Multidimensional bin packing

January 31, 2008

Epstein and van Stee, SODA 2004

- Extending bin packing to more dimensions
- The problem of packing the small items
- Analysis
- Lower bounds

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
- $\ldots$
- 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

Analysis of HARMONIC

- Analysis is done with weighting function
- Weight of item = amount of bin space that it occupies
- Asymptotic performance ratio
  = maximum weight per offline bin
- For HARMONIC, we find an upper bound of $\Pi_\infty = 1.691$

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 and HARMONIC are bounded space, but First Fit etc. are not

Previous results

- An algorithm with asymptotic performance ratio $\Pi_\infty^d = 1.691^d$ was given by Csirik and van Vliet (1993)
- No better offline algorithm is known!
- Only improvement is for $d = 2$: approximation ratio of $1 + \ln \Pi_\infty = 1.52$ (FOCS 2006)
- No APTAS is possible even for $d = 2$ (APX-hard)

Multidimensional packing

- In one dimension, packing small items is trivial (use NEXT FIT)
- In more dimensions, doing this *with bounded space* is the main problem
- Csirik and van Vliet use unbounded space
- Hard to pack all small items without wasting much space
- Other problem: how to deal with different dimensions?

Possible approaches

- Packing items in rows
- Shelf packing: classify items by height
  - what about dimensions $d \geq 3$?
- Cut bins into sub-bins
  - squares: i^2 items of type i per bin
  - how to pack small squares...?

Rectangle packing

- Consider a rectangle of 0.01 by 0.5: is it large or small?
- Csirik & van Vliet:
  arbitrarily large set of sub-bins available for items of similar size
- This can never be bounded space

Our algorithm

- We show how to pack items and when to close bins, without wasting too much space
- We use some ideas from Csirik and Raghavan(1989), Csirik and van Vliet (1993)
- C&vV give a lower bound of 1.691^d
- Our algorithm has this ratio
- For squares, ratio is optimal but we do not know what it is!

Algorithm for square packing (1)

- Parameters:
  - a small constant $\varepsilon > 0$
    small $\varepsilon \Rightarrow$ large additive constant in performance ratio
    large $\varepsilon \Rightarrow$ large performance ratio
  - a large integer M that depends on ε
- A square is small if width is at most $1/M$, else large

We actually define a group of algorithms which differ only in their choice of ε

Algorithm for square packing (2)

- We divide the squares into types based on their width
- For the large items, this is done just like HARMONIC
- Large squares: i^2 of type i per bin
- There are $M-1$ large types
- Small squares: M types, each type is packed separately
- Example: $M=3$. Intervals for large squares are $(1/3,1/2]$ and $(1/2,1]$.

Intervals for small squares ($M = 3$)

Type

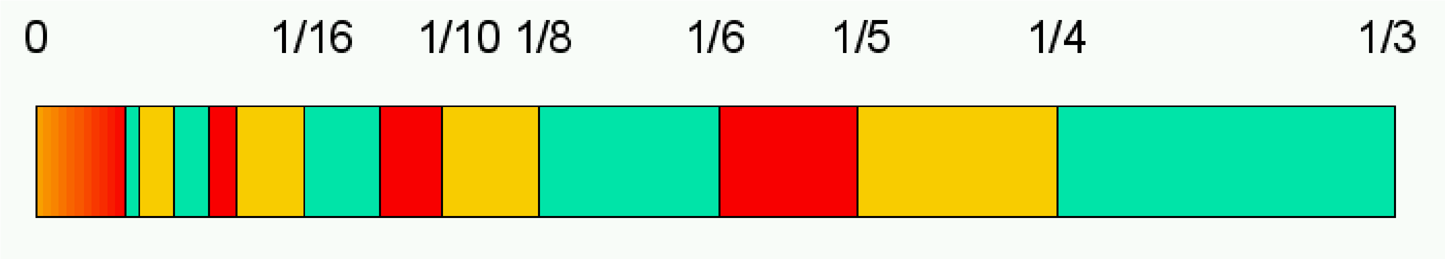
3. $(\frac{1}{4}, \frac{1}{3}] \cup (\frac{1}{8}, \frac{1}{6}] \cup (\frac{1}{16}, \frac{1}{12}] \cup \cdots = \cup_{i \geq 0} \left(\frac{1}{4 \cdot 2^i}, \frac{1}{3 \cdot 2^i} \right]$

4. $(\frac{1}{5}, \frac{1}{4}] \cup (\frac{1}{10}, \frac{1}{8}] \cup (\frac{1}{20}, \frac{1}{16}] \cup \cdots = \cup_{i \geq 0} \left(\frac{1}{5 \cdot 2^i}, \frac{1}{4 \cdot 2^i} \right]$

5. $(\frac{1}{6}, \frac{1}{5}] \cup (\frac{1}{12}, \frac{1}{10}] \cup (\frac{1}{24}, \frac{1}{20}] \cup \cdots = \cup_{i \geq 0} \left(\frac{1}{6 \cdot 2^i}, \frac{1}{5 \cdot 2^i} \right]$

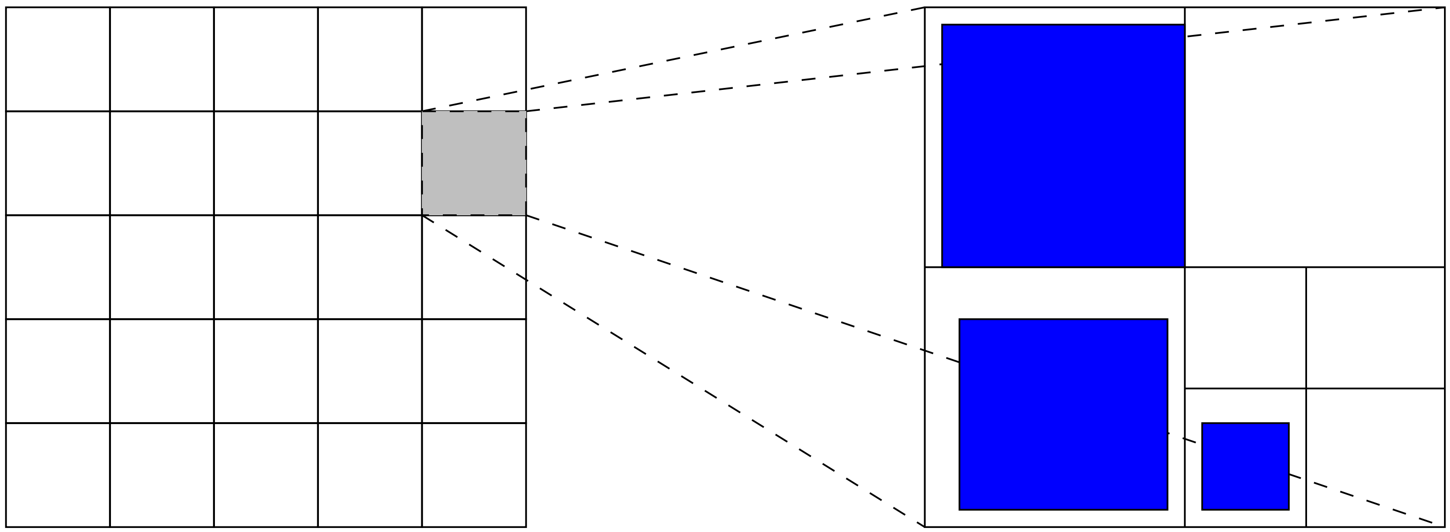
The types keep alternating as items get smaller

There is no single smallest type!



Packing small squares

- Type 5 items: when a new bin is opened, it is partitioned in 25 sub-bins of 1/5 by 1/5
- Item arrives: cut sub-bin repeatedly into 4 squares until correct size is reached



Packing small squares

- Never cut a large square if a smaller square exists
- If no free sub-bin larger than the item exists, close the bin and open a new one

Claim 1. *There are at most 3 open sub-bins of any size but the largest*

Proof: A sub-bin of a certain size is only created when all other sub-bins of this size are closed

We create four at a time, but one is immediately used: it is cut into smaller sub-bins or filled with an item

Claim 2. *Each closed bin with small items contains items of total area at least* $1-\varepsilon$

Proof: We have $i \geq M$, we choose M large enough

- a non-empty sub-bin is full by at least a fraction of $i^2/(i+1)^2$
- There are relatively few empty sub-bins: 3 per size, none of size $1/i$
- Total area of empty sub-bins is at most $3\sum_{k\geq 1}(2^k i)^{-2} = 1/i^2$.
- Occupied area is

$$(1-1/i^2)\cdot(i^2/(i+1)^2) = \frac{i^2-1}{(i+1)^2}.$$

Asymptotic performance ratio

- Use weighting function w_ε
- weight of item = fraction of bin that it occupies (items pay for bins they use)
- Large squares: weight of type i is $1/i^2$
- small square of width s has weight $s^2/(1-\varepsilon)$
- Performance ratio = maximum amount of weight that can be packed in one bin

Patterns

- Consider vectors $q = (q_1, \ldots, q_{M-1})$
- q is a pattern if there exists a feasible packing into a single bin which contains q_i items of type i ($i = 1, \ldots, M-1$)
- Let $A(q) = 1 - \sum_{i=1}^{M-1} \frac{q_i}{(i+1)^2}$
- $A(q)$ is an upper bound for the amount of space that is left in a bin with pattern q
- We define

$$w_\varepsilon(q) = \sum_{i=1}^{M-1} \frac{q_i}{i^2} + \frac{A(q)}{1-\varepsilon}.$$

Optimality of our algorithm

- Let
$$\alpha = \liminf_{\varepsilon \to 0} \max_q w_\varepsilon(q),$$
where the maximum is taken over all patterns q which are feasible for parameter ε
- (We use the $\liminf$ so that we do not have to prove that the limit exists)
- We show that no algorithm can have an asymptotic performance ratio strictly below α

In this sense, our algorithm (group of algorithms) is optimal

Proof of optimality

Suppose there is an algorithm with asymptotic performance ratio $(1-\varepsilon')\alpha$ for some $\varepsilon' > 0$

- We choose $\varepsilon < \varepsilon'$ such that our algorithm with parameter ε has ratio at most $(1+\varepsilon')\alpha$
- This is possible since the lim inf of the ratio is α for $\varepsilon \to 0$
- Let q be the pattern for which $w_\varepsilon(q)$ is maximal
- We write $w_\varepsilon(q) = (1+\varepsilon'')\alpha \leq (1+\varepsilon')\alpha$

Note: q specifies types, not specific items

Constructing an input set for a given q

- For each item of type i in q, we take a square of size $1/(i+1)+\delta$ for some very small $\delta > 0$
- Let $A_\delta = 1 - \sum_{i=1}^{M-1} q_i (1/(i+1)+\delta)^2$ be the free space
- Since q is a pattern, $A_\delta > 0$ for δ small enough
- We add a large amount of very small squares of total size A_δ such that they can all be packed together with the other items

Each item appears N times for some very large N

The lower bound

A bounded space algorithm must pack almost all items of a specific size together

- Phase i contains Nq_i items of size $1/(i+1)+\delta$, so algorithm needs $Nq_i/i^2 - O(1)$ bins for them
- Phase M contains small squares of total area NA_δ, so algorithm needs $NA_\delta - O(1)$ bins for them

Total amount of bins needed is $\sum_{i=1}^{M-1} Nq_i/i^2 + NA_\delta - O(M)$

A lower bound

- Total amount of bins needed is $\sum_{i=1}^{M-1} Nq_i/i^2 + NA_\delta - O(M)$
- The input can be packed into N bins
- Taking $\delta = 1/N$ and $N \to \infty$, this gives a lower bound of $\sum_{i=1}^{M-1} q_i/i^2 + A_\delta$ on the asymptotic performance ratio
- By our assumption, this is at most $(1-\varepsilon')\alpha$

The weight of this set

What is the weight of this set? Recall

- Item of type i has weight $1/i^2$ for $i = 1, \ldots, M$
- Small item of side s has weight $s^2/(1-\varepsilon)$

Contradiction

- The weight of this set of items tends to

$$\sum_{i=1}^{M-1} \frac{q_i}{i^2} + \frac{A_0}{1-\varepsilon} = w_\varepsilon(q) = (1+\varepsilon'')\alpha$$

as $\delta \to 0$.

- This implies

$$\begin{aligned}\sum_{i=1}^{M-1} \frac{q_i}{i^2} + A_0 &\geq (1-\varepsilon)(1+\varepsilon'')\alpha \\ &= (1-\varepsilon+\varepsilon''-\varepsilon\varepsilon'')\alpha > (1-\varepsilon')\alpha\end{aligned}$$

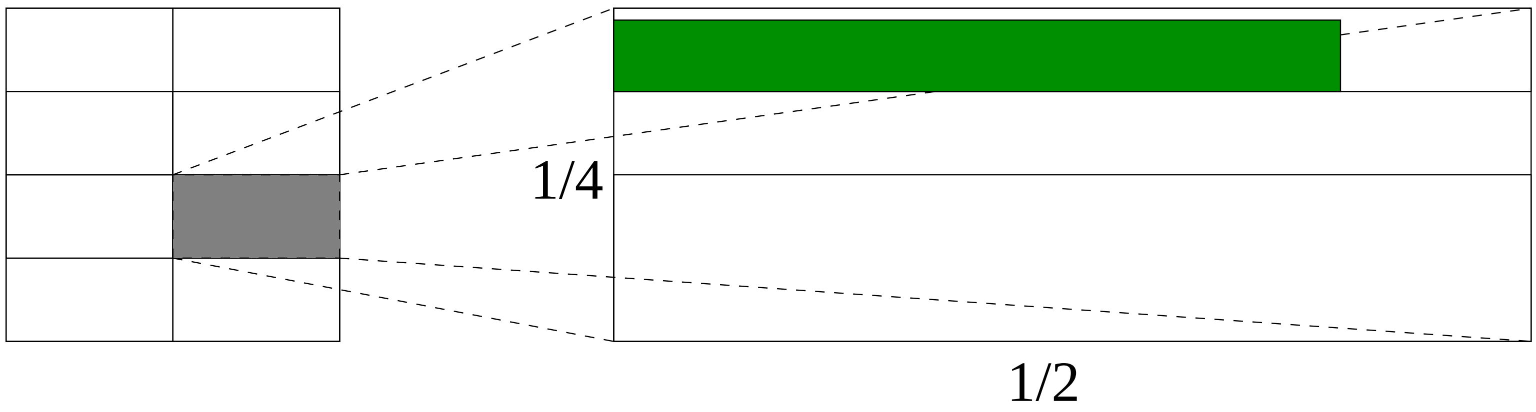
which is a contradiction.

Rectangle packing

- We now classify both the height and the width of an item
- There are $2M-1$ types for both
- In total there are $(2M-1)^2$ types
- A rectangle can be
  - large, large: treated similarly to squares
  - large, small / small, large
  - small, small

Example ($M = 3$)

- Rectangle of width 0.4 and height 0.06
- Type is $(2, 4)$ since $0.06 \in (\frac{1}{20}, \frac{1}{16})$
- A bin for type $(2, 4)$ is initially cut into sub-bins of width $1/2$ and height $1/4$
- A sub-bin is then cut further for items of small height (or width)
- We have only **one** sub-bin open for each size



Results

- This algorithm is also optimal among bounded space algorithms
- It can be extended to larger dimensions
- The asymptotic performance ratio is

$$1.691^d$$

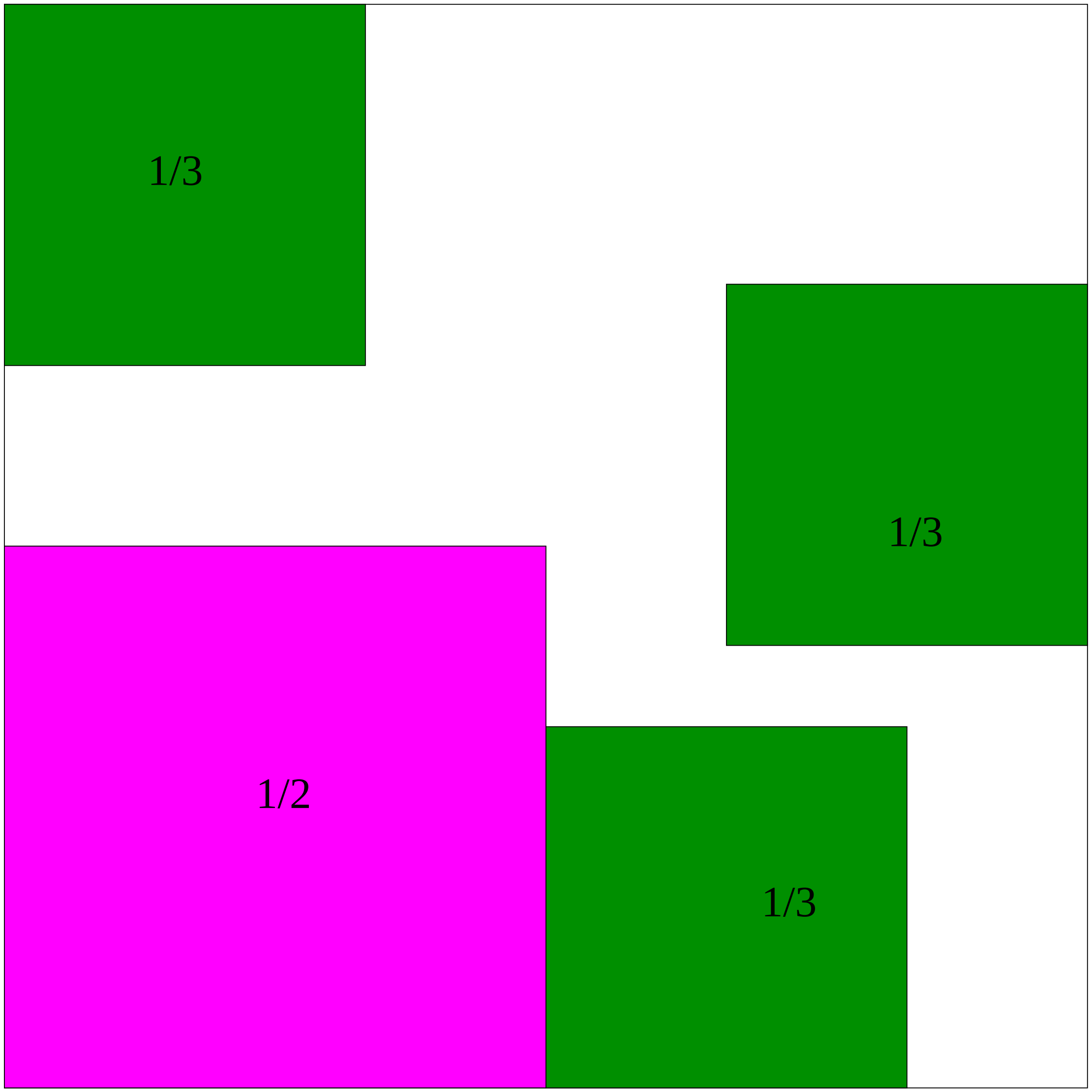
- This is optimal
- For hypercube packing, we have better bounds

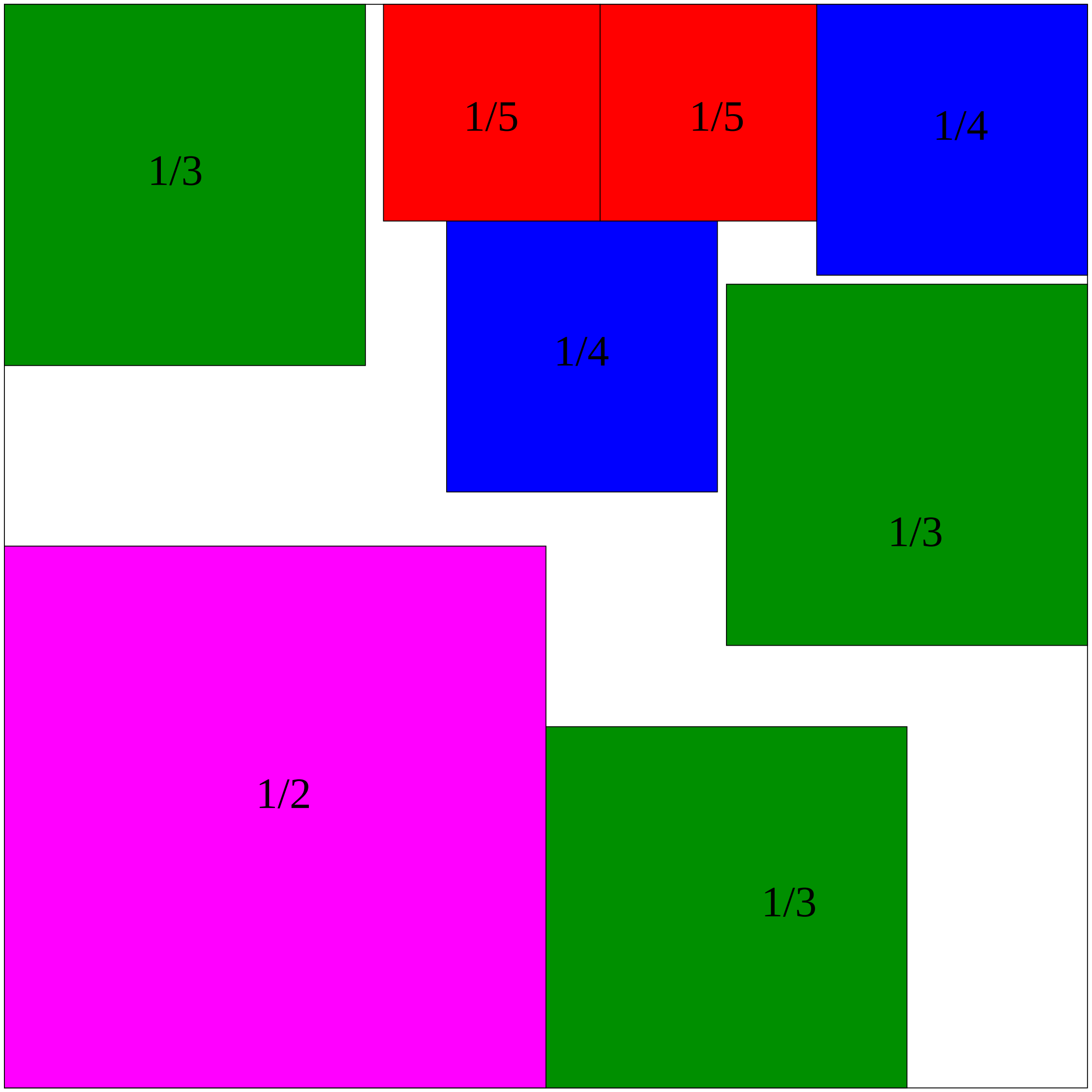
Packing squares into a square

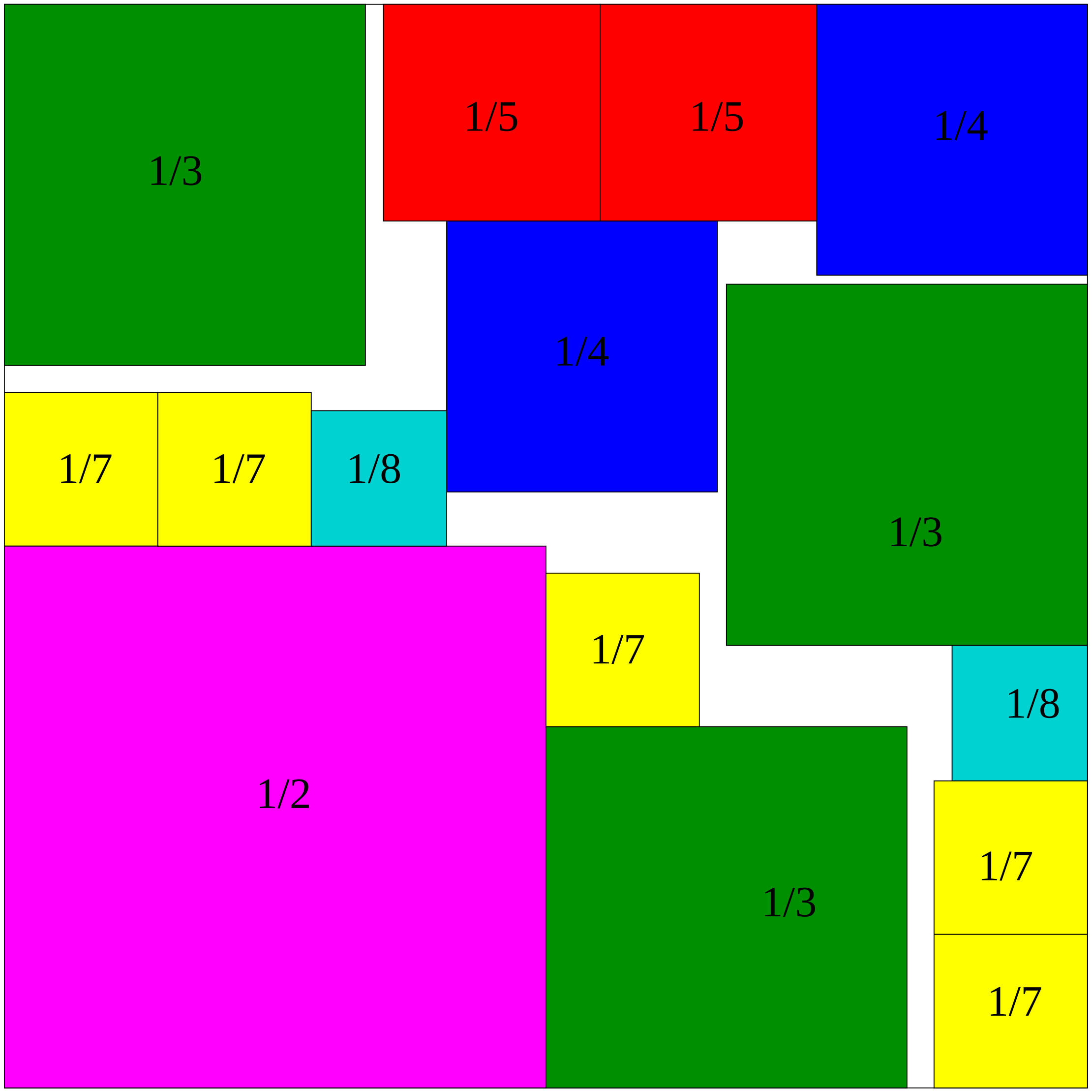
- Given a set of squares, can they be packed together in a single square?
- This problem is NP-hard! (Leung et al., 1990)
- We (probably. . .) cannot determine what is the maximum amount of weight packed in a bin
- Our algorithm is optimal but we do not know its ratio
- However, we can derive bounds on it

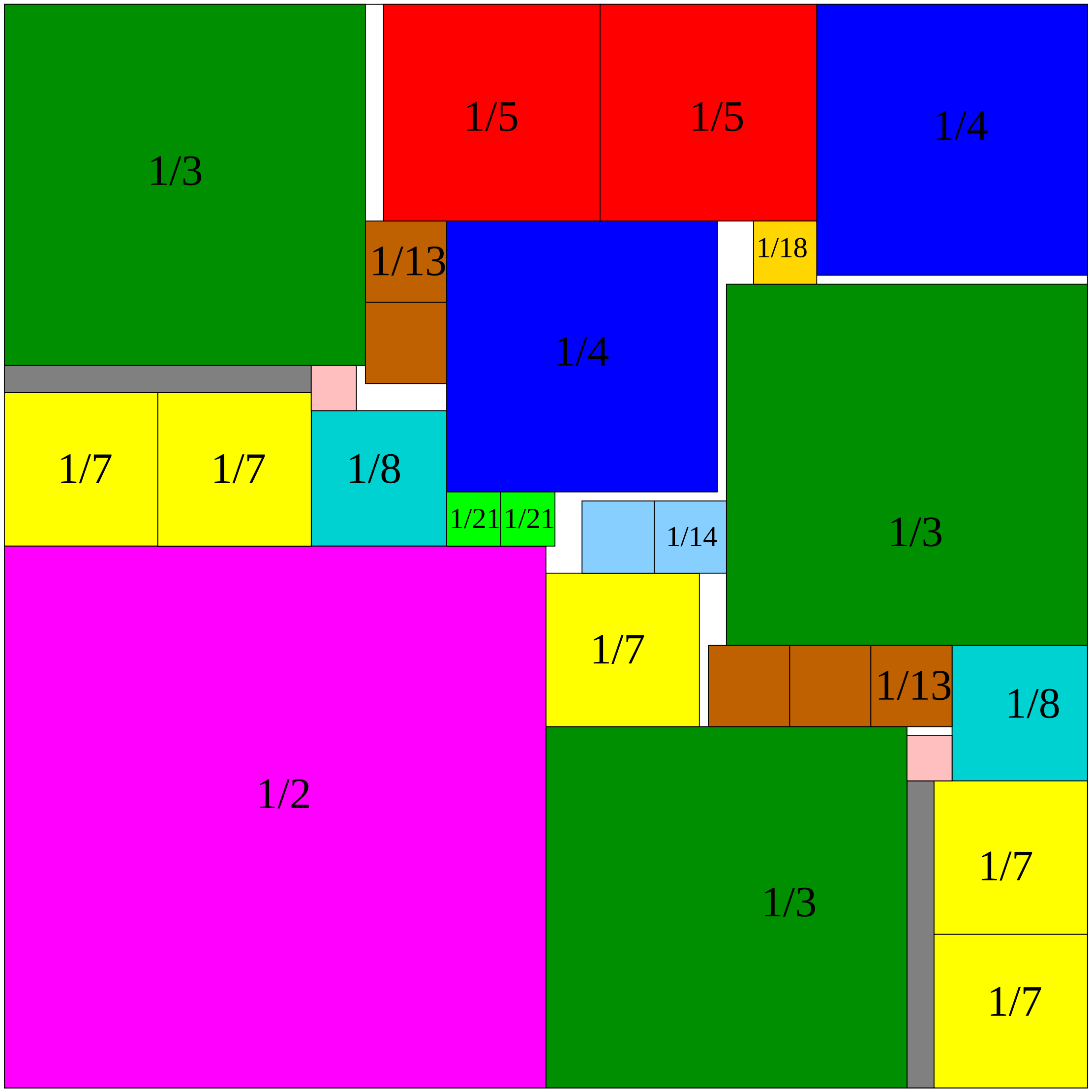
Square packing: lower bound

- As in one dimension, look for bin with maximal weight
- Use this to create a lower bound for bounded space algorithms
- How much weight can be packed in a square?
- Ad hoc packing, no algorithmic construction


1/3
1/3
1/2
1/3


1/3
1/5
1/5
1/4
1/4
1/3
1/2
1/3


1/3
1/5
1/5
1/4
1/4
1/7
1/7
1/8
1/3
1/7
1/8
1/2
1/7
1/3
1/7



LB = 2.3638

Square packing: upper bound

- To give a lower bound, it is sufficient to give a set of items and a packing for them in a square
- To prove an upper bound, you have to prove that some sets can be packed and some cannot
- This is much more difficult (NP-hard)

Square packing: upper bound

- We used a computer program to check all possible packings of crucial sets
- For instance, it is not possible to add an item of size more than $1/8$ to the given example
- We can prove an upper bound of 2.3692 (lower bound: 2.3638)

Hypercube packing: upper bound (1)

- Take $M = 2d/\log d$ (number of big types)
- Then for small items, an area of at least

$$\frac{i^d - 1}{(i+1)^d} \geq \frac{M^d - 1}{(M+1)^d} \geq \left(\frac{M}{M+1}\right)^{d+1}$$

is occupied, which is greater than

$$\left(\frac{M+1}{M}\right)^{-d} = \left(1 + \frac{1}{M}\right)^{-d} = \left(1 + \frac{\log d}{2d}\right)^{-d}$$

- This tends to

$$e^{-(\log d)/2} = (e^{\log d})^{-1/2} = 1/\sqrt{d}$$

Hypercube packing: upper bound (2)

- Denote the input by I
- Denote by I_i the subsequence of items of type i for $i = 1\ldots,M$
- Note that our algorithm uses separate bins for all these types
- Then $\mathrm{ALG}(I_i) = \mathrm{OPT}(I_i) \leq \mathrm{OPT}(I)$ for $i = 1,\ldots,M-1$
- Also $\mathrm{ALG}(I_M) = O(\sqrt{d}) \cdot \mathrm{OPT}(I_M) = O(\sqrt{d}) \cdot \mathrm{OPT}(I)$
- Therefore $\mathrm{ALG}(I) \leq (M-1)\mathrm{OPT}(I) + O(\sqrt{d}) \cdot \mathrm{OPT}(I) = O(d/\log d) \cdot \mathrm{OPT}(I)$

Hypercube packing: lower bound (1)

- To show a lower bound, we need to design an input on which a bounded space algorithm performs badly
- We use items of size $(1+\delta)/2^i$ for $i = 1, \ldots, \lceil \log d \rceil$
- In phase i, $N \cdot ((2^i - 1)^d - (2^i - 2)^d$ items of size $(1+\delta)/2^i$ arrive
- These items can be placed into N bins
- Along each coordinate axis, we reserve the space between $(1+\delta)(1-2^{1-i})$ and $(1+\delta)(1-2^{-i})$ for items of phase i

Hypercube packing: lower bound (2)

- How does a bounded space algorithm handle this input?
- For items of phase i, it needs

$$\frac{N \cdot ((2^i-1)^d - (2^i-2)^d}{(2^i-1)^d} = N \cdot \left(1 - \left(\frac{2^i-2}{2^i-1}\right)^d\right)$$

  bins
- This number is decreasing in i
- How many bins are needed for phase $\lceil \log d \rceil$?
- This is a lower bound for the amount of bins needed in each phase $1, \ldots, \lceil \log d \rceil$.

Hypercube packing: lower bound (3)

- In phase $i = \lceil \log d \rceil$, we need at least

$$
\begin{aligned}
N \cdot \left(1 - \left(\frac{2^i - 2}{2^i - 1}\right)^d\right) &= N \cdot \left(1 - \left(\frac{2d - 2}{2d - 1}\right)^d\right) \\
&= N \cdot \left(1 - \left(1 - \frac{1}{2d - 1}\right)^d\right) \\
&\geq N\left(1 - e^{-1/2}\right) \\
&> 0.39N
\end{aligned}
$$

  bins

- Thus in total, we need at least $0.39N \log d$ bins

- This proves a lower bound of $\log d$

Summary

- We give a bounded space online algorithm with ratio 1.691^d
- This matches the performance of the best known offline algorithm
- Compare this to results for one-dimensional bin packing
- For hypercube packing, the performance ratio of our algorithm is sublinear in d

Note: the best **lower bound** for hypercube packing (unbounded space!) is $4/3\ldots$