programming, which proved faster than using an ILP. Results are cached to avoid repeated computation.

As a final optimisation, the program stores all rounded sizes already decreased by 1 to simplify some formulae and avoid off-by-one errors.

These optimizations are how we achieved our results. The code is available on Github [10].

6 Conclusions

We found new algorithms for Online Bin Stretching on 4, 5, 6, 7 and 8 bins. Perhaps surprisingly, these new algorithms were found using discrete computer search, despite the problem inherently using real-valued input. This was possible thanks to a reduction from a game with infinite depth and branching factor to a game with both depth and branching factor being $O(mk)$ where m is the number of bins and k is our chosen granularity. We expect that these techniques will prove to be useful for other similar problems.

Acknowledgements. We would like to thank Martin Böhm for consultation and the reviewers for their valuable feedback.

References

1. Azar, Y., Regev, O.: On-line bin-stretching. Theoret. Comput. Sci. **268**(1), 17–41 (2001)
2. Böhm, M., Sgall, J., van Stee, R., Veselý, P.: A two-phase algorithm for bin stretching with stretching factor 1.5. J. Comb. Optim. **34**, 810–828 (2017)
3. Böhm, M., Simon, B.: Discovering and certifying lower bounds for the online bin stretching problem. arXiv:2001.01125 (2020)
4. Gabay, M., Brauner, N., Kotov, V.: Improved lower bounds for the online bin stretching problem. 4OR **15**, 183–199 (2017)
5. Gabay, M., Brauner, N., Kotov, V.: Semi-online bin stretching with bunch techniques. Theoret. Comput. Sci. **602**, 103–113 (2015)
6. Garey, M. R., Graham, R. L., Ullman J. D.: Worst-case analysis of memory allocation algorithms. In: Proceedings of the Fourth Annual ACM Symposium on Theory of Computing, STOC'72, pp. 143–150 (1972)
7. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Z.: Semi on-line algorithms for the partition problem. Oper. Res. Lett. **21**(5), 235–242 (1997)
8. Kellerer, H., Kotov, V.: An efficient algorithm for bin stretching. Oper. Res. Lett. **41**(4), 343–346 (2013)
9. Lhomme, A., Romane, O., Catusse, N., Brauner, N.: Online bin stretching lower bounds: Improved search of computational proofs. arXiv:2207.04931 (2022)
10. Lieskovský, M.: BinStretchingAlgorithmicSearch: Computer search for better bounds on the bin stretching problem. Github code repository. https://github.com/MatejLieskovsky/BinStretchingAlgorithmicSearch
11. von Neumann, J.: Zur Theorie der Gesellschaftsspiele. Mathematische Annalen **100**, 295–320 (1928)