Solving Woeginger's Hiking Problem
(Wonderful Partitions in Anonymous Hedonic Games)

Andrei Constantinescu, Pascal Lenzner, Rebecca Reiffenhäuser, Daniel Schmand, Giovanna Varricchio





ALGO'08

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Gerhard Woeginger
(1964—2022)

Comb. opt.

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling

Theoretical Computer Scientist

Gerhard Woeginger
(1964—2022)

Comb. opt., scheduling, graphs

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Gerhard Woeginger
(1964—2022)

Comb. opt., scheduling, graphs, complexity, game theory, ...

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory, ...

400^{+} papers

Gerhard Woeginger
(1964—2022)

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory, ...

400^{+} papers: ESA (18), TCS (16), Algorithmica (13), SODA (10), ICALP (8), STACS (6), TALG (3), FOCS (2), STOC (2) ...

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory, ...

400^+ papers: ESA (18), TCS (16), Algorithmica (13), SODA (10), ICALP (8), STACS (6), TALG (3), FOCS (2), STOC (2) ...

Program chair: **ICALP (2003)**

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory, ...

400^+ papers: ESA (18), TCS (16), Algorithmica (13), SODA (10), ICALP (8), STACS (6), TALG (3), FOCS (2), STOC (2) ...

Program chair: **ICALP (2003)**, ESA (1997)

Gerhard Woeginger
(1964—2022)

Theoretical Computer Scientist

Comb. opt., scheduling, graphs, complexity, game theory, ...

400^+ papers: ESA (18), TCS (16), Algorithmica (13), SODA (10), ICALP (8), STACS (6), TALG (3), FOCS (2), STOC (2) ...

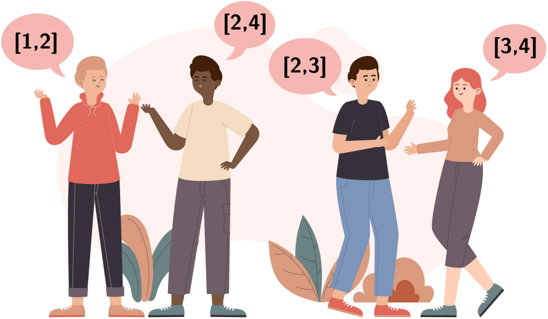
Program chair: **ICALP (2003)**, ESA (1997)

Liked Puzzles!

A Puzzle

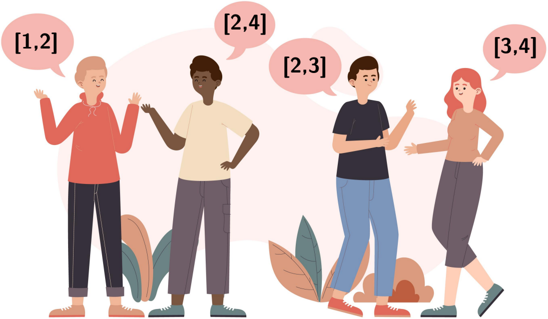
A Puzzle

Desired group sizes:



A Puzzle

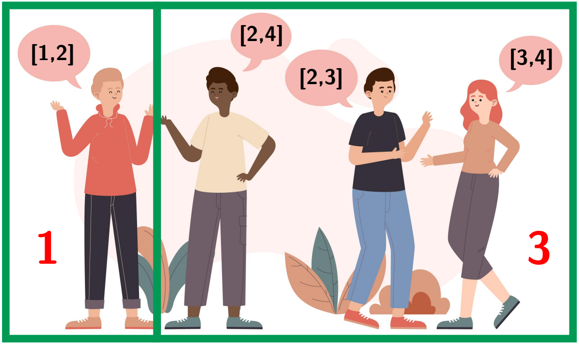
Desired group sizes:



Partition hikers into subgroups such that everyone is satisfied with their group size.

A Puzzle

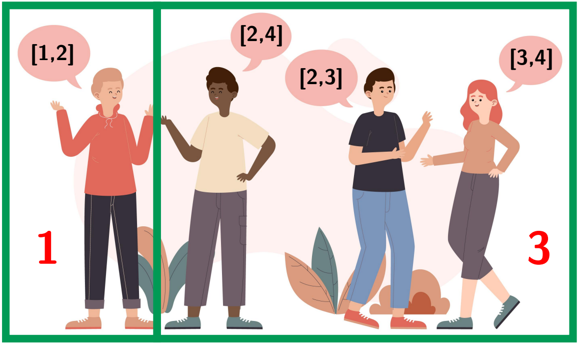
Desired group sizes:



Partition hikers into subgroups such that everyone is satisfied with their group size.

A Puzzle

Desired group sizes:



Partition hikers into subgroups such that everyone is satisfied with their group size. **Puzzle:** **Polynomial time?**

Hiking Problem

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.

Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.

Generalization

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| \leq 2$ **[Darmann et al.'18]**;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| = 2$ **[this paper]**;
- **NP-hard** if $|A_i| \leq 2$ **[Darmann et al.'18]**;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Complexity:

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| = 2$ **[this paper]**;
- **NP-hard** if $|A_i| \leq 2$ **[Darmann et al.'18]**;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \dots, r_i\}$**)**

Complexity: **open** **[Woeginger'13]**

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| = 2$ **[this paper]**;
- **NP-hard** if $|A_i| \leq 2$ **[Darmann et al.'18]**;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

Hiking Problem

Input: n agents, for each agent i an interval $[\ell_i, r_i]$.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in $[\ell_i, r_i]$.
(i.e., $A_i = \{\ell_i, \ldots, r_i\}$**)**

Complexity: **open [Woeginger'13]** $\Rightarrow$ **Polynomial [this paper]**.

Generalization

Input: n agents, for each agent i a set A_i of 'approved' sizes.
Output: Partition of agents s.t. $\forall i$ the size of i's group is in A_i.

Complexity:

- **Polynomial** if $|A_i| = 1$;
- **NP-hard** if $|A_i| = 2$ **[this paper]**;
- **NP-hard** if $|A_i| \leq 2$ **[Darmann et al.'18]**;
- **NP-hard** if $|A_i| \leq 3$ **[Woeginger'13].**

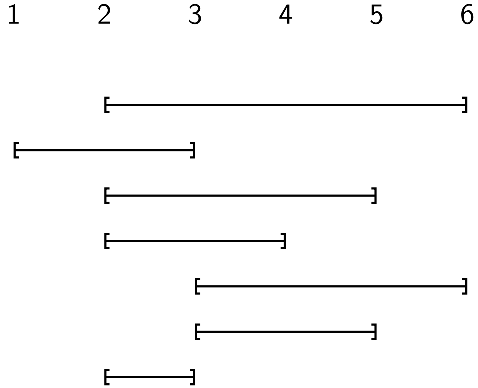
Reformulation

Reformulation

Select a red point on each of n intervals such that the number of red points c_i at coordinate each $x = i$ is divisible by i.

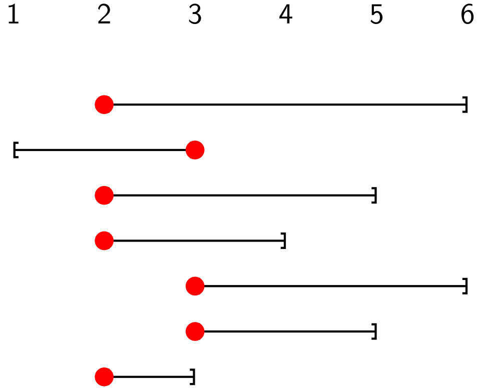
Reformulation

Select a red point on each of n intervals such that the number of red points c_i at coordinate each $x = i$ is divisible by i.



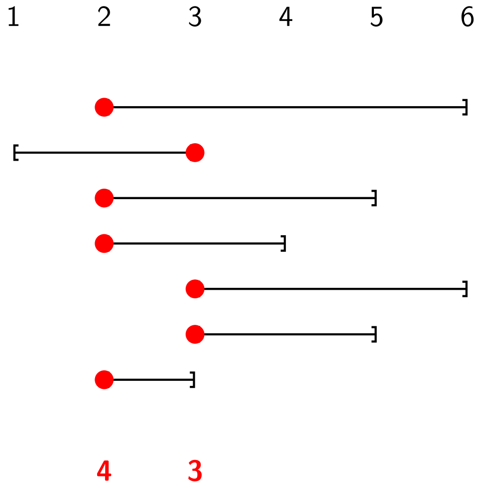
Reformulation

Select a red point on each of n intervals such that the number of red points c_i at coordinate each $x = i$ is divisible by i.



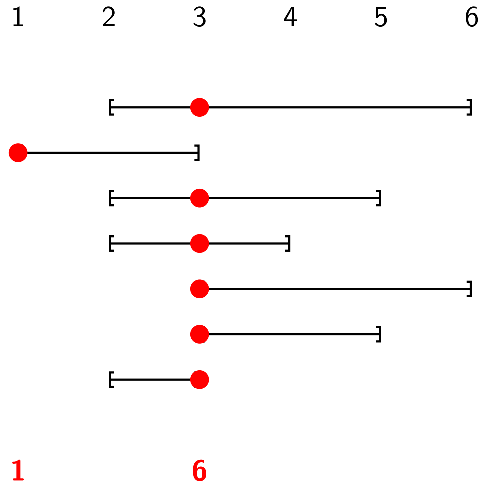
Reformulation

Select a red point on each of n intervals such that the number of red points c_i at coordinate each $x = i$ is divisible by i.



Reformulation

Select a red point on each of n intervals such that the number of red points c_i at coordinate each $x = i$ is divisible by i.



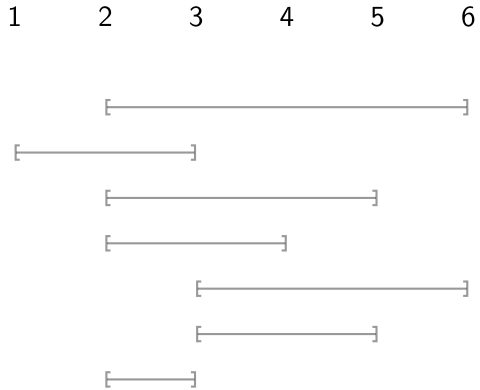
Solving with red counts advice

Solving with red counts advice

Assume red counts are known

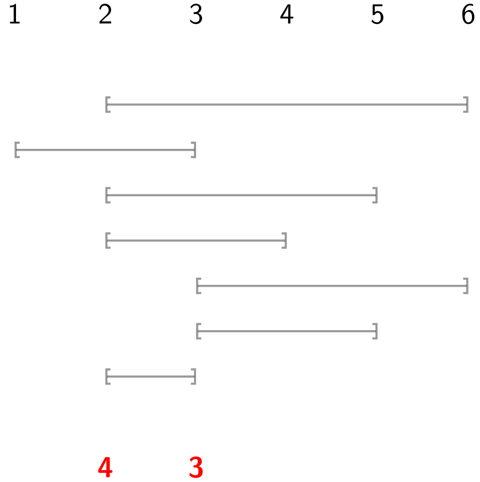
Solving with red counts advice

Assume red counts are known



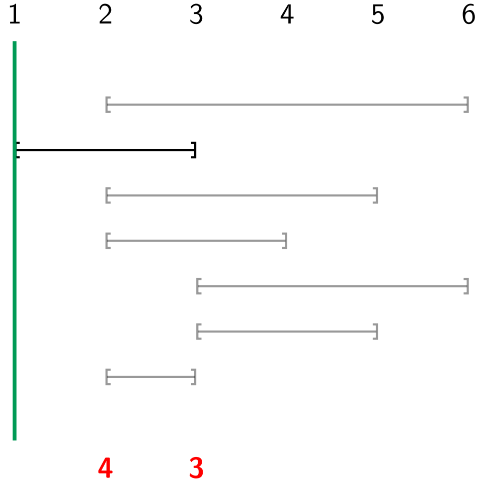
Solving with red counts advice

Assume red counts are known — find out if a solution exists.



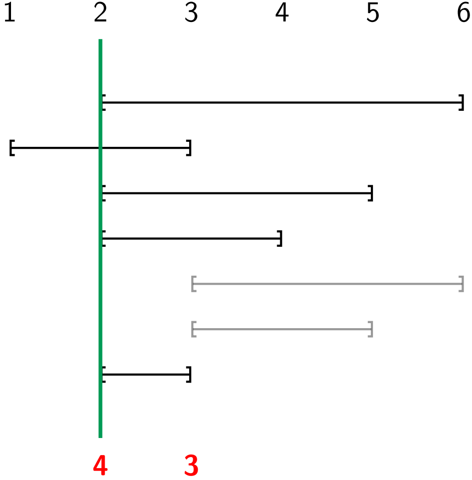
Solving with red counts advice

Assume red counts are known — find out if a solution exists.



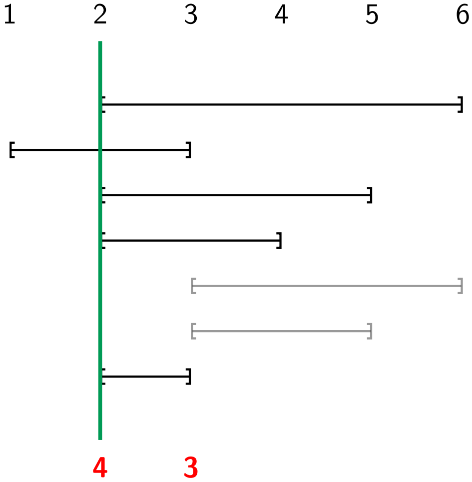
Solving with red counts advice

Assume red counts are known — find out if a solution exists.



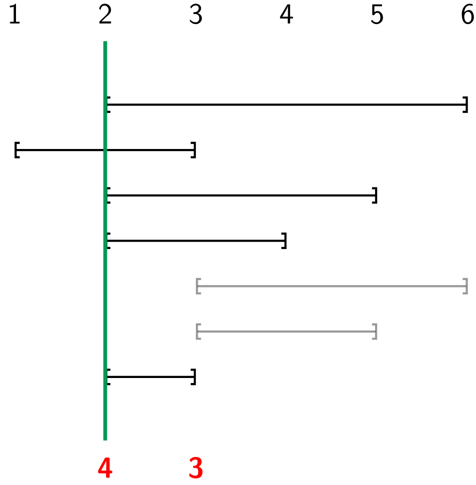
Solving with red counts advice

Assume red counts are known — find out if a solution exists.



Solving with red counts advice

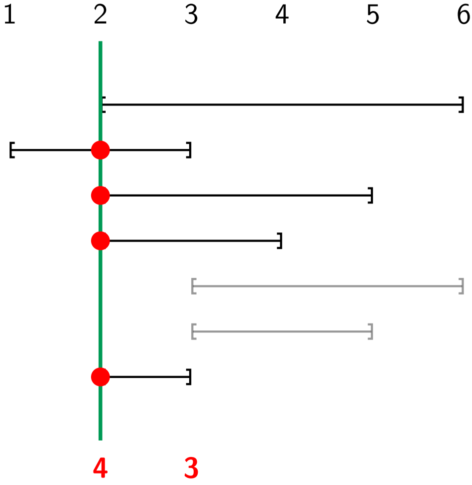
Assume red counts are known — find out if a solution exists.



Select 4 out of 5. Which?

Solving with red counts advice

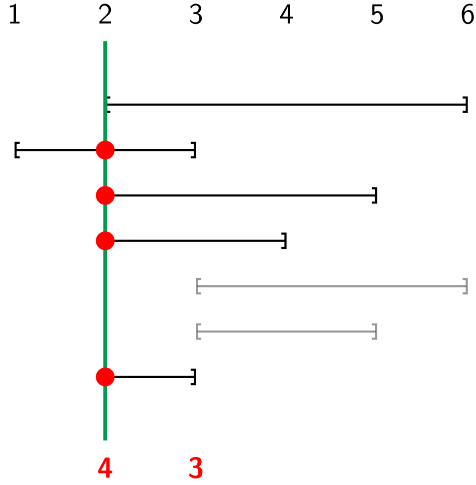
Assume red counts are known — find out if a solution exists.



Select 4 out of 5. Which?

Solving with red counts advice

Assume red counts are known — find out if a solution exists.

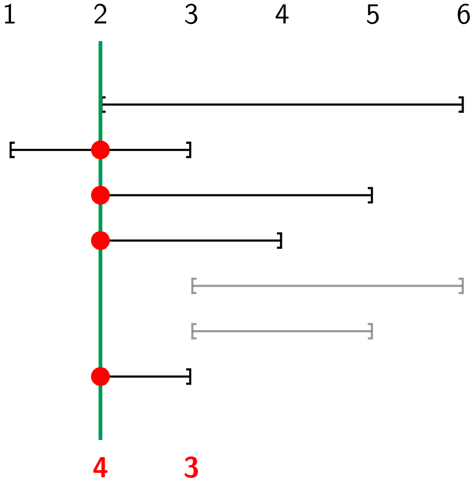


Select 4 out of 5. Which?

Lowest 'active' r_i's!

Solving with red counts advice

Assume red counts are known — find out if a solution exists.



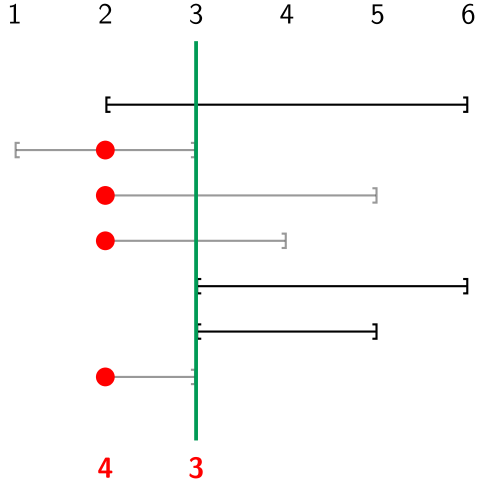
Select 4 out of 5. Which?

Lowest 'active' r_i's!

("Earliest-due-date" in scheduling)

Solving with red counts advice

Assume red counts are known — find out if a solution exists.



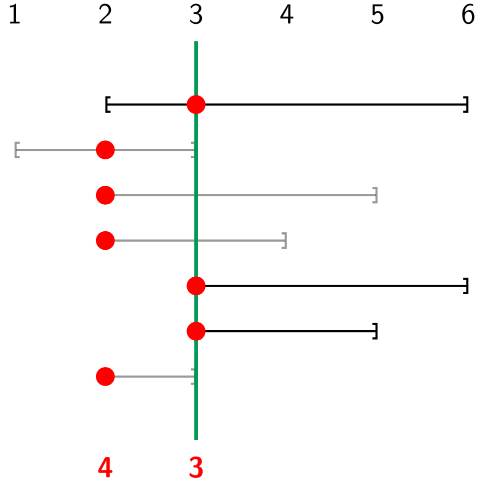
Select 4 out of 5. Which?

Lowest 'active' r_i's!

("Earliest-due-date" in scheduling)

Solving with red counts advice

Assume red counts are known — find out if a solution exists.



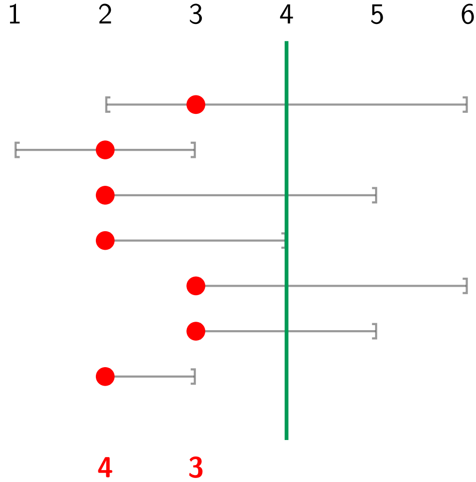
Select 4 out of 5. Which?

Lowest 'active' r_i's!

("Earliest-due-date" in scheduling)

Solving with red counts advice

Assume red counts are known — find out if a solution exists.



Select 4 out of 5. Which?

Lowest 'active' r_i's!

("Earliest-due-date" in scheduling)

But without advice?

Great, but needs red counts advice.

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm;

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm; at each stage, **guess** (in all possible ways) the red count

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm; at each stage, **guess** (in all possible ways) the red count and use earliest-due-date;

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm; at each stage, **guess** (in all possible ways) the red count and use earliest-due-date;
2. $=$ recursion with states (i, A), where $1 \leq i \leq n$ and A is the set of currently "active" intervals;

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm; at each stage, **guess** (in all possible ways) the red count and use earliest-due-date;
2. = recursion with states (i, A), where $1 \leq i \leq n$ and A is the set of currently "active" intervals;
3. + memoization/dynamic programming (DP);

But without advice?

Great, but needs red counts advice.

Simulate red counts advice?

Attempt 1

1. Run the previous algorithm; at each stage, **guess** (in all possible ways) the red count and use earliest-due-date;
2. $=$ recursion with states (i, A), where $1 \leq i \leq n$ and A is the set of currently "active" intervals;
3. $+$ memoization/dynamic programming (DP);

Problem: exponentially many states!

Attempt 2

Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

Attempt 2

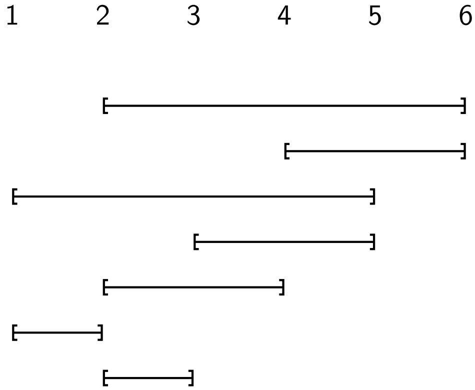
Build an **earliest-due-date** soln., but **not** “in the natural order.”

Idea **[Even et al.’2008]**:

Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

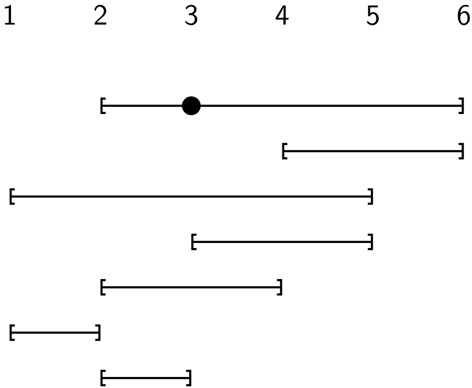
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

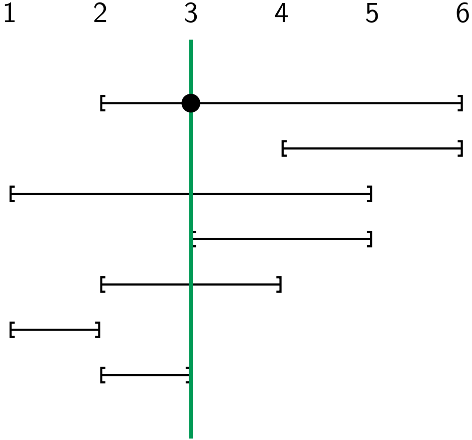
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

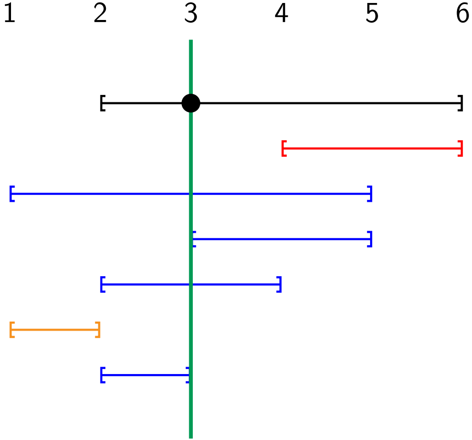
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

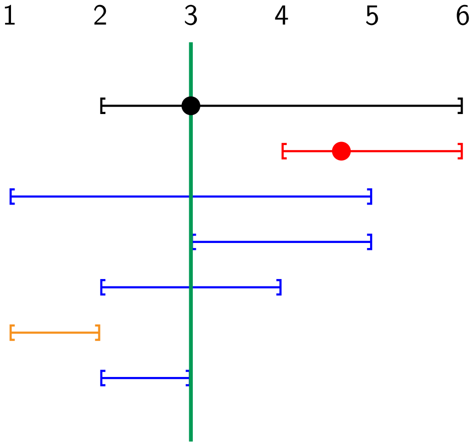
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

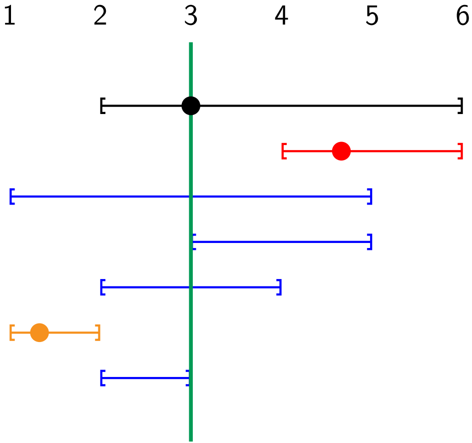
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** "in the natural order."

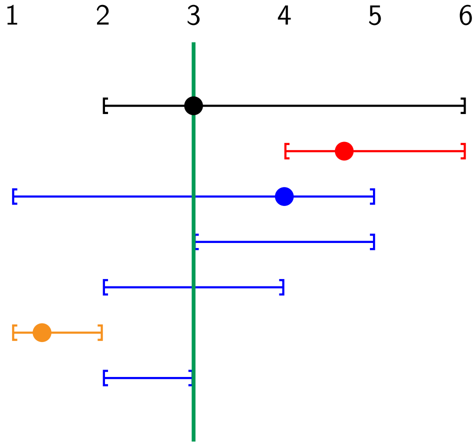
Idea **[Even et al.'2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

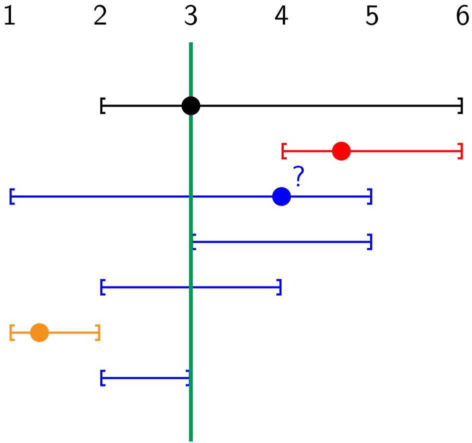
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** "in the natural order."

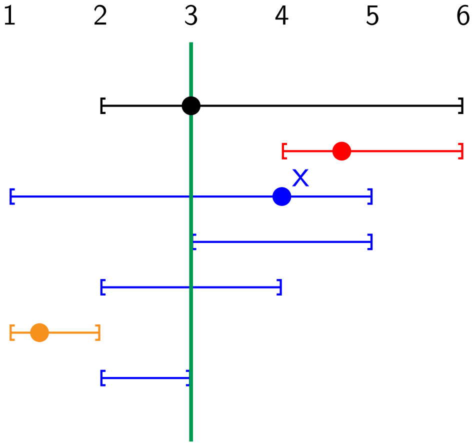
Idea **[Even et al.'2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** "in the natural order."

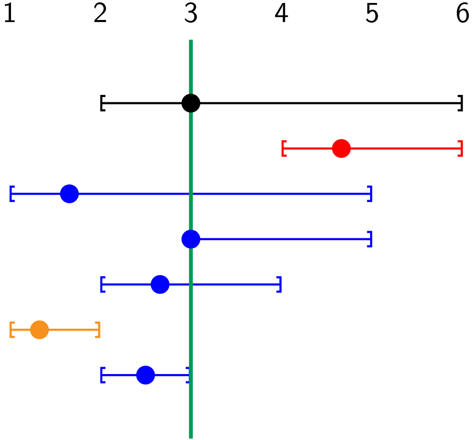
Idea **[Even et al.'2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

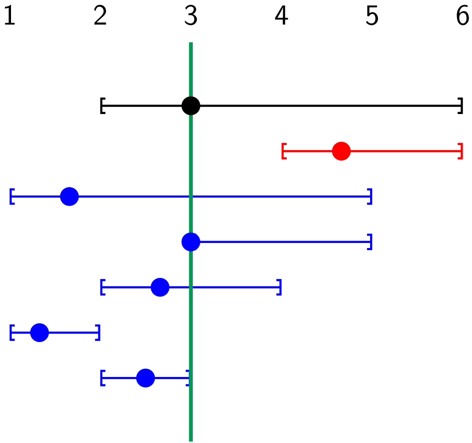
Idea **[Even et al.’2008]**:



Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

Idea **[Even et al.’2008]**:



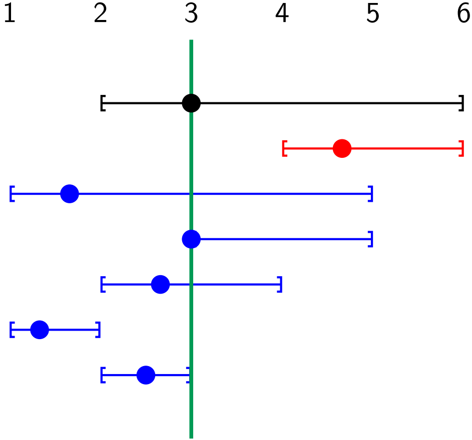
$= \{i \mid \ell_i \leq 3\}$

$= \{i \mid \ell_i > 3\}$

Attempt 2

Build an **earliest-due-date** soln., but **not** "in the natural order."

Idea **[Even et al.'2008]**:



$\vdash\!\dashv = \{i \mid \ell_i \leq 3\}$
$\vdash\!\dashv = \{i \mid \ell_i > 3\}$

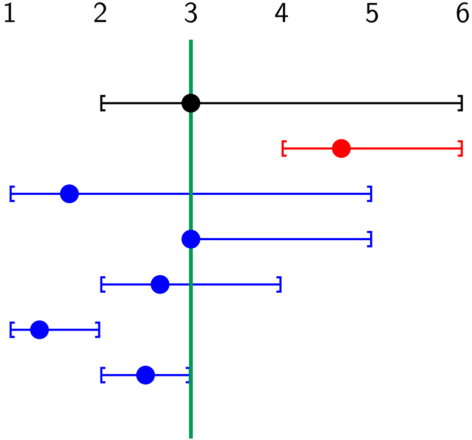
Crucial:

$\bullet \leq 3 < \bullet$

Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

Idea **[Even et al.’2008]**:



$= \{i \mid \ell_i \leq 3\}$
$= \{i \mid \ell_i > 3\}$

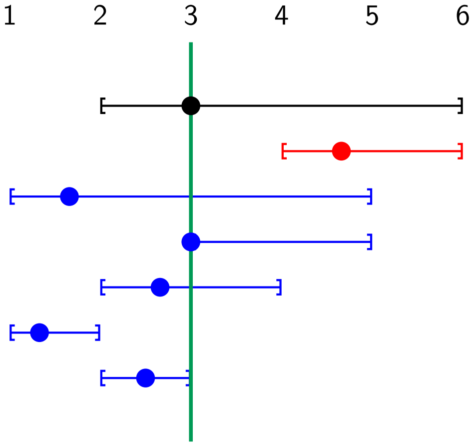
Crucial:
$\leq 3 <$

Solve for $\in [1, 3]$ recursively.

Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

Idea **[Even et al.’2008]**:



$\mapsto$ $= \{i \mid \ell_i \leq 3\}$

$\mapsto$ $= \{i \mid \ell_i > 3\}$

Crucial:

● $\leq 3 <$ ●

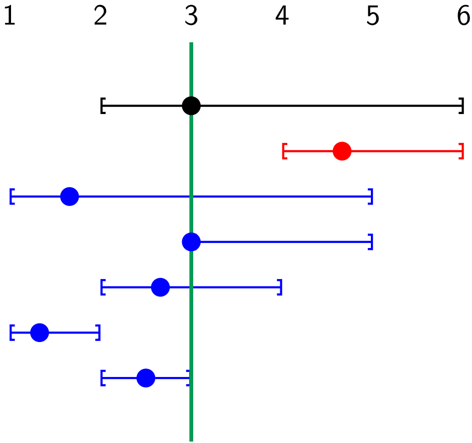
Solve $\mapsto$ for ● $\in [1, 3]$ recursively.

Solve $\mapsto$ for ● $\in [4, 6]$ recursively.

Attempt 2

Build an **earliest-due-date** soln., but **not** “in the natural order.”

Idea **[Even et al.’2008]**:



$\vdash\!\dashv = \{i \mid \ell_i \leq 3\}$

$\vdash\!\dashv = \{i \mid \ell_i > 3\}$

Crucial:

● $\leq 3 <$ ●

Solve $\vdash\!\dashv$ for ● $\in [1, 3]$ recursively. **(don’t forget ●!)**

Solve $\vdash\!\dashv$ for ● $\in [4, 6]$ recursively.

The Recursion

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as:** $r_j \in [1, r_i]$**)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement:

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as:** $r_j \in [1, r_i]$**)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement: w.l.o.g. $j = i$ is **active**.

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement: w.l.o.g. $j = i$ is **active**. **Try out** all possibilities ● $\in [\ell_i, r_i] \cap [x_1, x_2]$ for i's point,

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement: w.l.o.g. $j = i$ is **active**. **Try out** all possibilities ● $\in [\ell_i, r_i] \cap [x_1, x_2]$ for i's point, recurse **left** and **right** for each.

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement: w.l.o.g. $j = i$ is **active**. **Try out** all possibilities ● $\in [\ell_i, r_i] \cap [x_1, x_2]$ for i's point, recurse **left** and **right** for each.

Time complexity: $\mathcal{O}(n^5)$

The Recursion

Assume $r_1 \leq \ldots \leq r_n$.

Recursion with state (x_1, x_2, i, c): **(memoization/DP)**

- Consider only intervals j s.t.: **(active set)**
  - $\ell_j \in [x_1, x_2]$ and
  - $j \leq i$. **(roughly same as: $r_j \in [1, r_i]$)**
- Select a point for each interval in $[x_1, x_2]$ to satisfy the divisibility constraints.
- Assuming c **exogenous points** ● already present at x_2.

To implement: w.l.o.g. $j = i$ is **active**. **Try out** all possibilities ● $\in [\ell_i, r_i] \cap [x_1, x_2]$ for i's point, recurse **left** and **right** for each.

Time complexity: $\mathcal{O}(n^5)$ **(open: better?)**

Extensions

Extensions

- Hiking-Min-Delete: Delete a minimum number of agents to make the rest feasible.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- HIKING-MAX-SATISFIED: Satisfy as many agents as possible.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- HIKING-MAX-SATISFIED: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k.

Extensions

- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- HIKING-MAX-SATISFIED: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$

Extensions

- Hiking-Min-Delete: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- Hiking-X-Delete: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- Hiking-Max-Satisfied: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$ such that $(N \cup D_k) \setminus N'$ is feasible, where D_k are k dummies.

Extensions

- Hiking-Min-Delete: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- Hiking-X-Delete: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- Hiking-Max-Satisfied: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$ such that $(N \cup D_k) \setminus N'$ is feasible, where D_k are k dummies. This is done with Hiking-X-Delete.

Extensions

- Hiking-Min-Delete: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- Hiking-X-Delete: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- Hiking-Max-Satisfied: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$ such that $(N \cup D_k) \setminus N'$ is feasible, where D_k are k dummies. This is done with Hiking-X-Delete.
  - **Time complexity:** $\mathcal{O}(n^7 \log n)$;

Extensions

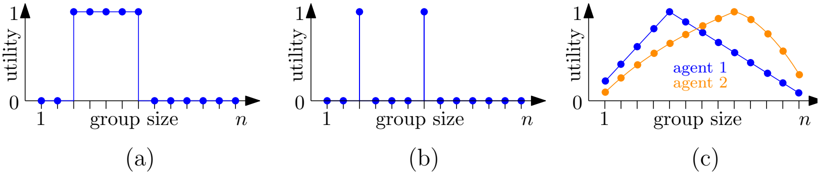
- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- HIKING-MAX-SATISFIED: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$ such that $(N \cup D_k) \setminus N'$ is feasible, where D_k are k dummies. This is done with HIKING-X-DELETE.
  - **Time complexity:** $\mathcal{O}(n^7 \log n)$;
  - Binary search is not needed actually, so $\mathcal{O}(n^7)$.

Extensions

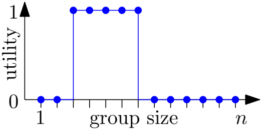
- HIKING-MIN-DELETE: Delete a minimum number of agents to make the rest feasible.
  - Same DP, but make it return the minimum number of deleted agents instead of yes/no. **Time complexity:** $\mathcal{O}(n^5)$
- HIKING-X-DELETE: Delete exactly x agents to make the rest feasible.
  - Add another variable to the DP. **Time complexity:** $\mathcal{O}(n^7)$
- HIKING-MAX-SATISFIED: Satisfy as many agents as possible.
  - Binary search for the number of unsatisfied agents k. Need to find $N' \subseteq N$ with $|N'| = k$ such that $(N \cup D_k) \setminus N'$ is feasible, where D_k are k dummies. This is done with HIKING-X-DELETE.
  - **Time complexity:** $\mathcal{O}(n^7 \log n)$;
  - Binary search is not needed actually, so $\mathcal{O}(n^7)$.
- weighted extensions ...

A Different Problem

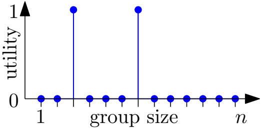
A Different Problem



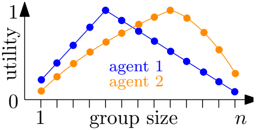
A Different Problem



(a)



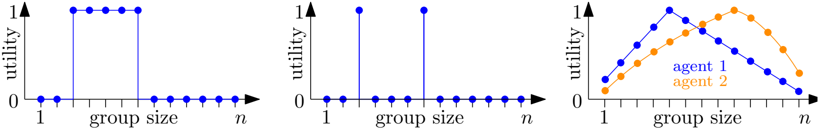
(b)



(c)

Given:

A Different Problem

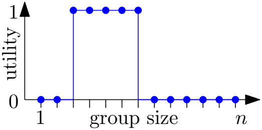


(a) (b) (c)

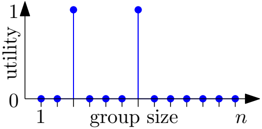
Given:

- n agents, each agent i with preferred group size s_i;

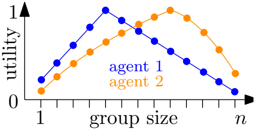
A Different Problem



(a)



(b)

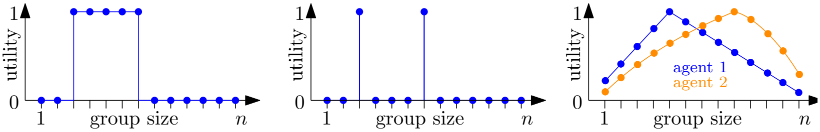


(c)

Given:

- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;

A Different Problem

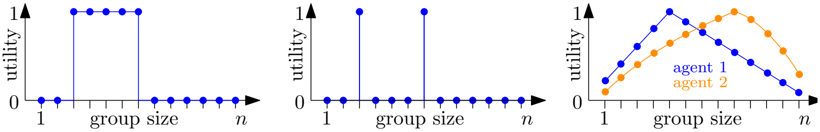


(a) (b) (c)

Given:

- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

A Different Problem



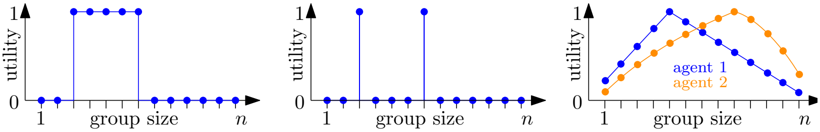
(a) (b) (c)

Given:

- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

Goal:

A Different Problem



(a) (b) (c)

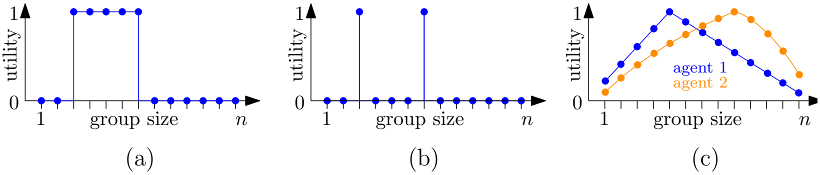
Given:

- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

Goal:

- Partition that minimizes the total/maximum cost of an agent.

A Different Problem



Given:

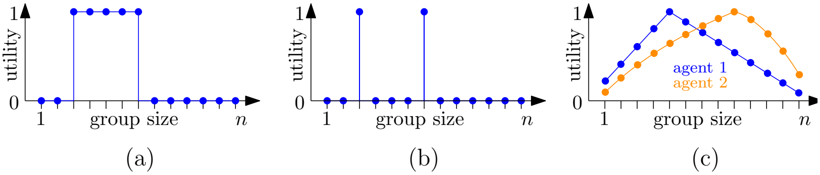
- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

Goal:

- Partition that minimizes the total/maximum cost of an agent.

Results:

A Different Problem



Given:

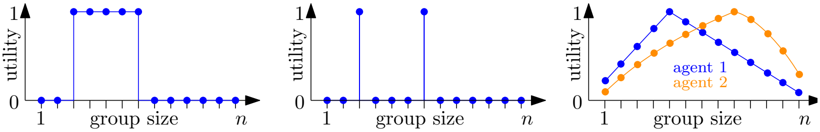
- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

Goal:

- Partition that minimizes the total/maximum cost of an agent.

Results: Under assumptions on f, we solve both versions in $\mathcal{O}(n^2)$ time.

A Different Problem



(a) (b) (c)

Given:

- n agents, each agent i with preferred group size s_i;
- Global function $f : \mathbb{R}^2 \to \mathbb{R}$; e.g., $f(x, y) = |x - y|$;
- Agent i being in a group of size s incurs cost $f(s_i, s)$.

Goal:

- Partition that minimizes the total/maximum cost of an agent.

Results: Under assumptions on f, we solve both versions in $\mathcal{O}(n^2)$ time. And can even find the best that can be achieved by removing at most α agents in time $\mathcal{O}(n^2(\alpha + 1))$.

Open Problems

- Approximation algorithms/FPT for the NP-hard cases.

Open Problems

- Approximation algorithms/FPT for the NP-hard cases.
- Strategic aspects for the optimization variants.

Open Problems

- Approximation algorithms/FPT for the NP-hard cases.
- Strategic aspects for the optimization variants.

Gerhard Woeginger (1964—2022) RIP