Online bin packing

- Problem definition
- First Fit and other algorithms
- The asymptotic performance ratio
- Weighting functions
- Lower bounds
Bin packing: problem definition (repeat)

- Input: $n$ items with sizes $a_1, \ldots, a_n \in (0, 1]$
- Goal: pack these items into a **minimal number of bins**
- Each bin has size 1
Bin packing: problem definition (repeat)

- Input: $n$ items with sizes $a_1, \ldots, a_n \in (0, 1]$
- Goal: pack these items into a minimal number of bins
- Each bin has size 1
Next Fit (repeat)

- Keep a single open bin
- When an item does not fit, close this bin and open a new bin
- Closed bins are never used again
- Approximation ratio is 2
  - each pair of successive bins contains more than 1
  - matching lower bound
First Fit, Best Fit, Worst Fit

- Keep all bins open
- An item is assigned to the first / best / worst bin in which it fits
  - **best** (worst) bin = bin in which the item fits and where it leaves the **smallest** (largest) empty space
- If item does not fit in any open bin, open a new bin

First Fit and Best Fit are better than Next Fit

Disadvantage: no bin is ever closed
Asymptotic performance ratio

- A performance measure for bin packing algorithms
- Similar to competitive ratio
- Idea: compare number of bins used to optimal number of bins, for large inputs
- Definition:

\[
R_A^\infty = \limsup_{n \to \infty} \sup_{\sigma} \left\{ \frac{A(\sigma)}{OPT(\sigma)} \middle| OPT(\sigma) = n \right\}.
\]
Asymptotic performance ratio

- It is the infimum $R$ for which

$$A(\sigma) \leq R \cdot \text{OPT}(\sigma) + c$$

for every input sequence $\sigma$

- Here $c$ is a non-negative constant which does not depend on the input

- We still calculate the ratio between the costs of ALG and the costs of OPT

- However, we give ALG several bins for free
Performance ratio of FF and BF

- These algorithms have a ratio of 1.7
- We show a simple lower bound
- Asymptotic ratio $\Rightarrow$ we need an input sequence of arbitrary length $n$

- Sequence: $6n$ items of size 0.15, $6n$ items of size 0.34, $6n$ items of size 0.51
- These items can be packed into $6n$ bins by OPT
How do FF and BF pack this input?

- 6n items of size 0.15 ⇒ packed in n bins
  All these bins are 0.9 full

- 6n items of size 0.34 ⇒ packed in 3n bins
  All these bins are 0.68 full

- 6n items of size 0.51 ⇒ packed in 6n bins

- Total 10n bins used

- Lower bound of 10n/6n = 5/3
Bounded space algorithms

- keep only a constant number of bins open at any time
- gives a constant stream of output (closed bins)
- Idea: pack similar items together
- NF is bounded space, but FF, BF and WF are not
The HARMONIC algorithm

This algorithm classifies items into types according to their size

- Size $\in (\frac{1}{2}, 1]$: type 1, pack 1 per bin
- Size $\in (\frac{1}{3}, \frac{1}{2}]$: type 2, pack 2 per bin
- ...
- Size $\in (\frac{1}{k}, \frac{1}{k-1}]$: type $k-1$, pack $k-1$ per bin
- Size $\in (0, \frac{1}{k}]$: use Next Fit
The HARMONIC algorithm

- Items of type $i$ are packed $i$ per bin for $i = 1, \ldots, k - 1$
- Items of type $k$ are packed using Next Fit: use one bin until next item does not fit, then start a new bin
- How do we analyze such an algorithm?

Notes:

- Items of different types are packed independently
- Size of an item is irrelevant, only its type matters

Of course this is not true for the optimal solution.
Weighting functions

- Idea: give a **weight** to every type

- The weight represents the **amount of bin space** that this type occupies

- For a given input, the **total** weight $W$ is then the total number of bins used

- OPT needs to pack this same amount of weight ($W$)

- OPT seeks to minimize the number of bins $\Leftrightarrow$ maximize the **weight per offline bin**
Weights for HARMONIC

- Items of type $i$ are packed $i$ per bin for $i = 1, \ldots, k - 1 \Rightarrow$ weight is $1/i$

- Items of type $k$ are packed using **Next Fit**. When an item does not fit in a bin, the bin is at least $\frac{k-1}{k}$ full (size of each item $\leq \frac{1}{k}$) $\Rightarrow$ weight of item of size $x$ is $\frac{k}{k-1}x$

- For any input sequence $\sigma$, the number of bins that HARMONIC uses is at most the total weight of all the items $+k$
Weighting functions (ctd.)

- Denote the total weight of a given input by $W$
- HARMONIC uses at most $W + k$ bins
- Denote the optimal number of bins by $b$
- We have
  $$\frac{HARMONIC(\sigma)}{OPT(\sigma)} = \frac{W + k}{b}$$
- Asymptotic performance ratio
  = maximum weight per offline bin
- How much weight can there be in an offline bin?
A bin with maximal weight

- We are looking for a set of items such that
  - their total size is at most 1
  - their weight is maximized

- We should use “smallest possible” size for each type:
  e.g. $\frac{1}{2} + \varepsilon$ for type 1, for $\varepsilon \to 0$

- Consider the ratio of weight to size for each type
The ratio of weight to size

- For type $i$, this is $\left(\frac{1}{i}\right)/\left(\frac{1}{i+1}\right) = (i + 1)/i$ for $i = 1, \ldots, k - 1$

- For type $k$, this ratio is $\frac{k}{k-1}$ (item of size $x$ has weight $\frac{k}{k-1}x$)

- These ratios are strictly decreasing from type 1 until type $k$
  Ratios are $2, 3/2, 4/3, \ldots, k/(k - 1), k/(k - 1)$

- We can find a maximum-weight set as follows:
  - start with an item of type 1 (i.e. size $\frac{1}{2} + \varepsilon$)
  - repeatedly add the largest item which will fit
A set of maximum weight

<table>
<thead>
<tr>
<th>Type</th>
<th>size</th>
<th>weight</th>
<th>Type</th>
<th>size</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>1</td>
<td>1/7+eps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/3</td>
<td>1/2</td>
<td>type 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/4</td>
<td>–</td>
<td>1/3+eps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1/5</td>
<td>–</td>
<td></td>
<td>type 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1/6</td>
<td>–</td>
<td></td>
<td></td>
<td>1/2+eps</td>
</tr>
<tr>
<td>6</td>
<td>1/7</td>
<td>1/6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... sum 1.691
Lower bound for bounded space algorithms

- We have found the bin with maximum weight
- We can turn this into a lower bound for BS algorithms
- Each item in the bin occurs $n$ times ($n \to \infty$)
- A bounded space algorithm must pack almost all items in bins that have one type of item – i.e., like HARMONIC!
- Thus, the number of bins needed for each $n$-tuple is $n$ times the weight of this item
- Total amount of bins needed is $n$ times total weight of input
- OPT can pack these items in $n$ bins
Weighting functions: summary

A weighting function can be used:

- to find an upper bound for a bin packing algorithm
- to find a lower bound for bounded space algorithms

Notes:

- Finding the right weighting function can be hard
- For some problems, we can give an optimal bounded space algorithm, but we do not know its performance ratio!
A general lower bound

What about non-bounded space algorithms?

The previous lower bound construction does not work!

We will show a lower bound of 3/2.
A general lower bound

Consider the following input:

- $n$ items of size $1/7 + \varepsilon$ (small)
- $n$ items of size $1/3 + \varepsilon$ (medium-sized)
- $n$ items of size $1/2 + \varepsilon$ (large)

Depending on what the algorithm does, the input may already end after the first or second phase.

The small items can be packed in $n/6$ bins.

ALG is better than $3/2$-competitive? Then it uses at most $n/4$ bins for these items
General lower bound (2)

- We consider how ALG packs the small items.
- It can use bins with 1, 2, ..., 6 small items.
- It wants to leave space for future items.
- E.g. no bin will have 5 small items.
- The small and medium items \((1/3 + \varepsilon)\) together can be packed in \(n/2\) bins.
- Each bin contains two small and two medium items.
- ALG may not use more than \(3n/4\) bins for these items.
General lower bound (3)

☐ Let a set of items of total size at most 1 be a configuration

☐ We can determine bounds on how often ALG uses each configuration

☐ E.g. it cannot use only bins with 1 small item in the first phase (then it needs $n$ bins for them and the input stops)

☐ This gives us a system of equations

☐ Working it out, we find that ALG cannot be better than $3/2$-competitive
Improving on Harmonic

- The upper bound for Harmonic is given by a bin with maximum weight.

- This weight is $1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{42} + \cdots = 1.691$.

- The highest contribution is from type 1, items in $(1/2, 1]$.

- A bin with an item of size $1/2 + \varepsilon$ is half empty.

- We would like to avoid this situation and use the wasted space for other items.

- Idea: **subdivide** the interval $(1/2, 1]$ in two types.
Refined Harmonic

- Use a parameter $\Delta \in (1/3, 1/2)$

- Define extra types:
  0. size $> 1 - \Delta$
  1. size $\in (1/2, 1 - \Delta]$
  2. size $\in (\Delta, 1/2]$
  3. size $\in (1/3, \Delta]$
  4. size $\in (1/4, 1/3]$, etc.

The idea is to combine type 3 with type 1

However, we don’t know how many items of any type will arrive
Refined Harmonic (2)

- We need an additional parameter $\alpha$.

- This parameter indicates which fraction of the type 3 items are placed alone in bins (red items).

- The remaining items are paired into bins (blue items).

- Whenever a type 3 item arrives, we determine its color by considering the current ratio of red to blue items ($\geq \alpha / \leq \alpha$).

- **Red**: place item in a bin with a type 1 item, or in a new bin.

- **Blue**: place item in a bin with a blue item, or in a new bin.
Refined Harmonic (3)

☐ Whenever a type 1 item arrives, we check if a bin with a red item is available

☐ If not, we pack it in a new bin (and keep it open for a possible red item)

☐ If we pick $\alpha$ too large, no items of type 1 will arrive

☐ If we pick $\alpha$ too small, many items of type 1 arrive

☐ We need to find good values for $\alpha$ and $\Delta$

☐ How can we show an upper bound?
Weighting functions for RH

- Idea: define **two** weighting functions

- Let
  - $x$ be the number of bins with red items
  - $y$ be the number of type 1 items
  - $z$ be the number of bins with other items

- Refined Harmonic uses

  $$\max(x, y) + z = \max(x + z, y + z)$$

  bins in total

- We define one weighting function for each argument
Weighting functions for RH

- Suppose there are many type 1 items
- Then we can ignore the red items
- If there are \( N \) type 3 items, \( \alpha N \) of them are red
- The rest are packed 2 per bin
- There are \((1 - \alpha)N\) blue items each of weight \( 1/2 \)
- There are \( \alpha N \) red items each of weight 0
- Average weight per item of type 3 is \((1 - \alpha)/2\)
Weighting functions for RH

- Suppose there are many red items
- Then we can ignore the type 1 items
- If there are $N$ type 3 items, $\alpha N$ of them are red
- The rest are packed 2 per bin
- There are $(1 - \alpha)N$ blue items each of weight $1/2$
- There are $\alpha N$ red items each of weight 1
- Average weight per item of type 3 is $(1 + \alpha)/2$
### Weighting functions for RH

<table>
<thead>
<tr>
<th>Type</th>
<th>Interval</th>
<th>( W )</th>
<th>( V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((1 - \Delta, 1])</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>((1/2, 1 - \Delta])</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>((\Delta, 1/2])</td>
<td>(1/2)</td>
<td>(1/2)</td>
</tr>
<tr>
<td>3</td>
<td>((1/3, \Delta])</td>
<td>((1 - \alpha)/2)</td>
<td>((1 + \alpha)/2)</td>
</tr>
<tr>
<td>3</td>
<td>((1/4, 1/3])</td>
<td>(1/3)</td>
<td>(1/3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>((1/19, 1/18])</td>
<td>(1/18)</td>
<td>(1/18)</td>
</tr>
<tr>
<td>19</td>
<td>((0, 1/19])</td>
<td>(\frac{19}{18}x)</td>
<td>(\frac{19}{18}x)</td>
</tr>
</tbody>
</table>
Upper bound for RH

- For both weighting functions, we need to find a maximum weight bin.
- The overall maximum weight will be an upper bound for RH.
- We can choose $\alpha$ and $\Delta$ such that the maximum is minimized.
- This gives an upper bound of 1.639.
- With bounded space, better than 1.691 is impossible.
Current state of research

- Using a longer lower bound structure, it is possible to show a lower bound of 1.5401 (van Vliet, 1992)

- Using a MUCH more complicated structure than RH, an upper bound of 1.58889 can be shown (Seiden, 2001: HARMONIC++)

- Improving either of these bounds is an open problem
Randomization

□ Does not help!

□ The lower bounds work in phases

□ In each phase, only one type of item arrives

□ The algorithm distributes them in some way

□ Deterministic: fractions (1/3 of the items is packed in this way, ...)

□ Randomized: expectations (The expected fraction of items packed in this way is 1/3, ...)

□ Final result is the same