Improved Lower Bounds for the Online Bin Stretching Problem

Michaël Gabay[a,*], Nadia Brauner[a], Vladimir Kotov[b]

[a] *Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France*
[b] *Belarusian State University, FPMI DMA department, 4 Nezavisimosti avenue 220030 Minsk Belarus*

Abstract

We use game theory techniques to automatically compute improved lower bounds on the competitive ratio for the bin stretching problem. Using these techniques, we improve the best lower bound for this problem to 19/14. We explain the technique and show that it can be generalized to compute lower bounds for any online or semi-online packing or scheduling problem.

Keywords: Bin Stretching, Scheduling, Online Algorithms, Lower Bounds

1. Introduction

In the online bin stretching problem, we are given a sequence of items defined by their weights $w_i \in [0;1]$. They all have to be packed into m bins with infinite capacities. We know in advance that all the items can be packed into m bins with unit size. The items are available and packed in the order of the sequence, without any knowledge on the number of remaining items and their weights. The value of a solution is equal to the size of the most stretched bin, which is the maximum between 1 and the size of the largest bin. An algorithm with *stretching factor* c for the online bin stretching problem is an online algorithm which successfully packs into m bins of size c, any sequence of items fitting into m unit sized bins. That is, for any instance I, the algorithm outputs a solution with value at most c. The aim is to find an algorithm having a stretching factor as small as possible.

This problem is equivalent to the scheduling problem $Pm|online-list|C_{max}$ where we additionally know that the optimal makespan is smaller than or equal to a given value C ($Pm|online-list, known-OPT|C_{max}$ is a subcase of this problem). The parameter $online-list$ means that, as soon as a job is presented, all its characteristics are known (its processing time in our case) and this job has to be scheduled before the next job is seen. The reader can refer to Borodin and El-Yaniv (1998); Fiat and Woeginger (1998) for more details about online algorithms and computation and to Pruhs et al. (2004) for online scheduling problems.

The bin stretching problem has been introduced by Azar and Regev (2001). They proposed an algorithm of stretching factor 1.625 and proved that 4/3 is the optimal stretching factor with two bins. Other algorithms with improved stretching factor have then been proposed by Kellerer and Kotov (2013); Gabay et al. (2013, 2015); Böhm et al. (2015) who respectively proposed algorithms with stretching factors $11/7 \approx 1.5714$, $26/17 \approx 1.5294$ and then 1.5. The best known upper bound with 3 bins is 1.375 and is due to Böhm et al. (2015).

The upper bound on the competitive ratio (the stretching factor) for this problem has been improved while, in the meantime, the best known lower bound remained the same: 4/3. In this paper, we present a computational approach to derive improved lower bounds for this problem. We use this approach to obtain a new lower bound with value $19/14 \approx 1.3571$. This lower bound has been proven for instances with 3 or 4

*Corresponding author
Email addresses: michael.gabay@g-scop.grenoble-inp.fr (Michaël Gabay), nadia.brauner@g-scop.grenoble-inp.fr (Nadia Brauner), kotovvm@bsu.by (Vladimir Kotov)

bins, leaving a gap of $1/56 \approx 0.018$ between the best known lower bound and upper bound for this problem with 3 bins and $1/7 \approx 0.143$ with 4 or more bins.

In the following section, we define worst-case competitive analysis and present the classical $4/3$ lower bound.

1.1. A lower bound

An online algorithm A is *c-competitive* if, for any instance I, algorithm A provides a solution with value at most c times greater than the optimal value, *i.e.* for all instance I, we have $A(I) \leq c \times OPT(I)$. For the bin stretching problem, this yields $A(I) \leq c$ (we are guaranteed that $OPT(I) = 1$).

Our objective is to improve lower bounds on c for a given problem. Ultimately, the aim is to find the smallest competitive ratio c^* among all online algorithms for the problem. This corresponds to finding the largest value c^* such that for any online algorithm A, there exists an instance I for which $A(I) \geq c^* \times OPT(I)$.

We now present the classical online scheduling lower bound for makespan minimization, adapted to the bin stretching problem. Consider the problem with 2 bins ($m = 2$) and the two following sequences of items in the input:

$$\pi = \left(\frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}\right) \qquad \pi' = \left(\frac{1}{3}, \frac{1}{3}, 1\right)$$

Obviously, both of these sequences of items can be packed into two unit sized bins. Consider a c-competitive deterministic online algorithm A for the bin stretching problem. Algorithm A must pack both of these sequences of items with stretching factor at most c.

Either A packs both of the first two items, of size $\frac{1}{3}, \frac{1}{3}$, in the same bin or in different bins. In the first case, with the sequence π, the smallest bin is filled to at least $4/3$, hence $c \geq \frac{4}{3}$. Otherwise, with sequence π', the smallest bin is filled to at least $4/3$, hence $c \geq \frac{4}{3}$. In both cases, $c \geq \frac{4}{3}$. Therefore, the stretching factor of any online algorithm is greater than or equal to $\frac{4}{3}$.

Azar and Regev (2001) generalized this bound to any number of bins. This bound, however, has not been improved ever since. We remark that the lower bound from Azar and Regev (2001) can be extended to prove a lower bound of $7/6$ for any randomized algorithms for the bin stretching and with any number of bins larger than or equal to two. This bound can be obtained by applying a uniform probability distribution on the two inputs considered by the authors.

Our aim is to improve the $4/3$ lower bound. Obviously, one cannot work with all possible algorithms and instances. Yet, in order to prove that a lower bound is valid, it has to be proven for all deterministic algorithms. We remark that on a given input, considering all assignments for all items is the same as considering all algorithms. In the following, we model the problem of finding lower bounds as a game and restrict the choices of the adversary. This restriction limits the set of considered instances.

1.2. Contribution

We derive a new worst-case lower bound, with value $19/14 \approx 1.3571$. In order to obtain this bound, we model the problem as a request-answer game against an adaptive off-line adversary (Ben-David et al., 1994). That is, the problem is modeled as a two-player, zero-sum game. Then, we use the so-called adversary method in which a malicious, omnipotent, adversary is playing against the algorithm to derive improved lower bounds. In online scheduling literature, layering techniques are often used to derive lower bounds for deterministic algorithms, see e.g. Albers (1999); Bartal et al. (1994); Rudin and Chandrasekaran (2003). However, since the optimum is known in advance in the bin stretching problem, this approach is very unlikely to work. We use an automated approach based on the minimax algorithm (Neumann, 1928), with alpha-beta pruning (Pearl, 1982) to solve the game where the adversary has restricted choices on the weights of the items. Moreover, to comply with the known feasibility of the corresponding bin packing problem with unit sized bins, we use constraint programming to compute feasible decisions of the adversary.

The algorithm outputs a decision tree as a proof. All decisions of the adversary are provided in this tree, for all decisions of any algorithm. The proof for the $19/14$ lower bound with 3 bins is provided in Appendix A. The proof with 4 bins is not included in this paper as it is much larger and do not seem to provide more insights on a possible structure to prove the $19/14$ lower bound for any number of bins.

Similar approaches have already been applied to other problems (Gormley et al., 2000). This computational approach relies on several classical tools of computer science and combinatorial optimization and can be generalized and applied to any online or semi-online problem. In this paper, we demonstrate how we apply it to the bin stretching problem and how the different components are connected together.

1.3. Outline

In Section 2, we model the problem of finding lower bounds for bin stretching algorithms as a game. Then, in Section 3, we present the algorithm and cuts we use to solve this game and compute lower bounds. Further research directions are proposed in Section 4. Eventually, the proof of the lower bound with 3 bins is included in Appendix A.

2. The bin stretching game

We model the problem of finding lower bounds for the bin stretching problem as the following two-player, zero-sum infinite game:

BIN STRETCHING GAME

Player 1 chooses a positive integer m. Then, successively, until Player 1 chooses `Stop`:

1. Player 1 (the *adversary*) chooses a feasible weight defining an item or `Stop`.
2. Player 2 (the *algorithm*) selects an integer $i \in \{1, \dots, m\}$ and packs the item into the bin B_i.

The payoff of Player 1 is equal to $\max(1, \max_{i=1,\dots,m} w(B_i))$, where $w(B_i) = \sum_{j \in B_i} w_j$.

Let w_j be the weight selected by Player 1 on iteration j. The weight w_j is feasible if and only if the bin packing problem with m bins of unit capacities and items with weights $w_1, \dots, w_j$ is feasible. The bin packing problem is strongly $\mathcal{NP}$-hard (Garey and Johnson, 1979). However, we can consider that the adversary is an oracle and can easily compute this problem.

Additionally, this is a game with complete information which means that both players know all the decisions taken and recall the history of the game.

The payoff of Player 1 is c, the stretching factor, while the payoff of Player 2 is $-c$. This game is a minimax game where Player 1 aims at maximizing c while Player 2 aims at minimizing c. An algorithm for the bin stretching problem defines a behavior for Player 2. The worst-case competitive ratio of an algorithm is equal to the supremum of c when Player 2 acts according to the algorithm. The supremum on the payoff of Player 1 in this game is equal to the value c^*.

It is easy to see that this game is infinite since the adversary can provide the input $w_j = 1/2^j$, for $j = 1, \dots, \infty$. Hence, we cannot explore all feasible choices of the adversary unless we restrain them. To cope with this issue, we actually consider that Player 1 has the following behavior: at the beginning of a game, Player 1 chooses a positive integer C. Then, all the weights chosen by Player 1 are in $\{1/C, 2/C, \dots, 1\}$ (and he can choose `Stop` as well). Considering this subset of adversaries, the game is finite: Player 1 has at most mC choices before the game is over.

In order to prove that a value c is a lower bound on c^*, it is "sufficient" to show that for any algorithm, there is an instance such that the stretching factor of the algorithm is greater than or equal to c. We cannot consider all algorithms but, on a given instance, there is a finite number of decisions for Player 2 and considering all decisions is actually the same as considering all algorithms. Hence, we only need to show that, for any decision of Player 2, there is a sequence of decisions from Player 1 leading to a solution with value at least c. Figure 1 illustrates this for the 4/3 lower bound. All decisions from Player 2 are considered while only one decision for each branch is provided for Player 1.

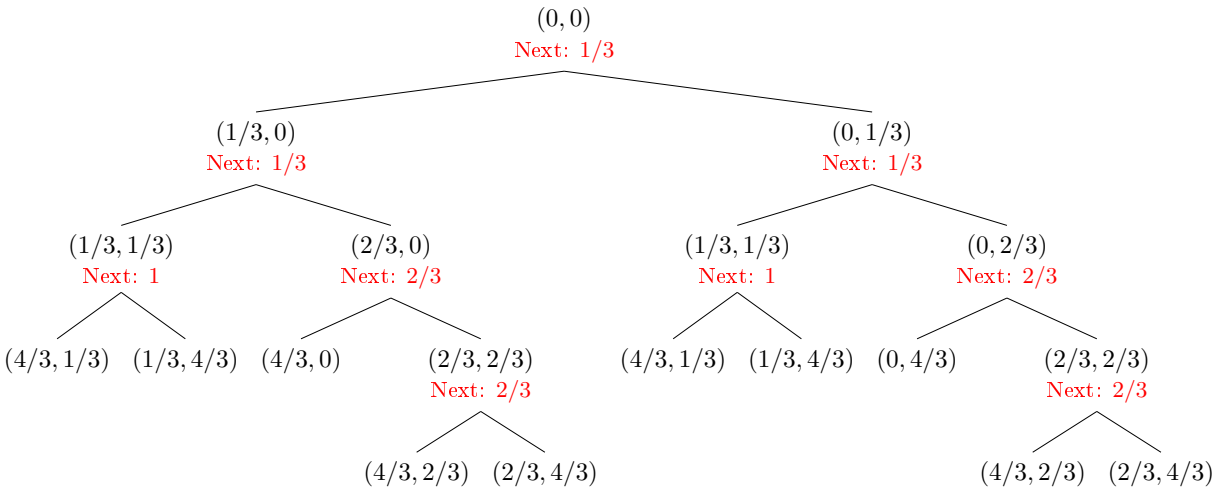

Figure 1: 4/3 lower bound decision tree. Player 1 decisions are the "Next: w_i". The pairs (w_1, w_2) are corresponding to the space used in the bins.

3. Implementation

In order to solve the game, that is, find a strategy for Player 1 maximizing c, we implement the minimax algorithm (depth-first search) for the game previously described. We apply the alpha-beta pruning with several additional cuts. Remark that considering unit capacities and weights in $\{1/C, 2/C, \ldots, 1\}$ is the same as considering the capacities of the bins to be C and weights in $\{1, \ldots, C\}$. Hence, we represent an item by an integer in $\{1, \ldots, C\}$ and a bin by a list of integers, corresponding to the items in the bin.

3.1. Decisions on item weights and assignments

In order to decide whether an item can be proposed by the adversary, we use lower and upper bounds on the corresponding bin packing problem, including the additional new item. Some of these bounds are described in the following paragraphs.

Let $w_1, \ldots, w_j$ be the weights of the items, sorted in non-increasing order. We verify that $\sum_{i=1}^{j} w_i \leq mC$ and $w_m + w_{m+1} \leq C$. Let $k = \max\{i | w_i > C/2\}$ ($k = 0$ if there are no such items) and $l = \max\{i | w_i = C/2\}$ ($l = k$ if there are no such items), we also ensure that $2k + l \leq 2m$. If any of the previous inequalities is not verified, then the weight is infeasible. At this step, we can also compute refined lower bounds such as L_2 and L_3 from Martello and Toth (1990). However, we choose to not compute these bounds since, in our experiments, subproblems are small and it is computationally more efficient to immediately solve these problems with an exact solver. With larger number of bins, one should consider computing these bounds before computing the exact solution of the problem.

Then, if the problem was not proven infeasible, we compute the best fit decreasing heuristic on the input. If it is feasible, then the new item is accepted. Otherwise, we need an exact approach to determine whether current item is feasible.

In our case, we use constraint programming to solve the bin packing problem. This choice was motivated by the small sizes of the problems that have to be solved. We did not implement a dedicated approach since the time spent checking feasibility is dominated by the time spent in the rest of the algorithm.

In general, for a semi-online problem, one can use any approach, including integer programming, branch and bound or any exact dedicated approach to determine whether a move for the adversary is feasible. We can also use heuristic approaches with the risk of not being able to find a lower bound because of a missed feasible move. However, when a move is validated, it has to be really feasible in order to ensure the correctness of the results of the algorithm.

Eventually, it is not necessary to verify feasibility for all items: once an item is proven feasible, all smaller items are feasible as well. Hence, by considering adversary choices by decreasing order of the weights of the items, we only have to find the first feasible item; then all other choices are smaller items, hence they are feasible.

3.2. Cuts

The size of the minimax tree is exponential in m and C. Hence, we have to find a way to cut branches in order to be able to compute optimal solutions of the restricted game. The first step to reduce the minimax tree is to break symmetries on the game: permutations on the bins are actually corresponding to identical solutions. Moreover, from Player 2 point of view, the items in the bins do not matter. Only the bin sizes matter. So, the configuration $((6,3),(4,5),\emptyset)$ is actually the same as $((7,1,1),\emptyset,(3,6))$. However, these two configuration are different from Player 1 point of view since he needs to ensure that the resulting bin packing problem is feasible. Yet, to both Player 1 and Player 2, the following configurations are equivalent: $((6,5,1,2),(7,7),\emptyset)$, $((6,1,7),(5,2,7),\emptyset)$. These two nodes can actually be described as: $(\{(0,1),(14,2)\},\{(1,1),(2,1),(6,1),(7,2)\})$ which is the same node to both players. The first set of pairs gives the weights of the bins and their multiplicities while the second set of pairs denotes the weights of the items and their multiplicities. All nodes having the same encoding are equivalent.

We use this encoding to represent a (partial) solution and we take advantage of it in two ways: when Player 2 packs an item, the number of edges to explore is equal to the cardinality of the first set of the pair, which is less than or equal to m. Moreover, we use memoization (Michie, 1968) (and compression) to store and recall the results of the nodes we have already computed and of bin packing problems which have already been non-trivially solved. Since we use an alpha-beta pruning, which is further described, we also have to store the values of α and β on the node, in order to be able to determine whether the value of a node shall be recomputed when it is recalled.

We apply an alpha-beta pruning to the minimax algorithm. The idea is to maintain a lower bound α and an upper bound β on the stretching factor. The pruning works as follows: on a maximizer node, once it is known that the solution of this node will be better than the solution of another node having the same parent (this parent is a minimizer), it is not necessary to explore any other choice. And similarly for minimizer nodes.

Since the adversary is computing a solution against all algorithms, we can consider several particular algorithms. Especially, we can consider the algorithm packing all remaining items into the currently smallest bin. We do not know the remaining items, but we know that the sum of their weights cannot exceed $mC - \sum_{k=1}^{j} w_k$. Let B_i be the smallest bin, if $w(B_i) + mC - \sum_{k=1}^{j} w_k \leq \alpha$ then we can immediately proceed to a β cut-off.

Additionally, we are aiming at strictly improving known lower bounds and we know that some competitive ratio can be achieved by some deterministic algorithms. So, we start the exploration with a lower bound which is equal to the best known lower bound ($\alpha = \lfloor C\tilde{c} \rfloor$, where $\tilde{c}$ is the best known lower bound on c^*) and an upper bound which is equal to the competitive ratio of the best algorithm: $\beta = 26C/17$ since 26/17 was the best upper bound available at the time of computation.

For most values, the lower bound will not be increased, so we improve the approach by dividing it into two steps: in the first step, we determine whether the lower bound can be improved. If so, in a second step, we determine the new best lower bound. Otherwise, we go on to the next value. Thus, we start with $\alpha = \lfloor C\tilde{c} \rfloor$ and $\beta = \lfloor C\tilde{c} \rfloor + 1$; such close values allow very early cut-offs. When the algorithm is over, the value is either α or β. In the first case, the lower bound cannot be improved for current values of m and C. It is over, we can try a new set of parameters m, C. In the other case, we know that $\frac{\lfloor C\tilde{c} \rfloor + 1}{C}$ is a new, strictly larger lower bound. We re-run the algorithm with $\beta = 26C/17$ to see if this new lower bound can be further improved.

3.3. Results

We implemented the algorithm in `Python` and used Choco (Jussien et al., 2008) as a constraint programming solver to solve bin packing problems (we also implemented approaches using integer programming).

The source code of our implementation is available online[1].

Running the program with parameters $m = 3$ and $C = 14$, we obtain the improved lower bound $19/14 \approx 1.357$. We backtracked the results and verified them manually. The proof is provided in Appendix A. With $m = 4$ and $C = 14$, the algorithm also finds the 19/14 lower bound and proves it. We suspect that 19/14 is a valid lower bound for any number of bins but could not go further with the algorithm because of the combinatorial explosion.

We used `PyPy` interpreter on a computer running Linux and equipped with an Intel Core i7-2600K Processor (clock speed 3.40GHz) and 8GB of RAM to compute lower bounds with our algorithm. Some experimental results are presented in Table 1. Using our approach, we were able to compute the results for $m = 3$ and C up to 20, and $m = 4$ and C up to 14. With larger values of m, we are only able to compute results with small C and we do not get improved lower bounds. For larger values of C, the number of nodes is too large. We remark that the limiting factor is the combinatorial explosion (see column #nodes, Table 1) and not any algorithmic factor. Optimizing the code or running it on faster computers would barely allow to compute solutions for the next values of C.

The results presented in Table 1 (except the last column) concern the single step approach, with pruning parameters initialized to $\alpha = \lfloor 4C/3 \rfloor$ and $\beta = 26C/17$. The last column gives the number of nodes in the first stage of the two-steps approach, with $\alpha = \lfloor C\tilde{c} \rfloor$ and $\beta = \lfloor C\tilde{c} \rfloor + 1$.

The column #calls corresponds to the number of times an item feasibility was verified. The column #exact is the number of calls to the exact method (that is when the item was not proven to be feasible or infeasible by a heuristic or a lower bound). The combinatorial explosion is very well illustrated in column #nodes where we can see that even with many efficient cuts, we cannot tackle much larger problems. Table 1 also shows that the time spent verifying items feasibility is negligible compared to the whole time spent. Time spent is approximately linear in the number of nodes.

Note that for the largest values of C ($m = 4$ and $C = 13, 14$), we limited the amount of memoized data (using an lru cache) and ran computations with the single step approach only. So the number of nodes displayed in the table is actually larger than the number of nodes that would be obtained with unlimited memory. For $m = 5$, we used a server with $32GB$ of RAM and also limited the number of memoized nodes (to 6×10^7 nodes).

In order to improve this algorithm, one could use a breadth-first search and for each depth, use a heuristic to select a sample of least promising nodes. Exploring these nodes in depth, will allow some early cut-offs. Another approach is to set $C = 1$ and select random weights in $]0; 1]$. Then, we can run the algorithm on many random samples of items, hopefully resulting in an improved lower bound. We ran several tests using random weights distributions but we did not obtain improved lower bounds with this approach.

4. Conclusion

By modeling the bin stretching problem as a game and solving this game with computer science techniques we provided a first improved lower bound and proved it is valid with both 3 and 4 bins. For the case with 3 bins, the new 19/14 lower bound reduces the gap between lower and upper bounds by more than a factor of two compared to the 4/3 lower bound.

Based on the tree, it is not obvious to find out whether this bound can be generalized to any number of bins $m \geq 5$. Yet, the bound is valid for both 3 and 4 bins so it is likely that it remains valid with more bins.

The approach can be generalized and applied to many other packing or scheduling, online or semi-online problems. Compared to layering techniques, for multiple bins problem, there is however a trade-off on the generality of the bound: the lower bound cannot easily be generalized to any number of bins.

Because of the initial knowledge that all items can be packed into m unit sized bins, there is little hope that layering techniques could work. Future research could focus on finding more general lower bounds. For instance, by trying to design a computational approach whose results could be generalized for all values of

[1] https://github.com/mgabay/Bin-Stretching-Lower-Bounds

m	C	c	*feasibility check*			*overall*		*first step*
			#calls	*#exact*	*time (s)*	*#nodes*	*time (s)*	*#nodes*
3	10	—	4,687	37	0.4	49,055	1.4	41,753
	11	—	14,802	141	1.2	168,380	3.4	141,176
	12	—	9,125	63	0.5	118,925	2.2	98,186
	13	—	32,538	209	1.1	458,183	5.8	384,052
	14⋆	**19/14**	82,868	644	2.2	1,240,619	14.0	286,845
	15	—	55,929	344	1.2	890,291	10.1	702,449
	16	—	196,835	1142	3.3	3,384,144	35.5	2,901,483
	17	23/17	207,133	1,804	4.0	3,728,386	40.8	1,620,468
	18	—	303,725	1,646	4.3	5,692,383	57.2	4,652,427
	19	—	1,045,692	4,958	21.0	21,262,246	1225.1	18,653,870
	20	27/20	977,992	6,191	21.9	20,283,070	1046.7	11,446,232
4	7	—	6,622	50	0.4	50,642	1.6	39,946
	8	—	28,099	182	1.1	254,344	4.5	193,474
	9	—	30,991	98	0.8	331,112	4.8	266,926
	10	—	127,063	721	2.6	1,442,281	19.5	1,106,147
	11	—	503,560	3114	7.5	6,365,822	81.7	5,195,618
	12	—	491,497	1974	5.8	6,718,232	89.7	5,158,805
	13	—	1,642,949	8103	19.8	24,277,322	344.6	—
	14⋆	**19/14**	4,139,364	≈18300	54.0	69,421,282	1503.4	—
5	14	?		>182900		>46,962,700,000	>1,095,060	—

Table 1: Numerical results on few inputs. Column 3, c is the best lower bound on the competitive ratio obtained for the instance. "—" means that the 4/3 lower bound was not improved. In the last column, "—" means that only the first step was computed (so the actual number of nodes in the first step is in column *overall #nodes*)

m. One could also consider reducing the search space by exploring the search tree for particular families of algorithms.

Another subject for further research is to find good distributions of randomized item weights. By imposing a structure on the distribution of the weights of the items and running the algorithm on many inputs it is maybe feasible to improve the lower bounds. A critical factor to improve lower bounds is that the chosen weights can sum up to 1 in many different ways. This is best achieved with fractions of the same integer but the whole range of numerators might not be needed.

Acknowledgments

This research has been partially supported by project ICS No 5379 and Belarusian BRFFI grant (Project F13K-078). The research of the first and the second author has been partially supported by the LabEx PERSYVAL-Lab (ANR–11-LABX-0025).

References

Albers, S., 1999. Better bounds for online scheduling. SIAM Journal on Computing 29 (2), 459–473.

Azar, Y., Regev, O., 2001. On-line bin-stretching. Theoretical Computer Science 268 (1), 17–41.

Bartal, Y., Karloff, H., Rabani, Y., 1994. A better lower bound for on-line scheduling. Information Processing Letters 50 (3), 113–116.

Ben-David, S., Borodin, A., Karp, R., Tardos, G., Wigderson, A., 1994. On the power of randomization in on-line algorithms. Algorithmica 11 (1), 2–14.

Borodin, A., El-Yaniv, R., 1998. Online computation and competitive analysis. Vol. 53. Cambridge University Press.

Böhm, M., Sgall, J., van Stee, R., Veselý, P., 2015. Better algorithms for online bin stretching. In: Bampis, E., Svensson, O. (Eds.), Approximation and Online Algorithms. Vol. 8952 of Lecture Notes in Computer Science. pp. 23–34.

Fiat, A., Woeginger, G. J., 1998. Online algorithms: The state of the art. Springer Berlin Heidelberg.

Gabay, M., Kotov, V., Brauner, N., 2013. Semi-online bin stretching with bunch techniques. Les Cahiers Leibniz 208, 1–10.

Gabay, M., Kotov, V., Brauner, N., 2015. Online bin stretching with bunch techniques. Theoretical Computer Science 602, 103 – 113.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. WH Freeman and Company, New York.

Gormley, T., Reingold, N., Torng, E., Westbrook, J., 2000. Generating adversaries for request-answer games. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, pp. 564–565.

Jussien, N., Rochart, G., Lorca, X., et al., 2008. Choco: an open source java constraint programming library. In: CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08). pp. 1–10.

Kellerer, H., Kotov, V., 2013. An efficient algorithm for bin stretching. Operations Research Letters 41 (4), 343–346.

Martello, S., Toth, P., 1990. Lower bounds and reduction procedures for the bin packing problem. Discrete Applied Mathematics 28 (1), 59–70.

Michie, D., 1968. Memo functions and machine learning. Nature 218 (5136), 19–22.

Neumann, J. v., 1928. Zur theorie der gesellschaftsspiele. Mathematische Annalen 100 (1), 295–320.

Pearl, J., 1982. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communications of the ACM 25 (8), 559–564.

Pruhs, K., Sgall, J., Torng, E., 2004. Online scheduling. In: Leung, J. Y. (Ed.), Handbook of scheduling: algorithms, models, and performance analysis. CRC Press.

Rudin, J., Chandrasekaran, R., 2003. Improved bounds for the online scheduling problem. SIAM Journal on Computing 32 (3), 717–735.

Appendix A. Proof of the lower bound

The following tree proves the 19/14 lower bound for the bin stretching problem with 3 bins. This lower bound was obtained using our algorithm with parameters $m = 3$ and $C = 14$.

In the proof, a single decision of the adversary is provided for each decision of the algorithm. We do not explore branches where the algorithm packs the item in a bin, making it larger than or equal to 19. Moreover, we stop exploring a branch when there is a feasible item making all algorithms fail. We denote these latter nodes by "`cut: Wmin+`w_j`>=UB`". We recall the input sequence on the leaves. The next items are not added to this sequence. For instance, for a leaf "`input: [2,1,7] / cut: Wmin + 3 >= UB`" the whole input sequence is (2,1,7,3).

In order to make the proof easier to read, we divide the tree in two levels: the first level is a root tree and the second level is a set of subtrees, one for each leaf of the root tree.

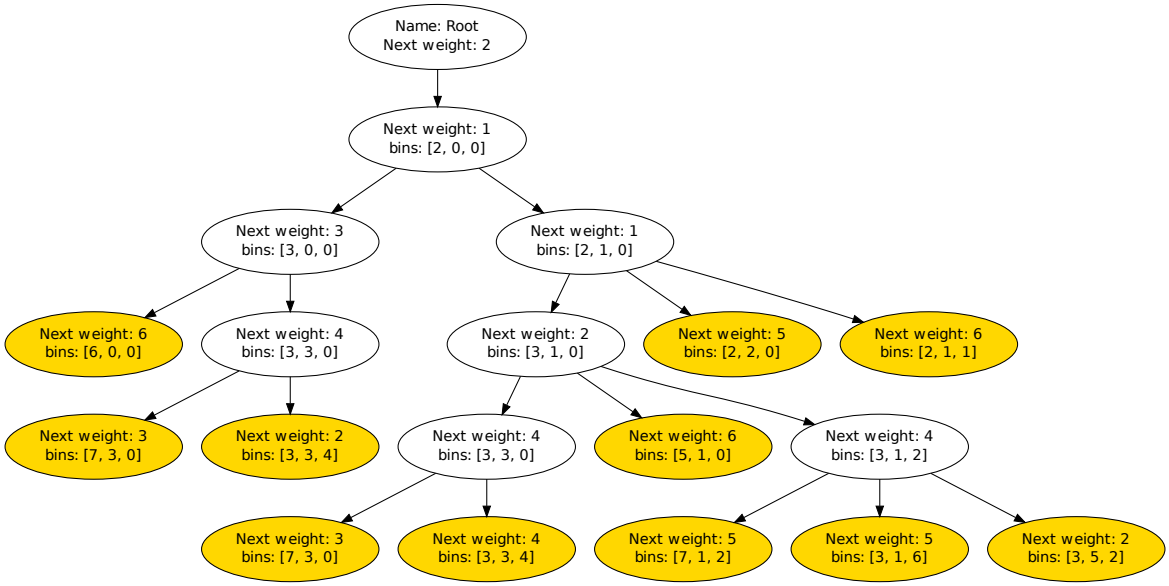

Figure A.2: Root tree

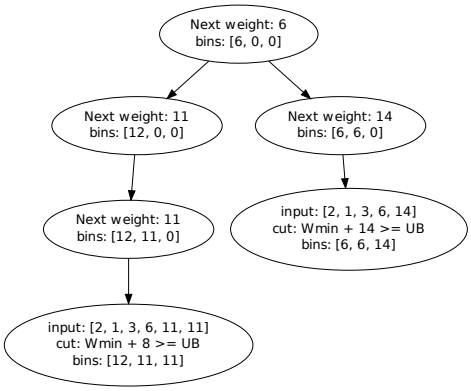

Figure A.3: Subtree 1, layout $[6, 0, 0]$, items (2,1,3).

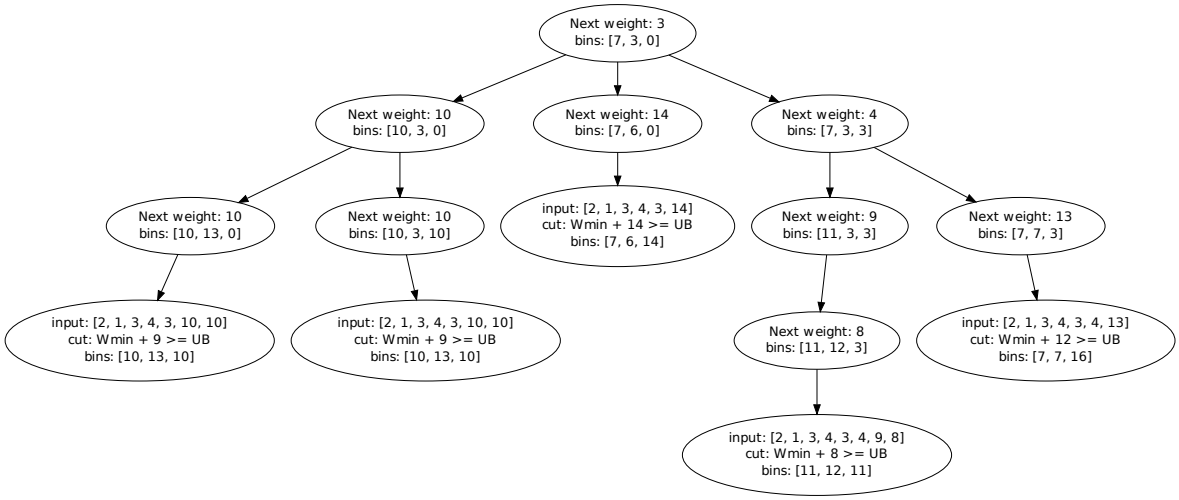

Figure A.4: Subtree 2, layout $[7, 3, 0]$, items (2,1,3,4).

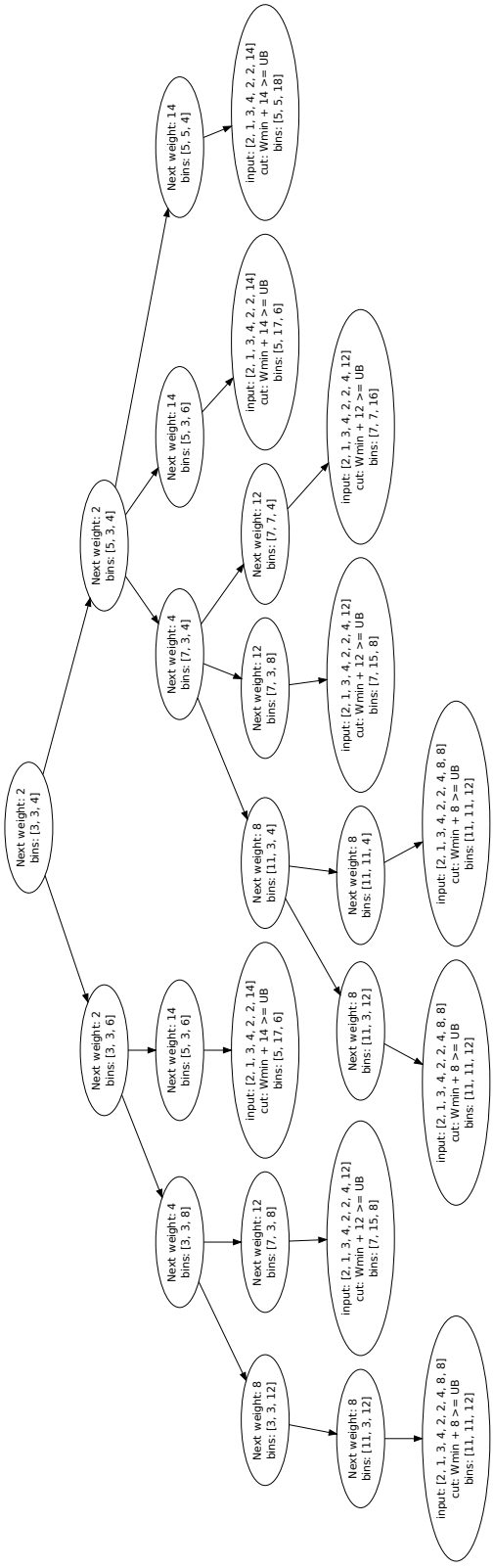

Figure A.5: Subtree 3, layout $[3, 3, 4]$, items (2,1,3,4).

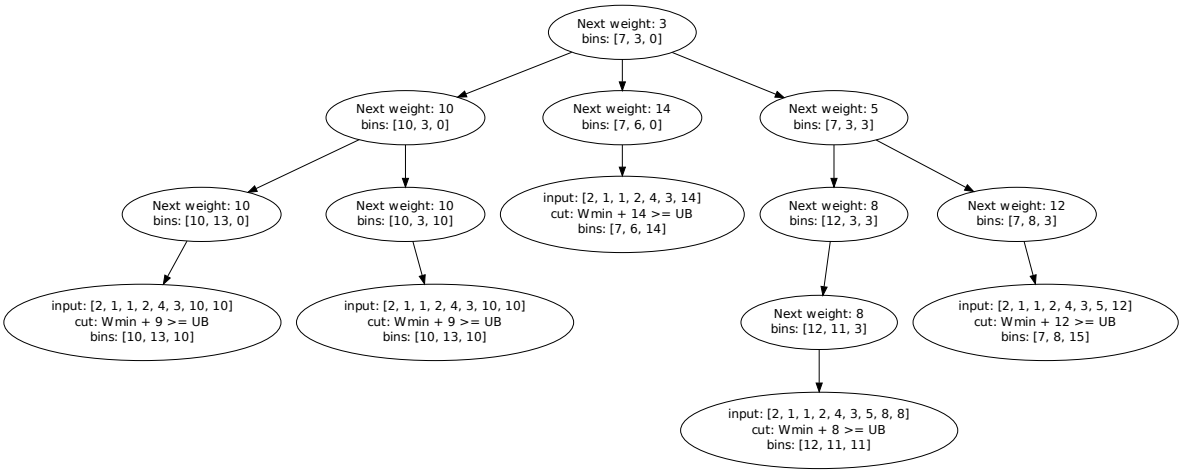

Figure A.6: Subtree 4, layout $[7, 3, 0]$, items (2,1,1,2,4).

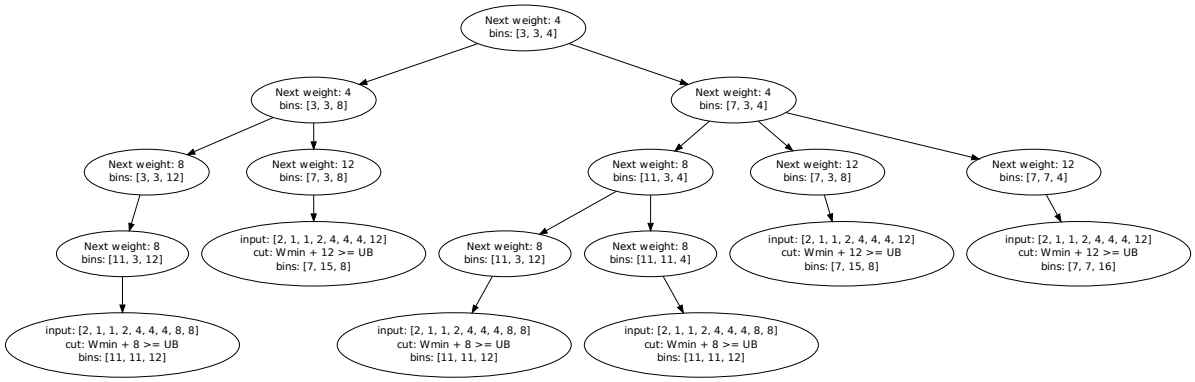

Figure A.7: Subtree 5, layout $[3, 3, 4]$, items (2,1,1,2,4).

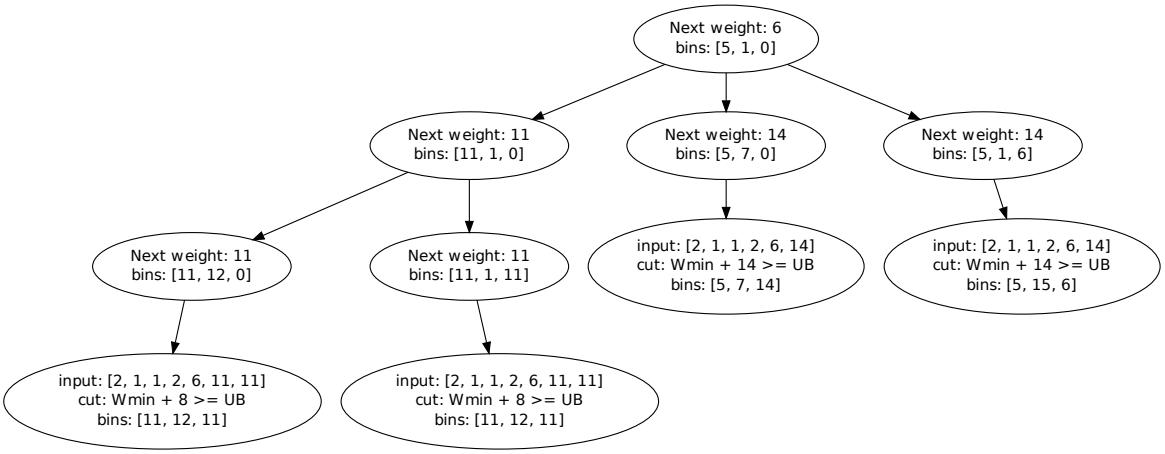

Figure A.8: Subtree 6, layout $[5, 1, 0]$, items (2,1,1,2).

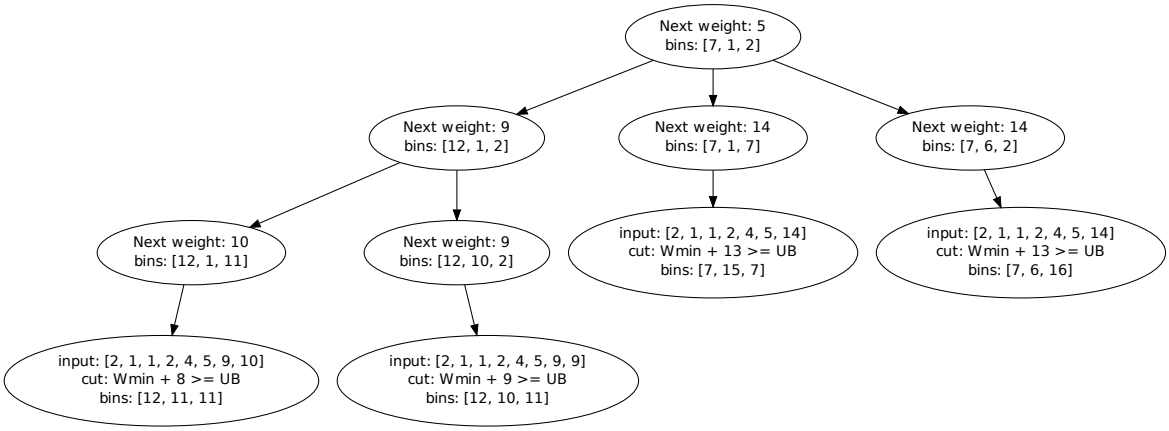

Figure A.9: Subtree 7, layout $[7, 1, 2]$, items (2,1,1,2,4).

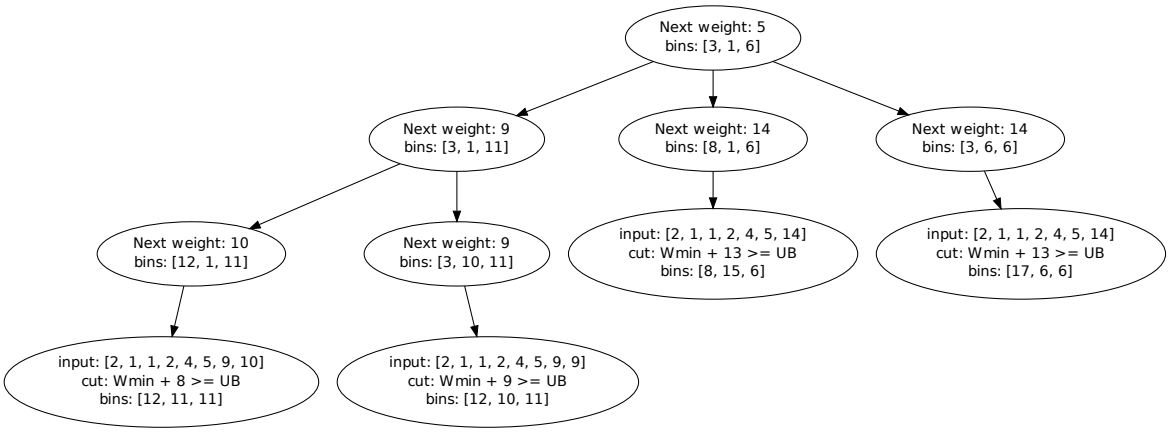

Figure A.10: Subtree 8, layout $[3, 1, 6]$, items (2,1,1,2,4).

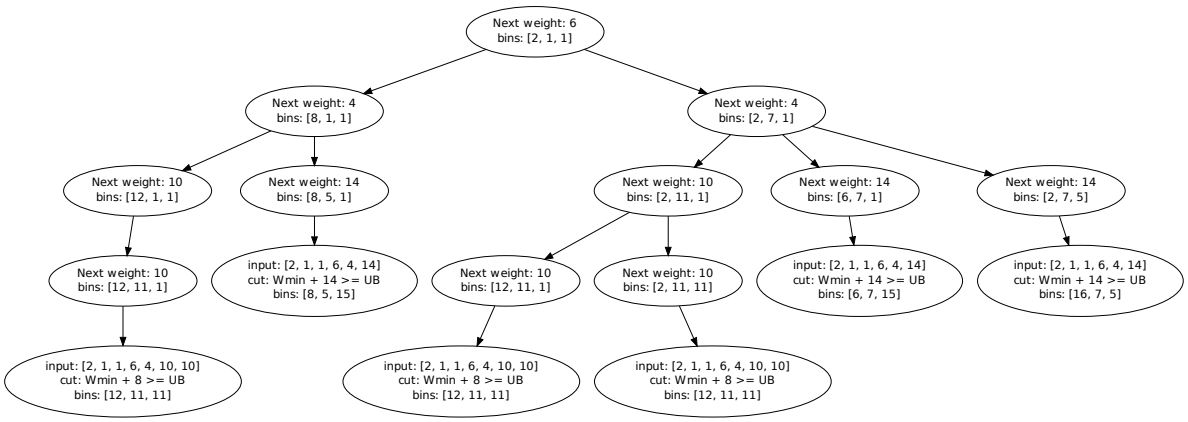

Figure A.11: Subtree 11, layout $[2, 1, 1]$, items (2,1,1).

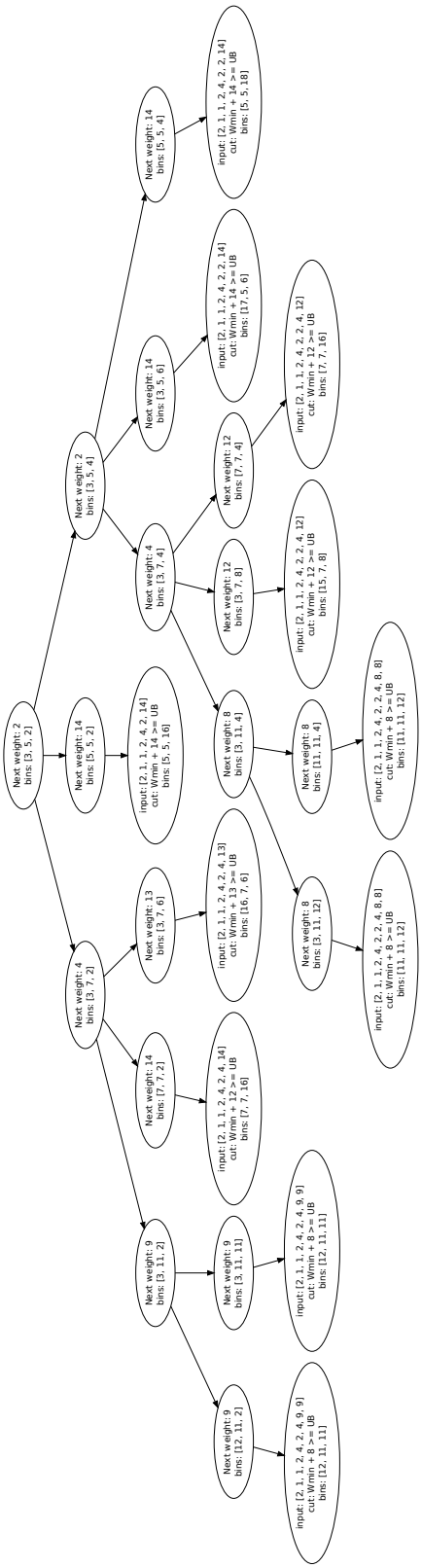

Figure A.12: Subtree 9, layout $[3, 5, 2]$, items (2,1,1,2,4).

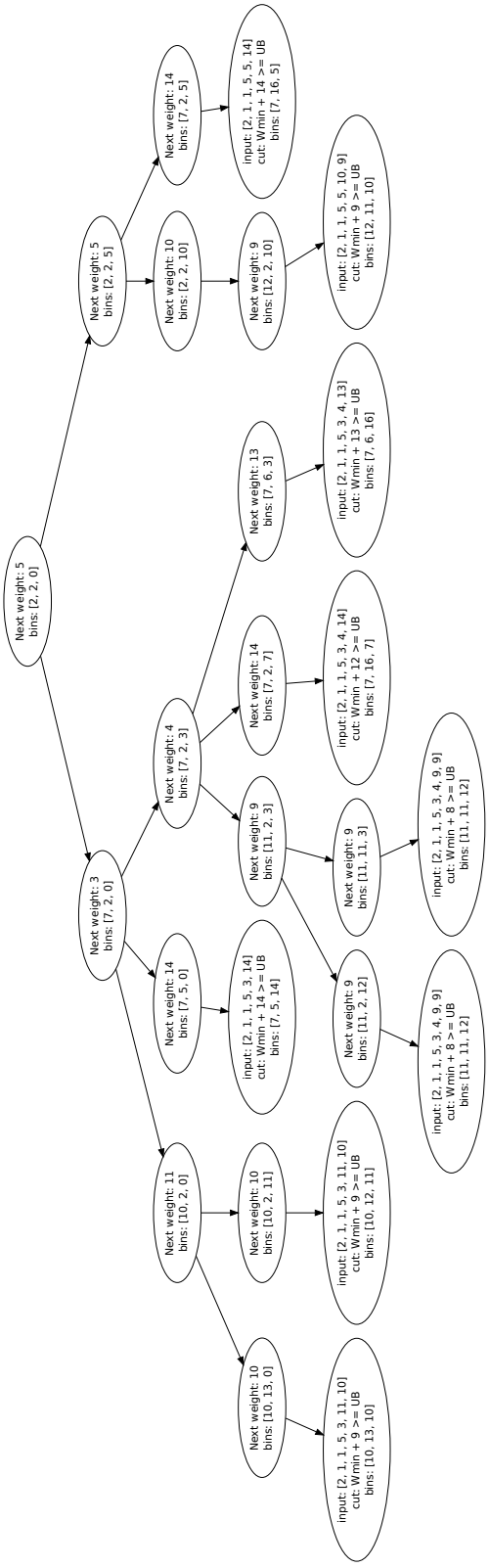

Figure A.13: Subtree 10, layout $[2, 2, 0]$, items (2,1,1).