Greedy Approximations
The Pricing Method: Vertex Cover
Return of the Knapsack

Algorithm Design & Analysis

Spring 2018
Weighted Set Cover: The Pricing Method

Observations
Observations

- Each \( s \in S \) got a price \( c_s \) (based on first \( S_i \) to cover \( s \))

Weighted Set Cover: The Pricing Method

Then we showed that there was a (small) value \( H \) such that for any set cover \( C^* \),

\[
P \leq H \cdot w(C^*)
\]

So \( P \) is an upper bound on the weight of the greedy solution and (within a small factor of) a lower bound on the weight of every other solution.

This idea of using a pricing method to measure goodness of approximation is quite powerful.

Let's try the same method on a related problem: Vertex Cover.
Observations

- Each $s \in S$ got a price $c_s$ (based on first $S_i$ to cover $s$)
- The weight of the greedy cover
  \[ w(C) = \sum_{S_i \in C} w(S_i) = \sum_{s \in U} c_s = P \]
Weighted Set Cover: The Pricing Method

Observations

- Each \( s \in S \) got a price \( c_s \) (based on first \( S_i \) to cover \( s \))
- The weight of the greedy cover
  \[ w(C) = \sum_{S_i \in C} w(S_i) = \sum_{s \in U} c_s = P \]
- Then we showed that there was a (small) value \( H \) such that for any set cover \( C^* \), \( P \leq Hw(C^*) \)

This idea of using a pricing method to measure goodness of approximation is quite powerful.
Observations

- Each $s \in S$ got a price $c_s$ (based on first $S_i$ to cover $s$)
- The weight of the greedy cover
  \[ w(C) = \sum_{S_i \in C} w(S_i) = \sum_{s \in U} c_s = P \]
- Then we showed that there was a (small) value $H$ such that for any set cover $C^*$, $P \leq Hw(C^*)$
- So $P$ is an upper bound on the weight of the greedy solution and (within a small factor of) a lower bound on the weight of every other solution!
Weighted Set Cover: The Pricing Method

Observations

• Each $s \in S$ got a price $c_s$ (based on first $S_i$ to cover $s$)
• The weight of the greedy cover
  
  \[ w(C) = \sum_{S_i \in C} w(S_i) = \sum_{s \in U} c_s = P \]
• Then we showed that there was a (small) value $H$ such that for any set cover $C^*$, $P \leq Hw(C^*)$
• So $P$ is an upper bound on the weight of the greedy solution and (within a small factor of) a lower bound on the weight of every other solution!

This idea of using a pricing method to measure goodness of approximation is quite powerful
Weighted Set Cover: The Pricing Method

Observations

• Each \( s \in S \) got a price \( c_s \) (based on first \( S_i \) to cover \( s \))

• The weight of the greedy cover
  \[ w(C) = \sum_{S_i \in C} w(S_i) = \sum_{s \in U} c_s = P \]

• Then we showed that there was a (small) value \( H \) such that for any set cover \( C^* \), \( P \leq Hw(C^*) \)

• So \( P \) is an upper bound on the weight of the greedy solution and (within a small factor of) a lower bound on the weight of every other solution!

This idea of using a pricing method to measure goodness of approximation is quite powerful

Let’s try the same method on a related problem: Vertex Cover
The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight.
The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight

Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights)

Perhaps we can do better?
The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight. Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights).

- For $G = (V, E)$, $U = E$ and the sets are $S_v = \{ e \in E : e = \{u, v\} \}$
Approximation Via Reduction: Weighted Vertex Cover

The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight. Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights):

- For $G = (V, E)$, $U = E$ and the sets are $S_v = \{ e \in E : e = \{u, v\} \}$
- A set cover of $U$ by $S_{v_1}, \ldots, S_{v_k}$ corresponds to a vertex cover of $E$ by $v_1, \ldots, v_k$

Perhaps we can do better?
The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight.
Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights).

- For $G = (V, E)$, $U = E$ and the sets are $S_v = \{ e \in E : e = \{ u, v \} \}$
- A set cover of $U$ by $S_{v_1}, \ldots S_{v_k}$ corresponds to a vertex cover of $E$ by $v_1, \ldots, v_k$
- Thus a minimum weight vertex cover of $G$ corresponds to a minimum weight set cover of $U$
Approximation Via Reduction: Weighted Vertex Cover

The Problem: Given a graph $G = (V, E)$ with vertex weights $w_v$, find a vertex cover of low weight.

Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights)

- For $G = (V, E)$, $U = E$ and the sets are $S_v = \{e \in E : e = \{u, v\}\}$
- A set cover of $U$ by $S_{v_1}, \ldots, S_{v_k}$ corresponds to a vertex cover of $E$ by $v_1, \ldots, v_k$
- Thus a minimum weight vertex cover of $G$ corresponds to a minimum weight set cover of $U$
- So GREEDYSETCOVER can be used to get a $O(\log n)$ approximation for VERTEXCOVER
The Problem: Given a graph \( G = (V, E) \) with vertex weights \( w_v \), find a vertex cover of low weight.

Recall that SETCOVER can be used to solve VERTEXCOVER (even with weights):

- For \( G = (V, E) \), \( U = E \) and the sets are \( S_v = \{ e \in E : e = \{u, v\} \} \).
- A set cover of \( U \) by \( S_{v_1}, \ldots S_{v_k} \) corresponds to a vertex cover of \( E \) by \( v_1, \ldots, v_k \).
- Thus a minimum weight vertex cover of \( G \) corresponds to a minimum weight set cover of \( U \).
- So GREEDYSETCOVER can be used to get a \( O(\log n) \) approximation for VERTEXCOVER.
- Perhaps we can do better?
Weighted Vertex Covers via the Pricing Method

Idea: An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.
Weighted Vertex Covers via the Pricing Method

**Idea:** An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.

- The set $\{p_e : e \in E\}$ of prices is *fair* if, for each $v \in V$
  $$\sum_{e=\{u,v\}} p_e \leq w_v \quad (v \text{ is not overcharging})$$

Claim: For any vertex cover $S$ and any set of fair prices $\{p_e : e \in E\}$:
$$\sum_{e \in E} p_e \leq w(S)$$

Proof:
$$\sum_{e \in E} p_e \leq \sum_{v \in S} \sum_{e=\{u,v\}} p_e \leq \sum_{v \in S} w_v = w(S)$$

So, in particular, if $S^*$ is a minimum weight vertex cover, we have
$$\sum_{e \in E} p_e \leq w(S^*)$$

That is, the sum of edge prices is a lower bound on the weight of a minimum weight vertex cover.
Weighted Vertex Covers via the Pricing Method

Idea: An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.

- The set $\{p_e : e \in E\}$ of prices is *fair* if, for each $v \in V$
  \[ \sum_{e \in \{u,v\}} p_e \leq w_v \] (v is not overcharging)

Claim: For any vertex cover $S$ and *any set* of fair prices
$\{p_e : e \in E\}$: $\sum_{e \in E} p_e \leq w(S)$
Weighted Vertex Covers via the Pricing Method

Idea: An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.

- The set $\{p_e : e \in E\}$ of prices is fair if, for each $v \in V$
  \[ \sum_{e=\{u,v\}} p_e \leq w_v \text{ (v is not overcharging)} \]

Claim: For any vertex cover $S$ and any set of fair prices
$\{p_e : e \in E\}$: $\sum_{e \in E} p_e \leq w(S)$

Proof:
\[
\sum_{e \in E} p_e \leq \sum_{v \in S} \sum_{e=\{u,v\}} p_e \leq \sum_{v \in S} w_v = w(S)
\]
Weighted Vertex Covers via the Pricing Method

Idea: An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.

- The set $\{p_e : e \in E\}$ of prices is fair if, for each $v \in V$
  $$\sum_{e=\{u,v\}} p_e \leq w_v \text{ (} v \text{ is not overcharging)}$$

Claim: For any vertex cover $S$ and any set of fair prices $\{p_e : e \in E\}$: $\sum_{e \in E} p_e \leq w(S)$

Proof:

$$\sum_{e \in E} p_e \leq \sum_{v \in S} \sum_{e=\{u,v\}} p_e \leq \sum_{v \in S} w_v = w(S)$$

So, in particular, if $S^*$ is a minimum weight vertex cover, we have
$$\sum_{e \in E} p_e \leq w(S^*)$$
Weighted Vertex Covers via the Pricing Method

Idea: An edge $e$ pays a vertex $v$ some price $p_e$ to cover it.

- The set \( \{p_e : e \in E\} \) of prices is fair if, for each $v \in V$
  \[ \sum_{e=\{u,v\}} p_e \leq w_v \quad (v \text{ is not overcharging}) \]

Claim: For any vertex cover $S$ and any set of fair prices \( \{p_e : e \in E\} \): \( \sum_{e \in E} p_e \leq w(S) \)

Proof:

\[
\sum_{e \in E} p_e \leq \sum_{v \in S} \sum_{e=\{u,v\}} p_e \leq \sum_{v \in S} w_v = w(S)
\]

So, in particular, if $S^*$ is a minimum weight vertex cover, we have \( \sum_{e \in E} p_e \leq w(S^*) \)

That is, the sum of edge prices is a lower bound on the weight of a minimum weight vertex cover.
A Price-Setting Greedy Algorithm

Idea: Build a vertex cover while greedily setting prices; show that small multiple of sum of prices is (upper) bound of cover weight
A Price-Setting Greedy Algorithm

Idea: Build a vertex cover while greedily setting prices; show that small multiple of sum of prices is (upper) bound of cover weight

Def’n: A vertex $v$ is tight if $\sum_{e=\{u,v\}} p_e = w_v$
A Price-Setting Greedy Algorithm

Idea: Build a vertex cover while greedily setting prices; show that small multiple of sum of prices is (upper) bound of cover weight

Def’n: A vertex $v$ is tight if $\sum_{e=\{u,v\}} p_e = w_v$

Algorithm 3 PriceFixing

```
procedure PRICEFIXING($G = (V, E), w[-]$)
    Set all prices $p[e]$ to 0
    while Some edge $e$ has neither vertex tight do
        Select such an edge $e = \{u, v\}$
        Increase $p[e]$ until first of $u$ or $v$ becomes tight
    Return set $S$ of all tight nodes
end procedure
```
A Price-Setting Greedy Algorithm

Idea: Build a vertex cover while greedily setting prices; show that small multiple of sum of prices is (upper) bound of cover weight

Def’n: A vertex \( v \) is tight if \( \sum_{e=\{u,v\}} p_e = w_v \)

Algorithm 4 PriceFixing

procedure \textsc{PriceFixing}(G = (V, E), w[\_])

Set all prices \( p[e] \) to 0

while Some edge \( e \) has neither vertex tight do

Select such an edge \( e = \{u, v\} \)

Increase \( p[e] \) until first of \( u \) or \( v \) becomes tight

Return set \( S \) of all tight nodes

end procedure

Observe: (1) Tight vertices form a cover; (2) tight vertices stay tight; (3) prices remain fair
How Good is PriceFixing?

Claim:
The $S$ and $p$ returned by PriceFixing satisfy $w(S) \leq 2 \sum_{e \in E} p_e$.

Proof:

$$w(S) = \sum_{v \in S} w(v) = \sum_{v \in S} \sum_{e = uv} p_e \leq 2 \sum_{e \in E} p_e$$

Corollary:
For any vertex cover $S^*$, $w(S) \leq 2 w(S^*)$.

Proof:

$$w(S) \leq 2 \sum_{e \in E} p_e$$

and

$$\sum_{e \in E} p_e \leq w(S^*)$$

Corollary:
The weight of $S$ is within a factor of 2 of optimal vertex cover.
Claim: The $S$ and $p[−]$ returned by PriceFixing satisfy
\[ w(S) \leq 2 \sum_{e \in E} p_e. \]
How Good is PriceFixing?

Claim: The $S$ and $p[\vdash]$ returned by PriceFixing satisfy $w(S) \leq 2 \sum_{e \in E} p_e$.

Proof:

$$w(S) = \sum_{v \in S} w_v = \sum_{v \in S} \sum_{e = \{u, v\}} p_e \leq 2 \sum_{e \in E} p_e$$
How Good is PriceFixing?

Claim: The $S$ and $p[−]$ returned by PriceFixing satisfy $w(S) \leq 2\sum_{e \in E} p_e$.

Proof:

$$w(S) = \sum_{v \in S} w_v = \sum_{v \in S} \sum_{e = \{u, v\}} p_e \leq 2\sum_{e \in E} p_e$$

Corollary: For any vertex cover $S^*$, $w(S) \leq 2w(S^*)$
**How Good is PriceFixing?**

**Claim:** The $S$ and $p[-]$ returned by PriceFixing satisfy $w(S) \leq 2 \sum_{e \in E} p_e$.

**Proof:**

$$w(S) = \sum_{v \in S} w_v = \sum_{v \in S} \sum_{e = \{u, v\}} p_e \leq 2 \sum_{e \in E} p_e$$

**Corollary:** For any vertex cover $S^*$, $w(S) \leq 2w(S^*)$

**Proof:**

$$w(S) \leq 2 \sum_{e \in E} p_e \text{ and } \sum_{e \in E} p_e \leq w(S^*)$$
How Good is PriceFixing?

Claim: The $S$ and $p[−]$ returned by PriceFixing satisfy
\[
w(S) \leq 2 \sum_{e \in E} p_e.
\]

Proof:
\[
w(S) = \sum_{v \in S} w_v = \sum_{v \in S} \sum_{e \in \{u,v\}} p_e \leq 2 \sum_{e \in E} p_e
\]

Corollary: For any vertex cover $S^*$, \(w(S) \leq 2w(S^*)\)

Proof:
\[
w(S) \leq 2 \sum_{e \in E} p_e \text{ and } \sum_{e \in E} p_e \leq w(S^*)
\]

Corollary: The weight of $S$ is within a factor of 2 of optimal vertex cover
The Knapsack Problem Revisited

The Problem: Given $n$ items, each with (integer) value $v_i$ and weight $w_i$, and an integer $W$ (*knapsack capacity*), find a subset $I \subseteq \{1, \ldots, n\}$ of items such that
The Problem: Given \( n \) items, each with (integer) value \( v_i \) and weight \( w_i \), and an integer \( W \) (knapsack capacity), find a subset \( I \subseteq \{1, \ldots, n\} \) of items such that

\[
V = \sum_{i \in I} v_i \text{ is maximized subject to } \sum_{i \in I} w_i \leq W
\]
The Knapsack Problem Revisited

The Problem: Given $n$ items, each with (integer) value $v_i$ and weight $w_i$, and an integer $W$ (knapsack capacity), find a subset $I \subseteq \{1, \ldots, n\}$ of items such that

$$V = \sum_{i \in I} v_i \text{ is maximized subject to } \sum_{i \in I} w_i \leq W$$

Recall (DP): Let $opt(i, w)$ be maximum value using only elements from $\{1, \ldots, i\}$ and capacity at most $w$. We want $opt(n, W)$
The Knapsack Problem Revisited

The Problem: Given \( n \) items, each with (integer) value \( v_i \) and weight \( w_i \), and an integer \( W \) (knapsack capacity), find a subset \( I \subseteq \{1, \ldots, n\} \) of items such that

\[
V = \sum_{i \in I} v_i \text{ is maximized subject to } \sum_{i \in I} w_i \leq W
\]

Recall (DP): Let \( \text{opt}(i, w) \) be maximum value using only elements from \( \{1, \ldots, i\} \) and capacity at most \( w \). We want \( \text{opt}(n, W) \)

Recall (DP): OO: Either item \( i \) is in the solution, or it isn’t....
The Knapsack Problem Revisited

The Problem: Given $n$ items, each with (integer) value $v_i$ and weight $w_i$, and an integer $W$ (knapsack capacity), find a subset $I \subseteq \{1, \ldots, n\}$ of items such that

$$V = \sum_{i \in I} v_i$$

is maximized subject to

$$\sum_{i \in I} w_i \leq W$$

Recall (DP): Let $opt(i, w)$ be maximum value using only elements from $\{1, \ldots, i\}$ and capacity at most $w$. We want $opt(n, W)$

Recall (DP): OO: Either item $i$ is in the solution, or it isn’t....

$$opt(i, w) = \begin{cases} 
  opt(i - 1, w), & \text{if } w < w_i \ (\text{Avoid } w - w_i < 0) \\
  \max\{opt(i - 1, w), v_i + opt(i - 1, w - w_i)\} & \text{otherwise}
\end{cases}$$

Pseudo-Polynomial: Run-time $O(nW)$ (good for small weights)
The Knapsack Problem Revisited

The Problem: Given \( n \) items, each with (integer) value \( v_i \) and weight \( w_i \), and an integer \( W \) (knapsack capacity), find a subset \( I \subseteq \{1, \ldots, n\} \) of items such that

\[
V = \sum_{i \in I} v_i \text{ is maximized subject to } \sum_{i \in I} w_i \leq W
\]

Recall (DP): Let \( \text{opt}(i, w) \) be maximum value using only elements from \( \{1, \ldots, i\} \) and capacity at most \( w \). We want \( \text{opt}(n, W) \)

Recall (DP): OO: Either item \( i \) is in the solution, or it isn’t....

\[
\text{opt}(i, w) = \begin{cases} 
\text{opt}(i - 1, w), & \text{if } w < w_i \ (\text{Avoid } w - w_i < 0) \\
\max\{\text{opt}(i - 1, w), v_i + \text{opt}(i - 1, w - w_i)\} & 
\end{cases}
\]

Pseudo-Polynomial: Run-time \( O(nW) \) (good for small weights)
A First Approximation Algorithm For Knapsack

Many greedy approaches fail miserably to produce a good approximation
A First Approximation Algorithm For Knapsack

Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
- Highest weight first

Fortunately, the unit value approach can be salvaged...
Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
- Highest weight first
- Highest value first
A First Approximation Algorithm For Knapsack

Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
- Highest weight first
- Highest value first
- Lowest value first
- Greatest unit value ($v_i/w_i$) first (!)

Fortunately, the unit value approach can be salvaged....
Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
- Highest weight first
- Highest value first
- Lowest value first
- Greatest unit value \( v_i/w_i \) first (!)
Many greedy approaches fail miserably to produce a good approximation

- Lowest weight first
- Highest weight first
- Highest value first
- Lowest value first
- Greatest unit value ($v_i/w_i$) first (!)

Fortunately, the unit value approach can be salvaged....
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm \textit{UnitGreed}
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm \textit{UnitGreed}

1. Sort items so that $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm \textit{UnitGreed}

1. Sort items so that $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$

2. Find largest $k$ such that

$$\sum_{i=1}^{k} w_i \leq W$$

Claim: \textit{UnitGreed} produces a result within a factor of 2 of the maximum.
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm *UnitGreed*

1. Sort items so that $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$

2. Find largest $k$ such that

   $$\sum_{i=1}^{k} w_i \leq W$$

This gives value $V = \sum_{i=1}^{k} v_i$
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm $UnitGreed$

1. Sort items so that $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$
2. Find largest $k$ such that

$$\sum_{i=1}^{k} w_i \leq W$$

This gives value $V = \sum_{i=1}^{k} v_i$

3. If $V > v_{k+1}$ take items $1, \ldots, k$, otherwise take item $k + 1$
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm UnitGreed

1. Sort items so that $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$
2. Find largest $k$ such that

$$\sum_{i=1}^{k} w_i \leq W$$

This gives value $V = \sum_{i=1}^{k} v_i$
3. If $V > v_{k+1}$ take items $1, \ldots, k$, otherwise take item $k + 1$

Claim: UnitGreed produces a result within a factor of 2 of the maximum.
A 2-Approximation for Knapsack

Proof: Consider a \textit{fractional} version of \textit{Knapsack} in which any fractional portion of an item can be selected. Observe that
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
- $opt_f(n, W)$ can be achieved by taking items 1, . . . , $k$ from UnitGreed and a portion of item $k + 1$ to fill the knapsack
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
- $opt_f(n, W)$ can be achieved by taking items 1, \ldots, $k$ from UnitGreed and a portion of item $k + 1$ to fill the knapsack
- Precisely, if $w = \sum_{i=1}^{k} w_i$, take $(W - w)/w_{k+1}$ of item $k + 1$ for value $v_{k+1}(W - w)/w_{k+1}$
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
- $opt_f(n, W)$ can be achieved by taking items 1, \ldots, $k$ from UnitGreed and a portion of item $k + 1$ to fill the knapsack
- Precisely, if $w = \sum_{i=1}^{k} w_i$, take $(W - w)/w_{k+1}$ of item $k + 1$ for value $v_{k+1}(W - w)/w_{k+1}$
- So $\sum_{i=1}^{k} v_i + v_{k+1}(W - w)/w_{k+1} = opt_f(n, W) \geq opt(n, W)$
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let \( \text{opt}_f(n, W) \) be the optimal value for the fractional knapsack problem.

- \( \text{opt}_f(n, W) \geq \text{opt}(n, W) \)
- \( \text{opt}_f(n, W) \) can be achieved by taking items 1, \ldots, \( k \) from UnitGreed and a portion of item \( k + 1 \) to fill the knapsack
- Precisely, if \( w = \sum_{i=1}^{k} w_i \), take \((W - w)/w_{k+1}\) of item \( k + 1 \) for value \( v_{k+1}(W - w)/w_{k+1} \)
- So \( \sum_{i=1}^{k} v_i + v_{k+1}(W - w)/w_{k+1} = \text{opt}_f(n, W) \geq \text{opt}(n, W) \)
- But UnitGreed yields \( V^* = \max\{\sum_{i=1}^{k} v_i, v_{k+1}\} \).
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
- $opt_f(n, W)$ can be achieved by taking items 1, \ldots, $k$ from UnitGreed and a portion of item $k + 1$ to fill the knapsack
- Precisely, if $w = \sum_{i=1}^{k} w_i$, take $(W - w)/w_{k+1}$ of item $k + 1$ for value $v_{k+1}(W - w)/w_{k+1}$
- So $\sum_{i=1}^{k} v_i + v_{k+1}(W - w)/w_{k+1} = opt_f(n, W) \geq opt(n, W)$
- But UnitGreed yields $V^* = \max\{\sum_{i=1}^{k} v_i , v_{k+1}\}$.

Note: $v_{k+1} \geq v_{k+1}(W - w)/w_{k+1}$, since $W - w < w_{k+1}$
A 2-Approximation for Knapsack

Proof: Consider a fractional version of Knapsack in which any fractional portion of an item can be selected. Observe that

Let $opt_f(n, W)$ be the optimal value for the fractional knapsack problem.

- $opt_f(n, W) \geq opt(n, W)$
- $opt_f(n, W)$ can be achieved by taking items 1, \ldots, $k$ from UnitGreed and a portion of item $k + 1$ to fill the knapsack
- Precisely, if $w = \sum_{i=1}^{k} w_i$, take $(W - w)/w_{k+1}$ of item $k + 1$ for value $v_{k+1}(W - w)/w_{k+1}$
- So $\sum_{i=1}^{k} v_i + v_{k+1}(W - w)/w_{k+1} = opt_f(n, W) \geq opt(n, W)$
- But UnitGreed yields $V^* = \max\{\sum_{i=1}^{k} v_i, v_{k+1}\}$.
  Note: $v_{k+1} \geq v_{k+1}(W - w)/w_{k+1}$, since $W - w < w_{k+1}$
- So $opt(n, W) \leq opt_f(n, W) = \sum_{i=1}^{k} v_i + v_{k+1}(W - w)/w_{k+1} \leq 2V^*$
α-Approximations: Maximization vs Minimization

Note: Since this is a maximization problem, we get a result of the form

$$2 \times \text{Approximate solution} \geq \text{Optimal solution}$$
Note: Since this is a maximization problem, we get a result of the form

\[ 2 \times \text{Approximate solution} \geq \text{Optimal solution} \]

Compare this to, say, the Vertex Cover approximation:

\[ \text{Approximate solution} \leq 2 \times \text{Optimal solution} \]
\( \alpha \text{-Approximations: Maximization vs Minimization} \)

Note: Since this is a maximization problem, we get a result of the form

\[
2 \times \text{Approximate solution} \geq \text{Optimal solution}
\]

Compare this to, say, the Vertex Cover approximation:

\[
\text{Approximate solution} \leq 2 \times \text{Optimal solution}
\]

An \( \alpha \)-approximation for a minimization problem guarantees

\[
1 \leq \frac{\text{Approximate solution}}{\text{Optimal solution}} \leq \alpha
\]
\(\alpha\)-Approximations: Maximization vs Minimization

Note: Since this is a maximization problem, we get a result of the form

\[2 \ast \text{Approximate solution} \geq \text{Optimal solution}\]

Compare this to, say, the Vertex Cover approximation:

\[\text{Approximate solution} \leq 2 \ast \text{Optimal solution}\]

An \(\alpha\)-approximation for a minimization problem guarantees

\[1 \leq \frac{\text{Approximate solution}}{\text{Optimal solution}} \leq \alpha\]

An \(\alpha\)-approximation for a maximization problem guarantees

\[1 \leq \frac{\text{Optimal solution}}{\text{Approximate solution}} \leq \alpha\]
A Value-Oriented DP Algorithm

Idea: Consider \textit{smallest} weight using items \{1, \ldots, i\} achieving value \textit{at least} \( V \)

**A Value-Oriented DP Algorithm**

**Idea:** Consider *smallest* weight using items \( \{1, \ldots, i\} \) achieving value *at least* \( V \)

- Denote this weight by \( \overline{opt}(i, V) \), where \( V = 0, \ldots, \sum_{j=1}^{i} v_j \)
A Value-Oriented DP Algorithm

**Idea:** Consider *smallest* weight using items \( \{1, \ldots, i\} \) achieving value at least \( V \)

- Denote this weight by \( \text{opt}(i, V) \), where \( V = 0, \ldots, \sum_{j=1}^{i} v_j \)
- **Note:** Maximum value of \( V \) depends on \( i \)
A Value-Oriented DP Algorithm

Idea: Consider *smallest* weight using items \(\{1, \ldots, i\}\) achieving value *at least* \(V\)

- Denote this weight by \(\text{opt}(i, \ V)\), where \(V = 0, \ldots, \sum_{j=1}^{i} v_j\)
- **Note**: Maximum value of \(V\) depends on \(i\)
- **Note**: If \(v^* = \max_i v_i\), then \(V \leq nv^*\)
A Value-Oriented DP Algorithm

Idea: Consider \textit{smallest} weight using items \{1, \ldots, i\} achieving value \textit{at least} \( V \)

- Denote this weight by \( \text{opt}(i, V) \), where \( V = 0, \ldots, \sum_{j=1}^{i} v_j \)
- \textbf{Note:} Maximum value of \( V \) depends on \( i \)
- \textbf{Note:} If \( v^* = \max_i v_i \), then \( V \leq n v^* \)
- Knapsack solution is the \textit{largest} \( V \) for which \( \text{opt}(n, V) \leq W \)
Idea: Consider smallest weight using items \(\{1, \ldots, i\}\) achieving value at least \(V\)

- Denote this weight by \(\text{opt}(i, V)\), where \(V = 0, \ldots, \sum_{j=1}^{i} v_j\)
- **Note:** Maximum value of \(V\) depends on \(i\)
- **Note:** If \(v^* = \max_i v_i\), then \(V \leq n v^*\)
- Knapsack solution is the largest \(V\) for which \(\text{opt}(n, V) \leq W\)
- That is, \(\text{opt}(n, W) = \max_V \{\text{opt}(n, V) \leq W\}\)
A Value-Oriented DP Algorithm

**Idea:** Consider *smallest* weight using items \( \{1, \ldots, i\} \) achieving value *at least* \( V \)

- Denote this weight by \( \text{opt}(i, V) \), where \( V = 0, \ldots, \sum_{j=1}^{i} v_j \)
- **Note:** Maximum value of \( V \) depends on \( i \)
- **Note:** If \( v^* = \max_i v_i \), then \( V \leq nv^* \)
- Knapsack solution is the *largest* \( V \) for which \( \text{opt}(n, V) \leq W \)
- That is, \( \text{opt}(n, W) = \max_V \{ \text{opt}(n, V) \leq W \} \)
- Algorithm is pseudo-polynomial in the *values*
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \not\in \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n$
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n)$
**A Recurrence For \( \overline{\text{opt}}(i, V) \)**

Let \( \mathcal{O} \) be an optimal solution (set of indices). Then

- If \( n \notin \mathcal{O} \), then \( \overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V) \)
- If \( n \) is the only item in \( \mathcal{O} \), then \( \overline{\text{opt}}(n, V) = w_n \)
- If \( n \in \mathcal{O} \) is not the only item in \( \mathcal{O} \), then
  \[ \overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n) \]
- **Note:** If \( V > \sum_{i=1}^{n-1} v_i \), then it must be that the third case holds
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \notin \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds

- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of
A Recurrence For $\text{opt}(i, V)$

Let $O$ be an optimal solution (set of indices). Then

- If $n \not\in O$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $O$, then $\text{opt}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then $\text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds
- If not, then $\text{opt}(n, V)$ is the smallest of
  - $\text{opt}(n - 1, V)$
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $O$ be an optimal solution (set of indices). Then

- If $n \notin O$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $O$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds

- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of
  - $\overline{\text{opt}}(n - 1, V)$
  - $w_n$ (if $v_n \geq V$)
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \notin \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds

- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of
  - $\overline{\text{opt}}(n - 1, V)$
  - $w_n$ (if $v_n \geq V$)
  - $w_n + \overline{\text{opt}}(n - 1, V - v_n)$
A Recurrence For $\text{opt}(i, V)$

Let $O$ be an optimal solution (set of indices). Then

- If $n \notin O$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $O$, then $\text{opt}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then $\text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds.

- If not, then $\text{opt}(n, V)$ is the smallest of
  - $\text{opt}(n - 1, V)$
  - $w_n$ (if $v_n \geq V$)
  - $w_n + \text{opt}(n - 1, V - v_n)$

- where $\text{opt}(i, 0) = 0$, for $i = 1, \ldots, n$
A Recurrence For $\overline{opt}(i, V)$

Let $\mathcal{O}$ be an optimal solution (set of indices). Then

- If $n \notin \mathcal{O}$, then $\overline{opt}(n, V) = \overline{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{opt}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then
  $$\overline{opt}(n, V) = w_n + \overline{opt}(n - 1, V - v_n)$$

  **Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the third case holds

- If not, then $\overline{opt}(n, V)$ is the smallest of
  - $\overline{opt}(n - 1, V)$
  - $w_n$ (if $v_n \geq V$)
  - $w_n + \overline{opt}(n - 1, V - v_n)$

- where $\overline{opt}(i, 0) = 0$, for $i = 1, \ldots, n$
- and $\overline{opt}(0, v) = \infty$ for $v \geq 1$