Noname manuscript No.
(will be inserted by the editor)

Online Bin Stretching with Three Bins

Martin Böhm · Jiří Sgall · Rob van Stee · Pavel Veselý

the date of receipt and acceptance should be inserted later

Abstract ONLINE BIN STRETCHING is a semi-online variant of bin packing in which the algorithm has to use the same number of bins as an optimal packing, but is allowed to slightly overpack the bins. The goal is to minimize the amount of overpacking, i.e., the maximum size packed into any bin.

We give an algorithm for ONLINE BIN STRETCHING with a stretching factor of $11/8 = 1.375$ for three bins. Additionally, we present a lower bound of $45/33 = 1.\overline{36}$ for ONLINE BIN STRETCHING on three bins and a lower bound of $19/14$ for four and five bins that were discovered using a computer search.

1 Introduction

The most famous algorithmic problem dealing with online assignment is arguably ONLINE BIN PACKING. In this problem, known since the 1970s, items of size between 0 and 1 arrive in a sequence and the goal is to pack these items into the least number of unit-sized bins, packing each item as soon as it arrives.

ONLINE BIN STRETCHING, which was introduced by Azar and Regev in 1998 [3,4], deals with a similar online scenario. Again, items of size between 0 and 1 arrive in a sequence, and the algorithm needs to pack them as soon as each item arrives, but it has two advantages: (i) The packing algorithm knows m, the number of bins that an optimal offline algorithm would use, and must also use only at most m bins, and (ii) the packing algorithm can use bins of capacity R for some $R \geq 1$. The goal is to minimize the stretching factor R.

While formulated as a bin packing variant, ONLINE BIN STRETCHING can also be thought of as a semi-online scheduling problem, in which we schedule jobs in an online manner on exactly m machines, before any execution starts. We have a guarantee that the optimum offline algorithm could schedule all jobs with makespan 1. Our task is to present an online algorithm with makespan of the schedule being at most R.

1.1 Motivation

We give two applications of ONLINE BIN STRETCHING.

Server upgrade. This application has originally appeared in [3]. In this setting, an older server (or a server cluster) is streaming a large number of files to the newer server without any guarantee on file order. The files cannot be split between drives. Both servers have m disk drives, but the newer server has a larger capacity of each drive. The goal is to present an algorithm that stores all incoming files from the old server as they arrive.

Shipment checking. A number m of containers arrive at a shipping center. It is noted that all containers are at most $p \leq 100$ percent full. The items in the containers are too numerous to be individually labeled, yet all items must be unpacked and scanned for illicit and dangerous material. After the scanning, the items must be speedily repackaged into the containers for further shipping. In this scenario, an algorithm with stretching factor $100/p$ can be used to repack the objects into containers in an online manner.

Supported by the project 14-10003S of GA ČR and by the GAUK project 548214.

M. Böhm, J. Sgall, P. Veselý
Computer Science Institute of Charles University, Prague, Czech Republic. E-mail: {bohm,sgall,vesely}@iuuk.mff.cuni.cz.

R. van Stee
Department of Computer Science, University of Leicester, Leicester, UK. E-mail: rob.vanstee@leicester.ac.uk.

1.2 History

ONLINE BIN STRETCHING was first proposed by Azar and Regev in [3,4]. Already before this, a matching upper and lower bound of $4/3$ for two bins had appeared [17]. Azar and Regev extended this lower bound to any number of bins and gave an online algorithm with a stretching factor 1.625.

The problem has been revisited recently, with both lower bound improvements and new efficient algorithms. On the algorithmic side, Kellerer and Kotov [16] have achieved a stretching factor $11/7 \approx 1.57$ and Gabay et al. [13] have achieved $26/17 \approx 1.53$. The best known general algorithm with stretching factor 1.5 was presented by the authors of this paper in [8].

In the case with only three bins, the previously best algorithm was due to Azar and Regev [3], with a stretching factor of 1.4.

On the lower bound side, the lower bound $4/3$ of [3] was surpassed only for the case of three bins by Gabay et al. [11], who show a lower bound of $19/14 \approx 1.357$, using an extensive computer search. The preprint [11] was updated in 2015 [12] to include a lower bound of $19/14$ for four bins.

1.3 Our contributions

In Section 2 we present an algorithm for three bins of capacity $11/8 = 1.375$. This is the first improvement of the stretching factor 1.4 of Azar and Regev [3]. In Section 3, we present a new lower bound of $45/33 = 1.\overline{36}$ for ONLINE BIN STRETCHING on three bins, along with a lower bound of $19/14$ on four and five bins which is the first non-trivial lower bound for four and five bins. We build on the paper of Gabay et al. [11] but significantly change the implementation, both technically and conceptually. The lower bound of $19/14$ for four bins is independently shown in [12].

A preliminary version of this work appeared in WAOA 2014 [7] and SOFSEM 2016 [6].

1.4 Related work.

The NP-hard problem BIN PACKING was originally proposed by Ullman [18] and Johnson [15] in the 1970s. Since then it has seen major interest and progress, see the survey of Coffman et al. [9] for many results on classical Bin Packing and its variants. While our problem can be seen as a variant of BIN PACKING, note that the algorithms cannot open more bins than the optimum and thus general results for BIN PACKING do not translate to our setting.

As noted, ONLINE BIN STRETCHING can be formulated as the online scheduling on m identical machines with known optimal makespan. Such algorithms were studied and are important in designing constant-competitive algorithms without the additional knowledge, e.g., for scheduling in the more general model of uniformly related machines [2,5,10].

For scheduling, also other types of semi-online algorithms are studied. Historically first is the study of ordered sequences with non-decreasing processing times [14]. Most closely related is the variant with known sum of all processing times studied in [17] and the currently best results are a lower bound of 1.585 and an algorithm with ratio 1.6, both from [1]. Note that this shows, somewhat surprisingly, that knowing the actual optimum gives a significantly bigger advantage to the online algorithm over knowing just the sum of the processing times (which, divided by m, is a lower bound on the optimum).

1.5 Problem statement and basic notation

We formally define ONLINE BIN STRETCHING as follows:

Input: an integer m and a sequence of items $I = i_1, i_2, \ldots$ given online one by one. Each item has a *size* $s(i) \in [0,1]$ and must be packed immediately and irrevocably.
Parameter: The *stretching factor* $R \geq 1$.
Output: Partitioning (packing) of I into bins $B_1, \ldots, B_m$ so that $\sum_{i \in B_j} s(i) \leq R$ for all $j = 1, \ldots, m$.
Guarantee: there exists a packing of all items in I into m bins of capacity 1.
Goal: Design an online algorithm with the stretching factor R as small as possible which packs all input sequences satisfying the guarantee.

For a bin B, we define the *size of the bin* $s(B) = \sum_{i \in B} s(i)$. Unlike $s(i)$, $s(B)$ can change during the course of the algorithm, as we pack more and more items into the bin. To easily differentiate between items, bins and lists of bins, we use lowercase letters for items (i, b, x), uppercase letters for bins and other sets of items (A, B, X), and calligraphic letters for lists of bins ($\mathscr{A}$, $\mathscr{C}$, $\mathscr{L}$).

We rescale the item sizes and bin capacities for simplicity. Therefore, in our setting, each item has an associated size $s(i) \in [0,k]$, where $k \in \mathbb{N}$ is also the capacity of the bins which the optimal offline algorithm uses. The online algorithm for ONLINE BIN STRETCHING uses bins of capacity $t \in \mathbb{N}$, $t \geq k$. The resulting stretching factor is thus t/k.

Our final online algorithm uses several subroutines, one of which is the classical online algorithm FIRST FIT:

Subroutine FIRST FIT**:**
(1) Set an ordering of your bins.
(2) For every incoming item i:
(3) Pack i into the first bin where i fits below or to the limit t.
(4) If no such bin exists, report failure.

Throughout the proof, we will need to argue about loads of the bins A, B, C before various items arrived. The following notation will help us in this endeavour:

Suppose that A is a bin and x is an item that gets packed at some point of the algorithm (not necessarily into A). Then $A_{\leftarrow x}$ will indicate the set of items that are packed into A just before x arrived.

2 Algorithm

We scale the input sizes by 16. The stretched bins in our setting therefore have capacity 22 and the optimal offline algorithm can pack all items into three bins of capacity 16 each. We prove the following theorem.

Theorem 2.1 *There exists an algorithm that solves* Online Bin Stretching *for three bins with stretching factor* $1+3/8=1.375$.

The three bins of our setting are named A, B, and C. We exchange the names of bins sometimes during the course of the algorithm.

A natural idea is to try to pack first all items in a single bin, as long as possible. In general, this is the strategy that we follow. However, somewhat surprisingly, it turns out that from the very beginning we need to put items in two bins even if the items as well as their total size are relatively small.

It is clear that we have to be very cautious about exceeding a load of 6. For instance, if we put 7 items of size 1 in bin A, and 7 such items in B, then if two items of size 16 arrive, the algorithm will have a load of at least 23 in some bin. Similarly, we cannot assign too much to a single bin: putting 20 items of size 0.5 all in bin A gives a load of 22.5 somewhere if three items of size 12.5 arrive next. (Starting with items of size 0.5 guarantees that there is a solution with bins of size 16 at the end.)

On the other hand, it is useful to keep one bin empty for some time; many problematic instances end with three large items such that one of them has to be placed in a bin that already has high load. Keeping one bin free ensures that such items must have size more than 11 (on average), which limits the adversary's options, since all items must still fit into bins of size 16.

Deciding when exactly to start using the third bin and when to cross the threshold of 6 for the first time was the biggest challenge in designing this algorithm: both of these events should preferably be postponed as long as possible, but obviously they come into conflict at some point.

2.1 Good situations

Before stating the algorithm itself, we list a number of *good situations* (GS). These are configurations of the three bins which allow us to complete the packing regardless of the following input.

It is clear that the identities of the bins are not important here; for instance, in the first good situation, all that we need is that *any* two bins together have items of size at least 26. We have used names only for clarity of presentation and of the proofs.

Definition 2.2 A *partial packing* of an input sequence S is a function $p : S_1 \to \{A,B,C\}$ that assigns a bin to each item from a prefix S_1 of the input sequence S.

Good Situation 1 *Given a partial packing such that* $s(A)+s(B)\geq 26$*, there exists an online algorithm that can finish the packing with capacity* 22.

Proof Since the optimum can pack into three bins of size 16, the total size of items in the instance is at most $3\cdot 16=48$. If two bins have size $s(A)+s(B)\geq 26$, all the remaining items (including the ones already placed on C) have size at most 22. Thus we can pack them all into bin C. □

Good Situation 2 *Given a partial packing such that* $s(A)\in[4,6]$*, there exists an online algorithm that can finish the packing with capacity* 22.

Proof Let A be the bin with size between 4 and 6 and B be one of the other bins (choose arbitrarily). Put all the items greedily into B. When an item x does not fit, put it into A, where it fits, as $s(A_{\leftarrow x})\leq 6$. Now $s(B_{\leftarrow x})+s(x)>22$. In addition, $s(A_{\leftarrow x})\geq 4$ by the assumption. Together we have $s(A_{\leftarrow x})+s(B_{\leftarrow x})+s(x)\geq 26$, which is GS1. □

From now on, we assume that each bin $X\in\{A,B,C\}$ satisfies $s(X)\notin[4,6]$, otherwise we are in GS2.

Good Situation 3 *Given a partial packing such that* $s(A)\geq 15$ *and either* (i) $s(B)+s(C)\geq 22$ *or* (ii) $s(C)<4$ *and* $s(B)$ *is arbitrary, there exists an online algorithm that can finish the packing with capacity* 22.

Proof (i) We have $\max(s(B),s(C))\geq 11$, so we are in GS1 on bins A and B or on bins A and C.

(ii) We pack arriving items into B. If $s(B)\geq 11$ at any time, we apply GS1 on bins A and B. Thus we can assume $s(B)<11$ and we cannot continue packing into B any further. This implies that an item i arrives such that $s(i)>11$. As $s(C_{\leftarrow i})<4$, we pack i into it and apply GS1 on bins A and C. □

Good Situation 4 *Given a partial packing such that* $s(A)+s(B)\geq 15+\frac{1}{2}s(C)$*,* $s(B)<4$*, and* $s(C)<4$*, there exists an online algorithm that can finish the packing with capacity* 22.

Proof Let c be the value of $s(C)$ when the conditions of this good situation hold for the first time. We run the following algorithm until we reach GS1 or GS3:

(1) If the incoming item i has $s(i) \geq 11 - \frac{1}{2}c$, pack i into B.
(2) Else, if i fits on A, pack it there.
(3) Otherwise pack i into C.

If at any time an item is to be packed into B by Step (1) (it always fits since we maintain $s(B) < 4$), then $s(A) + s(B) \geq 26$ and we reach GS1. In the event that no item is packed into B, we reach GS3 (with B in the role of C) whenever the algorithm brings the size of A to or above 15.

The only remaining case is when $s(A) < 15$ throughout the algorithm and several items with size in the interval $I := (22 - s(A), 11 - \frac{1}{2}c)$ arrive. These items are packed into C. Note that $I \subseteq (7, 11)$ and that the lower bound of I may decrease during the course of the algorithm.

The first two items with size in I will fit together, since $2(11 - \frac{1}{2}c) + c = 22$. With two such items packed into C, we know that the load $s(A) + s(C)$ is at least $s(A) + 2(22 - s(A)) = 44 - s(A) > 29$ and we have reached GS1, finishing the analysis. □

Good Situation 5 *Given a partial packing such that a new item a with $s(a) > 6$ is packed into bin A, $s(B_{\leftarrow a}) \in [3, 4)$, and $s(C_{\leftarrow a}) = 0$, there exists an online algorithm that can finish the packing with capacity* 22.

Proof Pack all incoming items into A as long as it is possible. If at some point $s(A) \geq 12$, we are in GS4, and so we assume the contrary. Therefore, $s(A) < 12$ and an item i arrives which cannot be packed into A.

Place i into B. If $s(i) \geq 12$, we apply GS3. We thus have $s(i) \in (10, 12)$ and $s(A_{\leftarrow i}) > 22 - s(i) > 10$. Continue with FIRST FIT on bins B, A, and C in this order. (That is, pack an incoming item into the first bin X in which the item fits. If there is no such bin, stop.)

We claim that GS1 is reached at the latest after FIRST FIT has packed two items, x and y, on bins other than B. If one of them (say x) is packed into bin A, this holds because $s(x) + s(B_{\leftarrow x}) > 22$ and $s(A_{\leftarrow x}) > 10$—enough for GS1. If both items do not fit in A, they are both larger than 10, since $s(A_{\leftarrow i}) < 12$ and nothing gets packed into A after item i. We will show by contradiction that this cannot happen.

As $s(A_{\leftarrow x}) < 12$ from our previous analysis, we note that $s(x), s(y) > 10$. We therefore have three items i, x, y with $s(i), s(x), s(y) > 10$ and an item $s(a) > 6$ from our initial conditions. These four items cannot be packed together by any offline algorithm into three bins of capacity 16, and so we have a contradiction with $s(x), s(y) > 10$. □

Good Situation 6 *If $s(C) < 4$, $s(B) > 6$ and $s(A) \geq s(B) + 4 - s(C)$, there exists an online algorithm that can finish the packing with capacity* 22.

Proof Pack all items into A, until an item x does not fit. At this point $s(A_{\leftarrow x}) + s(x) > 22$. If x fits on B, we put it there and reach GS1 because $s(B_{\leftarrow x}) > 6$. Otherwise, x definitely fits on C because $s(C_{\leftarrow x}) < 4$ by assumption. By the condition on $s(A)$, we have $s(x) + s(A_{\leftarrow x}) + s(C) \geq s(x) + s(B) + 4 > 26$, and we are in GS1 again. □

Good Situation 7 *Consider the arrival of an item x. If it holds that*

- $s(A_{\leftarrow x}) < 4$,
- $s(C_{\leftarrow x}) < 4$,
- $s(B_{\leftarrow x}) \leq 9 + \frac{1}{2}(s(A_{\leftarrow x}) + s(C_{\leftarrow x}))$,
- *and* $s(B_{\leftarrow x}) + s(x) > 22$,

then there exists an online algorithm that packs all remaining items into three bins of capacity 22.

Proof We have $s(B_{\leftarrow x}) > 22 - s(x) > 6$ and

$$s(x) > 22 - s(B_{\leftarrow x}) \geq 13 - \frac{1}{2}(s(A_{\leftarrow x}) + s(C_{\leftarrow x})) \geq s(B_{\leftarrow x}) + 4 - s(A_{\leftarrow x}) - s(C_{\leftarrow x}).$$

Placing x on A we increase $s(A)$ to at least $s(B_{\leftarrow x}) + 4 - s(C_{\leftarrow x})$ and we reach GS6. □

2.2 Good Situation First Fit

Throughout our algorithm, we often use a special variant of FIRST FIT which tries to reach good situations as early as possible. This variant can be described as follows:

Definition 2.3 Let $\mathscr{L} = (X|_k, Y|_l, \ldots)$ denote a list of bins $X, Y, \ldots$ where each bin X has an associated capacity k satisfying $s(X) \leq k$. GSFF($\mathscr{L}$) (Good Situation First Fit) is an online algorithm for bin stretching that works as follows:

Subroutine GSFF($\mathscr{L}$)**:** For each item i:
If it is possible to pack i into any bin (including bins not in $\mathscr{L}$, and using capacities of 22 for all bins) so that a good situation is reached, do so and continue with the algorithm of the relevant good situation.
Otherwise, traverse the list $\mathscr{L}$ in order and pack i into the first bin X such that $X|_k \in \mathscr{L}$ and $s(X) + s(i) \leq k$. If there is no such bin, stop.

For example, GSFF($A|_4, B|_{22}$) checks whether either the packing $(A \cup \{i\}, B, C)$, $(A, B \cup \{i\}, C)$ or $(A, B, C \cup \{i\})$ is a partial packing of any good situation. If this is not the case, the algorithm packs i into bin A provided that $s(A) + s(i) \leq 4$. If $s(A) + s(i) > 4$, the algorithm packs i into bin B with capacity 22. If i cannot be placed into B, GSFF($A|_4, B|_{22}$) halts and another online algorithm must be applied to pack i and subsequent items.

2.3 The algorithm

In a way, any algorithm for online bin stretching for three bins must be designed so as to avoid several *bad situations:* the two most prominent ones being either two items of size $R/2$ or three items of size $R/3$, where R is the volume of the remaining items.

Our algorithm – especially Steps (4) and (10) – are designed to primarily evade such bad situations, while making sure that no good situation is missed. This evasive nature gives it its name.

Algorithm EVASIVE:

(1) Run GSFF($A|_4, B|_4$).
(2) Rename the bins so that $s(A) \geq s(B)$.
(3) If the next item j satisfies $s(j) > 6$:
(4) Set $p := 6 + s(j)$; apply GSFF($A|_p, B|_4$).
(5) If the next item w fits into $A|_{22}$:
(6) GSFF($A|_{22}, B|_{22}, C|_{22}$).
(7) Else:
(8) GSFF($A|_p, B|_{22}, C|_{22}$).
(9) Else (j satisfies $s(j) < 4$):
(10) GSFF($A|_4, B|_q, C|_4$) where $q := 9 + \frac{1}{2}(s(A) + s(C))$. Whenever $s(A)$ or $s(C)$ change in this step, update q and continue Step (10).
(11) GSFF($A|_4, B|_{22}, C|_{22}$).
(12) GSFF($A|_{22}, B|_{22}, C|_{22}$).

2.4 Analysis

Let us start the analysis of the algorithm EVASIVE in Step (3), where the algorithm branches on the size of the item j.

We first observe that our algorithm can be in two very different states, based on whether $s(j) > 6$ or $s(j) < 4$. Note that the case $s(j) \in [4, 6]$ is immediately settled using GS2, and that in either case it must be true that $s(j) > 2$; an item j with a smaller size would either fit into $A|_4$, $B|_4$ or trigger GS2.

Observation 2.4 *Assume that* $2 < s(j) < 4$. *We have that* $s(A_{\leftarrow j}) \in (3, 4)$ *and* $s(B_{\leftarrow j}) + s(j) \in (6, 8)$ *where A and B are bins after renaming in Step* (2). *Thus both A and B received some items during Step* (1). *Moreover, there is at most one item either in* $A_{\leftarrow j}$, *or in* $B_{\leftarrow j}$.

Proof Since $s(j) < 4$ and $s(B_{\leftarrow j}) < 4$, the item j is assigned to B in Step (10), which by Step (2) is the least loaded bin among A and B after Step (1). For this bin, we have $s(B_{\leftarrow j}) > 2$. If the opposite were true, we would either reach GS2 by packing j into $B_{\leftarrow j}$, or j fits into $B|_4$, a contradiction with the definition of j.

This implies that both A and B received items in Step (1), so $s(A_{\leftarrow j}) + s(B_{\leftarrow j}) > 6$, else a good situation would have been reached before j arrived. It follows that $s(A_{\leftarrow j}) \in (3, 4)$ and $s(B_{\leftarrow j}) + s(j) \in (6, 8)$.

Since any item that is put into B during Step (1) must have size of more than two (otherwise it fits into $A|_6$), only one such item can be packed into B which proves the last statement. □

Contrast the preceding observation with the next one, which considers $s(j) > 6$:

Observation 2.5 *Assume that* $s(j) > 6$. *Then,* $s(A_{\leftarrow j}) < 3$, $s(B_{\leftarrow j}) = 0$.

Proof If $s(A_{\leftarrow j}) \geq 3$ we reach GS5 by packing j into B. However, if $s(A_{\leftarrow j}) < 3$ then $s(A_{\leftarrow j}) + s(B_{\leftarrow j}) < 6$ which can be true only if $s(B_{\leftarrow j}) = 0$; indeed, we would have never packed an item z into a previously empty bin B if it were true that $s(A_{\leftarrow z}) + s(z) < 6$, $s(z) < 3$ and $s(A_{\leftarrow z}) < 3$. □

Both the analysis and the algorithm differ quite a lot based on the size of j. If it holds that $s(j) > 6$, we enter the *large case* of the analysis, while $2 < s(j) < 4$ will be analyzed as the *standard case*. Intuitively, if $s(j) > 6$, the offline optimum is now constrained as well; for instance, no three items of size 10 can arrive in the future. This makes the analysis of the large case comparatively simpler.

2.4.1 The large case

We now assume that $s(j) > 6$. Our goal in both the large case and the standard case will be to show that in the near future either a good situation is reached or several large items arrive, but EVASIVE is able to pack them nonetheless.

Let us start by recalling the relevant steps of the algorithm:

(4) Set $p := 6 + s(j)$; apply GSFF($A|_p, B|_4$).
(5) If the next item w fits into $A|_{22}$:
(6) GSFF($A|_{22}, B|_{22}, C|_{22}$).
(7) Else:
(8) GSFF($A|_p, B|_{22}, C|_{22}$)

By choosing the limit p to be $s(j) + 6$ in Step (4), we make enough room for j to be packed into A. We also ensure that any item i larger than 6 that cannot be placed into A with capacity 22 must satisfy $s(i) + s(j) > 16$ and so i cannot be with j in the same bin in the offline optimum packing.

Let us define A_S as the set of items in A of size less than 6 (packed before or after j). We note the following:

Observation 2.6

1. *During Step* (4), *if B contains any item, it is true that* $s(A_S) + s(B) > 6$.

2. *If no good situation is reached, the item w ending Step* (4) *satisfies* $s(w) > 6$.

Proof The first point follows immediately from our choice of p and $\text{GSFF}(A|_p, B|_4)$.

For the second part of the observation, consider the item w that ends Step (4) and assume $s(w) \le 6$. The possibility that $s(w) \in [4,6]$ is excluded due to GS2. The case $s(B_{\leftarrow w}) \ge 3$ is also excluded, as this would imply GS5 with j in A.

Since $s(B_{\leftarrow w}) + s(w) > 6$, the only remaining possibility is $s(B_{\leftarrow w}) \in [2,3), s(w) \in (3,4)$. Even though w does not fit into $A|_p$, if we were to pack w into $A|_{22}$, we can use the first point of this observation and get $s(A) + s(B) \ge s(j) + \big(s(A_S) + s(B_{\leftarrow w})\big) + s(w) > 6+6+3 = 15$, enough for GS4 as $s(C) = 0$. The algorithm $\text{GSFF}(A|_p, B|_4)$ in Step (4) will notice this possibility and will pack w into A, where it will always fit, as $s(A_{\leftarrow w}) < 15$ by GS4. □

We now split the analysis based on which branch is entered in Step (5):

Case 1: Item w fits into bin A; we enter Step (6).

We first note that $s(A) + s(B) < 15$, else we are in GS4 since C is still empty. This inequality also implies $s(B) = 0$, otherwise we have

$$s(A) + s(B) = s(w) + s(j) + (s(A_S) + s(B)) > 18$$

via Observation 2.6 and this is enough for GS4.

We continue with Step (6) until we reach a good situation or the end of input. Suppose three items x, y, z arrive such that none of them can be packed into A and we do not reach a good situation. We will prove that this cannot happen. We make several quick observations about those items:

1. We have $s(x) > 7$ because $s(A_{\leftarrow x}) < 15$ or we reach GS4. The item x is packed into B.
2. At any point, B contains at most one item, otherwise $s(A) + s(B) > 22 + 7 > 26$, reaching GS1.
3. We have $s(y) > 9$ because $\min(s(A_{\leftarrow y}), s(B_{\leftarrow y})) < 13$ by GS1. The item y is packed into C.
4. The bin C contains also at most one item, similarly to B.
5. Again, we have $s(z) > 9$ similarly to y. The item z does not fit into any bin.

From our observations above, we get $s(x) + s(y) > 22$, $s(x) + s(z) > 22$, $s(y) + s(z) > 22$. Therefore, at least two of the items $\{x, y, z\}$ are of size at least 11 and the third one is larger than 6. However, both items j and w have size at least 6, and there is no way to pack j, w, x, y, z into three bins of capacity 16, a contradiction.

Case 2: Item w does not fit into bin $A|_{22}$. The choice of p gives us $s(j) + s(w) > 16$. Item w is placed on B.

The limit p gives us an upper bound on the volume of small items A_S in A, namely $s(A_S) \le 6$. An easy argument gives us a similar bound on B, namely if $B_S := B \setminus \{w\}$, then $s(B_S) < 4$. Indeed, we have $26 > s(A) + s(B) > 22 + s(B_S)$, the first inequality implied by not reaching GS1.

In Case 2, it is sufficient to consider two items x, y that do not fit into $A|_p$ or $B|_{22}$. We have:

1. Using $s(B_S) < 4$, we have $s(x) + s(w) > 18$ and $s(y) + s(w) > 18$.
2. None of the items x, y fits into $A|_{22}$. If say x did fit, then we use the fact that x does not fit into $B|_{22}$ and get $s(B) + s(A) = \big(s(B_{\leftarrow x}) + s(x)\big) + s(A_{\leftarrow x}) > 22 + s(j) > 26$ and we reach GS1.
3. The choice of the limit p on $s(A)$ implies $s(x) + s(j) > 16$ and $s(y) + s(j) > 16$.
4. Since $\min(s(A), s(B)) < 13$ at all times by GS1, we have $s(x) > 9$ and $s(y) > 9$.
5. The items x and y do not fit together into C, or we would have $s(C) + s(A) > 22 + s(y) > 26$. This implies $s(x) + s(y) > 22$.

From the previous list of inequalities and using $s(j) + s(w) > 16$, we learn that no two items from the set $\{j, w, x, y\}$ can be together in a bin of size 16. Again, this is a contradiction with the assumptions of ONLINE BIN STRETCHING.

2.4.2 The standard case

From now on, we can assume that $s(j) < 4$, j is packed into B and Step (10) of EVASIVE is reached. Recall that by Observation 2.4 $s(A_{\leftarrow j}) \in (3,4)$, $s(B_{\leftarrow j}) + s(j) \in (6,8)$, and there is exactly one item either in A, or in B; we denote this item by e. We repeat the steps done by EVASIVE in the standard case:

(10) $\text{GSFF}(A|_4, B|_q, C|_4)$ where $q := 9 + \frac{1}{2}(s(A) + s(C))$. Whenever $s(A)$ or $s(C)$ change in this step, update q and continue Step (10).
(11) $\text{GSFF}(A|_4, B|_{22}, C|_{22})$.
(12) $\text{GSFF}(A|_{22}, B|_{22}, C|_{22})$.

Recall that $s(A) > 3$ by Observation 2.4. Assuming that no good situation is reached before Step (10), we observe the following:

Observation 2.7 *In Step* (10)*, as long as C is empty, packing any item of size at least* 4 *leads to a good situation. Thus while C is empty, all items that arrive in Step* (10) *and are not put on A have size in* $(6 - s(A), 4)$.

Proof Any item with size in $[4 - s(A), 6 - s(A)] \cup [4,6]$ leads to GS2. Any item with size more than 6 is assigned to B if it fits there, reaching GS5, and else to A or C, reaching GS7 since $s(B) \le 9 + \frac{1}{2}(s(A) + s(C))$. The only remaining possible sizes of items that are not packed into A are $(6 - s(A), 4)$. □

Corollary 2.8 *After Step* (10)*, C contains exactly one item r and* $s(A) + s(C) > 6$.

Proof From the previous observation it is clear that C receives at least one item r in Step (10). No second item r_2 can be packed into $C|_4$ in Step (10) as $s(r)+s(r_2)$ would be at least $2(6-s(A)) > 4$. □

Step (10) terminates with a new item x which fits into $B|_{22}$ (otherwise we would reach GS7), but not below the limit $q = 9+\frac{1}{2}(s(A)+s(C))$. We pack x into B in Step (11), getting $s(B) > 9+\frac{1}{2}(s(A)+s(C)) > 12$.

A possible bad situation for our current packing is when three items b_1, b_2, b_3 arrive, where the items are such that no two items of this type fit together into any bin, and no single item of this type fits on the largest bin, which is B in our case. In fact, we will prove later that this is the only possible bad situation.

We claim that this potential bad situation cannot occur:

Claim 2.9 *Suppose that the algorithm* Evasive *reaches no good situation in the standard case. Then,* $s(C) \geq s(r) > 2.8$ *and after placing* x *into* B *in Step* (11) *it holds that*

$$s(B) < 12.8.$$

Furthermore, suppose that among items that arrive after x*, there are three items* b_1, b_2, b_3 *such that*

$$\min(s(b_1), s(b_2), s(b_3)) > 8.$$

Then, it holds that

$$\min(s(b_1), s(b_2), s(b_3)) < 9.2.$$

We now show how Claim 2.9 finishes the analysis of Evasive. After that we show the claim using linear programming; a formal proof is in Appendix B.

After Step (10), assuming no good situation is reached, the algorithm places x into $B|_{22}$ and moves to Step (11), which is GSFF$(A|_4, B|_{22}, C|_{22})$. Claim 2.9 gives us $s(B) < 12.8$ after placing x, while the fact that we exited Step (10) means that $s(B) > q = 9+(s(A)+s(C))/2 > 12$.

Consider the first item b_1 that does not fit into $A|_4$. We have that $s(b_1) > 2$, otherwise GS2 is reached. However, any item that fits into B (as long as $s(C) \leq 4$) triggers GS4, because $s(A)+s(B)+s(b_1) \geq 6+12 > 15+s(C)/2$.

We now know that the first item b_1 does not fit into both $A|_4$ and $B|_{22}$. We place it into C, noting that $s(b_1) > 22 - s(B) \geq 22-12.8 = 9.2$.

We keep packing items into $A|_4$, waiting for the second item b_2 that does not fit into $A|_4$ in Step (11). Again, $s(b_2) > 2$. Suppose that b_2 fits into $B|_{22}$ or $C|_{22}$. Claim 2.9 gives us $s(r) > 2.8$; we thus sum up bins B and C and get $s(B_{\leftarrow b_2})+s(b_2)+s(r)+s(b_1) > 12+2+2.8+9.2 = 26$, which is enough for GS1. Our assumption was false, the item b_2 does fit into neither $B|_{22}$ nor $C|_{22}$, in particular we have that $s(b_2) > 9.2$.

We move to Step (12), pack b_2 into $A|_{22}$ and continue with GSFF$(A|_{22}, B|_{22}, C|_{22})$. If at any time $s(A) \geq 14$, we enter GS1 on A and B. Otherwise, if an item b_3 does not fit into $A|_{22}$, it must satisfy $s(b_3) > 8$.

We now apply the full strength of Claim 2.9. The smallest item of b_1, b_2, b_3 must have size less than 9.2, and because of our argument, it must be b_3 – but this means it fits into B, as $s(B) < 12.8$. GS1 on bins A and B finishes the packing, since $s(A)+s(b_3)+s(B_{\leftarrow b_3}) > 22+12 > 26$.

2.4.3 *Proof of Claim 2.9*

Our current goal is to prove Claim 2.9. As in the large case, we would now like to appeal to the offline layout of the larger items currently packed. Unlike the large case, none of the items we have packed before Step (11) is guaranteed to be over 6.

Sidestepping this obstacle, we will argue about the offline layout of the smaller items. We now list several items that are packed before Step (12) and will be important in our analysis:

Definition 2.10 The four items e, j, r, x are defined as follows:

1. The item $e, 2 < s(e) < 4$: the only item packed into B in Step (1) by Observation 2.4. (Note that e might end up on A after renaming the bins.)
2. The item $j, 2 < s(j) < 4$, defined in Step (3).
3. The item $r, 2 < s(r) < 4$, placed into C in Step (10) by Observation 2.8; r is the only item in C until Step (11).
4. The item x which terminated Step (10).

There are four such items and only three bins, meaning that in the offline optimum layout with capacity 16, two of them are packed in the same bin. We will therefore argue about every possible pair, proving that each pair is of size more than 6.8.

Our main tool in proving the mentioned lower bounds are the inequalities that must be true during various stages of algorithm Evasive, since a good situation was not reached. We now list all the major inequalities that we will use.

- Observation 2.4:
  $$s(A_{\leftarrow j}) > 3. \tag{1}$$
- Beginning of Step (10):
  $$s(B_{\leftarrow j}) + s(j) > 6. \tag{2}$$
- GS2 not reached in Step (10):
  $$s(A_{\leftarrow i}) + s(i) > 6. \tag{3}$$
- r does not fit into $B|_q$:
  $$s(B_{\leftarrow r}) + s(r) > 9 + \frac{s(A_{\leftarrow r})}{2}. \tag{4}$$

– No GS4 if r packed into B:

$$(s(B_{\leftarrow r}) + s(r)) + s(A_{\leftarrow r}) < 15. \quad (5)$$

– No GS4 when x packed into $B|_{22}$:

$$(s(B_{\leftarrow x}) + s(x)) + s(r) < 15 + \frac{s(A_{\leftarrow x})}{2}. \quad (6)$$

– No GS4 when x packed into $B|_{22}$:

$$(s(B_{\leftarrow x}) + s(x)) + s(A_{\leftarrow x}) < 15 + \frac{s(r)}{2}. \quad (7)$$

– x does not fit into $B|_q$:

$$s(B_{\leftarrow x}) + s(x) > 9 + \frac{s(A_{\leftarrow x}) + s(r)}{2}. \quad (8)$$

– No GS6 if x packed into A:

$$s(A_{\leftarrow x}) + s(x) < s(B_{\leftarrow x}) + (4 - s(r)). \quad (9)$$

– No GS6 if x packed into A:

$$s(B_{\leftarrow x}) < s(A_{\leftarrow x}) + s(x) + \big(4 - s(r)\big). \quad (10)$$

– No GS6 if x packed into C:

$$s(r) + s(x) < s(B_{\leftarrow x}) + \big(4 - s(A_{\leftarrow x})\big). \quad (11)$$

– No GS6 if x packed into C:

$$s(B_{\leftarrow x}) < s(r) + s(x) + (4 - s(A_{\leftarrow x})). \quad (12)$$

We first show the claim using infeasible linear programming (LP) instances formed by the above inequalities. The specific instances can be found in Appendix A and online at `http://github.com/bohm/binstretch/`.

We write our LPs in the GNU MathProg language, thus they can be verified using GNU Linear Programming Kit[1]. In Appendix B we provide formal proofs of these lemmas for completeness.

In our LPs we use a variable `i` for $s(i)$, the size of item i, and `X` for $s(X)$ where X is a bin. Instead of $s(X_{\leftarrow i})$ we write `X_i`.

Since our inequalities are mostly strict and LPs cannot contain strict inequalities, we add a non-negative variable `eps` (epsilon) which allows us to turn strict inequalities to non-strict. More precisely, we change an inequality of type `A < B` into `A + eps <= B` and we maximize the value of `eps`; we can do this, because our LPs do not need another objective function. If the optimal value of `eps` is zero, or if the LP is infeasible, then also the original system or strict inequalities is infeasible as well. Otherwise, if the optimal value of `eps` is positive, then all the strict inequalities can be satisfied.

Our first lemma establishes that j is actually the only item that is packed into B during Step (10), which intuitively means that j is not too small:

[1] One can also use online solver at `https://www3.nd.edu/~jeff/mathprog/`.

Lemma 2.11 *Assume that no good situation is reached until Step* (11)*. Then it holds that during Step* (10)*, only j is packed into B.*

Proof We first prove that no two additional items j_2, j_3 can be packed into B during Step (10). Assuming the contrary, we get $s(B_{\leftarrow j_2}) + s(j_2) + s(j_3) > 6 + 2 + 2 = 10$. With that load on B, we consider the packing at the end of Step (10), when the item x arrived. If $s(x) + s(C_{\leftarrow x}) < 9$, we get GS6 by placing x into C since $s(A) > 3$, so it must be true that $s(x) + s(C_{\leftarrow x}) > 9$, which means $s(x) > 5$. This is enough for us to place x into $B|_{22}$ (where it fits, otherwise we are in GS7) and reach GS3.

This contradiction gives us that at most one additional item j_2 can be packed into B during Step (10). We will now prove that even j_2 does not exist, again by contradiction.

We split the analysis into two cases depending on which of j_2 and r arrives first.

Case 1. The item r is packed before j_2, meaning $s(B_{\leftarrow x}) = s(B_{\leftarrow r}) + s(j_2)$. We create a linear program from inequalities (1), (4), (7), (10) and $s(A_{\leftarrow r}) \geq s(A_{\leftarrow j})$ (since r arrives after j). We also add $s(r) < 4$ and $s(j_2) > 2$, since $s(A_{\leftarrow j_2}) < 4$ and j_2 did not fit into $A|_6$. We obtain LP1 for which the optimal value is 0, a contradiction.

Case 2. In the remaining case, j_2 arrives before r. We create an LP from (2), (3), (5), (7), (12), $s(r) < 4$, $s(A_{\leftarrow j_2}) \leq s(A_{\leftarrow r}) \leq s(A_{\leftarrow x})$, and $s(B_{\leftarrow x}) = s(B_{\leftarrow r}) = s(B_{\leftarrow j}) + s(j) + s(j_2)$ (the last two are only true here in Case 2, where r arrived later than j_2). The resulting LP2 is infeasible. □

Having established that only one item j is packed into B during Step (10), we can start deriving lower bounds on pairs of items from the set $\{e, j, r, x\}$. We will prove these bounds similarly to Lemma 2.11 by infeasible LPs from bounds that arise from evading various good situations.

Lemma 2.12 *Suppose that e and r are items as described in Definition 2.10 and suppose also that no good situation was reached during Step* (10) *of the algorithm* EVASIVE*. Then,* $s(e) + s(r) \geq s(B_{\leftarrow j}) + s(r) > 6.8$.

Proof First of all, it is important to note that the item e may be packed on A or on B. Since either $B_{\leftarrow j}$, or $A_{\leftarrow j}$ contains solely e by Observation 2.4, we get that either $s(B_{\leftarrow j}) = s(e)$, or $s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) = s(e)$. Thus it is sufficient to prove $s(B_{\leftarrow j}) + s(r) > 6.8$.

We use (4), (8), (9), $s(j) < 4$,

$$s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) \leq s(A_{\leftarrow r}) \leq s(A_{\leftarrow x}),$$

and a converse of the claim and obtain LP3 with the optimal value equal to 0. □

Lemma 2.13 *Suppose that e and j are items as described in Definition 2.10 and suppose also that no good situation* *was reached by the algorithm* EVASIVE. *Then,* $s(e)+s(j) \geq s(B_{\leftarrow j})+s(j) > 7.6$.

Proof The same argument as in Lemma 2.12 gives us $s(e)+s(j) \geq s(B_{\leftarrow j})+s(j)$. We therefore aim to prove $s(B_{\leftarrow j})+s(j) > 7.6$. We create LP4 from (8), (11), $s(B_{\leftarrow j})+s(r) > 6.8$ by Lemma 2.12, and $s(B_{\leftarrow j} \leq s(A_{\leftarrow x})$ for which 0 is the optimum again. □

Lemma 2.14 *Suppose that j and r are items as described in Definition 2.10 and suppose also that no good situation was reached by the algorithm* EVASIVE. *Then,* $s(r)+s(j) > 7$.

Proof Starting with (4):

$$s(B_{\leftarrow j})+s(j)+s(r) > 9+\frac{s(A_{\leftarrow r})}{2}$$

and using $s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) \leq s(A_{\leftarrow r})$ with $s(B_{\leftarrow j}) < 4$, we have:

$$s(j)+s(r) > 9+\left(\frac{s(A_{\leftarrow r})}{2}-s(B_{\leftarrow j})\right) \geq 9-\frac{s(B_{\leftarrow j})}{2} > 7. \qquad \square$$

Lemma 2.15 *Suppose that* x,e,j,r *are items as described in Definition 2.10. Suppose also that no good situation was reached by the algorithm* EVASIVE. *Then,*

$$s(x) > 4 \quad \text{and} \quad s(x)+\min(s(j),s(e),s(r)) > 6.8.$$

Proof With all the previous lemmas in place, the proof is simple enough. We first observe that $s(x) > 4$; this is true because $s(B_{\leftarrow j})+s(j) < 4+4 = 8$ and $s(B_{\leftarrow x})+s(x) > q \geq 12$.

Since the sizes of the remaining three items $\{e,r,j\}$ are bounded from above by 4 but their pairwise sums are always at least 6.8, we have that $\min\{e,r,j\} > 2.8$, which along with $s(x) > 4$ gives us the required bound. □

From Lemmata 2.12, 2.13, 2.14 and 2.15 we get a portion of Claim 2.9: if three big items b_1,b_2,b_3 exist in the offline layout, then one of these items needs to be packed together with at least two items from the set $\{e,j,r,x\}$, and therefore $\min(s(b_1),s(b_2),s(b_3)) < 9.2$. The second bound $s(r) > 2.8$ follows from Lemma 2.12 and the fact that $s(e) < 4$.

All that remains is to prove the bound on $s(B)$, which we do in the following lemma:

Lemma 2.16 *Suppose that no good situation was reached in the algorithm* EVASIVE *during Step* (10)*. Then, after placing x into B in Step* (11)*, it holds that* $s(B) < 12.8$.

Proof As before, we will use our inequalities to derive an LP showing the desired bound. As we have argued above, Lemma 2.12 gives us that $s(r) > 2.8$. We also use inequalities (7), (11), $s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) \leq s(A_{\leftarrow x})$ (this is true because we reorder the bins B, A in Step (2)), and $s(j) < 4$ to get LP5 with 0 being the optimal value. □

With Lemma 2.16 proven, we have finished the proof of Claim 2.9 and completed the analysis of the algorithm EVASIVE.

3 Lower bound

In this section, we describe our lower bound technique for a small number of bins. We build on the paper of Gabay, Brauner and Kotov [11] 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 [11] can be skipped entirely.

On the technical side, we reimplement the algorithm of [11], gaining significant speedup from the reimplementation alone. While the lower bound search program of [11] 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 a new lower bound for ONLINE BIN STRETCHING for three bins, namely $45/33 = 1.\overline{36}$.

We also present the lower bound of $19/14 \approx 1.357$ for $m = 4$ and $m = 5$. Note that this is the first non-trivial lower bound for $m = 5$ and that our result is independent from the lower bound of $19/14$ for $m = 4$ by Gabay et al. [11].

To see the strength of our improvements, consider the scaling factor K and items of integer size. It is easy to see that a general game tree search requires exponential running time with respect to K. The algorithm of [11] is able to check all $K \leq 20$ (for $m = 3$) 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 $m = 5$.

3.1 Lower bound technique

We now 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 3.4.

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 online algorithm for ONLINE BIN STRETCHING. The other player (ADVERSARY) selects an 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.

In this model, existence of an algorithm for ONLINE BIN STRETCHING with stretching factor S is equivalent to knowing that the player ALGORITHM has a winning strategy for the game along with a deterministic algorithm for computing said strategy.

We are interested primarily in the lower bound. Therefore, it makes sense to slightly reformulate the 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 (as defined in Section 1.5) is satisfied – that the presented input can be packed by an offline algorithm into bins with capacity 1.

This way, a winning strategy 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 arbitrarily 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 [11].

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

Definition 3.1 For given parameters $S \in \mathbb{N}, T \in \mathbb{N}$, a **bin configuration** is a tuple $(a,b,c,\mathscr{I})$, where
- $a,b,c \in \{0,1,\dots,S\}$ denote the current sorted loads of the bins, i.e., $a \geq b \geq c$,
- $\mathscr{I}$ is a multiset with ground set $\{1,2,\dots,T\}$ which lists the items used in the bins.

Additionally, in a bin configuration, it must hold:
- that there exists a packing of items from $\mathscr{I}$ into three bins with loads exactly a,b,c,
- that there exists a packing of items from $\mathscr{I}$ 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 with 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 3.2 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:

– There are two players named ADVERSARY and ALGORITHM. The player ADVERSARY starts.
– Each turn of the player ADVERSARY is associated with a bin configuration $B = (a,b,c,\mathscr{I})$. The start of the game is associated with the bin configuration $(0,0,0,\emptyset)$.
– The player ADVERSARY receives a bin configuration B. Then, ADVERSARY selects a number i such that the multiset $\mathscr{I} \cup \{i\}$ can be packed by an offline optimum into three bins of capacity T. The pair (B,i) is then sent to the player ALGORITHM.
– The player ALGORITHM receives a pair (B,i). Its task is 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.

For a bin configuration B we define recursively whether it is won or lost for player ADVERSARY:

– 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.
– 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.
– 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 3.3 We say that a game $\mathrm{BSG}(S,T)$ gives a **lower bound** of S/T if and only if the bin configuration $(0,0,0,\emptyset)$ is won for the player ADVERSARY.

See Figure 1 for an illustration of a game that gives a lower bound of $4/3$ for three bins.

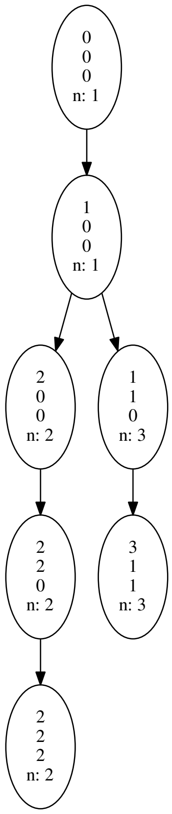

Fig. 1 A simple lower bound tree of value $4/3$ for 3 bins. The root of the tree is the configuration $(0,0,0,\emptyset)$, and each vertex is one bin configuration. To make the presentation more compact, we do not show the configurations which lead to packing of capacity $4/3$ and above.

3.2 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. [11] 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.

The following procedures return 0 if the bin configuration is won for the player ADVERSARY; otherwise they return 1 (player ALGORITHM wins).

Procedure EVALUATEADVERSARY:
Input is a bin configuration $B = (a,b,c,\mathscr{I})$.
(1) Check if the bin configuration is cached; if so, return the value found in cache and end.
(2) Create a list L of items which can be sent as the next step of the player ADVERSARY.
(3) For every item size i in the list L:
(4) Compute EVALUATEALGORITHM(B,i).
(5) If EVALUATEALGORITHM(B,i) returns 0, stop the cycle, store the configuration in the cache and end EVALUATEADVERSARY with return 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.

Procedure EVALUATEALGORITHM:
Input is a bin configuration $B = (a,b,c,\mathscr{I})$ and item i.
(1) If applicable, prune the tree using known algorithms (Section 3.2.3).
(2) For any one of the three bins:
(3) If i can be packed into the bin so that its load is less than S:
(4) Create a configuration B' that corresponds to this packing.
(5) Recurse on EVALUATEADVERSARY(B'). If EVALUATEADVERSARY(B') returns 1, exit the procedure with return 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 $B = (a,b,c,\mathscr{I})$.
(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 report success and output the game tree.

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 satisfying the ONLINE BIN STRETCHING guarantee. We employ the following steps:

1. First, we calculate a lower and upper bound $LB \le UB$ on 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 $\mathscr{I}'$, where $\mathscr{I}'$ is $\mathscr{I}$ plus the item in question and $\mathscr{I}$ is the current multiset of items.
3. The first feasible item size is the desired value of y.

Note that in the second step we could also implement a binary search over the interval, but in our experiments the difference between UB and LB was very small (usually at most 4), thus a linear search is quicker.

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 $\mathscr{I}$ 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 $\mathscr{I}$, as the implementation of $\mathscr{I}$ stores them in a 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 making the ONLINE BIN STRETCHING input invalid.

Our upper bound UB is comparatively simpler; for a bin configuration $(a,b,c,\mathscr{I})$, 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 $3T$.

3.2.1 Procedure TEST

Procedure TEST is a sparse modification of the standard dynamic programming algorithm for KNAPSACK. Given a multiset $\mathscr{I}, |\mathscr{I}| = n$, 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 $\mathscr{I}[i+1]$ 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 $\mathscr{I}[i+1]$ 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 $\mathscr{O}(|\mathscr{I}| \cdot T^3)$ 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 $\mathscr{I}$.

(1) Create two queues Q_1, Q_2.
(2) Add the triple $(\mathscr{I}[1], 0, 0)$ to Q_1.
(3) For each item i in the multiset $\mathscr{I}$, starting with the second item:
(4) For each triple $(a,b,c) \in Q_1$:
(5) If $a + s(i) \le T$:
(6) Add the triple $(a+s(i), b, c)$ to Q_2 unless $F[a+s(i), b, c] = 1$.
(7) Set $F[a+s(i), b, c] = 1$.
(8) Do the same for triples $(a, b+s(i), c)$ and $(a, b, c+s(i))$.
(9) Swap the queues Q_1 and Q_2.
(10) 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 $a \ge b \ge c$, 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 [11]). However, we believe our sparse dynamic programming solution carries little overhead and for large instances it is much faster than the CSP/ILP solvers.

3.2.2 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. Note that we do not cache results of EVALUATEALGORITHM.

We store a large hash table of fixed size, with each entry being a separate chain. With each node in a chain we store the number of accesses. When a chain is to be filled over a fixed limit, we eliminate a node 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 $(a,b,c,\mathscr{I})$, 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.

Our hash function is based on the standard Zobrist hashing approach [19]. We now discuss the specifics of our function. For each bin configuration, we count the occurences of items, creating pairs

$$(i,f) \in \{1,\dots,T\} \times \{0,1,\dots,3T\},$$

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 random 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 $\mathscr{O}(1)$, 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.

So far, we have described caching 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 $B=(a,b,c,\mathscr{I})$ as input, but only the multiset $\mathscr{I}$. Therefore, we aim to eliminate overhead that is caused by calling TEST on a different bin configuration, but with the same multiset $\mathscr{I}$.

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

3.2.3 Tree pruning

Alongside the extensive caching described above, 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 situation are called *good situations*, same as in Section 2.1. We will make use of the first five good situations from Section 2.1.

Recall that in the bin stretching game $\mathrm{BSG}(S,T)$, the player ALGORITHM is trying to pack all three bins with capacity strictly below S, which we can think of as capacity $S-1$. Therefore, we set $S'=S-1$ and use S' in our definitions.

We restate the good situations GS1 to GS5 for an instance of $\mathrm{BSG}(S',T)$ for general S',T with $\alpha=S'-T$ satisfying $\alpha\geq T/3$, while in Section 2.1 we formulate the good situations only for $\mathrm{BSG}(22,18)$. The proofs are however equivalent and we omit them.

Good Situation 1 *For a bin configuration $(a,b,c,\mathscr{I})$ 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 *For a bin configuration $(a,b,c,\mathscr{I})$ 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 *For a bin configuration $(a,b,c,\mathscr{I})$ such that $a\in[\frac{3}{2}(T-\alpha),S']$ and either* (i) *$c\geq\alpha$ 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 *For a bin configuration $(a,b,c,\mathscr{I})$ such that*

$$a+b\geq\frac{3}{2}(T-\alpha)+c/2, b<T-2\alpha \text{ and } c<T-2\alpha,$$

there exists an online algorithm that packs all remaining items into three bins of capacity S'. □

Good Situation 5 *Suppose that we are given a bin configuration $(a,b,c,\mathscr{I})$ such that an item i with $s(i)>\alpha$ is present in the multiset $\mathscr{I}$ 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'.* □

3.3 Results

Table 1 summarizes our results. The paper of Gabay, Brauner and Kotov [11] 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 $1/41$.

3.4 Lower bound for four and five bins

The notion of bin configuration (Definition 3.1) as well as the core 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 3.2 is now $\mathscr{O}(|\mathscr{I}|\cdot T^m)$.

One notion that does not generalize very well are the good situations of Section 3.2.3. For instance, the formula $a+b\geq(m-1)T-\alpha$ 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 $m>3$.

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.

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. In order to normalize the speed of the program, the algorithm only checked for a lower bound and did not generate the entire tree in the **Yes** cases.

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

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.

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.

3.5 Verification of the results

We give a compact representation of our game tree for the lower bound of 45/33 for $m = 3$, which can be found in Appendix C. In particular, Figure 2 contains the top part of the tree while each of Figures 3, 4 and 5 contains one large branch of the tree. The fully expanded representation, as given by our algorithm, is a tree on 11053 vertices.

For our lower bounds of 19/14 for $m = 4$ and $m = 5$, the sheer size of the tree (e.g. 4665 vertices for $m = 5$) prevents us from presenting the game tree in its entirety. We therefore include the lower bound along with the implementations, publishing it online at `http://github.com/bohm/binstretch/`.

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 could be manageable. The verifier is available along with the rest of the programs and data.

References

1. S. Albers and M. Hellwig. Semi-online scheduling revisited. *Theor. Comput. Sci.*, 443:1–9, 2012.
2. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 44:486–504, 1997.
3. Y. Azar and O. Regev. On-line bin-stretching. In *Randomization and Approximation Techniques in Computer Science*, pages 71–81. Springer, 1998.
4. Y. Azar and O. Regev. On-line bin-stretching. *Theor. Comput. Sci.* 268(1):17–41, 2001.
5. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.
6. M. Böhm. Lower Bounds for Online Bin Stretching with Several Bins. Student Research Forum Papers and Posters at SOFSEM 2016, CEUR WP Vol-1548, 2016.
7. M. Böhm, J. Sgall, R. van Stee, and P. Veselý. Better algorithms for online bin stretching. In Approximation and Online Algorithms (pp. 23-34). Springer International Publishing, 2014.
8. M. Böhm, J. Sgall, R. van Stee, and P. Veselý. A Two-Phase Algorithm for Bin Stretching with Stretching Factor 1.5. ArXiv preprint arXiv:1601.08111v2, 2016.
9. E. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin Packing Approximation Algorithms: Survey and Classification, In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer New York, 2013.
10. T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53:504–522, 2009.
11. M. Gabay, N. Brauner, V. Kotov. Computing lower bounds for semi-online optimization problems: Application to the bin stretching problem. HAL preprint hal-00921663, version 2, 2013.
12. M. Gabay, N. Brauner, V. Kotov. Improved Lower Bounds for the Online Bin Stretching Problem. HAL preprint hal-00921663, version 3, 2015.
13. M. Gabay, V. Kotov, N. Brauner. Semi-online bin stretching with bunch techniques. HAL preprint hal-00869858, 2013.
14. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.
15. D. Johnson. *Near-optimal Bin Packing Algorithms*. Massachusetts Institute of Technology, project MAC. Massachusetts Institute of Technology, 1973.
16. H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013.
17. H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.
18. J. Ullman. The Performance of a Memory Allocation Algorithm. *Technical Report 100*, 1971.
19. Zobrist, Albert L. A new hashing method with application for game playing. ICCA journal 13.2: 69-73. 1970.

Appendix

A Linear programs used in the proofs

LP1

```
var A_j; var A_r; var A_x; var B_r; var B_x;
var j_2; var r; var x;
var eps >= 0;

maximize obj: eps;

AjLessThanAr: A_j <= A_r;
Bx: B_x = B_r + j_2;
eq1: A_j >= 3 + eps;
eq4: B_r + r >= 9 + A_r/2 + eps;
eq7: B_x + x + A_x <= 15 + r/2 - eps;
eq10: B_x <= A_x + x + (4 - r) - eps;
j2notSmall: j_2 >= 2 + eps;
rNotBig: r <= 4 - eps;
```

LP2

```
var A_j_2; var A_r; var A_x; var B_r; var B_x; var B_j;
var r; var j_2; var x; var j;
var eps >= 0;

maximize obj: eps;

Aj2LessThanAr: A_j_2 <= A_r;
ArLessThanAx: A_r <= A_x;
BxEqBr: B_x = B_r;
BrEq: B_r = B_j + j + j_2;
eq2: B_j + j >= 6 + eps;
eq3: A_j_2 + j_2 >= 6 + eps;
eq5: B_r + r + A_r <= 15 - eps;
eq7: B_x + x + A_x <= 15 + r/2 - eps;
eq12: B_x <= r + x + (4 - A_x) - eps;
rNotBig: r <= 4 - eps;
```

LP3

```
var A_r; var A_x; var B_j; var B_r; var B_x;
var r; var x; var j;
var eps >= 0;

maximize obj: eps;

BrEq: B_r = B_j + j;
BxEq: B_x = B_r;
eq4: B_r + r >= 9 + A_r/2 + eps;
eq8: B_x + x >= 9 + (A_x + r)/2 + eps;
eq9: A_x + x <= B_x + (4 - r) - eps;
BjLessThanAx: B_j <= A_x;
BjLessThanAr: B_j <= A_r;
jNotBig: j <= 4 - eps;
contradiction: B_j + r <= 6.8;
```

LP4

```
var A_x; var B_j; var B_x;
var r; var x; var j;
var eps >= 0;

maximize obj: eps;

BxEq: B_x = B_j + j;
eq8: B_x + x >= 9 + (A_x + r)/2 + eps;
eq11: r + x <= B_x + (4 - A_x) - eps;
BjPlus_rBound: B_j + r >= 6.8 + eps;
BjLessThanAx: B_j <= A_x;
contradiction: B_j + j <= 7.6;
```

LP5

```
var A_x; var B_j; var B_x;
var r; var x; var j;
var eps >= 0;

maximize obj: eps;

BjLessThanAx: B_j <= A_x;
BxEq: B_x = B_j + j;
eq7: B_x + x + A_x <= 15 + r/2 - eps;
eq11: r + x <= B_x + (4 - A_x) - eps;
jNotBig: j <= 4 - eps;
rBound: r >= 2.8 + eps;
contradiction: B_j + j + x >= 12.8;
```

B Formal proofs from Section 2.4.3

For completeness, we provide full proofs of lemmas in Section 2.4.3 which are shown using infeasible linear programs. Recall that these lemmas are parts of the proof of Claim 2.9. We use the notation introduced in Definition 2.10 and Equations (1)-(12).

Lemma B.1 (Lemma 2.11) *Assume that no good situation is reached until Step* (11)*. Then it holds that during Step* (10)*, only j is packed into B.*

Proof We first prove that no two additional items j_2, j_3 can be packed into B during Step (10). Assuming the contrary, we get $s(B_{\leftarrow j_2}) + s(j_2) + s(j_3) > 6+2+2 = 10$. With that load on B, we consider the packing at the end of Step (10), when the item x arrived. If $s(x) + s(C_{\leftarrow x}) < 9$, we get GS6 by placing x into C since $s(A) > 3$, so it must be true that $s(x) + s(C_{\leftarrow x}) > 9$, which means $s(x) > 5$. This is enough for us to place x into $B|_{22}$ (where it fits, otherwise we are in GS7) and reach GS3.

This contradiction gives us that at most one additional item j_2 can be packed into B during Step (10). We will now prove that even j_2 does not exist.

We split the analysis into two cases depending on which of j_2 and r arrives first.

Case 1. The item r is packed before j_2, meaning $s(B_{\leftarrow x}) = s(B_{\leftarrow r}) + s(j_2)$.

We start with inequalities (4), (7) and (10) in the following form:

$$9 + \frac{s(A_{\leftarrow r})}{2} < s(B_{\leftarrow r}) + s(r)$$

$$s(B_{\leftarrow r}) + s(j_2) + s(x) + s(A_{\leftarrow x}) < 15 + \frac{s(r)}{2}$$

$$s(B_{\leftarrow r}) + s(j_2) < s(A_{\leftarrow x}) + s(x) + \big(4 - s(r)\big)$$

We sum twice (4) with (7) and (10):

$$18 + s(A_{\leftarrow r}) + s(B_{\leftarrow r}) + s(j_2) + s(x) + s(A_{\leftarrow x}) + s(B_{\leftarrow r}) + s(j_2)$$
$$< 2s(B_{\leftarrow r}) + 2s(r) + 15 + \frac{s(r)}{2} + s(A_{\leftarrow x}) + s(x) + 4 - s(r)$$

$$s(A_{\leftarrow r}) + 2s(j_2) < \frac{3s(r)}{2} + 1$$

Using $s(A_{\leftarrow r}) \geq s(A_{\leftarrow j})$ (r arrives after j) with $s(A_{\leftarrow j}) > 3$ from Observation 2.4 and $s(r) < 4$ gives us:

$$3 + 2s(j_2) < 7$$
$$s(j_2) < 2$$

which is a contradiction, since $s(A_{\leftarrow j_2}) < 4$ and j_2 did not fit into $A|_6$.

Case 2. In the remaining case, j_2 arrives before r, which means

$$s(B_{\leftarrow x}) = s(B_{\leftarrow r}) = s(B_{\leftarrow j}) + s(j) + s(j_2).$$

We start by summing (5) and (7). We get:

$$\begin{aligned} s(B_{\leftarrow r}) + s(B_{\leftarrow x}) + s(x) + s(r) & \\ + s(A_{\leftarrow x}) + s(A_{\leftarrow r}) &< 30 + \frac{s(r)}{2} \\ 2s(B_{\leftarrow j}) + 2s(j) + 2s(j_2) + s(x) + s(r) & \\ + s(A_{\leftarrow r}) + s(A_{\leftarrow x}) &< 30 + \frac{s(r)}{2}. \end{aligned} \tag{13}$$

Keeping (13) in mind for later use, we continue by considering (2), (3) and (12) in the following form:

$$s(B_{\leftarrow j}) + s(j) > 6 \tag{14}$$
$$s(A_{\leftarrow j_2}) + s(j_2) > 6 \tag{15}$$
$$s(r) + s(x) + (4 - s(A_{\leftarrow x})) > s(B_{\leftarrow x}) = s(B_{\leftarrow j}) + s(j) + s(j_2)$$

Summing the three inequalities gives us:

$$\begin{aligned} s(B_{\leftarrow j}) + s(j) + s(A_{\leftarrow j_2}) + s(j_2) & \\ + s(r) + s(x) + (4 - s(A_{\leftarrow x})) &> 12 + s(B_{\leftarrow j}) + s(j) + s(j_2) \\ s(r) + s(x) + (s(A_{\leftarrow j_2}) - s(A_{\leftarrow x})) &> 8 \\ s(r) + s(x) &> 8 \end{aligned} \tag{16}$$

Summing two times (14), two times (15) and once (16) gives us:

$$2s(B_{\leftarrow j}) + 2s(j) + 2s(j_2) + 2s(A_{\leftarrow j_2}) + s(r) + s(x) > 32. \tag{17}$$

Using $s(A_{\leftarrow j_2}) \leq s(A_{\leftarrow r}) \leq s(A_{\leftarrow x})$ (which is only true here in Case 2, where r arrived later) and recalling (13) along with (17), we get $30 + s(r)/2 > 32$ and $s(r) > 4$, which is a contradiction with r fitting into $C|_4$. □

Lemma B.2 (Lemma 2.12) *Suppose that e and r are items as described in Definition 2.10 and suppose also that no good situation was reached during Step* (10) *of the algorithm* EVASIVE*. Then,* $s(e) + s(r) \geq s(B_{\leftarrow j}) + s(r) > 6.8$.

Proof First of all, it is important to note that the item e may be packed on A or on B. Since either $B_{\leftarrow j}$, or $A_{\leftarrow j}$ contains solely e by Observation 2.4, we get that either $s(B_{\leftarrow j}) = s(e)$, or $s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) = s(e)$. Thus it is sufficient to prove $s(B_{\leftarrow j}) + s(r) > 6.8$.

We start the proof of $s(B_{\leftarrow j}) + s(r) > 6.8$ by restating (4), (8), and (9) in the following form:

$$\begin{aligned} s(B_{\leftarrow j}) + s(j) + s(r) &> 9 + \frac{s(A_{\leftarrow r})}{2} \\ s(B_{\leftarrow j}) + s(j) + s(x) &> 9 + \frac{s(A_{\leftarrow x}) + s(r)}{2} \\ s(B_{\leftarrow j}) + s(j) + (4 - s(r)) &> s(A_{\leftarrow x}) + s(x). \end{aligned}$$

Before summing up the inequalities, we multiply the first one by 8, the second by 2 and the third by 2. In total, we have:

$$\begin{aligned} 12s(B_{\leftarrow j}) + 12s(j) + 8 + 6s(r) + 2s(x) > 90 + 3s(A_{\leftarrow x}) + 4s(A_{\leftarrow r}) \\ + s(r) + 2s(x). \end{aligned}$$

We know that $s(B_{\leftarrow j}) \leq s(A_{\leftarrow x})$ and $s(B_{\leftarrow j}) \leq s(A_{\leftarrow r})$, allowing us to cancel out the terms:

$$5s(B_{\leftarrow j}) + 5s(r) + 12s(j) > 82.$$

Finally, using the bound $s(j) < 4$ and noting that $(82 - 48)/5 = 6.8$, we get

$$s(B_{\leftarrow j}) + s(r) > 6.8. \quad \square$$

Lemma B.3 (Lemma 2.13) *Suppose that e and j are items as described in Definition 2.10 and suppose also that no good situation was reached by the algorithm* EVASIVE*. Then,* $s(e) + s(j) \geq s(B_{\leftarrow j}) + s(j) > 7.6$.

Proof The same argument as in Lemma 2.12 gives us $s(e) + s(j) \geq s(B_{\leftarrow j}) + s(j)$. We therefore aim to prove $s(B_{\leftarrow j}) + s(j) > 7.6$. Summing up (8) and (11) and using $s(B_{\leftarrow x}) = s(B_{\leftarrow j}) + s(j)$, we get

$$2s(B_{\leftarrow j}) + 2s(j) + s(x) + 4 - s(A_{\leftarrow x}) > 9 + \frac{s(A_{\leftarrow x}) + s(r)}{2} + s(r) + s(x)$$

$$2s(B_{\leftarrow j}) + 2s(j) > 5 + \frac{3}{2}\big(s(A_{\leftarrow x}) + s(r)\big).$$

We now apply the bound $s(A_{\leftarrow x}) + s(r) \geq s(B_{\leftarrow j}) + s(r) > 6.8$, the second inequality being Lemma 2.12. We get:

$$2s(B_{\leftarrow j}) + 2s(j) > 5 + 10.2,$$

and finally $s(B_{\leftarrow j}) + s(j) > 7.6$, completing the proof. □

Lemma B.4 (Lemma 2.16) *Suppose the algorithm* EVASIVE *reaches no good situation during Step* (10)*. Then, after placing x into B in Step* (11)*, it holds that* $s(B) < 12.8$.

Proof As before, we will use our inequalities to derive the desired bound. As we have argued above, Lemma 2.12 gives us that $s(r) > 2.8$.

We sum up inequalities (7) and (11), getting:

$$\begin{aligned} s(B_{\leftarrow j}) + s(j) + 2s(x) + s(A_{\leftarrow x}) + s(r) < 15 + \frac{s(r)}{2} + s(B_{\leftarrow j}) \\ + s(j) + 4 - s(A_{\leftarrow x}) \\ 2s(x) + 2s(A_{\leftarrow x}) < 19 - \frac{s(r)}{2}. \\ s(x) + s(A_{\leftarrow x}) < 9.5 - \frac{s(r)}{4}. \end{aligned}$$

To finish the bound we need $s(B_{\leftarrow j}) \leq s(A_{\leftarrow j}) \leq s(A_{\leftarrow x})$ (this is true because we reorder the bins B, A in Step (2)), $s(r) > 2.8$ and $s(j) < 4$. Plugging them in, we get:

$$s(B) = s(B_{\leftarrow j}) + s(j) + s(x) \leq s(A_{\leftarrow x}) + s(j) + s(x)$$

$$< 9.5 - \frac{s(r)}{4} + s(j) < 9.5 - 0.7 + 4 < 12.8. \quad \square$$

C Lower bound of 45/33

The lower bound of 45/33 for three bins is presented as Figure 2, 3, 4 and 5.

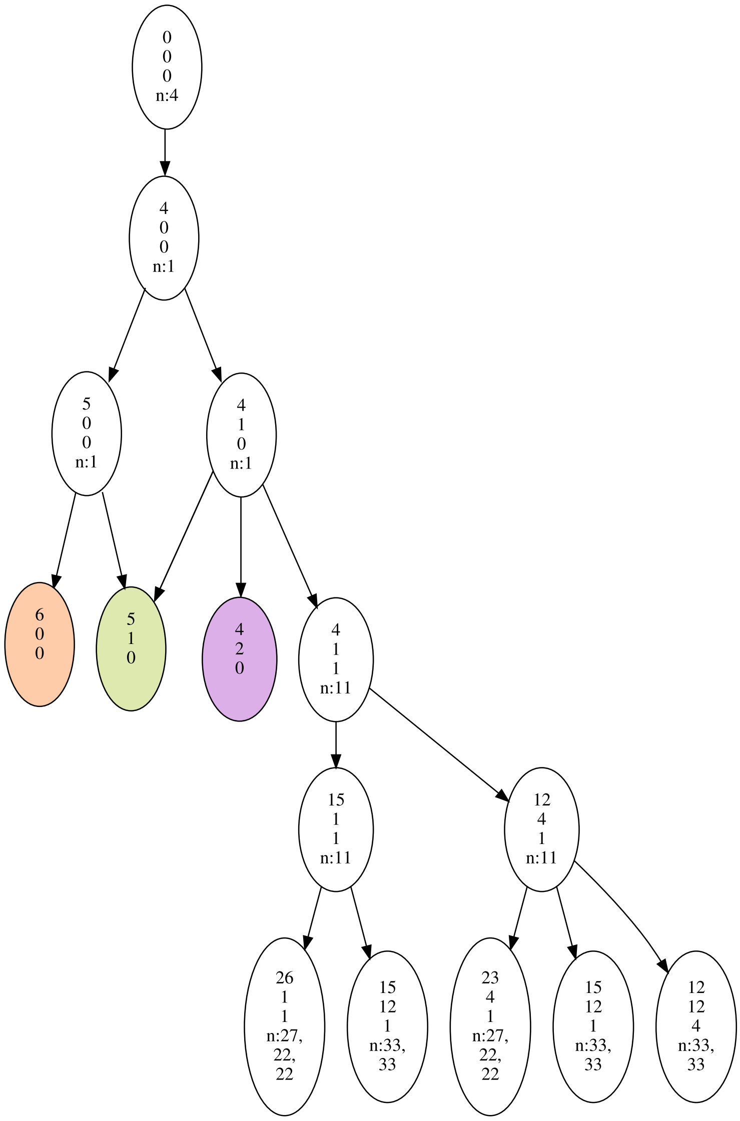

Fig. 2 The beginning moves of the $45/33$ lower bound, scaled so that $T = 33$ and $S = 45$. The vertices contain the current loads of all three bins, and a string `n: ` i with i being the next item presented by the ADVERSARY. If there are several numbers after `n:`, the items are presented in the given order, regardless of packing by the player ALGORITHM. The coloured vertices are expanded in later figures.

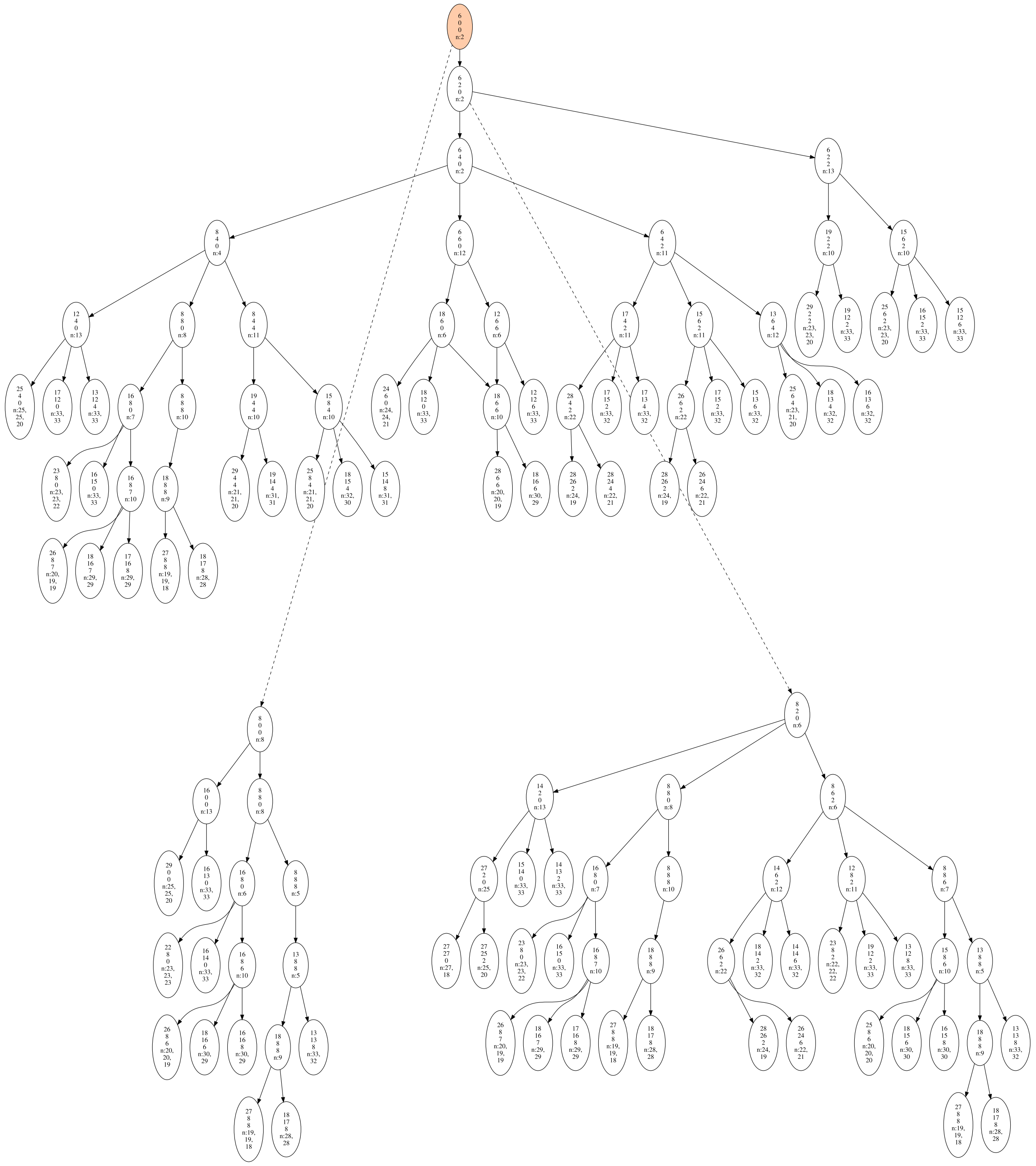

Fig. 3 Game tree for the lower bound of 45/33, starting with the bin configuration $(6,0,0,\{4,1,1\})$.

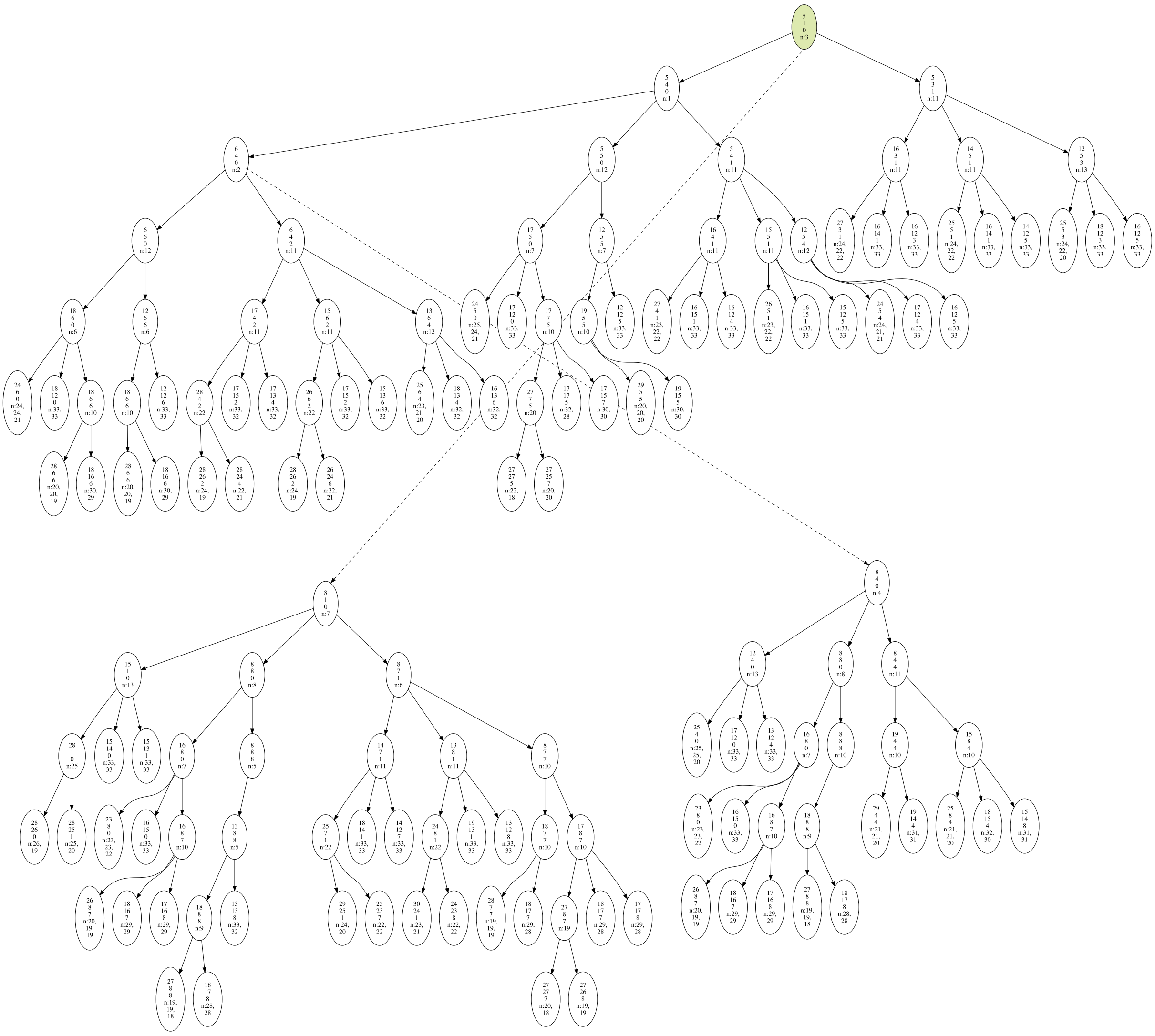

Fig. 4 Game tree for the lower bound of $45/33$, starting with the bin configuration $(5,1,0,\{4,1,1\})$.

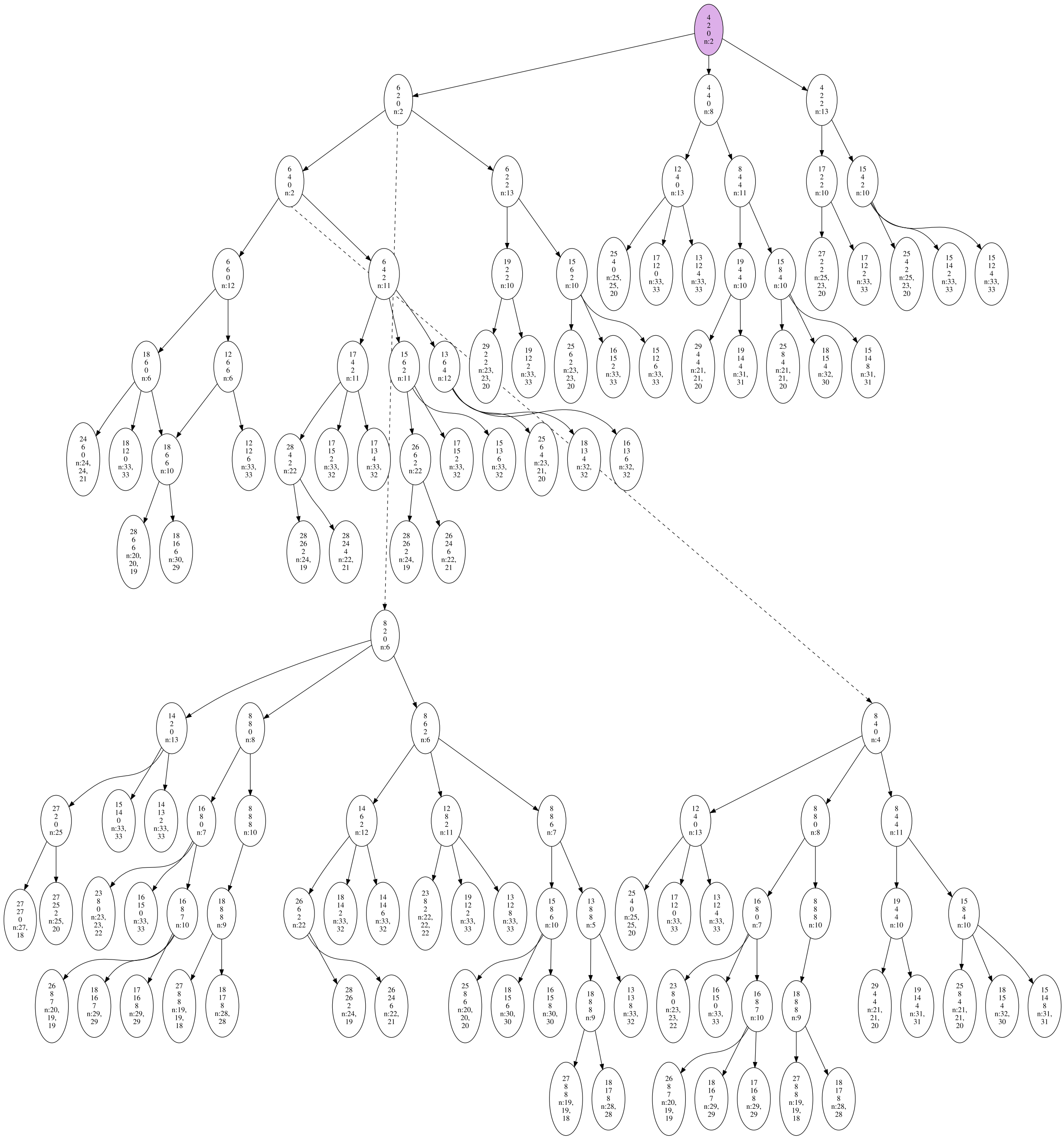

Fig. 5 Game tree for the lower bound of $45/33$, starting with the bin configuration $(4,2,0,\{4,1,1\})$.