

Lecture 12

The Knapsack problem

Chinmay Nirkhe | CSE 421 Spring 2025



Previously in CSE 421...

General dynamic programming algorithm

- **Iterate through subproblems:** Starting from the “smallest” and building up to the “biggest.” For each one:
 - Find the optimal value, using the previously-computed optimal values to smaller subproblems.
 - Record the choices made to obtain this optimal value. (If many smaller subproblems were considered as candidates, record which one was chosen.)
- **Compute the solution:** We have the value of the optimal solution to this optimization problem but we don't have the actual solution itself. Use the recorded information to actually reconstruct the optimal solution.

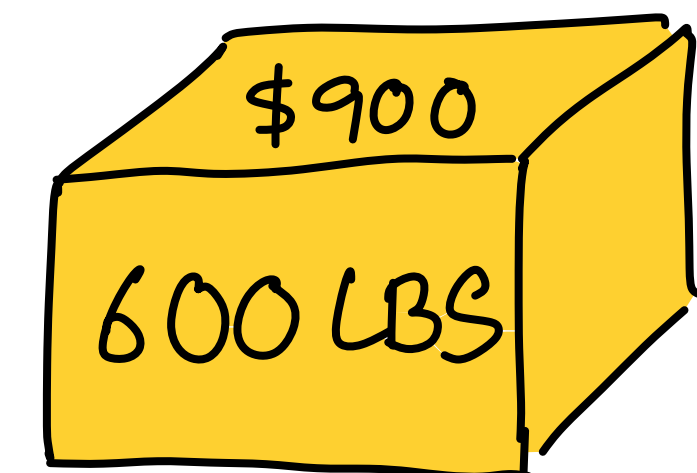
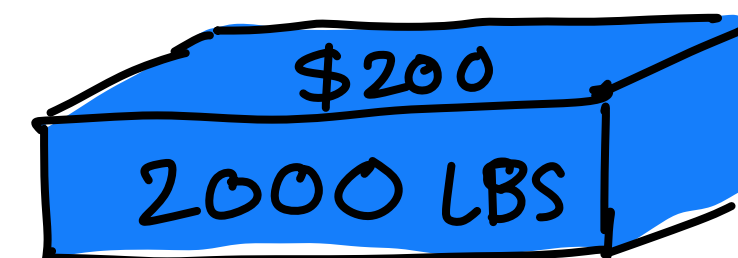
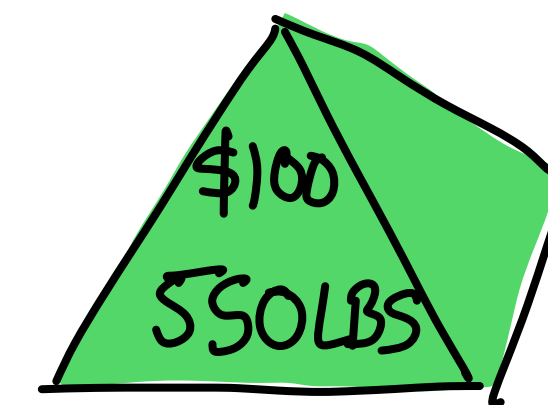
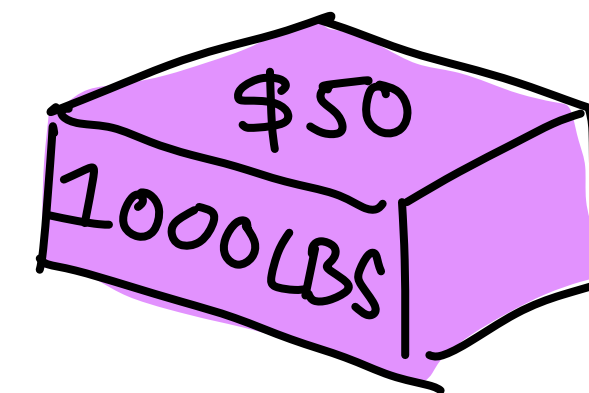
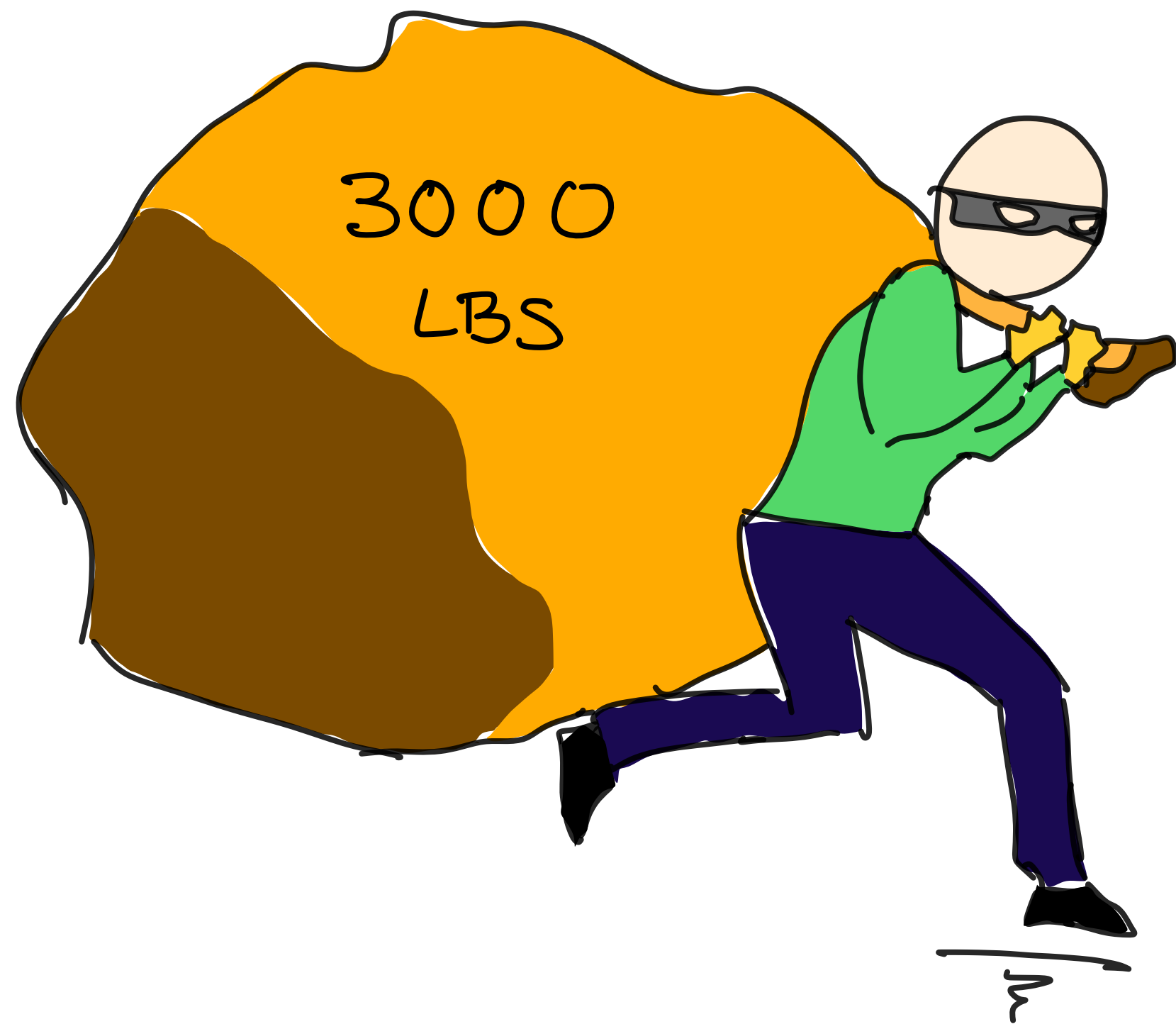
General dynamic programming runtime

$$\text{Runtime} = (\text{Total number of subproblems}) \times \left(\begin{array}{l} \text{Time it takes to solve problems} \\ \text{given solutions to subproblems} \end{array} \right)$$

Today

The Knapsack problem

Maximize the items grabbed subject to the weight constraint of the bag.



The Knapsack problem

- **Input:** Items with **integer** weights $w_1, \dots, w_n \in \mathbb{N}$ and values $v_1, \dots, v_n \in \mathbb{N}$ and a max weight $W \in \mathbb{N}$

- **Output:** Subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and maximizing $\sum_{i \in S} v_i$.

Let $V = \sum_{i=1}^n v_i$, the max value of any set.

- **Brute force solution:** Check all 2^n possible S and choose the optimal S amongst those satisfying the weight constraint.

- **Runtime:** $O(n \cdot 2^n \log VW)$ $= \log V + \log W$
arithmetic complexity of adding numbers $\leq W$ or $\leq V$.

A better dynamic programming algorithm

- **Observation:** Either item i is included in S or it is not
- Defining an appropriate subproblem
- Let $S(i, W')$ be the optimal subset $S \subseteq \{1, \dots, i\}$ such that S 's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value
- **Base cases:** $S(\cdot, 0) = S(0, \cdot) = \emptyset$, $V(\cdot, 0) = V(0, \cdot) = 0$.

A better dynamic programming algorithm

- Let $S(i, W')$ be the optimal subset $S \subseteq \{1, \dots, i\}$ such that S 's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value
- To calculate $S(i, W')$, if we include item i
 - Value of bag is at least v_i and bag now has remainder available weight $W' - w_i$
 - Need to recursively choose between items $\{1, \dots, i - 1\}$
- Else
 - Bag still has remainder available weight W'
 - Need to recursively choose between items $\{1, \dots, i - 1\}$

A better dynamic programming algorithm

- Let $S(i, W')$ be the optimal subset $S \subseteq \{1, \dots, i\}$ such that S 's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value

$$V(i, W') = \max \left\{ \begin{array}{l} V(i-1, W' - w_i) + v_i, \\ V(i-1, W') \end{array} \right\}$$

- Depending on maximization, $S(i, W') = S(i-1, W' - w_i) \cup \{i\}$ or $S(i, W') = S(i-1, W')$ respectively.

Memoization for Knapsack

Table of $V(i, w')$:

0					
0				$V(i, w')$	
0					
0					
0	0	0	0	0	0

Memoization for Knapsack

Table of $V(i, W')$:

$$V(i, W') = \max \left\{ \begin{array}{l} V(i-1, W' - w_i) + v_i, \\ V(i-1, W') \end{array} \right\}$$

0					
0				$V(i, W')$	
0		$V(i-1, W' - w_i)$		$V(i-1, W')$	
0					
0	0	0	0	0	0

Memoization for Knapsack

Table of $V(i, W')$:

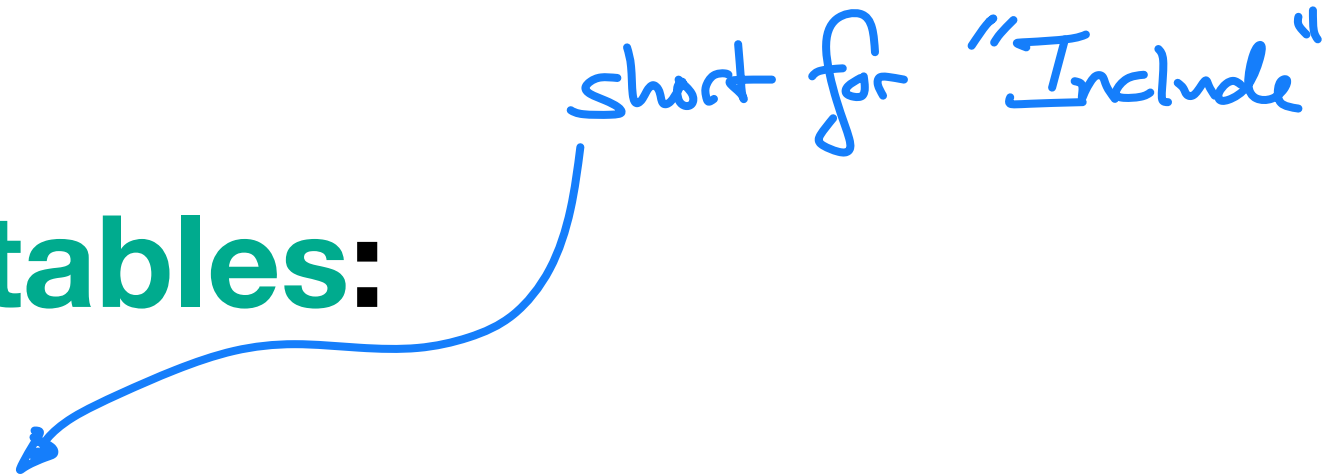
0					
0				$V(i, W')$	
0		$V(i-1, W'-w_i)$		$V(i-1, W')$	
0					
0	0	0	0	0	0

$$V(i, W') = \max \left\{ \begin{array}{l} V(i-1, W' - w_i) + v_i, \\ V(i-1, W') \end{array} \right\}$$

Each edge \downarrow or \swarrow goes from (i, \cdot) to $(i-1, \cdot)$.

Record for every (i, W') if we are going to include or exclude item i .

Knapsack dynamic programming algorithm

- **Generate tables:**  short for "Include"
 - Let V, Inc be $(n + 1) \times (W + 1)$ sized tables and set $V(0, \cdot) = V(\cdot, 0) \leftarrow 0$.
 - For i from 1 to n , W' from 1 to W
 - If $V(i - 1, W') > V(i - 1, W' - w_i) + v_i$
 - Then, set $V(i, W') \leftarrow V(i - 1, W')$ and set $\text{Inc}(i, W') = \text{false}$
 - Else, set $V(i, W') \leftarrow V(i - 1, W' - w_i) + v_i$ and set $\text{Inc}(i, W') = \text{true}$

Knapsack dynamic programming algorithm

- **Find optimal Knapsack:**
 - Set $(i, W') \leftarrow (n, W)$. Set $S \leftarrow \emptyset$.
 - While $i \neq 0$,
 - If $\text{Inc}(i, W') = \text{true}$,
 - Then, $S \leftarrow S \cup \{i\}$ and $(i, W') \leftarrow (i - 1, W' - w_i)$.
 - Else, $(i, W') \leftarrow (i - 1, W')$.
 - Return S .

Knapsack dynamic programming algorithm

Runtime analysis

- Tables are of size $O(nW)$ and computing each entry takes $O(\log VW)$ time given past entries
- Total compute time of tables is $O(nW)$
- To find the set S , path walks from (i, \cdot) to $(i - 1, \cdot)$ each step. The path has length $\leq n$.
- Computing S takes time $O(n)$.
- **Total computation time:** $O(nW \log VW)$.

Knapsack runtime

- The input for Knapsack is usually written in **binary** with each item weight w_i expressed with $O(\log W)$ bit numbers and value with $O(\log V)$ bit numbers
- Total input length is $O(n \log VW)$
- Runtime of Knapsack dynamic programming algorithm is exponential in the input length
- This is expected. The decision version of Knapsack is a NP-complete problem. We do not expect an efficient algorithm for Knapsack.


polynomial time in the input
length

Approximation algorithms

- We've only alluded to NP-completeness so far, but the NP-completeness of the Knapsack problem means that there is *no* algorithm for optimizing Knapsack that runs in time

$$O(\text{poly}(n \log W)) = O(n^c \text{polylog } W)$$

- Instead we will have to turn to **approximation algorithms**
- Given a Knapsack problem $(v_1, \dots, v_n, w_1, \dots, w_n, W)$, let OPT be the optimal value of subset of items weighing $\leq W$:

$$\text{OPT} = V(n, W)$$

Approximation algorithms

- Instead we will have to turn to **approximation algorithms**
- Given a Knapsack problem $(v_1, \dots, v_n, w_1, \dots, w_n, W)$, let OPT be the optimal value of subset of items weighing $\leq W$:

$$\text{OPT} = V(n, W)$$

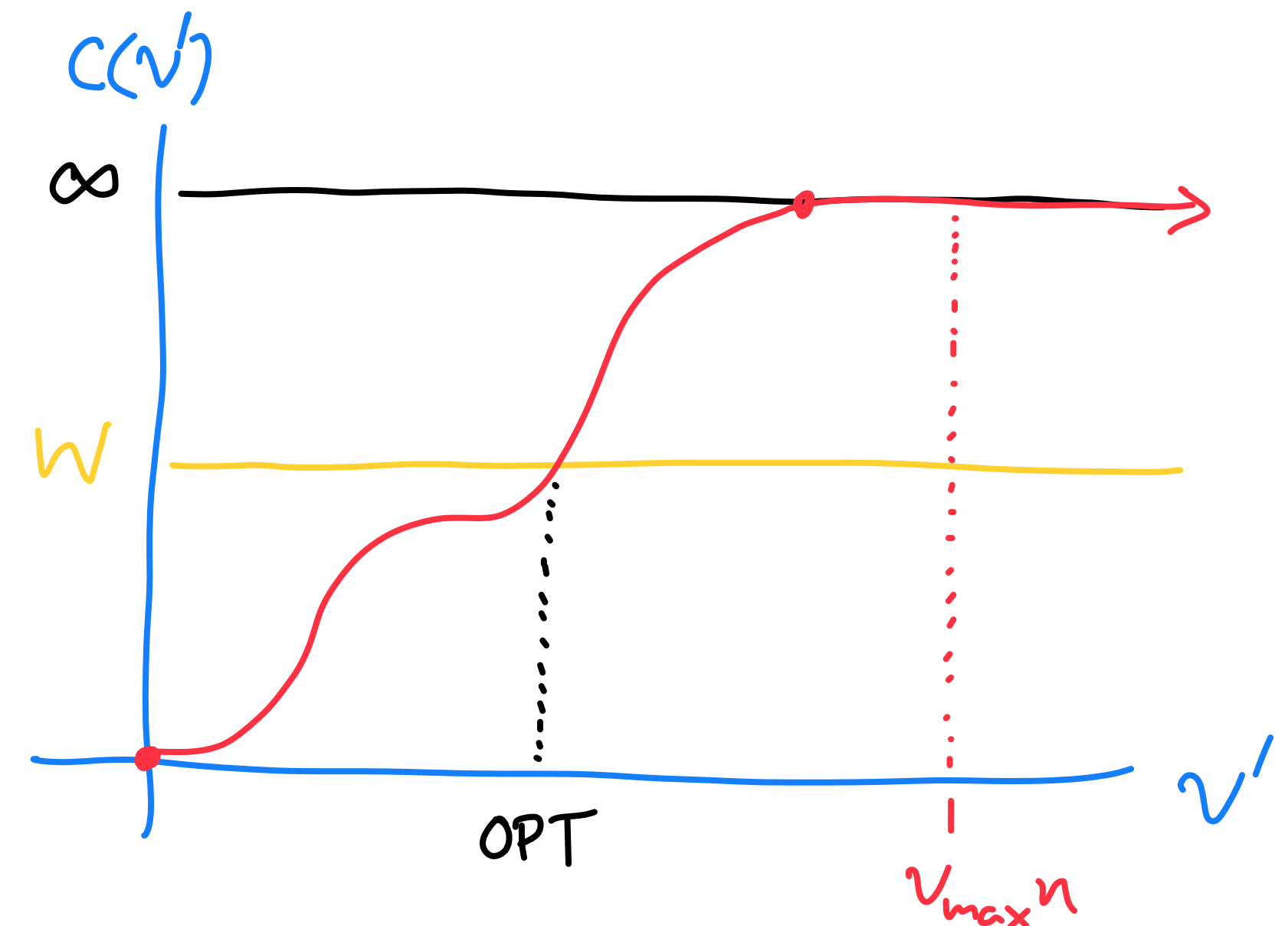
- An alg. \mathcal{A} is an **ϵ -approximation alg.** if \mathcal{A} always outputs a subset \tilde{S} such that (a) $\text{weight}(\tilde{S}) \leq W$ and (b) $\text{value}(\tilde{S}) \geq (1 - \epsilon) \cdot \text{OPT}$.

Knapsack approximation algorithm

- **Theorem:** For every $\epsilon > 0$, there exists an ϵ -approximation alg. for n -item Knapsack that runs in time $O\left(\frac{n^3 \log(VW)}{\epsilon}\right)$.
- The construction will be another dynamic programming algorithm.
- However, we will have to make adjustments to not depend on W .

A different DP algorithm for (exact) Knapsack

- Assume that $0 \leq w_i \leq W$ for all items.
- Let $v_{\max} = \max_i v_i$. Then, $v_{\max} \leq \text{OPT} \leq V$
- **Define:** $C(V')$ to be the minimum weight of a set S such that $\text{value}(S) \geq V'$
 - Let $C(V') = \infty$ if no set S exists of this value.
 - Base case of $C(0) \leq 0$
 - $C(V') = \infty$ for $V' > V$
 - $C(V')$ is monotonically increasing
- Then, Knapsack solution $\text{OPT} = \max \text{value } V' \text{ s.t. } C(V') \leq W$



A slightly different optimization

- $C(V')$ can be “morally” seen as a dual problem to maximization $V(W')$
- **Define:** $C(i, V')$ as the minimum weight of a set S such that $\text{value}(S) \geq V'$ using items only $\{1, \dots, i\}$
 - This new subproblem has a recursive definition similar to our previous example
- $$C(i, V') = \min \left\{ \begin{array}{l} C(i-1, V'), \\ C(i-1, V' - v_i) + w_i \end{array} \right\}$$
- OPT = the maximum value V' s.t. $C(n, V') \leq W$

A different Knapsack algorithm

- This new algorithm has a table of size $(n + 1) \times V$
- Each entry of the table can be constructed in $O(\log W + \log V)$ time
- Computing OPT after table involves binary searching along $C(n, \cdot)$
 - OPT = the maximum value V' s.t. $C(n, V') \leq W$
 - Requires $O(\log V(\log VW))$ total compute
- Yields a total runtime of $O(nV \log VW)$ runtime
 - No exponential dependence in terms of $\log W$
 - However, exponential dependence in terms of $\log V$

An approximation algorithm

- **Idea:** Compute $S \leftarrow \text{Knapsack}(\{\tilde{v}_i\}, \{w_i\}, W)$ for $\tilde{v}_i = \frac{v_i}{Z}$ and $Z = \frac{\epsilon v_{\max}}{n}$.
 - Since the weights are reduced, the runtime is shorter!
- **Runtime:** $O\left(\frac{nV \log VW}{Z}\right) = O\left(\frac{n^2V \log VW}{\epsilon v_{\max}}\right) \leq O\left(\frac{n^3 \log VW}{\epsilon}\right)$
- **Claim:** S is a feasible solution and $\text{value}(S) \geq (1 - \epsilon)\text{OPT}$.

An approximation algorithm

- **Idea:** Compute $S \leftarrow \text{Knapsack}(\{\tilde{v}_i\}, \{w_i\}, W)$ for $\tilde{v}_i = \frac{v_i}{Z}$ and $Z = \frac{\epsilon V_{\max}}{n}$.
 - Since the weights are reduced, the runtime is shorter!

For intuition, say $Z = 2^k$ for some k .

Then if we express v_i in binary: v_i

1	0	1	1	0	1	1	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---

\tilde{v}_i

1	0	1	1	0	1	1						
---	---	---	---	---	---	---	--	--	--	--	--	--

$\underbrace{\hspace{10em}}_k$

Keep only the significant digits. This alg. is morally rounding.

An approximation algorithm

Let $\tilde{v}_i = \left\lfloor \frac{v_i}{Z} \right\rfloor$ for $Z = \frac{\epsilon v_{\max}}{n}$. Output $S \leftarrow \text{Knapsack}(\{\tilde{v}_i\}, \{w_i\}, W)$.

Claim: S is a feasible solution to the original problem.

Proof: Since the weights $\{w_i\}$ and limit W are the same in both problems,

$$\text{then } \sum_{i \in S} w_i \leq W.$$

An approximation algorithm

Let $\tilde{v}_i = \left\lfloor \frac{v_i}{Z} \right\rfloor$ for $Z = \frac{\epsilon v_{\max}}{n}$. Output $S \leftarrow \text{Knapsack}(\{\tilde{v}_i\}, \{w_i\}, W)$.

Let $\text{value}(S) = \sum_{i \in S} v_i$, $\widetilde{\text{value}}(S) = \sum_{i \in S} \tilde{v}_i$.

Let O be the optimal sol. to $\text{Knapsack}(\{v_i\}, \{w_i\}, W)$.
So, $\text{OPT} = \text{value}(O)$.

Claim: $\text{value}(S) \geq (1 - \epsilon) \text{OPT}$.

An approximation algorithm

Claim: $\text{value}(S) \geq (1 - \epsilon) \text{OPT}$.

Proof: For any item i , $v_i - Z \tilde{v}_i = Z \left(\frac{v_i}{Z} - \left\lfloor \frac{v_i}{Z} \right\rfloor \right) \leq Z$.

Since O has $\leq n$ items, $\text{OPT} - Z \widetilde{\text{value}}(O) = \sum_{i \in S} v_i - Z \tilde{v}_i \leq nZ = \frac{\epsilon V}{n}$

$$Z \widetilde{\text{value}}(O) \geq \text{OPT} - \epsilon V_{\max} \geq (1 - \epsilon) \text{OPT}. \quad (1)$$

Next, $\widetilde{\text{value}}(S) \underset{(2)}{\geq} \widetilde{\text{value}}(O)$ since S is optimal sol. to $\text{Knapsack}(\{\tilde{v}_i\}, \{w_i\}, W)$

$$\text{So, } \text{value}(S) \geq Z \widetilde{\text{value}}(S) \underset{(2)}{\geq} Z \widetilde{\text{value}}(O) \underset{(1)}{\geq} (1 - \epsilon) \text{OPT}. \quad \square$$

Structure of approx. DP algorithm

- We came up with two DP algorithms for **exact** Knapsack based on the following recursive definitions
 - $V(i, W') = \max \text{ value with items } S \subseteq \{1, \dots, i\} \text{ s.t. } \text{weight}(S) \leq W'$
 - $C(i, V') = \min \text{ weight with items } S \subseteq \{1, \dots, i\} \text{ s.t. } \text{value}(S) \geq V'$
- Approx. alg. by rounding values $\tilde{v}_i = \lfloor v_i/Z \rfloor$ and running second alg.
- Is there an approx. alg. by rounding $\tilde{w}_i = \lfloor w_i/Z \rfloor$, $\tilde{W} = \lfloor W/Z \rfloor$ and running the first alg.?
 - Doing this will yield *some* subset $S \subseteq \{1, \dots, n\}$
 - Trouble is that this new set may not be **feasible** for the original weight constraints

Knapsack overview

- **Input:** n items of integer values v_i and weights w_i and weight threshold W .
- **Input length:** $O(n \log VW)$
- **Output:** optimal $S \subseteq [n]$ maximizing $\text{value}(S)$ s.t. $\text{weight}(S) \leq W$
- **Various algorithms:**
 - Brute force alg: Runtime of $O(n2^n \log VW)$
 - DP alg: Runtime $O(nW \log VW)$ or $O(nV \log VW)$
 - ϵ -approx. alg: Runtime $O\left(\frac{n^3 \log VW}{\epsilon}\right)$