Weighted interval scheduling
COMS20010 2020, Video 11-1

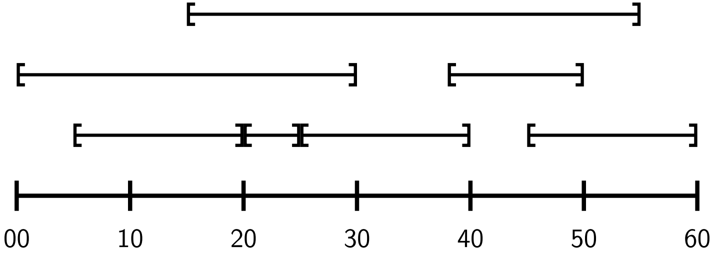
John Lapinskas, University of Bristol

Unweighted interval scheduling (recap from week 2)

Motivation: A satellite imaging service wants to use its camera to fill as many orders as possible, but it can only take one picture at once.

Input: A set of intervals, e.g.
$\mathcal{R} = \{(0,30),(5,20),(15,55),(20,25),(25,40),(38,50),(45,60)\}$.

Output: A maximum **compatible** set $\mathcal{R}' \subseteq \mathcal{R}$ — that is, for all $(s,f),(s',f') \in \mathcal{R}'$, we have either $s',f' \geq f$ or $s',f' \leq s$.



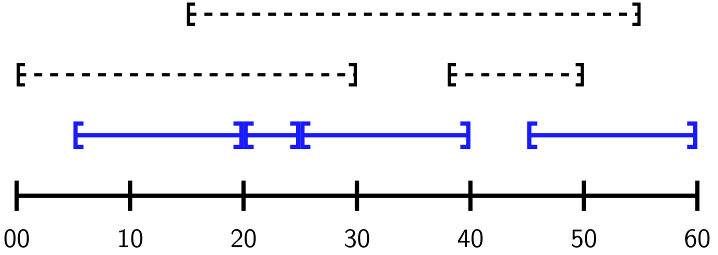
Algorithm: Sort $\mathcal{R}$ in increasing order of finishing time, then add them to the output greedily while maintaining compatibility.

Unweighted interval scheduling (recap from week 2)

Motivation: A satellite imaging service wants to use its camera to fill as many orders as possible, but it can only take one picture at once.

Input: A set of intervals, e.g.
$\mathcal{R} = \{(0,30),(5,20),(15,55),(20,25),(25,40),(38,50),(45,60)\}$.

Output: A maximum **compatible** set $\mathcal{R}' \subseteq \mathcal{R}$ — that is, for all $(s,f),(s',f') \in \mathcal{R}'$, we have either $s',f' \geq f$ or $s',f' \leq s$.



Algorithm: Sort $\mathcal{R}$ in increasing order of finishing time, then add them to the output greedily while maintaining compatibility.

Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right?

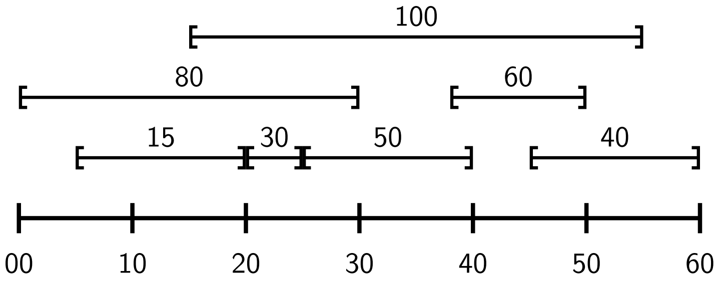
Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right?
We'd want to earn as much **money** as possible.

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.



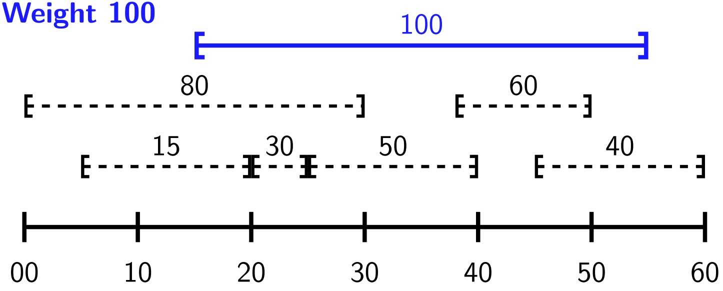
Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

Weight 100

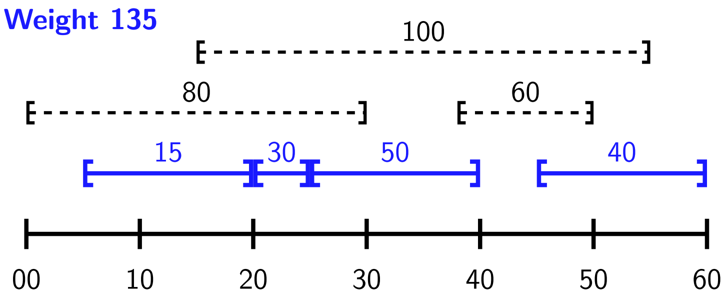


Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.



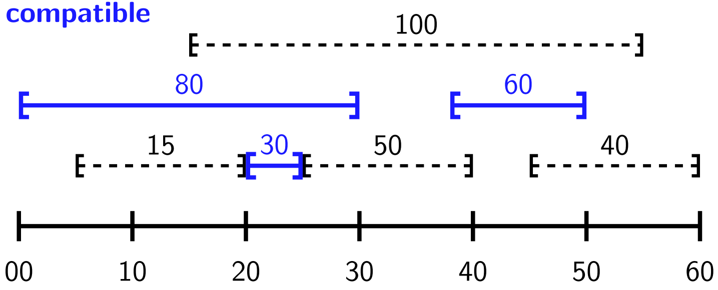
Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

Not compatible

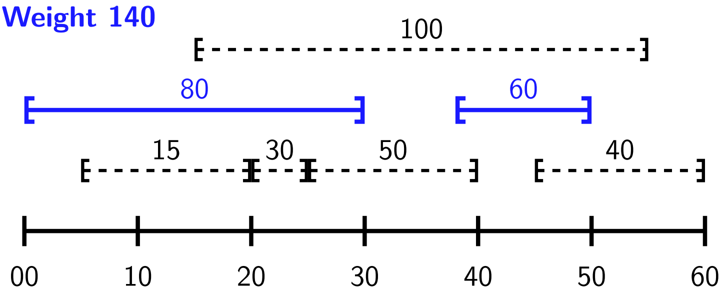


Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

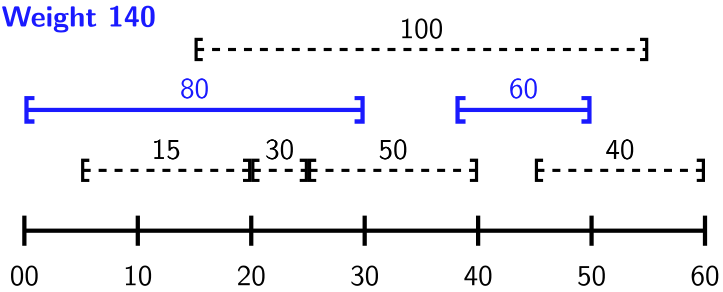


Weighted interval scheduling

But we wouldn't really want to fill as many orders as possible, right? We'd want to earn as much **money** as possible.

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.

Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.



In this case, the desired output has weight 140. But our old greedy algorithm fails! There is **no** known greedy algorithm for this problem.

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

Step 1: Come up with an **exponential-time** recursive algorithm for your problem by reducing it to multiple smaller versions of itself.

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

Step 1: Come up with an **exponential-time** recursive algorithm for your problem by reducing it to multiple smaller versions of itself.

Step 2: Arrange things so that most of the calls of your recursive algorithm are repeated, and use this to make it polynomial. (Hard!)

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

Step 1: Come up with an **exponential-time** recursive algorithm for your problem by reducing it to multiple smaller versions of itself.

Step 2: Arrange things so that most of the calls of your recursive algorithm are repeated, and use this to make it polynomial. (Hard!)

Step 3: Optionally, rewrite your algorithm as an iterative one. (Easier.)

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

Step 1: Come up with an **exponential-time** recursive algorithm for your problem by reducing it to multiple smaller versions of itself.

Step 2: Arrange things so that most of the calls of your recursive algorithm are repeated, and use this to make it polynomial. (Hard!)

Step 3: Optionally, rewrite your algorithm as an iterative one. (Easier.)

Why call it "dynamic programming"?

The idea: Dynamic programming

You've seen dynamic programming in year 1, but I'm assuming you have forgotten almost all of it!

Step 1: Come up with an **exponential-time** recursive algorithm for your problem by reducing it to multiple smaller versions of itself.

Step 2: Arrange things so that most of the calls of your recursive algorithm are repeated, and use this to make it polynomial. (Hard!)

Step 3: Optionally, rewrite your algorithm as an iterative one. (Easier.)

Why call it "dynamic programming"?

Because Richard Bellman needed to sell it to an idiot politician and "dynamic" was a fashionable word! If it had been invented today, it would have been called "agile blockchain programming in the cloud"...

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w \colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

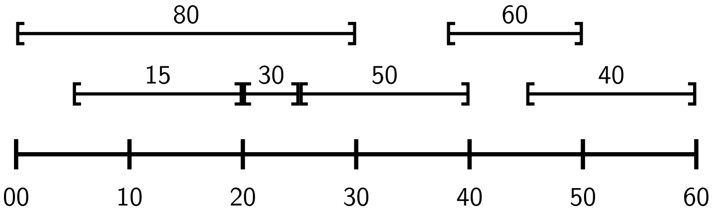
Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



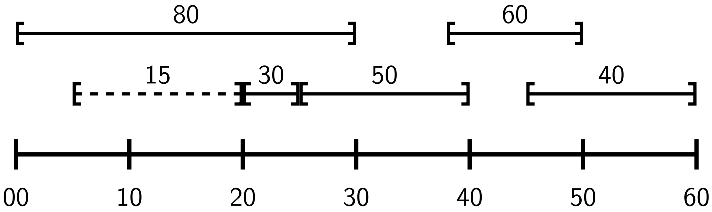
If we don't include some interval I:

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



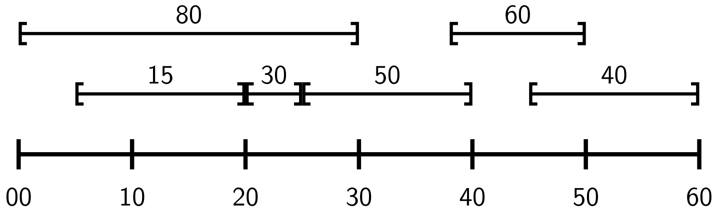
If we don't include some interval I: Then the maximum-weight compatible set will be the same as $\mathrm{WIS}(\mathcal{R} \setminus \{I\})$.

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



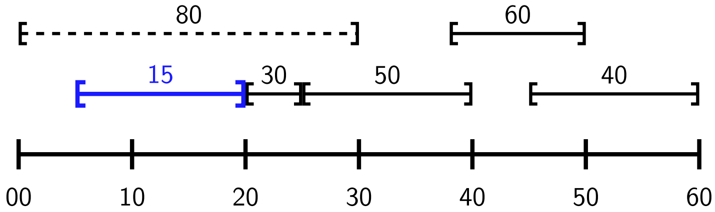
If we do include some interval I:

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R\in\mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



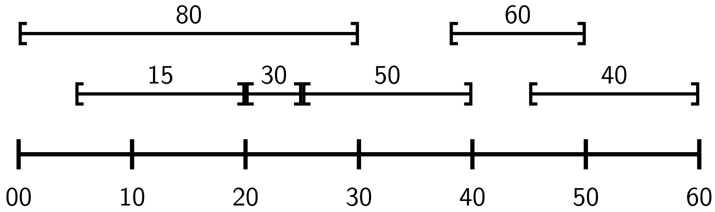
If we do include some interval I: Then we can't include any interval in the set $X_I \subseteq \mathcal{R}$ of intervals intersecting I, or we lose compatibility. But the maximum-weight compatible set will be I together with $\mathrm{WIS}(\mathcal{R} \setminus X_I)$.

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



So overall, the highest-weight compatible set will be either $\mathrm{WIS}(\mathcal{R} \setminus \{I\})$ or $\{I\} \cup \mathrm{WIS}(\mathcal{R} \setminus X_I)$, whichever has higher weight.

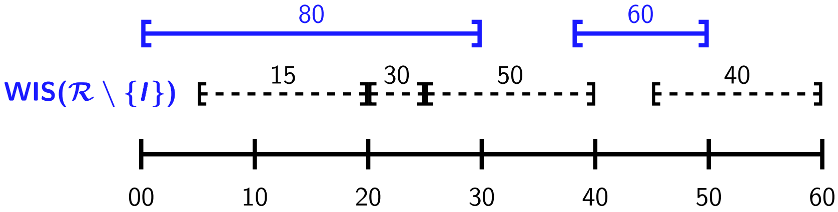
(See problem sheet 8 question 4 for more examples!)

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



So overall, the highest-weight compatible set will be either $\mathrm{WIS}(\mathcal{R} \setminus \{I\})$ or $\{I\} \cup \mathrm{WIS}(\mathcal{R} \setminus X_I)$, whichever has higher weight.

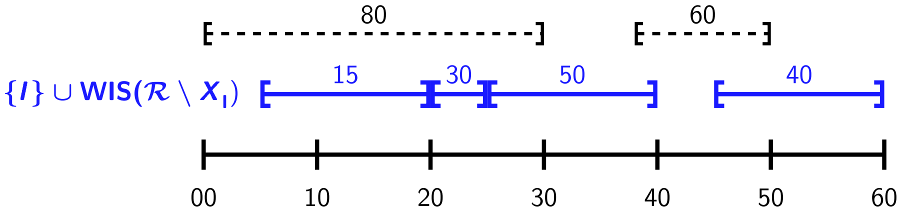
(See problem sheet 8 question 4 for more examples!)

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



So overall, the highest-weight compatible set will be either $\mathrm{WIS}(\mathcal{R} \setminus \{I\})$ or $\{I\} \cup \mathrm{WIS}(\mathcal{R} \setminus X_I)$, whichever has higher weight.

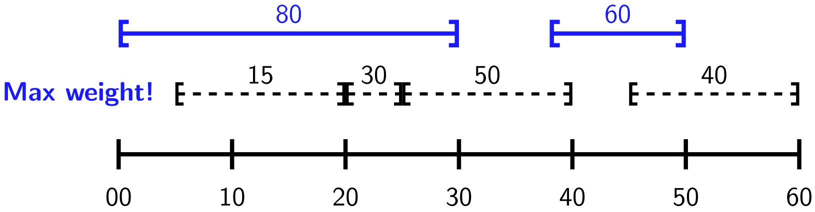
(See problem sheet 8 question 4 for more examples!)

Step 1: Reducing weighted interval scheduling to itself

Input: A set of intervals $\mathcal{R}$ and a **weight function** $w\colon \mathcal{R} \to \mathbb{Q}_{\geq 0}$.
Output: A compatible set $\mathcal{R}' \subseteq \mathcal{R}$ of maximum **weight** $\sum_{R \in \mathcal{R}'} w(R)$.

You can think of weighted interval scheduling as a sequence of **choices**: Do I include this interval in my output, or not?

Our greedy algorithm decided "yes" or "no" based only on finishing times. But to reduce a problem to itself, we consider the **effect** of each choice.



So overall, the highest-weight compatible set will be either $\mathrm{WIS}(\mathcal{R} \setminus \{I\})$ or $\{I\} \cup \mathrm{WIS}(\mathcal{R} \setminus X_I)$, whichever has higher weight.

(See problem sheet 8 question 4 for more examples!)

The full recursive algorithm

Algorithm: WIS

Input : An array $\mathcal{R}$ of n requests and a weight function w.
Output : A maximum-weight compatible subset of $\mathcal{R}$.

begin
- **if** $\mathcal{R} = \emptyset$ **then**
  - Return $\emptyset$.
- **else**
  - Choose $I \in \mathcal{R}$ arbitrarily.
  - Find the set X_I of intervals in $\mathcal{R}$ incompatible with I.
  - $S_{\mathrm{out}} \leftarrow \mathrm{WIS}(\mathcal{R} \setminus \{I\}, w)$.
  - $S_{\mathrm{in}} \leftarrow \{I\} \cup \mathrm{WIS}(\mathcal{R} \setminus X_I, w)$.
  - **if** $w(S_{\mathrm{out}}) > w(S_{\mathrm{in}})$ **then**
    - Return S_{out}.
  - **else**
    - Return S_{in}.

Note that this algorithm will work **regardless** of how we pick each I.

Next video, we will exploit this to make the algorithm run much faster...