Discovering and Certifying Lower Bounds for the Online Bin Stretching Problem

Martin Böhm

University of Wrocław
Wrocław, Poland

Bin Packing Seminar Series,
April 21, 2021
based on PhD thesis (2018) and work with B. Simon (2020)

Uniwersytet Wrocławski

SEMI-ONLINE SCHEDULING	
on m machines	
with known optimal makespan.	

Semi-online Scheduling	Online Bin Stretching
on m machines	with m bins
with known optimal makespan.	and known optimum.

Semi-online Scheduling	Online Bin Stretching
on m machines	with m bins
with known optimal makespan.	and known optimum.

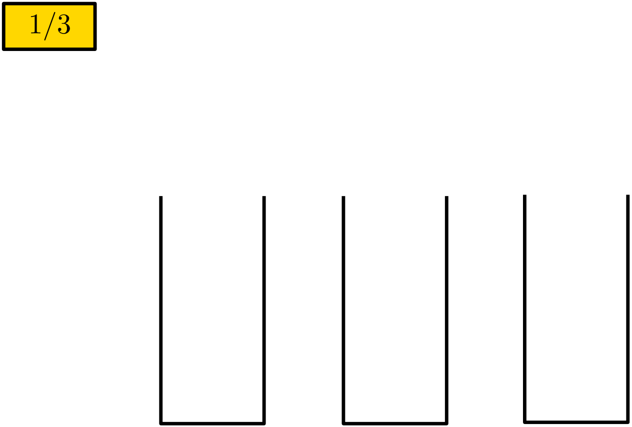
Online Bin Stretching

Input: A sequence of items, each with size in $[0, 1]$;
A number m – how many bins we can use.

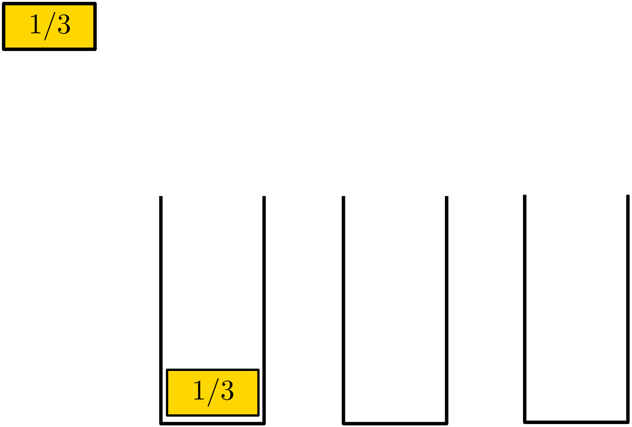
Guarantee: There exists an offline algorithm that can pack the sequence into m bins of capacity 1.

Goal: Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

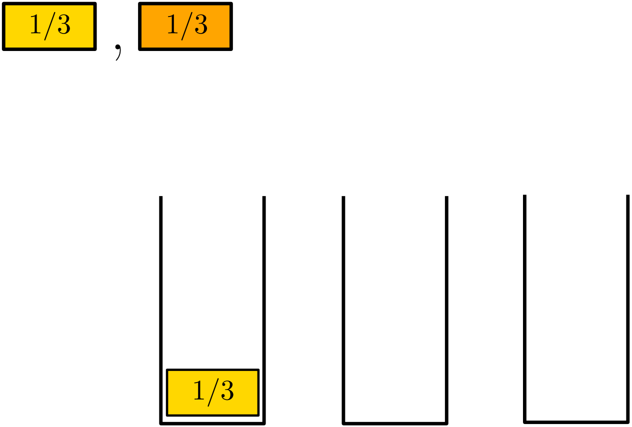
Example of Online Bin Stretching



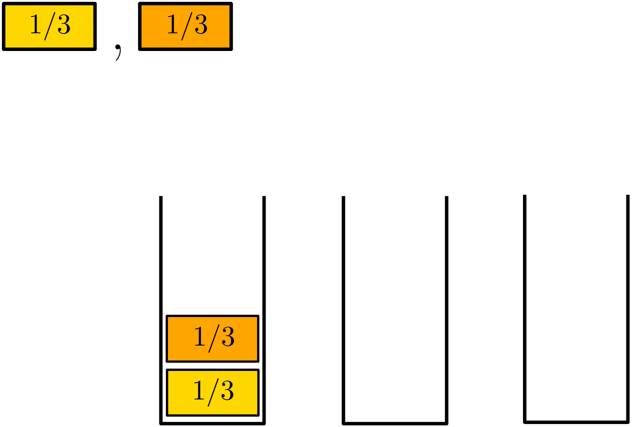
Example of Online Bin Stretching



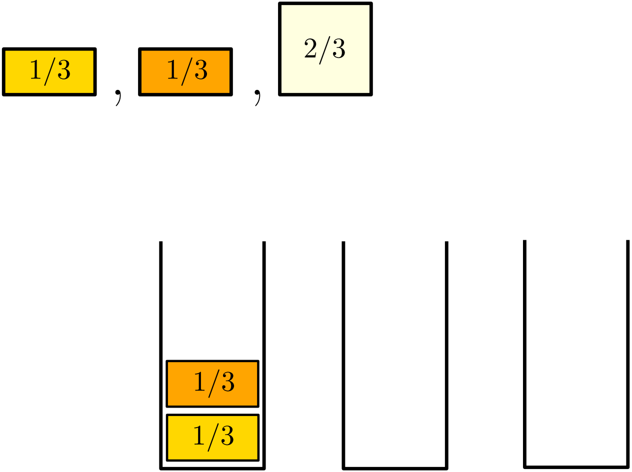
Example of Online Bin Stretching



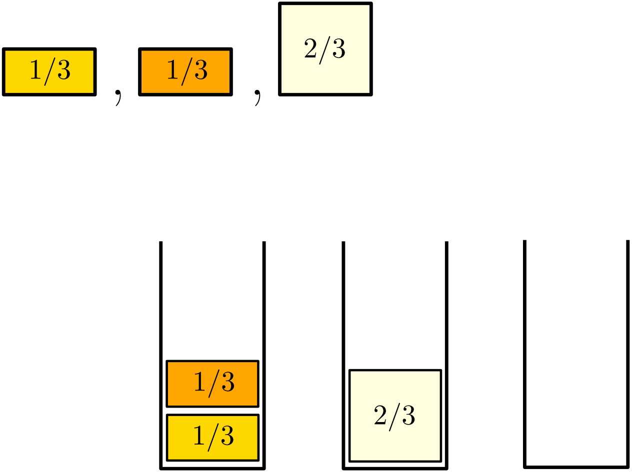
Example of Online Bin Stretching



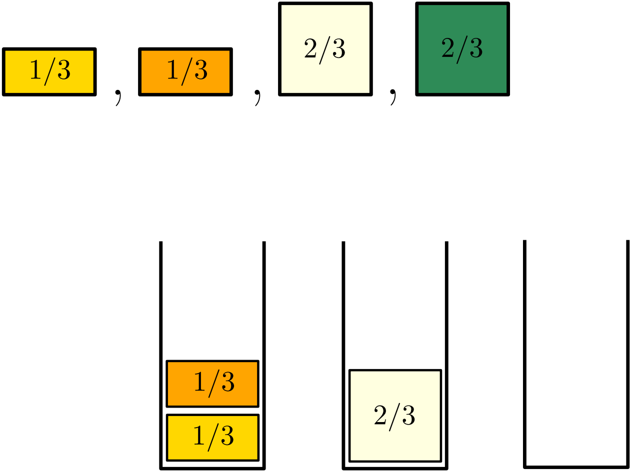
Example of Online Bin Stretching



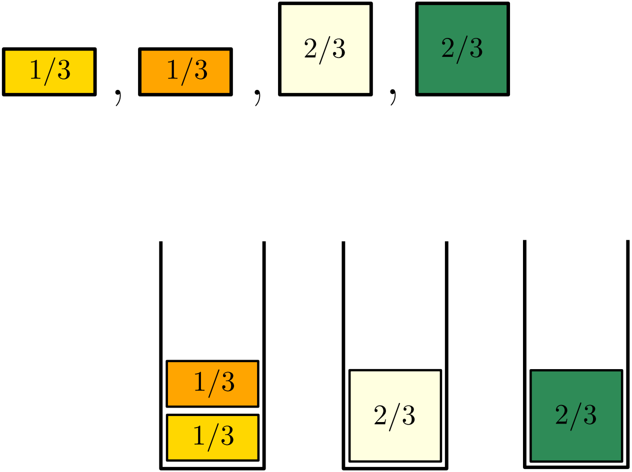
Example of Online Bin Stretching



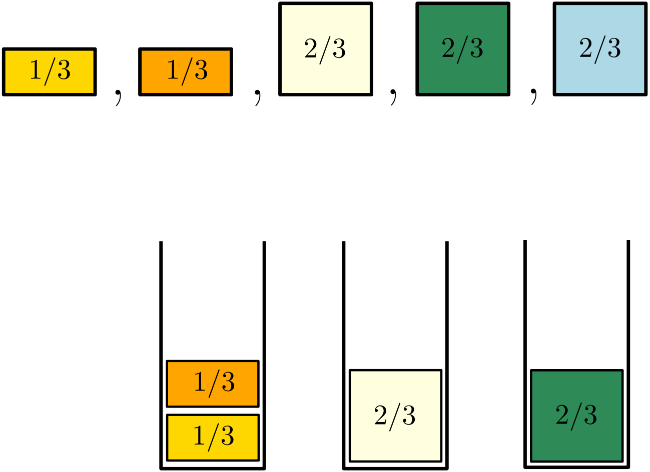
Example of Online Bin Stretching



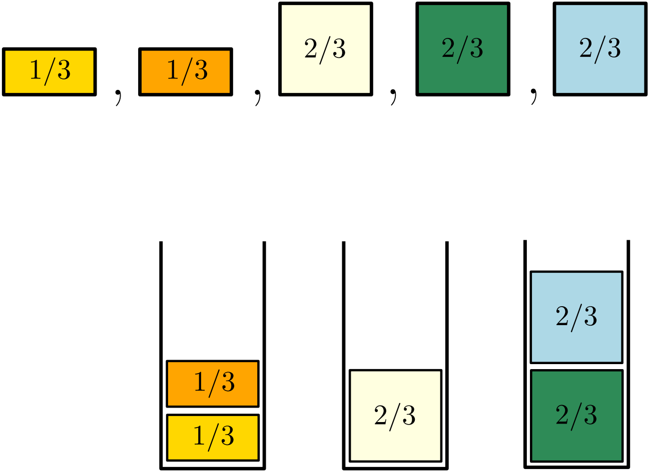
Example of Online Bin Stretching



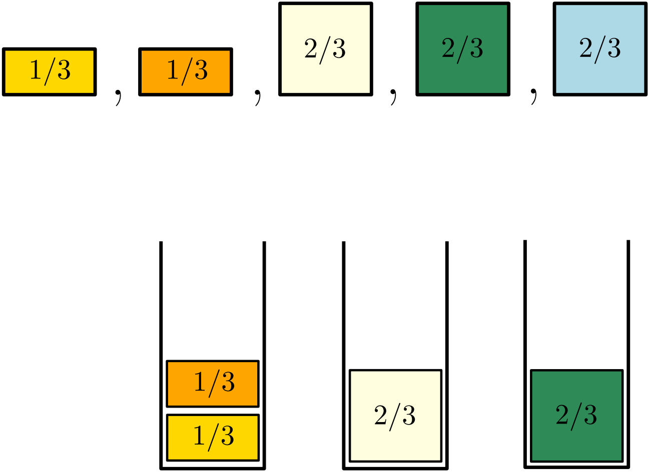
Example of Online Bin Stretching



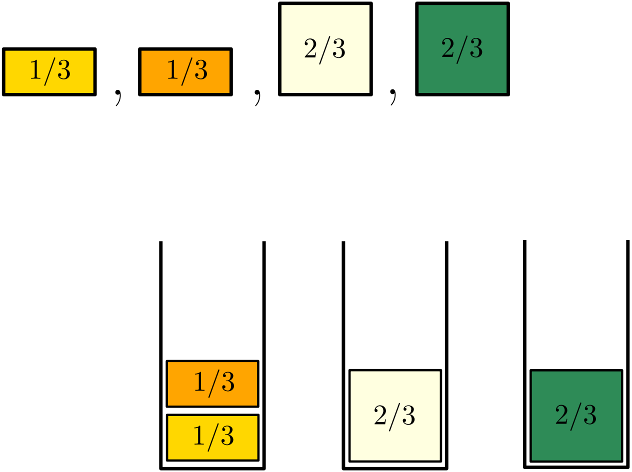
Example of Online Bin Stretching



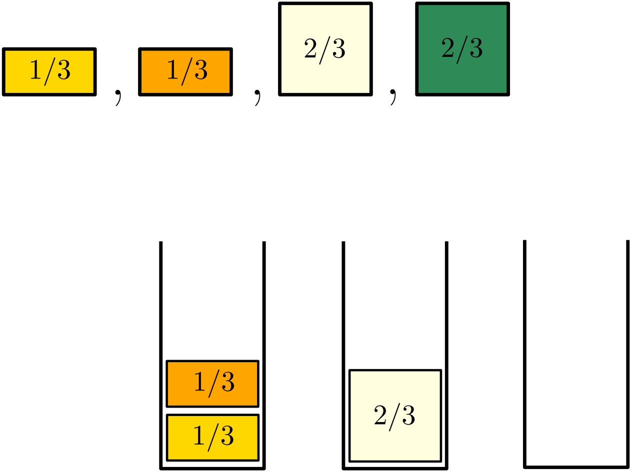
Example of Online Bin Stretching



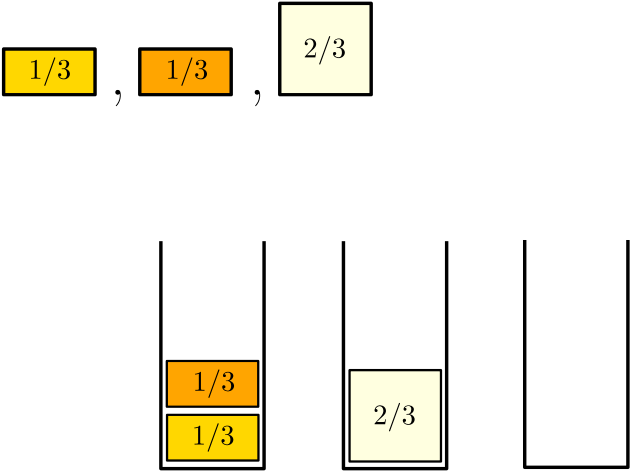
Example of Online Bin Stretching



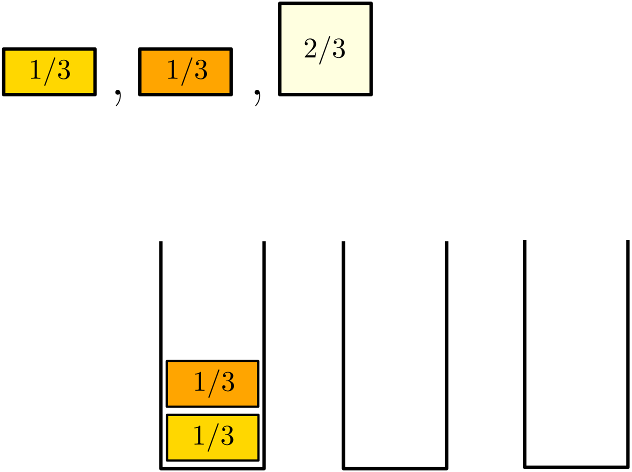
Example of Online Bin Stretching



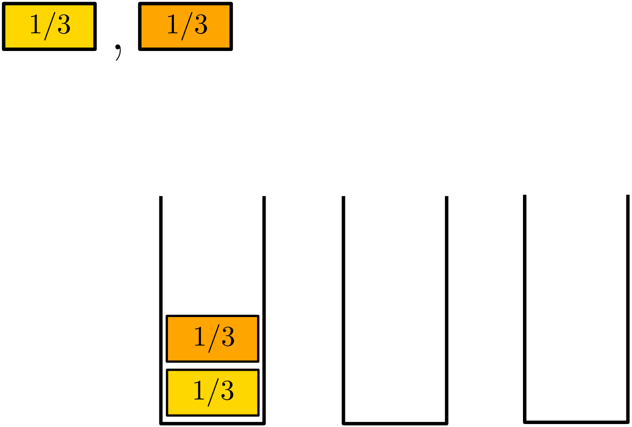
Example of Online Bin Stretching



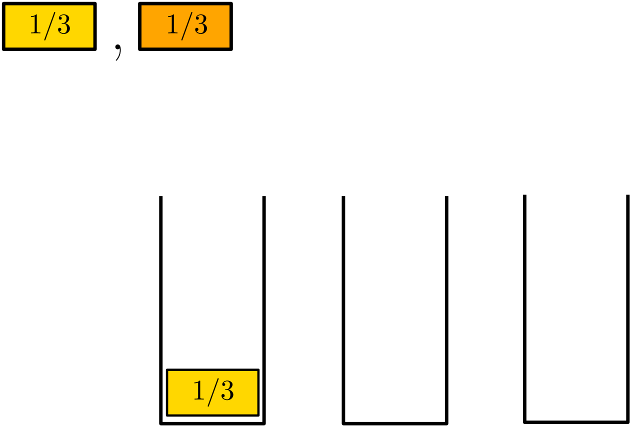
Example of Online Bin Stretching



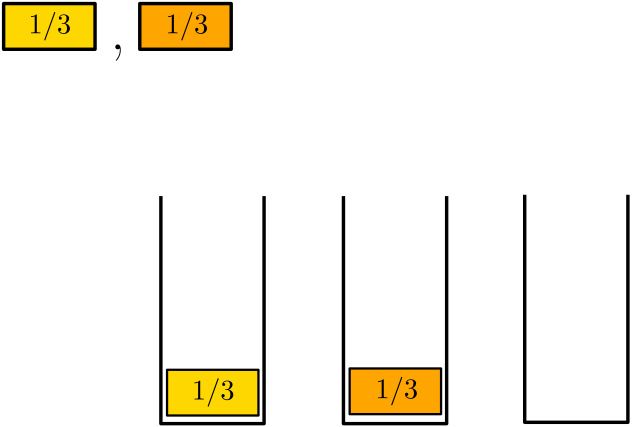
Example of Online Bin Stretching



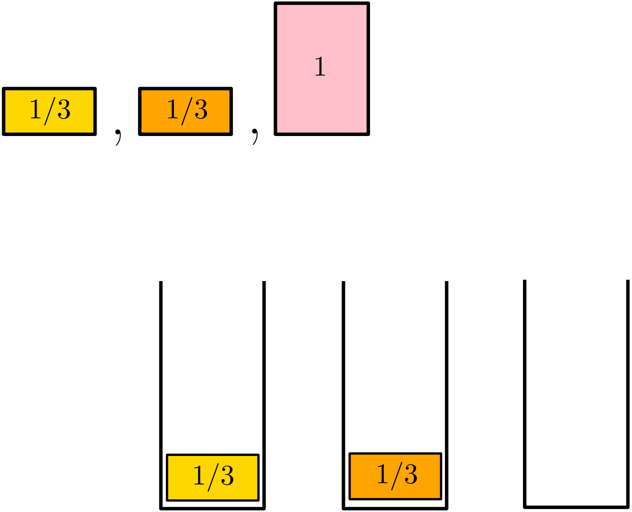
Example of Online Bin Stretching



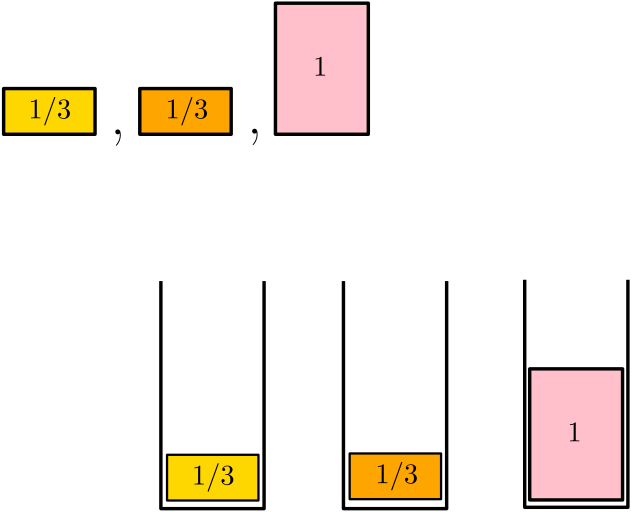
Example of Online Bin Stretching



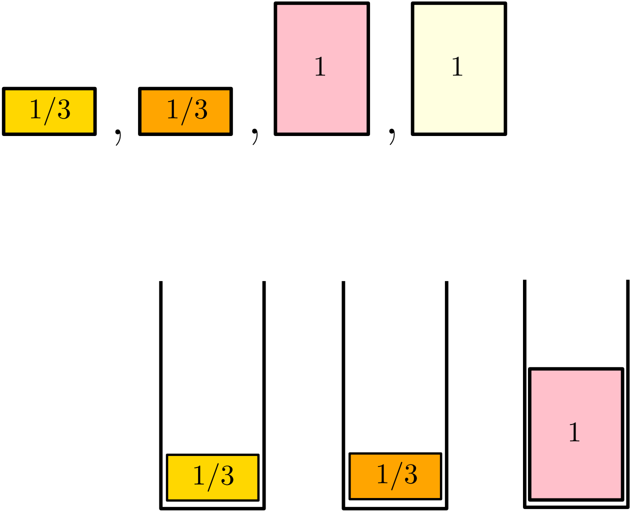
Example of Online Bin Stretching



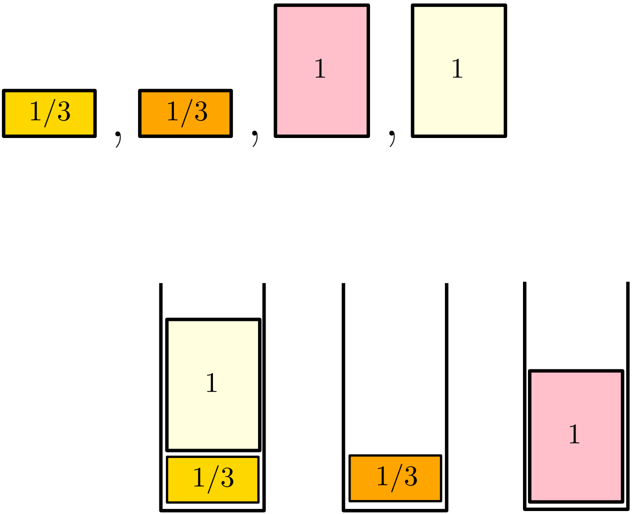
Example of Online Bin Stretching



Example of Online Bin Stretching



Example of Online Bin Stretching



State of the art

Recall:

- **Goal:** Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

Algorithms:

- [Azar, Regev '98]:
  - stretching factor 1.625.
- Currently best: [B., Sgall, van Stee, Veselý '14]
  - stretching factor 1.5.

Lower Bounds:

- [Azar, Regev '98]:
  - Stretching factor must be at least $4/3$. (We've just seen it!)

Restricted setting

Normal setting:

- Algorithm learns m at the start.
- One algorithm must be competitive for any m.

Restricted setting:

- OPT always uses exactly k bins.
- Easier to design algorithms.
- Much easier to create lower bounds.

Restricted setting: State of the art

Recall:

- **Goal:** Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

Three bins:

- [Azar, Regev '98]: Algorithm with stretching factor 1.4.
- [B., Sgall, van Stee, Veselý '14]: S. f. $11/8 = 1.375$.

LBs for $3 \leq m \leq 8$:

- [Gabay, Brauner, Kotov '14]:
  - A computer-found lower bound for three bins: $19/14 \approx 1.357$.
- [B. '16]: Improved to $45/33 \approx 1.\overline{36}$.

Restricted setting: State of the art

Recall:

- **Goal:** Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

Three bins:

- [Azar, Regev '98]: Algorithm with stretching factor 1.4.
- [B., Sgall, van Stee, Veselý '14]: S. f. $11/8 = 1.375$.

LBs for $3 \leq m \leq 8$**:**

- [Gabay, Brauner, Kotov '14]:
  - A computer-found lower bound for three bins: $19/14 \approx 1.357$.
- [B. '16]: Improved to $45/33 \approx 1.\overline{36}$.

Restricted setting: State of the art

Recall:

- **Goal:** Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

Three bins:

- [Azar, Regev '98]: Algorithm with stretching factor 1.4.
- [B., Sgall, van Stee, Veselý '14]: S. f. $11/8 = 1.375$.

LBs for $3 \leq m \leq 8$**:**

- [Gabay, Brauner, Kotov '14]:
  - A computer-found lower bound for three bins: $19/14 \approx 1.357$.
- [B. '16]: Improved to $45/33 \approx 1.\overline{36}$.

Restricted setting: State of the art

Recall:

- **Goal:** Pack all items into m bins of capacity c, with the **stretching factor** c being as small as possible.

Three bins:

- [Azar, Regev '98]: Algorithm with stretching factor 1.4.
- [B., Sgall, van Stee, Veselý '14]: S. f. $11/8 = 1.375$.

LBs for $3 \leq m \leq 8$**:**

- [Gabay, Brauner, Kotov '14]:
  - A computer-found lower bound for three bins: $19/14 \approx 1.357$.
- [B. '16]: Improved to $45/33 \approx 1.\overline{36}$.
- [B. '18, thesis]: Improved to $112/82 \approx 1.365$.
- [B. '18, thesis]: Lower bound $19/14$ for $4 \leq m \leq 8$.

Online problems are games

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- Adversary presents input;
- Algorithm packs the items.

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- ADVERSARY presents input;
- ALGORITHM packs the items.

Victory conditions:

- ADVERSARY wins when one bin is loaded to $\geq t/g$;
- ALGORITHM wins if all bins $< t/g$ and no more input.

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- Adversary presents input;
- Algorithm packs the items.

Victory conditions:

- Adversary wins when one bin is loaded to $\geq t/g$;
- Algorithm wins if all bins $< t/g$ and no more input.
- Winning strategy for Adversary $\leftrightarrow$ lower bound of t/g.
- Winning strategy for Algorithm $\leftrightarrow$ s. factor $< t/g$.

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- ADVERSARY presents input;
- ALGORITHM packs the items.

Victory conditions:

- ADVERSARY wins when one bin is loaded to $\geq t/g$;
- ALGORITHM wins if all bins $< t/g$ and no more input.
- Winning strategy for ADVERSARY $\leftrightarrow$ lower bound of t/g.
- Winning strategy for ALGORITHM $\leftrightarrow$ s. factor $< t/g$.

To make the tree finite:

- ADVERSARY can only send $1, 2, \ldots, g$.

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- ADVERSARY presents input;
- ALGORITHM packs the items.

Victory conditions:

- ADVERSARY wins when one bin is loaded to $\geq t/g$;
- ALGORITHM wins if all bins $< t/g$ and no more input.
- Winning strategy for ADVERSARY $\leftrightarrow$ lower bound of t/g.
- ~~Winning strategy for ALGORITHM $\leftrightarrow$ s. factor $< t/g$.~~

To make the tree finite:

- ADVERSARY can only send $1, 2, \ldots, g$.

Online problems are games

Setting: m bins, capacity (guarantee): g, stretched bin (target): t.

- Adversary presents input;
- Algorithm packs the items.

Victory conditions:

- Adversary wins when one bin is loaded to $\geq t/g$;
- Algorithm wins if all bins $< t/g$ and no more input.

- Winning strategy for Adversary $\leftrightarrow$ lower bound of t/g.
- ~~Winning strategy for Algorithm $\leftrightarrow$ s. factor $< t/g$.~~

To make the tree finite:

- Adversary can only send $1, 2, \ldots, g$.

Caveat: Adversary must at all times honor the guarantee:
Items can be packed into m bins of capacity g.

Lower bounds using the computer

- [Gabay, Brauner, Kotov '14]: lower bound $t/g = 19/14 \approx 1.357$ for three bins via computer.

Core idea:

1. Use the MINIMAX algorithm to evaluate the game tree.
2. Cache vertices to speed up computation.
3. Make sure that ADVERSARY honors the guarantee at all steps: *Items can be packed into m bins of capacity g.*

Lower bounds using the computer

- [Gabay, Brauner, Kotov '14]: lower bound $t/g = 19/14 \approx 1.357$ for three bins via computer.

Core idea:

1. Use the Minimax algorithm to evaluate the game tree.
2. Cache vertices to speed up computation.
3. Make sure that Adversary honors the guarantee at all steps: *Items can be packed into m bins of capacity g.*

- Written in Python.
- Uses CSP for the guarantee checking in every step of Adversary.
- Never empties cache, runs out of memory for $g = 20$.
- Authors believed that much larger instances cannot be tackled.

Lower bounds using the computer

- [Gabay, Brauner, Kotov '14]: lower bound $t/g = 19/14 \approx 1.357$ for three bins via computer.

Core idea:

1. Use the MINIMAX algorithm to evaluate the game tree.
2. Cache vertices to speed up computation.
3. Make sure that ADVERSARY honors the guarantee at all steps: *Items can be packed into m bins of capacity g.*

- Written in Python.
- Uses CSP for the guarantee checking in every step of ADVERSARY.
- Never empties cache, runs out of memory for $g = 20$.
- Authors believed that much larger instances cannot be tackled.

Improvements

1. ~~Written in Python.~~ Written in C++.

Improvements

1. ~~Written in Python.~~ Written in C++.
2. ~~Never empties cache.~~ Fix maximum cache size.

Improvements

1. ~~Written in Python.~~ Written in C++.
2. ~~Never empties cache.~~ Fix maximum cache size.
3. Fast verification of the guarantee.

Improvements

1. ~~Written in Python.~~ Written in C++.
2. ~~Never empties cache.~~ Fix maximum cache size.
3. Fast verification of the guarantee.
4. Prune tree with good situations.

Improvements

1. ~~Written in Python.~~ Written in C++.
2. ~~Never empties cache.~~ Fix maximum cache size.
3. Fast verification of the guarantee.
4. Prune tree with good situations.
5. Iterate over monotonicity.

Improvements

1. ~~Written in Python.~~ Written in C++.
2. ~~Never empties cache.~~ Fix maximum cache size.
3. Fast verification of the guarantee.
4. Prune tree with good situations.
5. Iterate over monotonicity.
6. Parallelization.

Improvements

1. Written in C++.
2. Fix maximum cache size.
3. **Fast verification of the guarantee.**
4. Prune tree with good situations.
5. Iterate over monotonicity
6. Parallelization.

Improvements: Guarantee checking

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

- *Constraint satisfaction:* Does the next item i satisfy all the constraints?

Improvements: Guarantee checking

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

- *Constraint satisfaction:* Does the next item i satisfy all the constraints?
- *ILP:* Does the next item i satisfy all the *linear integer* constraints?

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: **1**,2,1,11

Old queue:

New queue:

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: **1**,2,1,11

Old queue:

New queue: `(1,0,0)`

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,**2**,1,11

Old queue: `(1,0,0)`

New queue:

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,**2**,1,11

Old queue: `(1,0,0)`

New queue: `(3,0,0), (2,1,0)`

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,2,**1**,11

Old queue: `(3,0,0), (2,1,0)`

New queue:

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,2,**1**,11

Old queue: `(3,0,0), (2,1,0)`

New queue: `(4,0,0), (3,1,0),` ~~`(3,1,0)`~~`, (2,2,0), (2,1,1)`

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,2,1,**11**

Old queue: `(4,0,0), (3,1,0), (2,2,0), (2,1,1)`

New queue:

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,2,1,**11**

Old queue: `(4,0,0), (3,1,0), (2,2,0), (2,1,1)`

New queue: `(11,4,0), (14,1,0), (12,3,0), (11,3,1), (13,2,0), (11,2,2), (13,1,1), (12,2,1)`

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps:
Items can be packed into m bins of capacity g.

Example for 3 bins, $g = 14$:

Item list: 1,2,1,11

Old queue: `(4,0,0), (3,1,0), (2,2,0), (2,1,1)`

New queue: `(11,4,0), (14,1,0), (12,3,0), (11,3,1), (13,2,0), (11,2,2), (13,1,1), (12,2,1)`

Largest item that can be sent next: 14.

Fastest algorithm I know: Sparse Dynprog

Guarantee:

Make sure that ADVERSARY honors the guarantee at all steps: *Items can be packed into m bins of capacity g.*

Example for 3 bins, $g = 14$:

Item list: 1,2,1,11

Old queue: `(4,0,0), (3,1,0), (2,2,0), (2,1,1)`

New queue: `(11,4,0), (14,1,0), (12,3,0), (11,3,1), (13,2,0), (11,2,2), (13,1,1), (12,2,1)`

Largest item that can be sent next: 14.

- Use hashing to find duplicities.
- *Experiments:* The length of queues stays in 1000s $\Rightarrow$ try to keep hash table in CPU cache.

Improvements

1. Written in C++.
2. Fix maximum cache size.
3. Fast verification of the guarantee.
4. **Prune tree with good situations.**
5. Iterate over monotonicity.
6. Parallelization.

Improvements: Good situations

Good Situation: A partial packing which an algorithm can easily finalize with a good stretching factor (say $1 + \alpha$).

Assume 3 *bins:*

Improvements: Good situations

Good Situation: A partial packing which an algorithm can easily finalize with a good stretching factor (say $1 + \alpha$).

Assume 3 *bins:*

Good Situation 1: Two bins (A and B), $size(A) + size(B) \geq 2 - \alpha$. Last bin ($C$) arbitrary.

Improvements: Good situations

Good Situation: A partial packing which an algorithm can easily finalize with a good stretching factor (say $1 + \alpha$).

Assume 3 *bins:*

Good Situation 1: Two bins (A and B), $size(A) + size(B) \geq 2 - \alpha$. Last bin ($C$) arbitrary.

Algorithm: Simply pack remaining items into the last bin.

Improvements: Good situations

Good Situation: A partial packing which an algorithm can easily finalize with a good stretching factor (say $1 + \alpha$).

Assume 3 *bins:*

Good Situation 1: Two bins (A and B), $size(A) + size(B) \geq 2 - \alpha$. Last bin ($C$) arbitrary.

Algorithm: Simply pack remaining items into the last bin.

Proof: Remaining load is $\leq 3 - (2 - \alpha) = 1 + \alpha$: everything fits into bin C.

Improvements: Good situations

Good Situation: A partial packing which an algorithm can easily finalize with a good stretching factor (say $1 + \alpha$).

Assume 3 *bins:*

Good Situation 1: Two bins (A and B), $size(A) + size(B) \geq 2 - \alpha$. Last bin ($C$) arbitrary.

Algorithm: Simply pack remaining items into the last bin.

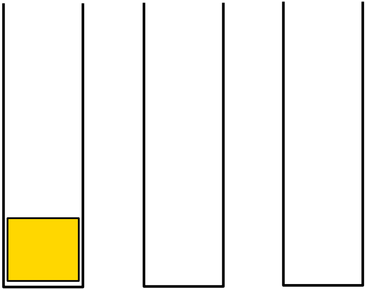
Proof: Remaining load is $\leq 3 - (2 - \alpha) = 1 + \alpha$: everything fits into bin C. ✓

Good Situation 2

Good Situation 2: One bin with $size(A) \in [1 - 2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.

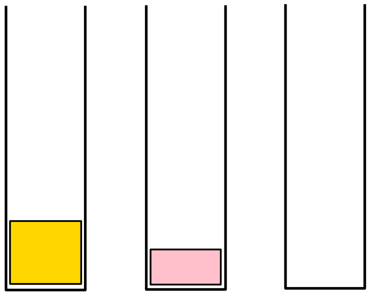
Good Situation 2

Good Situation 2: One bin with $size(A) \in [1 - 2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.



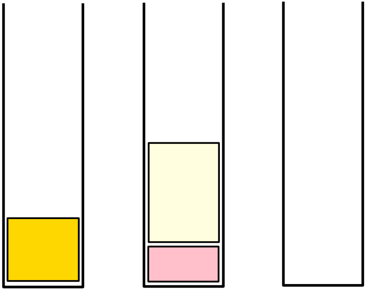
Good Situation 2

Good Situation 2: One bin with $size(A) \in [1 - 2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.



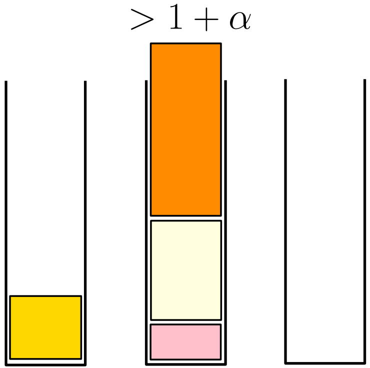
Good Situation 2

Good Situation 2: One bin with $size(A) \in [1 - 2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.



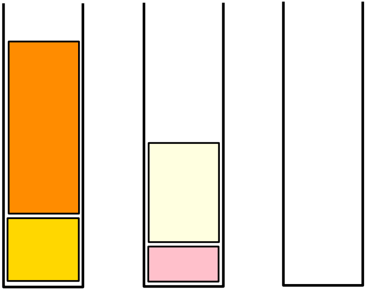
Good Situation 2

Good Situation 2: One bin with $size(A) \in [1-2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.



Good Situation 2

Good Situation 2: One bin with $size(A) \in [1 - 2\alpha, \alpha]$, other two bins can be arbitrary.
Algorithm: Pack items into some other bin, then into A.



Packed load $\geq 1 + \alpha + (1 - 2\alpha) = 2 - \alpha$. Use Good Situation 1. ✓

Improvements: Good situations

Recall:

Good Situation: A partial packing which the algorithm can easily finalize with the correct stretching factor.

Improvements: Good situations

Recall:

Good Situation: A partial packing which the algorithm can easily finalize with the correct stretching factor.

Good situation with factor $\frac{t-1}{g}$ $\Rightarrow$ winning position for $\textsc{Algorithm}$ $\Rightarrow$ pruning.

Improvements: Good situations

Recall:

Good Situation: A partial packing which the algorithm can easily finalize with the correct stretching factor.

Good situation with factor $\frac{t-1}{g}$ $\Rightarrow$ winning position for ALGORITHM $\Rightarrow$ pruning.

16 core server, CPU AMD Opteron 6134, 32GB RAM.
Lower bound for 3 bins, 45/33, no good situations: 294s.
Lower bound for 3 bins, 45/33, good situations active: 7s.

Improvements: Good situations

Recall:

Good Situation: A partial packing which the algorithm can easily finalize with the correct stretching factor.

Good situation with factor $\frac{t-1}{g}$ $\Rightarrow$ winning position for ALGORITHM $\Rightarrow$ pruning.

16 core server, CPU AMD Opteron 6134, 32GB RAM.
Lower bound for 3 bins, 45/33, no good situations: 294s.
Lower bound for 3 bins, 45/33, good situations active: 7s.

Lower bound for 7 bins, 19/14, no good situations: 91s.
Lower bound for 7 bins, 19/14, GS active: 57s.

Improvements

1. Written in C++.
2. Fix maximum cache size.
3. Fast verification of the guarantee.
4. Prune tree with good situations.
5. **Iterate over monotonicity.**
6. Parallelization.

Improvements: Monotonicity

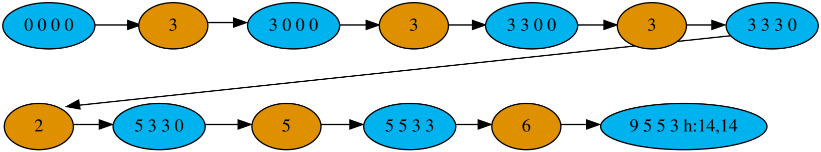


Figure: One branch from the lower bound of 19/14 for 4 bins.

- The branch is not non-decreasing . . . but only barely.

Improvements: Monotonicity

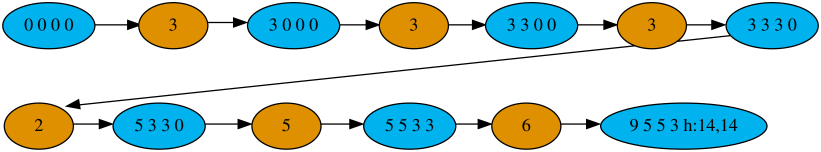


Figure: One branch from the lower bound of 19/14 for 4 bins.

- The branch is not non-decreasing ... but only barely.
- Define **monotonicity** k: after item of size s, you can send only item of size $s - k$ or higher.

Improvements: Monotonicity

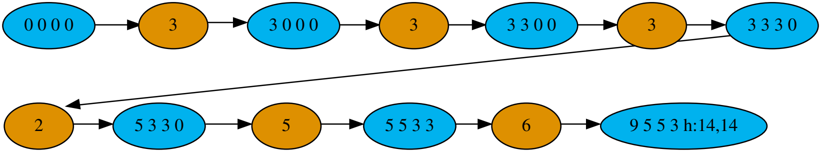


Figure: One branch from the lower bound of 19/14 for 4 bins.

- The branch is not non-decreasing ... but only barely.
- Define **monotonicity** k: after item of size s, you can send only item of size $s - k$ or higher.
- Monotonicity 0 $\rightarrow$ non-decreasing;

Improvements: Monotonicity

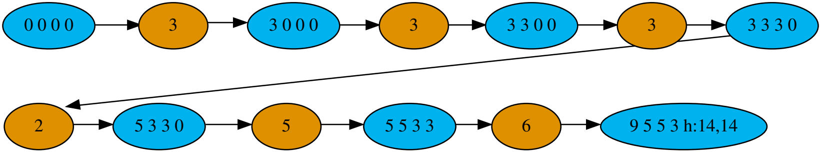


Figure: One branch from the lower bound of 19/14 for 4 bins.

- The branch is not non-decreasing . . . but only barely.
- Define **monotonicity** k: after item of size s, you can send only item of size $s - k$ or higher.
- Monotonicity 0 $\rightarrow$ non-decreasing;
- Monotonicity $g - 1$ $\rightarrow$ full generality.

Improvements: Monotonicity

Bins	Lower bound	Monotonicity required
3	$45/33$	1
3	$86/63$	6
3	$112/82$	8
4-7	$19/14$	0*
8	$19/14$	1*

Improvements: Monotonicity

Bins	Lower bound	Monotonicity required
3	45/33	1
3	86/63	6
3	112/82	8
4-7	19/14	0*
8	19/14	1*

**: The first item (size 5) does not count.*

Improvements: Monotonicity

Bins	Lower bound	Monotonicity required
3	$45/33$	1
3	$86/63$	6
3	$112/82$	8
4-7	$19/14$	0*
8	$19/14$	1*

**: The first item (size 5) does not count.*

16 core server, CPU AMD Opteron 6134, 32GB RAM.
Lower bound for 7 bins, $19/14$, full monotonicity: 1.53h.
Lower bound for 7 bins, $19/14$, monotonicity 0: 57s.

Improvements

1. Written in C++.
2. Fix maximum cache size.
3. Prune tree with good situations.
4. Fast verification of the guarantee.
5. Iterate over monotonicity.
6. **Parallelization.**

Improvements: Parallelization

Idea: Cut the tree in some depth, send tasks to remote workers.

Improvements: Parallelization

Idea: Cut the tree in some depth, send tasks to remote workers.

Implementation:

- Just cutting by depth $\rightarrow$ some tasks very easy, some very hard.

Improvements: Parallelization

Idea: Cut the tree in some depth, send tasks to remote workers.

Implementation:

- Just cutting by depth $\rightarrow$ some tasks very easy, some very hard.
- Workers on the same machine can share cache?

Improvements: Parallelization

Idea: Cut the tree in some depth, send tasks to remote workers.

Implementation:

- Just cutting by depth $\rightarrow$ some tasks very easy, some very hard.
- Workers on the same machine can share cache?
    - *If so:* need to implement proper locking.

Improvements: Parallelization

Idea: Cut the tree in some depth, send tasks to remote workers.

Implementation:

- Just cutting by depth $\rightarrow$ some tasks very easy, some very hard.
- Workers on the same machine can share cache?
  - *If so:* need to implement proper locking.
- OpenMPI for cluster-level parallelization, each machine: several Posix threads.

Summary of results

- [Gabay, Brauner, Kotov '14]:
  - lower bound $t/g = 19/14 \approx 1.357$ for three bins via computer.
  - No results for more than three bins.

Without parallelization:

- [B. '16]: Improved to $45/33 \approx 1.\overline{36}$ for three bins.
- [B. '16]: Lower bound of $19/14 \approx 1.357$ for $4, 5$ bins.

With parallelization:

- [B. '18, thesis]: For $m = 3$, improved to $112/82 \approx 1.365$.
- [B. '18, thesis]: For $3 \leq m \leq 8$, the lower bound of $19/14$ holds.

Verification via the Coq Proof Assistant.

Coq Proof Assistant

Proof assistant / interactive theorem prover

- It verifies the theorem for us, has a large collection of valid theorems built-in;
- We have to provide most of the proof ourselves.

Previous verification

- [B. '18, thesis]: A small C++ verification program, 1046 lines of code.

To get the result previously, the authors needed to:

Previous verification

- [B. '18, thesis]: A small C++ verification program, 1046 lines of code.

To get the result previously, the authors needed to:

1. Compute the lower bounds.
2. Write the verification program, debug it and trust it.

And a reviewer had to:

Previous verification

- [B. '18, thesis]: A small C++ verification program, 1046 lines of code.

To get the result previously, the authors needed to:

1. Compute the lower bounds.
2. Write the verification program, debug it and trust it.

And a reviewer had to:

1. Understand the problem, definitions and the claims.
2. Check the trees manually/separately *or*
3. Go through the verification program code and check it.

Verification with Coq

- [B., Simon '20]: Formal proof of all the lower bounds.

To get the result now, the authors have to:

Verification with Coq

- [B., Simon '20]: Formal proof of all the lower bounds.

To get the result now, the authors have to:

1. Compute the lower bounds.
2. Write the definitions of what a lower bound is in Coq.
3. Convert the lower bounds into Coq structures (proof advice).

Verification with Coq

- [B., Simon '20]: Formal proof of all the lower bounds.

To get the result now, the authors have to:

1. Compute the lower bounds.
2. Write the definitions of what a lower bound is in Coq.
3. Convert the lower bounds into Coq structures (proof advice).

And the reviewer has to:

1. Understand the problem, definitions and the claims.

Verification with Coq

- [B., Simon '20]: Formal proof of all the lower bounds.

To get the result now, the authors have to:

1. Compute the lower bounds.
2. Write the definitions of what a lower bound is in Coq.
3. Convert the lower bounds into Coq structures (proof advice).

And the reviewer has to:

1. Understand the problem, definitions and the claims.

That's it! The Coq system makes sure the trees are a valid proof for the given claims.

Code examples 1

The reviewer has to:

1. Understand the problem, definitions and the claims.

Two data structures: list of bins viewed as loads *or* as items packed in them:

```
Definition BinLoads    := list nat.
Definition BinExtended  := list nat.
Definition BinsExtended := list BinExtended.
```

Code examples 1

The reviewer has to:

1. Understand the problem, definitions and the claims.

Two data structures: list of bins viewed as loads *or* as items packed in them:

```
Definition BinLoads     := list nat.
Definition BinExtended  := list nat.
Definition BinsExtended := list BinExtended.
```

Simple recursive properties:

```
Fixpoint BinSum (B: BinExtended) := match B with
| nil    ⇒ 0
| x :: s ⇒ x + BinSum s
end.

Fixpoint MaxBinSum (P: BinsExtended) := match P with
| nil    ⇒ 0
| x :: s ⇒ max (BinSum x) (MaxBinSum s)
end.

Fixpoint MaxBinValue (St: BinLoads) := match St with
| nil    ⇒ 0
| x :: s ⇒ max x (MaxBinValue s)
end.
```

Code examples 2

- `AddToBin` – adding an item to the bins represented as loads.

```
Fixpoint AddToBin (St: BinLoads) (e: nat) (b: nat) := match St,b with
| nil   , b     ⇒ [e]
| x :: s, 0     ⇒ (x+e)::s
| x :: s, (S k) ⇒ x :: (AddToBin s e k)
end.
```

Code examples 2

- `AddToBin` – adding an item to the bins represented as loads.

```
Fixpoint AddToBin (St: BinLoads) (e: nat) (b: nat) := match St,b with
| nil   , b     ⇒ [e]
| x :: s, 0     ⇒ (x+e)::s
| x :: s, (S k) ⇒ x :: (AddToBin s e k)
end.
```

- `CompletePacking` – predicate is true when all items in ℓ appear in the bin configuration P.

```
Definition CompletePacking (ℓ : list nat) (P: BinsExtended) := forall e,
count_occ Nat.eq_dec ℓ e <= count_occ Nat.eq_dec (concat P) e.
```

Code examples 2

- `AddToBin` – adding an item to the bins represented as loads.

```
Fixpoint AddToBin (St: BinLoads) (e: nat) (b: nat) := match St, b with
| nil   , b     ⇒ [e]
| x :: s, 0     ⇒ (x+e) ::s
| x :: s, (S k) ⇒ x :: (AddToBin s e k)
end.
```

- `CompletePacking` – predicate is true when all items in ℓ appear in the bin configuration P.

```
Definition CompletePacking (ℓ : list nat) (P: BinsExtended) := forall e,
count_occ Nat.eq_dec ℓ e <= count_occ Nat.eq_dec (concat P) e.
```

- `SolutionPacking` – all items in ℓ appear and the bin configuration packs all items into m bins of capacity g.

```
Definition SolutionPacking (ℓ : list nat) (P: BinsExtended) :=
CompletePacking ℓ P ∧ length P = m ∧ MaxBinSum P <= g.
```

Code examples 3

- `OnlineInfeasible` – the main predicate for a lower bound.
- Parameters:
  - $X \in \mathbb{N}$ – a technical parameter for Coq induction.

```
Inductive OnlineInfeasible : nat → list nat → BinLoads → Prop :=
| Overflow X ℓ St:    t <= MaxBinValue St   → (exists P, SolutionPacking ℓ P)
   → OnlineInfeasible X ℓ St
| Deadend  X ℓ St:    length St <= m
   → (exists e, forall b, (b < m)
      → OnlineInfeasible X ( (e) :: ℓ) (AddToBin St e b) )
   → OnlineInfeasible (X+1) ℓ St
.
```

Theorem (Only theorem needed to prove)

For any ℓ, St, X, the proposition `OnlineInfeasible X ℓ St` *implies a lower bound for* Online Bin Stretching *for a bin configuration with ℓ items and loads of bins equal to St.*

Code examples 3

- `OnlineInfeasible` – the main predicate for a lower bound.
- Parameters:
  - $X \in \mathbb{N}$ – a technical parameter for Coq induction.
  - list ℓ – list of items

```
Inductive OnlineInfeasible : nat → list nat → BinLoads → Prop :=
| Overflow X ℓ St:   t <= MaxBinValue St  → (exists P, SolutionPacking  ℓ P)
   → OnlineInfeasible X ℓ St
| Deadend  X ℓ St:   length St <= m
   → (exists e, forall b, (b < m)
      → OnlineInfeasible X ( (e) :: ℓ) (AddToBin St e b) )
   → OnlineInfeasible (X+1) ℓ St
.
```

Theorem (Only theorem needed to prove)

For any ℓ, St, X, the proposition `OnlineInfeasible X ℓ St` *implies a lower bound for* Online Bin Stretching *for a bin configuration with ℓ items and loads of bins equal to St.*

Code examples 3

- `OnlineInfeasible` – the main predicate for a lower bound.
- Parameters:
  - $X \in \mathbb{N}$ – a technical parameter for Coq induction.
  - list ℓ – list of items
  - St – list of loads of bins corresponding to the items.

```
Inductive OnlineInfeasible : nat → list nat → BinLoads → Prop :=
| Overflow X ℓ St:   t <= MaxBinValue St  → (exists P, SolutionPacking ℓ P)
   → OnlineInfeasible X ℓ St
| Deadend  X ℓ St:   length St <= m
   → (exists e, forall b, (b < m)
      → OnlineInfeasible X ( (e) :: ℓ) (AddToBin St e b) )
   → OnlineInfeasible (X+1) ℓ St
.
```

Theorem (Only theorem needed to prove)

For any ℓ, St, X, the proposition `OnlineInfeasible X ℓ St` *implies a lower bound for* Online Bin Stretching *for a bin configuration with ℓ items and loads of bins equal to St.*

Technical challenges

In principle, this would be enough . . .

Technical challenges

In principle, this would be enough . . . but verification failed due to memory as well as time.

1. **DAG encoding.**
   - Functional programming base $\Rightarrow$ Coq embeds trees easily, DAGs not as easily.
   - Trees were too big for Coq prover to read and validate (duplicate objects).
   - We needed to encode DAGs as DAGs.

Technical challenges 2

2. **Last layer compression.**
   - The full DAGs were still beyond the memory limit of our verifier PCs (32 GB of RAM).

Technical challenges 2

2. **Last layer compression.**
   - The full DAGs were still beyond the memory limit of our verifier PCs (32 GB of RAM).

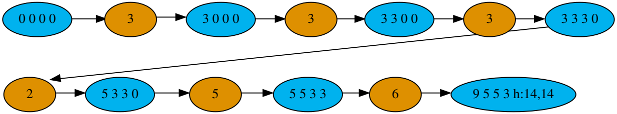


Figure: One branch from the lower bound of 19/14 for 4 bins.

   - **Solution:** Encode the last (two) item heuristic into Coq, saving space.

Technical challenges 2

2. **Last layer compression.**
   - The full DAGs were still beyond the memory limit of our verifier PCs (32 GB of RAM).

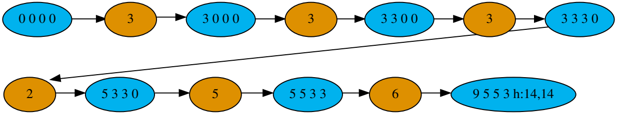


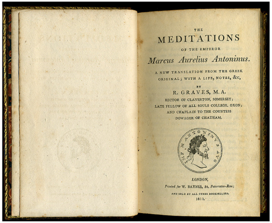
Figure: One branch from the lower bound of 19/14 for 4 bins.

   - **Solution:** Encode the last (two) item heuristic into Coq, saving space.
3. **Binary integers.**
   - Coq naturally works with Peano arithmetic – easy to axiomatize + already verified statements in the core.
   - But we needed to squeeze a bit more performance + memory savings.
   - We spent some effort to move to binary representation.

Graph size and results

Value of m	3	4	5	6	7	8
Lower bound	112/82	19/14	19/14	19/14	19/14	19/14
Tree nodes	186k	433	3908	3.8M	231M	2.5G
DAG nodes	103k	236	1271	38k	186k	1.6M
cDAG nodes	37k	102	408	7k	61k	598k
Time	38s	1s	2s	12s	4m30	2h

Size of the uncompressed and compressed DAGs and (approximate) time needed to load the trees and certify each lower bound. The running times were computed on a machine with the Intel Core i5-6600 CPU and 32 GB of RAM.


THE
MEDITATIONS
OF THE EMPEROR
Marcus Aurelius Antoninus.
A NEW TRANSLATION FROM THE GREEK ORIGINAL; WITH A LIFE, NOTES, &c,
BY
R. GRAVES, M.A.
RECTOR OF CLAVERTON, SOMERSET;
LATE FELLOW OF ALL SOULS COLLEGE, OXON;
AND CHAPLAIN TO THE COUNTESS
DOWAGER OF CHATHAM.
LONDON,
Printed for W. BAYNES, 54, *Paternoster-Row;*
AND SOLD BY ALL OTHER BOOKSELLERS.
1811.

&

Meditations & research directions

Meditations: Adaptive lower bounds

- Lower bound construction for online (scheduling, packing) problems tend to have a lower amount of adaptivity.
- Adaptivity often represented as uncertainty when sequence ends or several very different optimal solutions.
- *Possible reason:* Case analysis still feels inelegant to researchers.

Meditations: Adaptive lower bounds

- Lower bound construction for online (scheduling, packing) problems tend to have a lower amount of adaptivity.
- Adaptivity often represented as uncertainty when sequence ends or several very different optimal solutions.
- *Possible reason:* Case analysis still feels inelegant to researchers.
- BIN STRETCHING a sweet spot: no non-trivial lower bound for the general case despite effort.

Meditations: Adaptive lower bounds

- Lower bound construction for online (scheduling, packing) problems tend to have a lower amount of adaptivity.
- Adaptivity often represented as uncertainty when sequence ends or several very different optimal solutions.
- *Possible reason:* Case analysis still feels inelegant to researchers.
- BIN STRETCHING a sweet spot: no non-trivial lower bound for the general case despite effort.
- *Philosophical question:* Can we expect more adaptive (and harder to comprehend) lower bounds as we near optimality for other, major problems?

Meditations: Limits of Minimax

Why a Minimax approach works for Bin Stretching:

1. Finite number of bins $\Rightarrow$ limited configuration space.
2. Sending a large item restricts the optimum substantially.
3. Strategies with exponentially increasing items not applicable.

Meditations: Limits of Minimax

Why a Minimax approach works for BIN STRETCHING:

1. Finite number of bins $\Rightarrow$ limited configuration space.
2. Sending a large item restricts the optimum substantially.
3. Strategies with exponentially increasing items not applicable.

- *Research direction:* Apply the same approach for closely related problems (small m and known sum of processing times, small m and related machines).

Meditations: Limits of Minimax

Why a Minimax approach works for BIN STRETCHING:

1. Finite number of bins $\Rightarrow$ limited configuration space.
2. Sending a large item restricts the optimum substantially.
3. Strategies with exponentially increasing items not applicable.

- *Research direction:* Apply the same approach for closely related problems (small m and known sum of processing times, small m and related machines).
- *Philosophical question:* Is computer-aided search doomed for other problems where these advantages are not present?

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.
- *Question:* What is the (in practice) fastest way of solving this problem?

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.
- *Question:* What is the (in practice) fastest way of solving this problem?
- *Caveat:* Interesting on its own, but unclear how much it helps here (due to caching).

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.
- *Question:* What is the (in practice) fastest way of solving this problem?
- *Caveat:* Interesting on its own, but unclear how much it helps here (due to caching).

The Coq verification takes up more resources (memory, time) than the original C++ verifier.

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.
- *Question:* What is the (in practice) fastest way of solving this problem?
- *Caveat:* Interesting on its own, but unclear how much it helps here (due to caching).

The Coq verification takes up more resources (memory, time) than the original C++ verifier.

- *Question:* Is there a still simple enough, yet much faster, solution within Coq?

Research directions: Technical challenges

Solving a feasibility of the packing: Can the items on input fit into m bins of capacity g?

- Previous approach: CSP solver.
- Our approach: Sparse dynamic program.
- *Question:* What is the (in practice) fastest way of solving this problem?
- *Caveat:* Interesting on its own, but unclear how much it helps here (due to caching).

The Coq verification takes up more resources (memory, time) than the original C++ verifier.

- *Question:* Is there a still simple enough, yet much faster, solution within Coq?
- *Engineering challenge:* Find bottlenecks, possibly "give back" code to Coq itself.

Research directions: Larger computer

Cluster: heterogenous, 109 computation threads running.
Runtime: hours/single days.

Research directions: Larger computer

Cluster: heterogenous, 109 computation threads running.
Runtime: hours/single days.

Bigger computer/Longer computation:
Optimistic guess: find a lower bound of $19/14$ for $m \leq 10$ bins.

Research directions: Closing the gap for $m = 3$

Current gap: $[1.365, 1.375]$.

Research directions: Closing the gap for $m = 3$

Current gap: $[1.365, 1.375]$.

- *Educated guess:* Right number might be $41/30 = 1.3\overline{6}$.

Needs algorithmic improvements or stronger good situations.

Research directions: The curious case of $m = 4$

For 3 *bins:*

Potential ratio t/g	Lower bound found
$19/14$	Yes.
$34/25$	Yes.
$45/33$	Yes.

Research directions: The curious case of $m = 4$

For 3 *bins:*

Potential ratio t/g	Lower bound found
$19/14$	Yes.
$34/25$	Yes.
$45/33$	Yes.

For 4 *bins:*

Potential ratio t/g	Lower bound found
$19/14$	Yes.
$34/25$	No.
$45/33$	No.

Research directions: The curious case of $m = 4$

For 3 *bins:*

Potential ratio t/g	Lower bound found
$19/14$	Yes.
$34/25$	Yes.
$45/33$	Yes.

For 4 *bins:*

Potential ratio t/g	Lower bound found
$19/14$	Yes.
$34/25$	No.
$45/33$	No.

Conjecture: The optimal stretching factor for 4 bins is strictly smaller than the optimal factor for 3.

Meditations: Further progress via ML?

- Using a Minimax approach to solve the BIN STRETCHING GAME is reminiscent of Chess approaches of 15 years ago or more.
- AlphaGo: combination of Monte Carlo Tree Search with ML evaluation of Chess configurations when the depth is exceeded.
- AlphaGo does not solve Chess, but it moved the technology forward. Of course, massive computational power required.

Meditations: Further progress via ML?

- Using a Minimax approach to solve the Bin Stretching Game is reminiscent of Chess approaches of 15 years ago or more.
- AlphaGo: combination of Monte Carlo Tree Search with ML evaluation of Chess configurations when the depth is exceeded.
- AlphaGo does not solve Chess, but it moved the technology forward. Of course, massive computational power required.
- Possible research direction: Can we make use of that technology here? Can we teach computer to play as Algorithm better to get faster pruning?

Research directions: Summary

github.com/bohm/binstretch | github.com/bs24/LB_BinStretching

1. Stretching factor 19/14 for $m \leq 10$ bins: Bigger computer might suffice.
2. Three bins: close the gap on stretching factor: $[1.365, 1.375]$ (algorithmic).
   - *Educated guess:* Right number might be $41/30 = 1.3\overline{6}$.
3. Four bins: Show that tight bound is less than for three bins.

Research directions: Summary

github.com/bohm/binstretch | github.com/bs24/LB_BinStretching

1. Stretching factor $19/14$ for $m \leq 10$ bins: Bigger computer might suffice.
2. Three bins: close the gap on stretching factor: $[1.365, 1.375]$ (algorithmic).
   - *Educated guess:* Right number might be $41/30 = 1.3\overline{6}$.
3. Four bins: Show that tight bound is <u>less</u> than for three bins.
4. **Big open problem:** A better general lower bound.
   $4/3$ is easy, nothing else is known (for more than 8 bins).

Research directions: Summary

github.com/bohm/binstretch | github.com/bs24/LB_BinStretching

1. Stretching factor 19/14 for $m \leq 10$ bins: Bigger computer might suffice.
2. Three bins: close the gap on stretching factor: $[1.365, 1.375]$ (algorithmic).
   - *Educated guess:* Right number might be $41/30 = 1.3\overline{6}$.
3. Four bins: Show that tight bound is <u>less</u> than for three bins.
4. **Big open problem:** A better general lower bound.
   4/3 is easy, nothing else is known (for more than 8 bins).

Thank you!