Knapsack and rounding

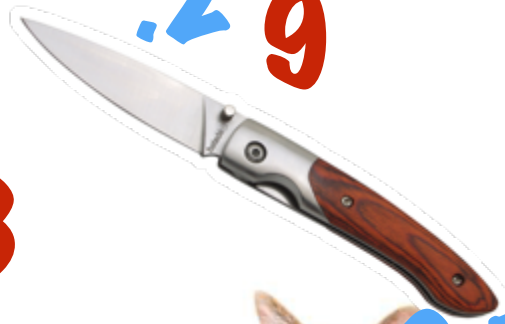
The knapsack problem: what should you put in your knapsack?

Weight
Value

.3
2

.15
5

.29



4 3

3.5
1

.23

.23

2.5
5

4 3

6.1

.05
1

2
.9

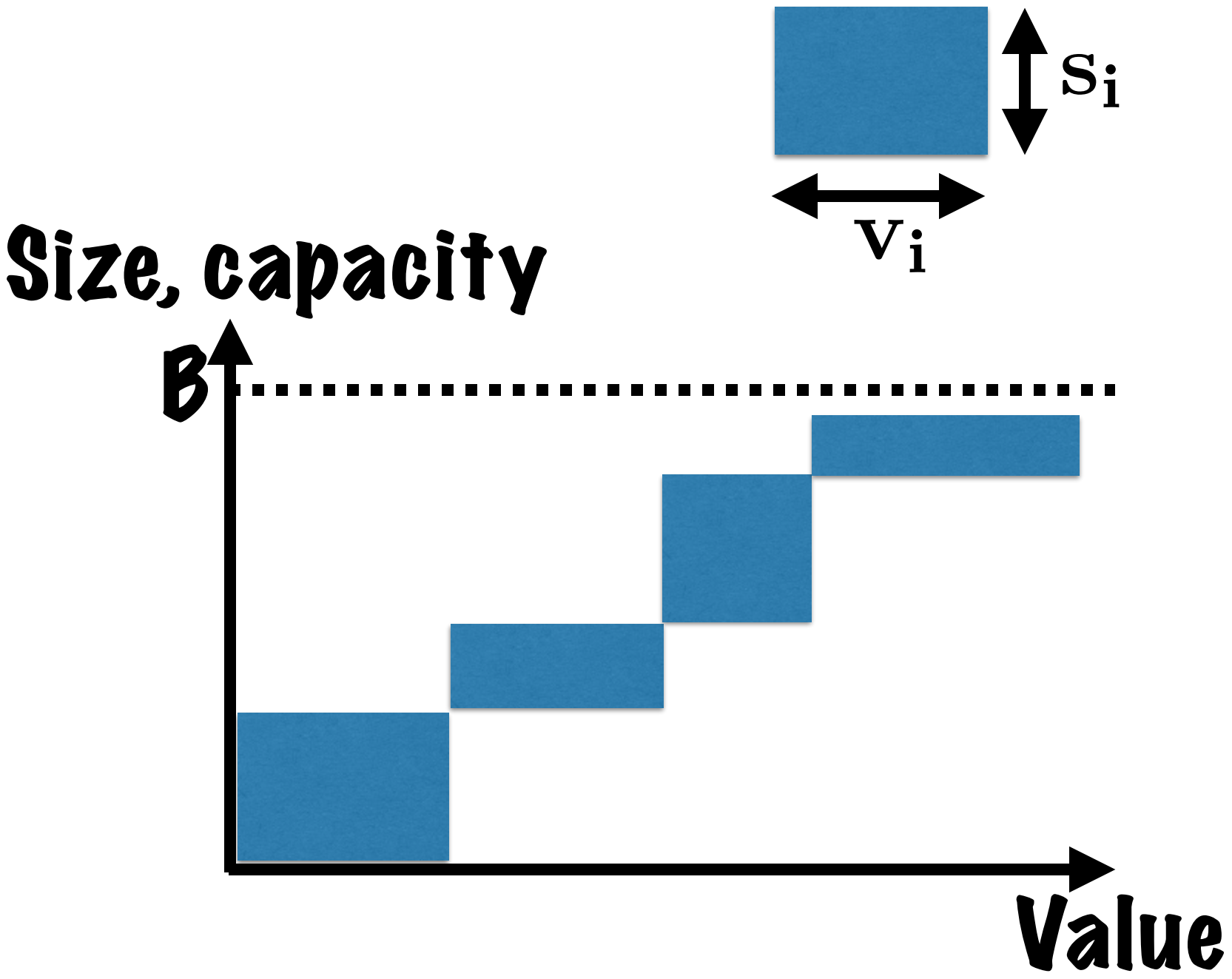
10

The knapsack problem

Given: capacity B knapsack, n items,
item i has size s_i and value v_i
Place some items in knapsack
Maximize value

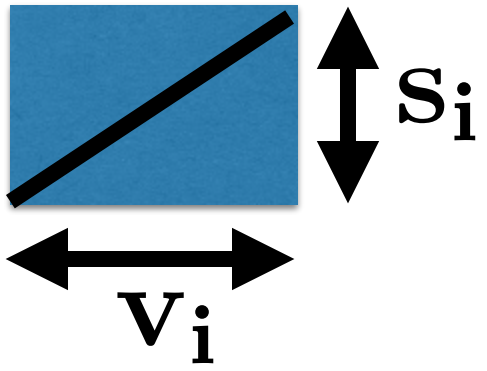
NP-hard

A general recipe:
for intuition,
graphical representation


s_i
v_i
Size, capacity
B
Value

Desire: small size, high value

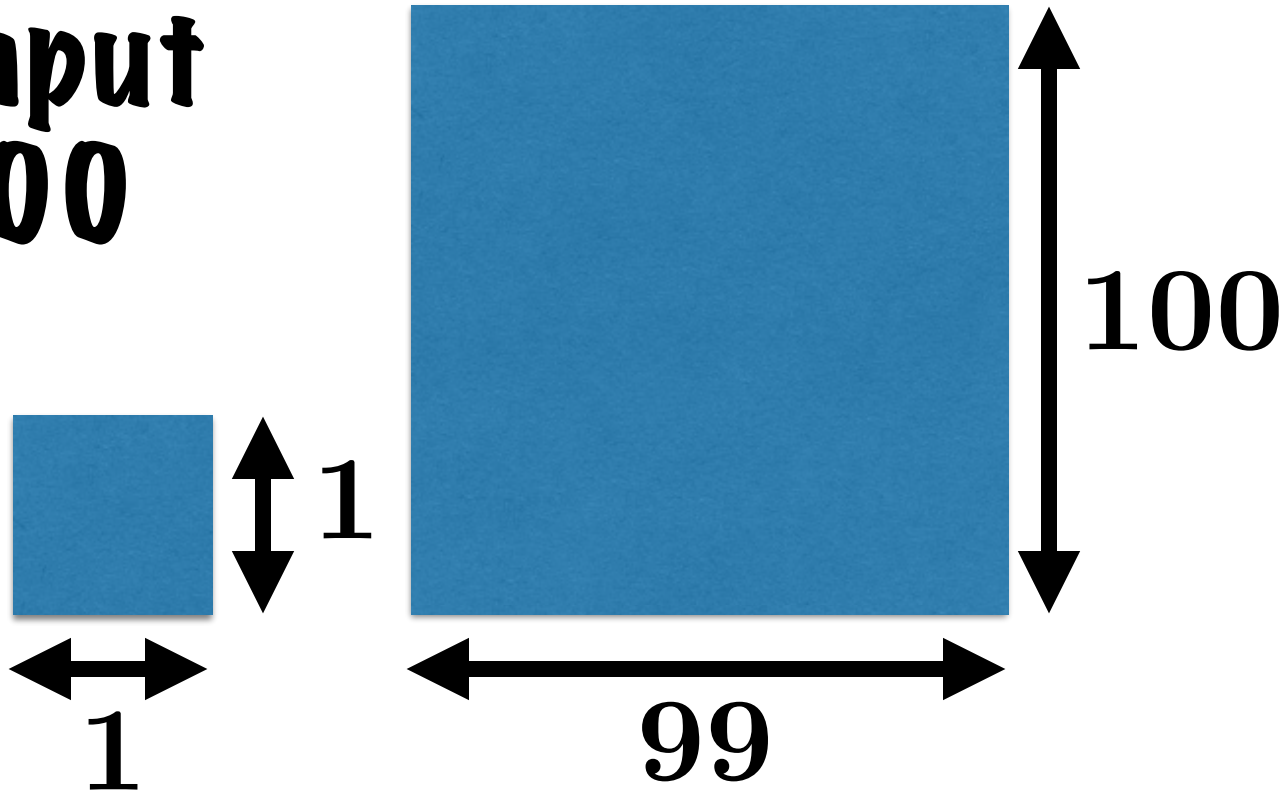
Naive greedy algorithm: take by order of increasing size/value

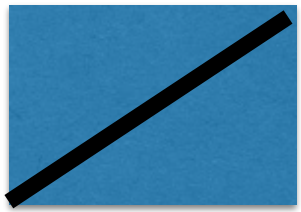


slope $\frac{s_i}{v_i}$

How good is Greedy?

Bad input
B=100





slope $\frac{s_i}{v_i}$

Greedy is bad

Another general recipe: for intuition, try special cases

If all items have the same size...

Greedy is good.

If all items have the same value...

Greedy is good.

If all items have size=value...

Knapsack and rounding

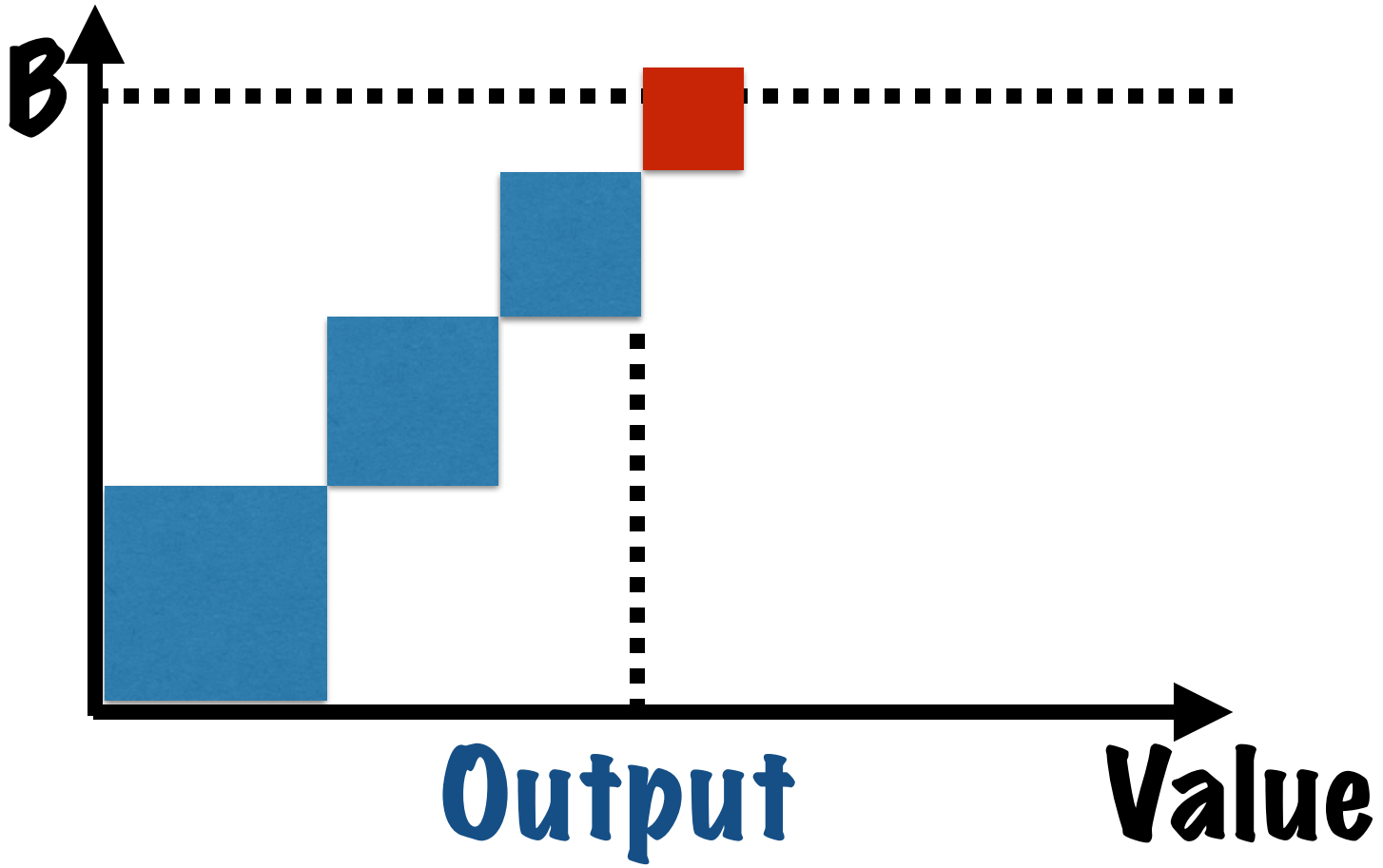
Knapsack and rounding

A greedy algorithm for special case size=value

Order items by decreasing value.

How good is that?

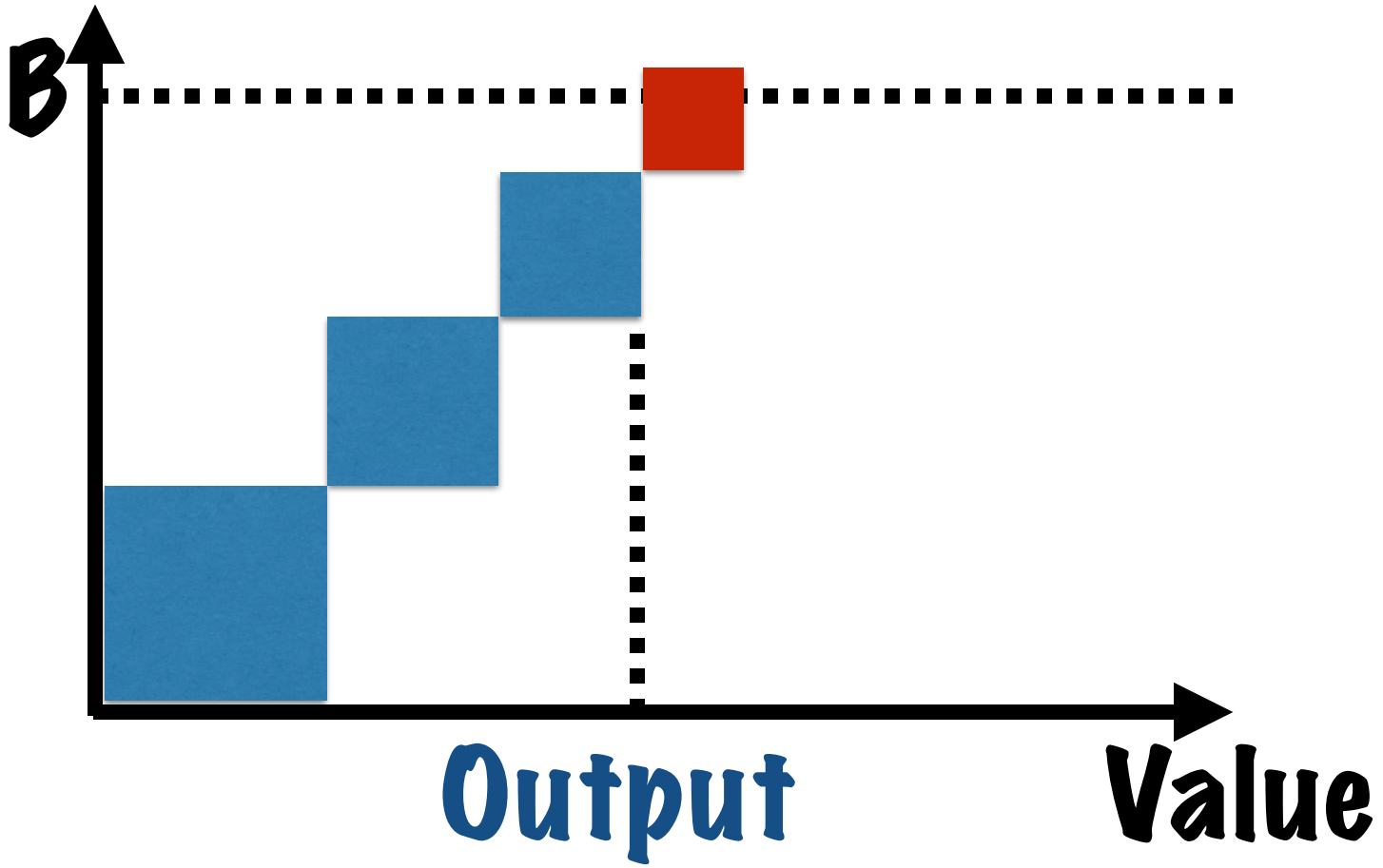
Observe: $\mathrm{OPT} \leq \mathrm{B}$



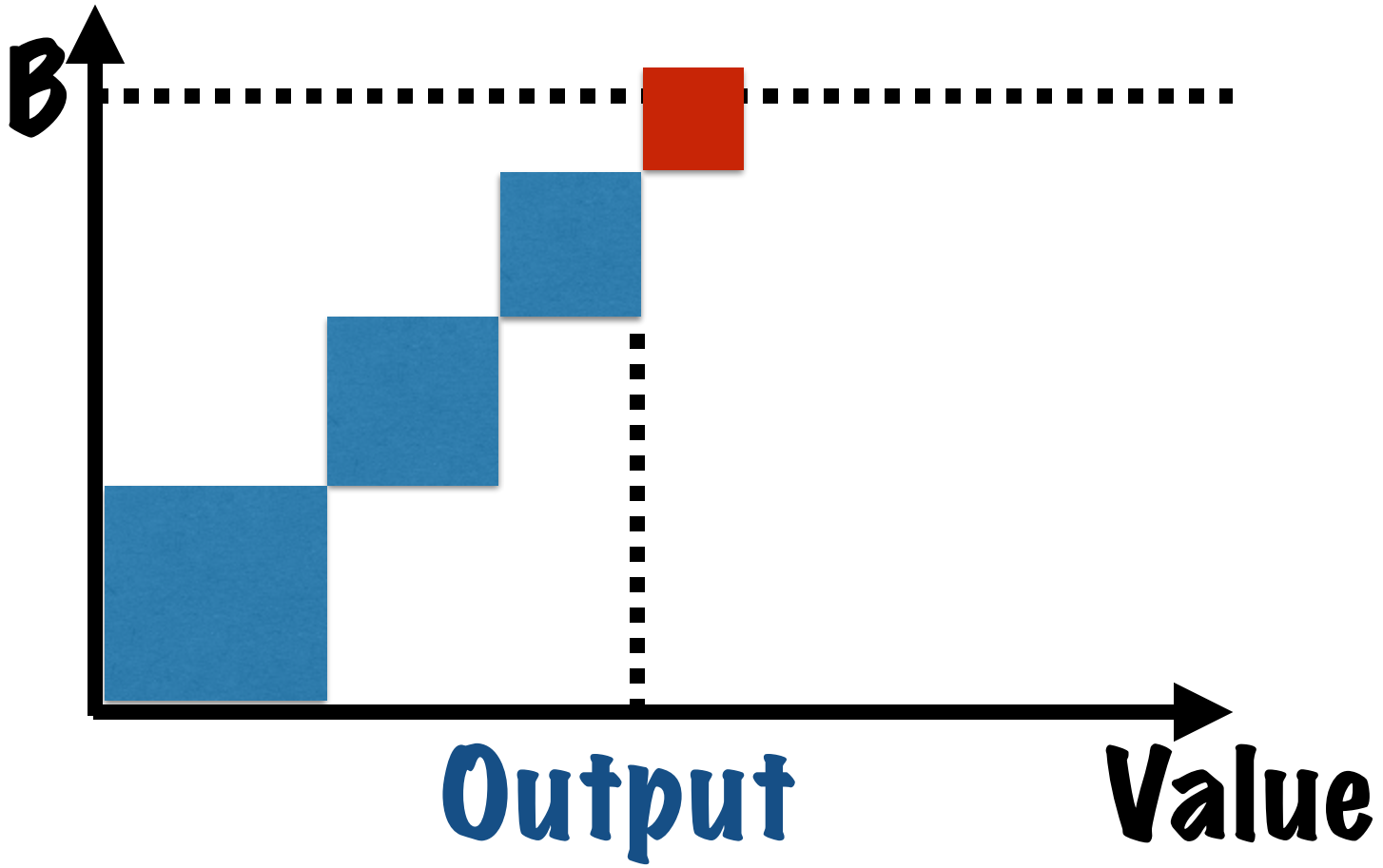
Observe: Output+1 item $> B$

Can assume:

Output has at least 1 item



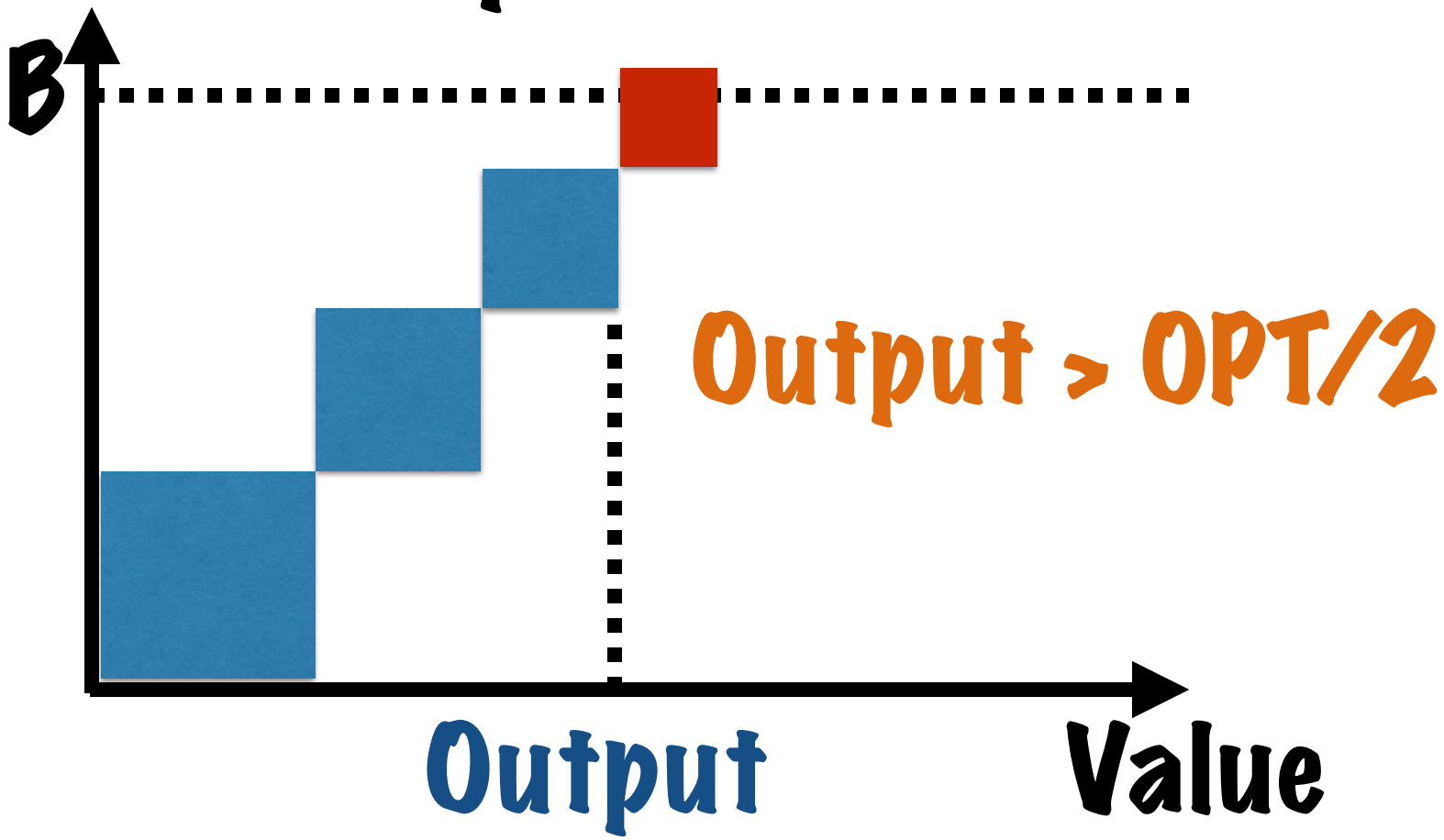
Observe: first item in output is better than item not in output



Combine: Output+red item > B

First output item > red item

Output > B/2



Theorem:
in special case size=value,
greedy is a 2-approximation.

Can we do better?

Knapsack and rounding

Knapsack and rounding

Knapsack special special case

All items have size = value

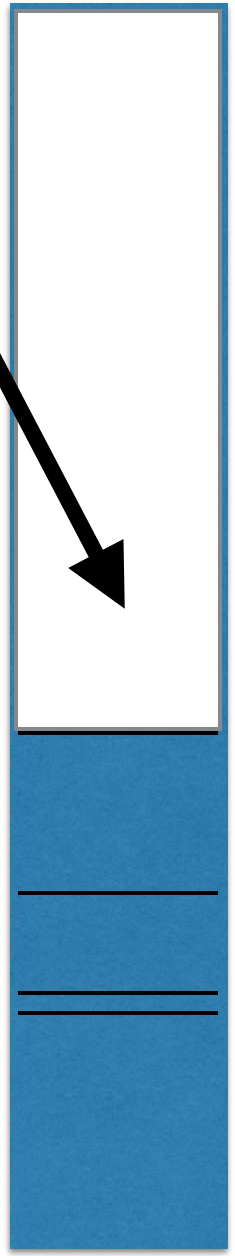
$$\in \{1, 2, \ldots, \mathrm{B}\}$$

and in addition

B is a "small" integer

Dynamic programming

add stuff here



interface

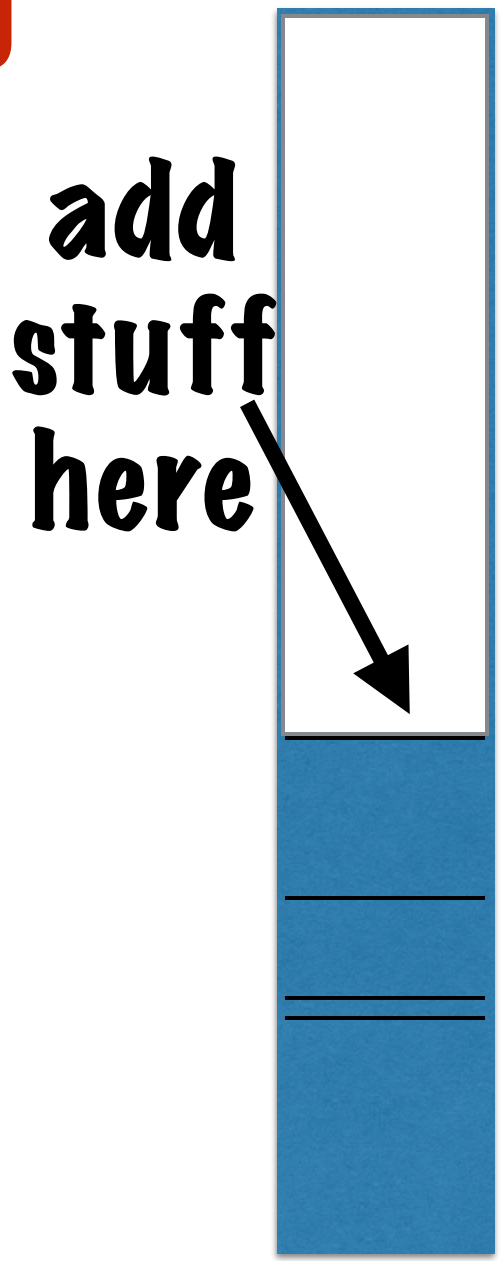
Given partial solution for first i items, what to remember to complete solution optimally?

Dynamic programming

Q: What to remember?

A: remember

total value so far



A[i,v]=whether
v achievable with
subset of first i items

Q: v achievable with
subset of first i items iff...
A: ...it depends on
whether subset contains i

If not:
  v must be reached
  with first i-1 items
If yes:
  $v - v_i$ must be reached
  with first i-1 items

A[i,v]=whether
v achievable with
subset of first i items

$$\mathbf{A[i,v] =}$$
$$\mathbf{A[i-1,v]\ or}$$
$$\mathbf{((v \geq v_i)\ and\ A[i-1,v-v_i])}$$

Algorithm

For $v = 0 \ldots B : A[1, v] \leftarrow$ false

$A[1, v_1] \leftarrow$ true, $A[1, 0] \leftarrow$ true

For $i = 2 \cdots n$,

  For $v = 0 \ldots v_i - 1 : A[i, v] \leftarrow A[i-1, v]$

  For $v = v_i \ldots B$:

    $A[i, v] \leftarrow A[i-1, v]$ or $A[i-1, v - v_i]$

Output $\max\{v : A[n, v]$ is true $\}$

Knapsack and rounding

Knapsack and rounding

Less special special case: values are small integers

All values

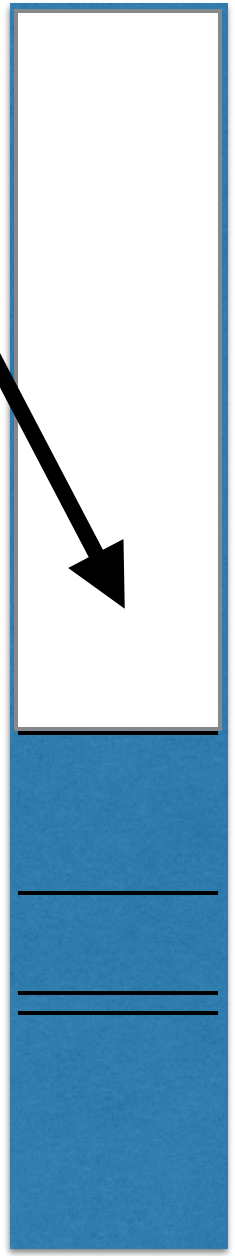
$$\in \{1, 2, \ldots, \mathrm{N}\}$$

N: "small" integer

Extend previous ideas

Dynamic programming

add stuff here



interface

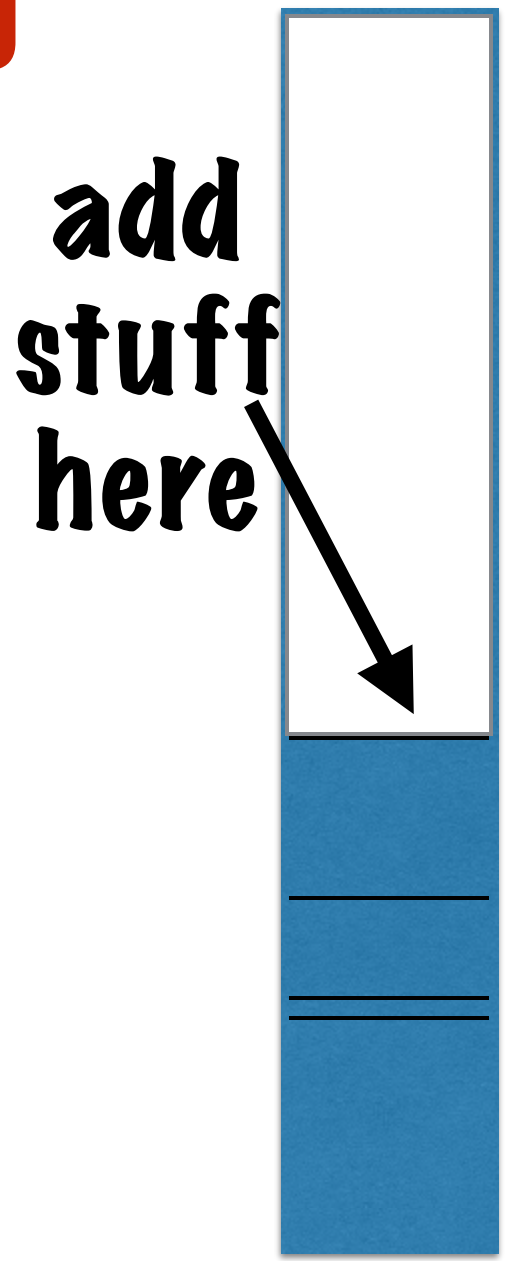
Given partial solution for first i items, what to remember to complete solution optimally?

Dynamic programming

Q: What to remember?

~~A[i,v]=whether v achievable with subset of first i items~~

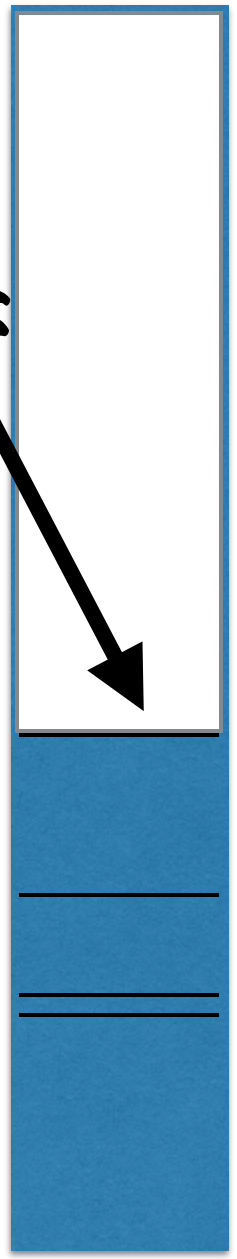
A[i,v]=must remember size



Dynamic programming

Q: What to remember?

add stuff here



A[i,v]=min size achievable for subset of first i items of value v

Q: v,s achievable with subset of first i items iff...

A: ...it depends on whether subset contains i

If not:

v,s reached with first i-1 items

If yes:

$v - v_i, s - s_i$ reached with first i-1 items

Dynamic program

A[i,v]=min size achievable for subset of first i items of value v

$$\textbf{If } \mathbf{v} \geq \mathbf{v_i}$$
$$\textbf{then } \mathbf{A[i,v]} =$$
$$\min(\mathbf{A[i-1,v]},$$
$$\mathbf{A[i-1,v-v_i]} + \mathbf{s_i})$$
$$\textbf{else } \ldots$$

For $v = 0 \ldots nN : A[1, v] \leftarrow B + 1$

$A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0$

For $i = 2 \cdots n$,

  For $v = 0 \ldots v_i - 1 : A[i, v] \leftarrow A[i - 1, v]$

  For $v = v_i, v_i + 1, \ldots, nN :$

    $A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)$

Output $\max\{v : A[n, v] \leq B\}$

Runtime: O(n^2N)

Q: What's the main idea?

For $v = 0 \ldots nN : A[1, v] \leftarrow B + 1$
$A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0$
For $i = 2 \cdots n$,
  For $v = 0 \ldots v_i - 1 : A[i, v] \leftarrow A[i - 1, v]$
  For $v = v_i, v_i + 1, \ldots, nN :$
    $A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)$
Output $\max\{v : A[n, v] \leq B\}$

A: The definition of A[i,v]

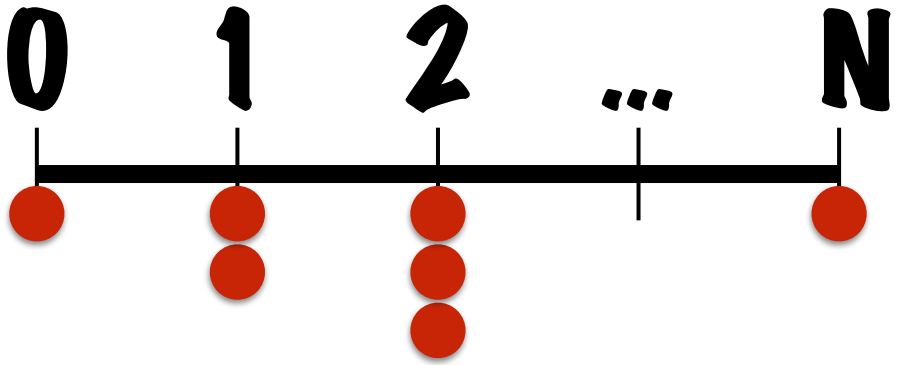
dynamic program key step = DP table definition

Knapsack and rounding

Knapsack and rounding

General case: algorithm

We already have an algorithm when values are small integers



Idea:
modify input
so that values are
small integers

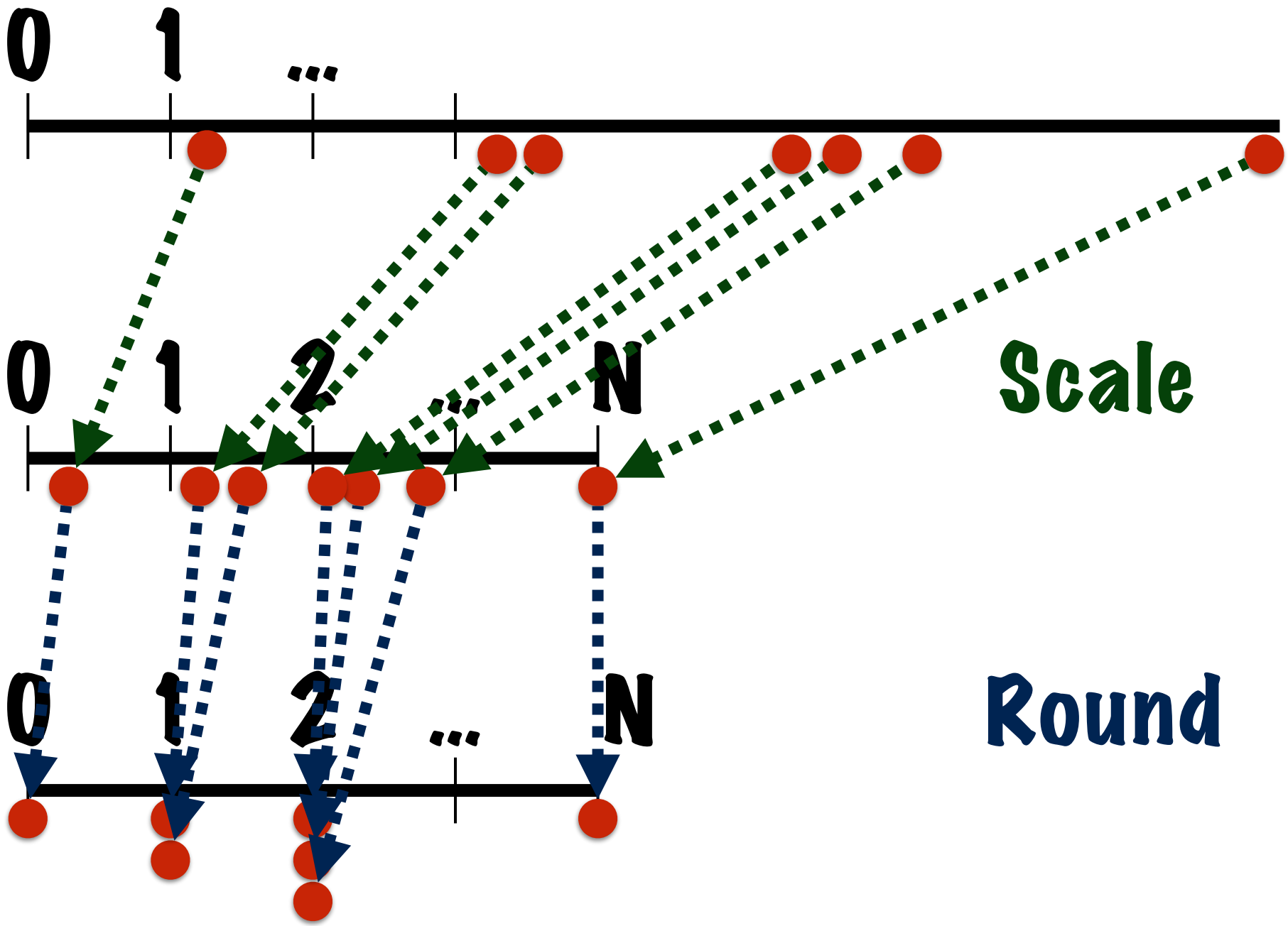
Q : How to modify input so that values are small integers?

A : Scale and round!

Discard items that don't fit

1. Multiply each value by something so that values are small
2. Round values to integers

Dynamic program for the scaled and rounded problem


0
1
...
0
1
2
...
N
Scale
0
1
2
...
N
Round

Effect of scaling:
Max scaled value = N

Multiply by $\alpha = \frac{\mathbf{N}}{\max \mathbf{v_i}}$

How do we pick N?

- small enough that the runtime is polynomial
- large enough that the resulting set of items has value close to the optimum

Algorithm Scale

1. Discard items not fitting
2. Let N=100 n
3. Let $v_i' \leftarrow \lfloor v_i \times \frac{N}{\max_j v_j} \rfloor$
4. Apply DP to $B, (s_i, v_i')_i$
5. Output corresponding items

Runtime

$$\textbf{For } \mathbf{v = 0 \ldots nN : A[1, v] \leftarrow B + 1}$$
$$\mathbf{A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0}$$
$$\textbf{For } \mathbf{i = 2 \cdots n},$$
$$\quad \textbf{For } \mathbf{v = 0 \ldots v_i - 1 : A[i, v] \leftarrow A[i - 1, v]}$$
$$\quad \textbf{For } \mathbf{v = v_i, v_i + 1, \ldots, nN :}$$
$$\quad\quad \mathbf{A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)}$$
$$\textbf{Output } \max\{\mathbf{v : A[n, v] \leq B}\}$$

$$\mathbf{N = 100 \times n \implies n^2 N = 100 \times n^3}$$

Knapsack and rounding

Knapsack and rounding

Analysis

- S: output items
- DP: S optimal for scaled rounded input
- S*: optimal items
- Scaling: S* optimal for scaled unrounded input

Relate S for original and modified input value

output value

$$\mathbf{Value(S)} = \sum_{\mathbf{S}} \mathbf{v_i}$$

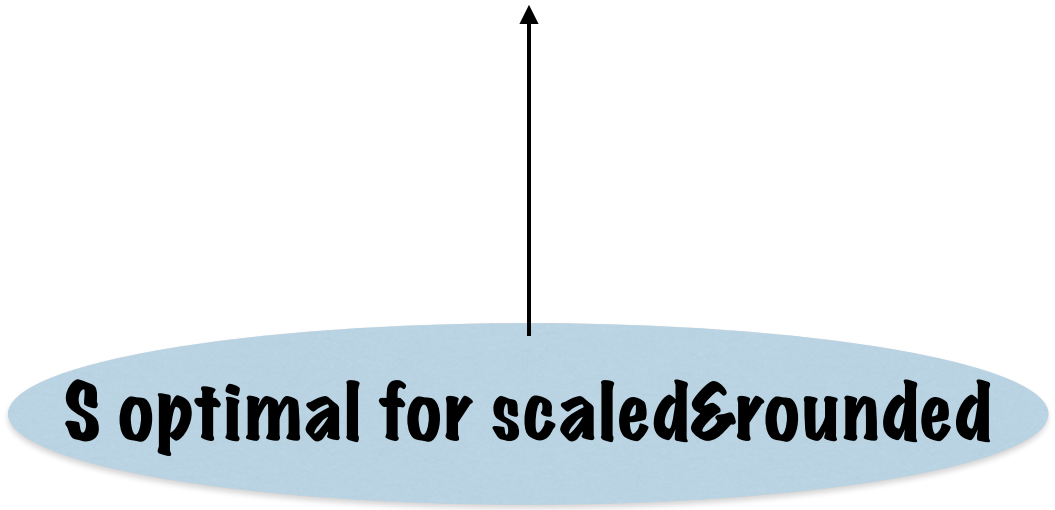
$$= \frac{\mathbf{1}}{\alpha} \sum_{\mathbf{S}} (\alpha \mathbf{v_i})$$

$$\geq \frac{\mathbf{1}}{\alpha} \sum_{\mathbf{S}} \mathbf{v'_i}$$

value for scaled&rounded input

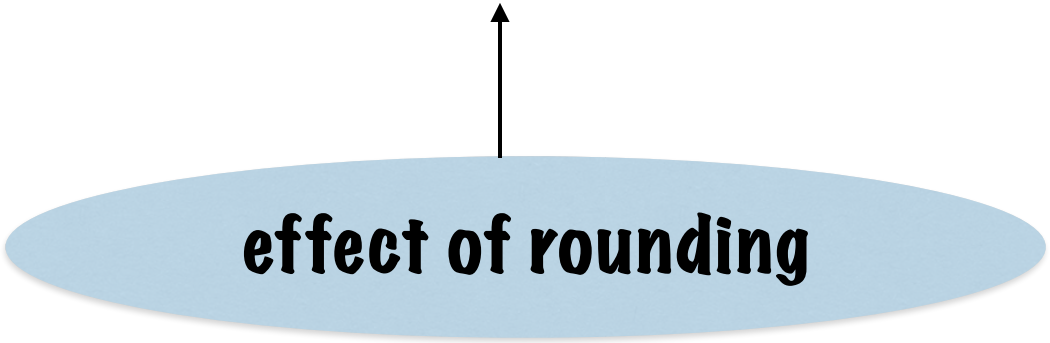
Relate S to S*
on modified input

$$\sum_S v'_i \geq \sum_{S^*} v'_i$$



Relate S* for original and modified input value

$$\mathbf{v_i'} > \alpha \mathbf{v_i} - 1$$



Sum:

$$\sum_{\mathbf{S}^*} \mathbf{v_i'} > \alpha \sum_{\mathbf{S}^*} \mathbf{v_i} - \mathbf{n}$$

Combine and substitute:

$$\begin{aligned}
\mathbf{Value(S)} &\geq \frac{1}{\alpha}\sum_{\mathbf{S}} \mathbf{v_i'} \\
&\geq \frac{1}{\alpha}\sum_{\mathbf{S^*}} \mathbf{v_i'} \\
&\geq \frac{1}{\alpha}[\alpha \sum_{\mathbf{S^*}} \mathbf{v_i} - \mathbf{n}] \\
&= \mathbf{OPT} - \frac{\mathbf{n}}{\alpha} \\
&= \mathbf{OPT} - \frac{\mathbf{n} \times \max \mathbf{v_i}}{\mathbf{N}}
\end{aligned}$$

Lower bound OPT

Discarded items that don't fit:

$$\mathbf{OPT} \geq \max \mathbf{v_i}$$

Wrapping up

$$\mathbf{Value(S) \geq OPT - \frac{n}{N} OPT}$$

$$\mathbf{N = 100 \times n}$$

$$\mathbf{Value(S) \geq .99 \times OPT}$$

Theorem: Solution to knapsack with value at least .99 OPT and runtime O(poly(n))

Knapsack and rounding

Knapsack and rounding

Approximation schemes

The general algorithm

1. Let $N = 1000\,n$
2. Let $v'_i \leftarrow \left\lfloor v_i \times \frac{N}{\max_j v_j} \right\rfloor$
3. Apply DP to $B, (s_i, v'_i)_i$
4. Output corresponding items

Theorem: this gives a solution
to knapsack
with value at least .999 OPT
and with runtime O(poly(n))

Approximation scheme : family of algorithms:
One for each $\epsilon > 0$
runtime polynomial in input
output value is near-optimal:

$$|\text{Value(Output)-OPT}| \leq \epsilon \times \text{OPT}$$

Theorem: knapsack has an approximation scheme

How ϵ comes in

Q: The smaller ϵ, the closer to OPT. Why not let it go to 0 and find the exact OPT in O(poly(n))?

A: Runtime increases as ϵ goes to zero. Runtime has N with $N=n/\epsilon$ so it goes to infinity.

Method:

1. Simplify the input
2. Design algorithm for "simple" inputs

Knapsack and rounding