Design and Analysis of Algorithms

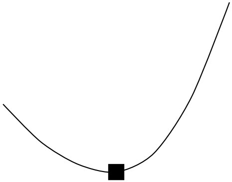
Presented by Dr. Li Ning

Shenzhen Institutes of Advanced Technology, Chinese Academy of Science
Shenzhen, China

Greedy Algorithm

Climb The Hill

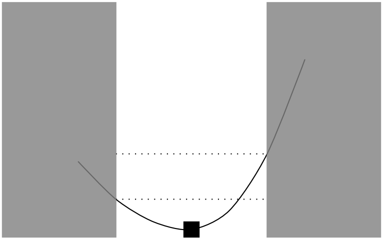
Climb The Hill



Climb The Hill

$$x = 0$$

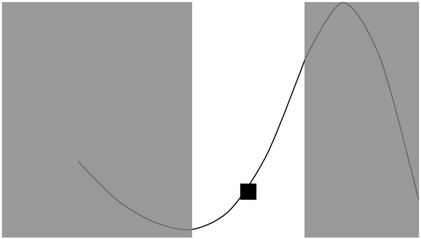
$$y \uparrow 0.6 \qquad y \uparrow 1$$



Climb The Hill

$$x = 1$$

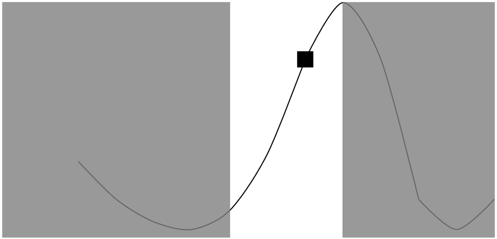
$$y \downarrow 1 \qquad y \uparrow 2$$



Climb The Hill

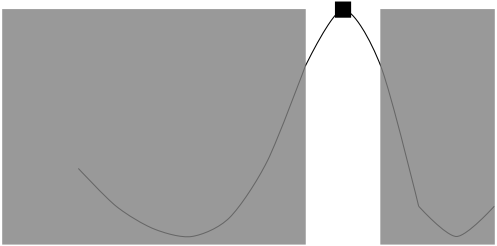
$$x = 2$$

$$y \downarrow 2 \qquad y \uparrow 0.5$$



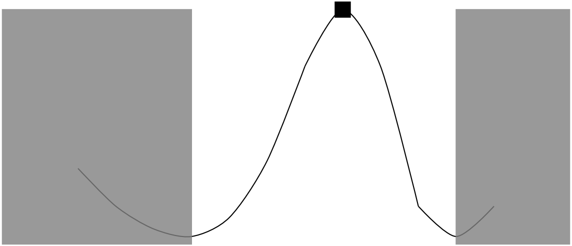
Climb The Hill

$$x = 3$$



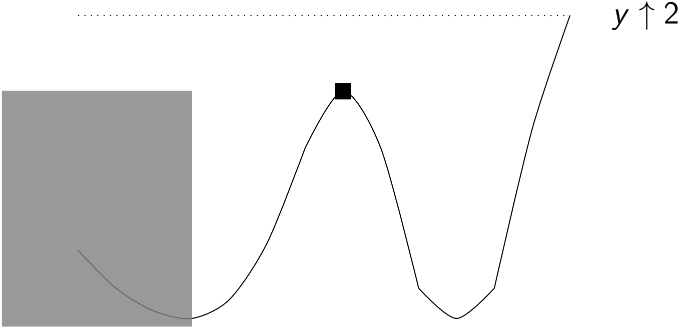
Climb The Hill

$$x = 3$$

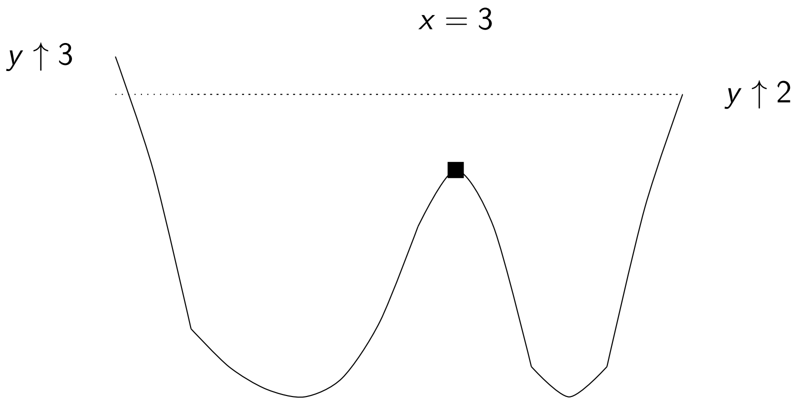


Climb The Hill

$$x = 3$$



Climb The Hill



Climb The Hill

Local search: Optimize the action within the observable local area.

Climb The Hill

Local search: Optimize the action within the observable local area.

- y increases by 1 if $x = x + 1$;
- y increases by 0.5 if $x = x - 1$;

Climb The Hill

Local search: Optimize the action within the observable local area.

- y increases by 1 if $x = x + 1$;
- y increases by 0.5 if $x = x - 1$;
- go with $x = x + 1$, if the **maximum** is desired.

Climb The Hill

Local search: Optimize the action within the observable local area.

- y increases by 1 if $x = x + 1$;
- y increases by 0.5 if $x = x - 1$;
- go with $x = x + 1$, if the **maximum** is desired.

How far can you see?

- narrow (local) area:
  - less choice
  - less calculation
- wide (local) area
  - more choice
  - more calculation

Revisit: Knapsack

The Knapsack Problem: 0/1 Version

Given

- a container $\mathcal{K}$ of capacity W
- n items $\{x_0, x_1, \ldots, x_{n-1}\}$
  - integral weight $w_i > 0$
  - value $v_i > 0$

Fill the knapsack so as to maximize the total value.

The Knapsack Problem: Fractional Version

When filling the knapsack, you can take part of an item.

Consider an item of weight 2 and value 4. Taking $1/2$ of the item results in

- weight 1
- value 2

The Knapsack Problem: Example

$\mathcal{K}$ of capacity $W = 10$

- x_0 : $w_0 = 1$, $v_0 = 1$
- x_1 : $w_1 = 2$, $v_1 = 6$
- x_2 : $w_2 = 5$, $v_2 = 18$
- x_3 : $w_3 = 6$, $v_3 = 22$
- x_4 : $w_4 = 7$, $v_4 = 28$

The Knapsack Problem: Example

0/1 version

- x_0, x_1, x_4
- total weight: 10
- total value: 35

Fractional version

- $\frac{1}{2}x_3, x_4$
- total weight: 10
- total value: $28 + \frac{1}{2}22 = 39$

$\mathcal{K}$ of capacity $W = 10$

- $x_0 : w_0 = 1,\ v_0 = 1$
- $x_1 : w_1 = 2,\ v_1 = 6$
- $x_2 : w_2 = 5,\ v_2 = 18$
- $x_3 : w_3 = 6,\ v_3 = 22$
- $x_4 : w_4 = 7,\ v_4 = 28$

The Knapsack Problem: Greedy

Select the item of the maximum v_i/w_i

$\mathcal{K}$ of capacity $W = 10$

- $x_0 : w_0 = 1, v_0 = 1$
- $x_1 : w_1 = 2, v_1 = 6$
- $x_2 : w_2 = 5, v_2 = 18$
- $x_3 : w_3 = 6, v_3 = 22$
- $x_4 : w_4 = 7, v_4 = 28$

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

The Knapsack Problem: Greedy

Select the item of the maximum v_i/w_i

- select x_4
  - capacity: $10 - 7 = 3$
  - value: 28

$\mathcal{K}$ of capacity $W = 10$

- $x_0 : w_0 = 1, v_0 = 1$
- $x_1 : w_1 = 2, v_1 = 6$
- $x_2 : w_2 = 5, v_2 = 18$
- $x_3 : w_3 = 6, v_3 = 22$
- $x_4 : w_4 = 7, v_4 = 28$

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

The Knapsack Problem: Greedy

Select the item of the maximum v_i/w_i

- select x_4
  - capacity: $10 - 7 = 3$
  - value: 28
- take $\frac{3}{w_3} = \frac{1}{2}$ of x_3
  - capacity: $3 - 3 = 0$
  - value: $\frac{1}{2}22 = 11$

$\mathcal{K}$ of capacity $W = 10$

- $x_0 : w_0 = 1, v_0 = 1$
- $x_1 : w_1 = 2, v_1 = 6$
- $x_2 : w_2 = 5, v_2 = 18$
- $x_3 : w_3 = 6, v_3 = 22$
- $x_4 : w_4 = 7, v_4 = 28$

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

The Knapsack Problem: Greedy

Select the item of the maximum v_i/w_i

- select x_4
  - capacity: $10 - 7 = 3$
  - value: 28
- take $\frac{3}{w_3} = \frac{1}{2}$ of x_3
  - capacity: $3 - 3 = 0$
  - value: $\frac{1}{2}22 = 11$
- total value: 39

$\mathcal{K}$ of capacity $W = 10$

- x_0 : $w_0 = 1$, $v_0 = 1$
- x_1 : $w_1 = 2$, $v_1 = 6$
- x_2 : $w_2 = 5$, $v_2 = 18$
- x_3 : $w_3 = 6$, $v_3 = 22$
- x_4 : $w_4 = 7$, $v_4 = 28$

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

The Knapsack Problem: Greedy

Select the item of the maximum v_i/w_i

- select x_4
  - capacity: $10 - 7 = 3$
  - value: 28
- take $\frac{3}{w_3} = \frac{1}{2}$ of x_3
  - capacity: $3 - 3 = 0$
  - value: $\frac{1}{2}22 = 11$
- total value: 39

Is 39 optimal?

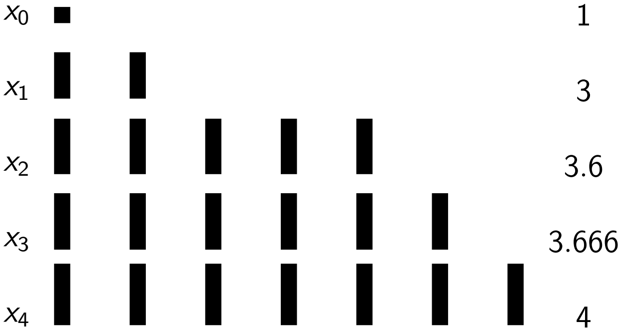
$\mathcal{K}$ of capacity $W = 10$

- $x_0 : w_0 = 1, v_0 = 1$
- $x_1 : w_1 = 2, v_1 = 6$
- $x_2 : w_2 = 5, v_2 = 18$
- $x_3 : w_3 = 6, v_3 = 22$
- $x_4 : w_4 = 7, v_4 = 28$

- $v_0/w_0 = 1/1 = 1$
- $v_1/w_1 = 6/2 = 3$
- $v_2/w_2 = 18/5 = 3.6$
- $v_3/w_3 = 22/6 = 3.666$
- $v_4/w_4 = 28/7 = 4$

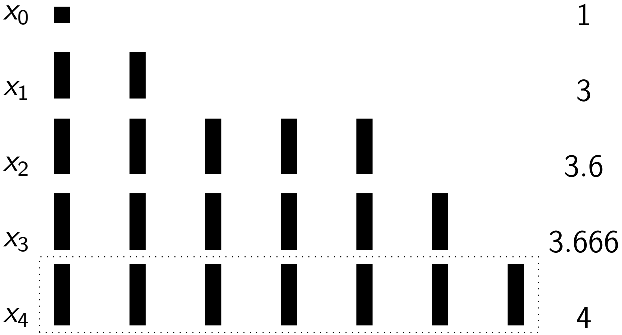
The Knapsack Problem: Greedy

Item x_i can be divided into w_i pieces. Thus each piece is of weight 1, and value v_i/w_i.



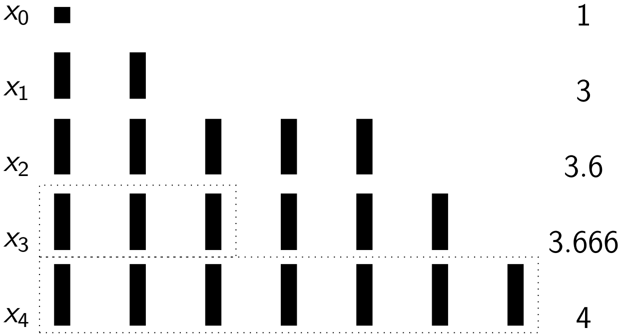
The Knapsack Problem: Greedy

Item x_i can be divided into w_i pieces. Thus each piece is of weight 1, and value v_i/w_i.

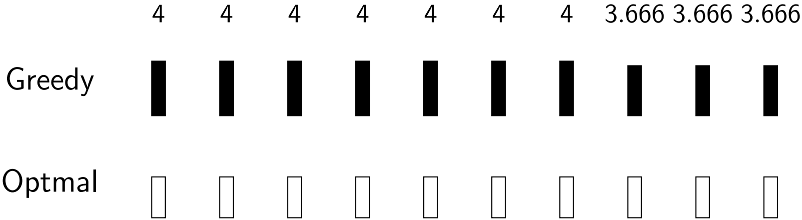


The Knapsack Problem: Greedy

Item x_i can be divided into w_i pieces. Thus each piece is of weight 1, and value v_i/w_i.



The Knapsack Problem: Transform



The Knapsack Problem: Transform

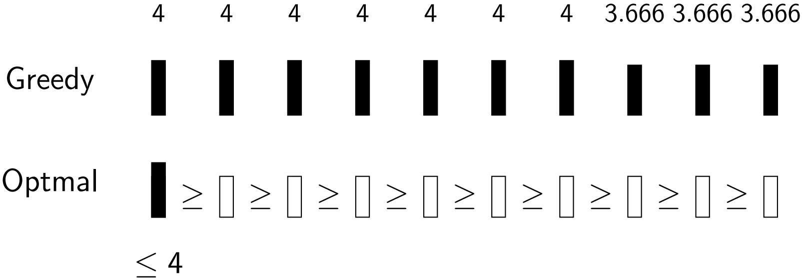
4 4 4 4 4 4 4 3.666 3.666 3.666

Greedy

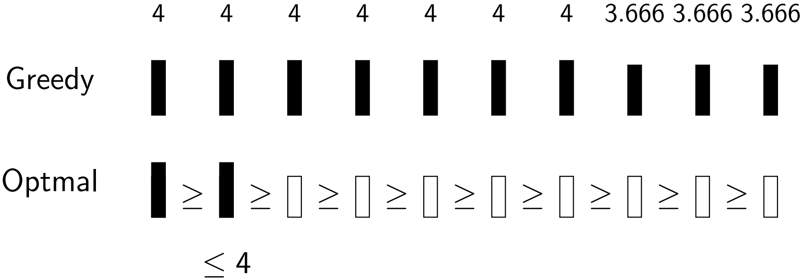
Optmal $\square \geq \square \geq \square \geq \square \geq \square \geq \square \geq \square \geq \square \geq \square \geq \square$

≤ 4

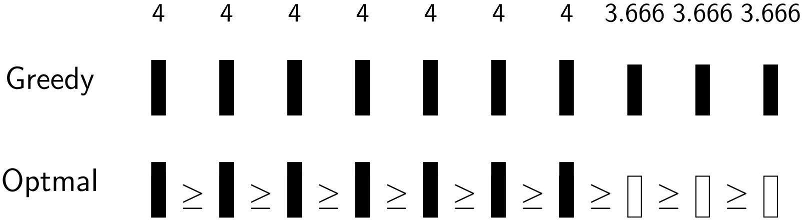
The Knapsack Problem: Transform



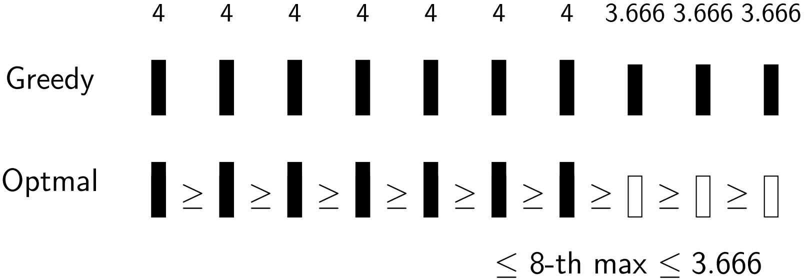
The Knapsack Problem: Transform



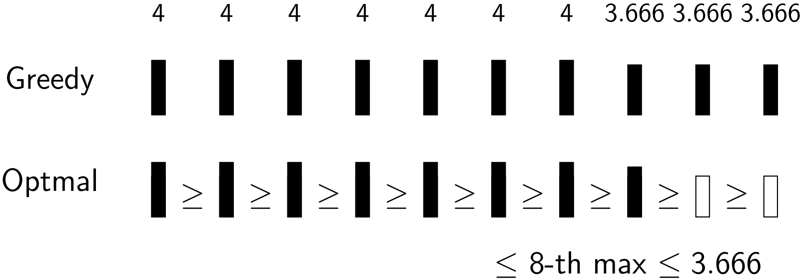
The Knapsack Problem: Transform



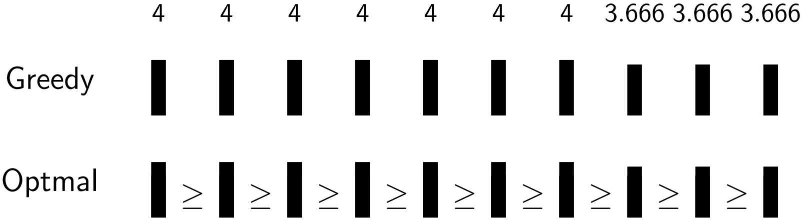
The Knapsack Problem: Transform



The Knapsack Problem: Transform



The Knapsack Problem: Transform



Paradigm of The Greedy Algorithms

Greedy Algorithms

Algorithm is **greedy** if

- it builds up a solution in small steps.
- it optimizes the action at each step according to the local observation.

Greedy Algorithms

Algorithm is **greedy** if

- it builds up a solution in small steps.
- it optimizes the action at each step according to the local observation.

The optimality of the greedy algorithm is shown by

- in every step, it is **not worse** than **any other** algorithm.
- every algorithm can be gradually transformed to the greedy one **without loosing any quality**.

Cash Only

Pay with Cash

Pay $B \in \mathbb{Z}^+$ with coins of value $v_0 > v_1 > \ldots > v_{n-1}$, where $v_i \in \mathbb{Z}^+$ and $v_{n-1} = 1$. Try to minimize the number of coins.

For $i = 0$ to $n - 1$

- find max n_i such that $n_i \cdot v_i \leq B$
- $B = B - n_i \cdot v_i$

Pay with n_i coins of value v_i.

Interval Scheduling

Scheduling

Given jobs

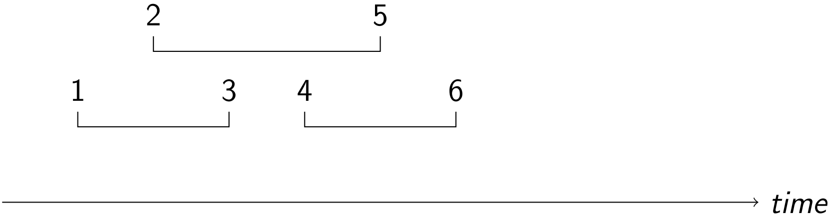
- Job 1: must be started at time 1 and finished at time 3
- Job 2: must be started at time 2 and finished at time 5
- Job 3: must be started at time 4 and finished at time 6

Restriction: two jobs can not be processed at the same time.

Problem: Try to process as many jobs as possible.

- All three jobs: overlap!
- Job 1 and Job 2: overlap!
- Job 2 and Job 3: overlap!
- Job 1 and Job 3: great!

Interval Scheduling



Problem: Find the largest subset of the given intervals, such that none two of them overlap.

Interval Scheduling: Greedy

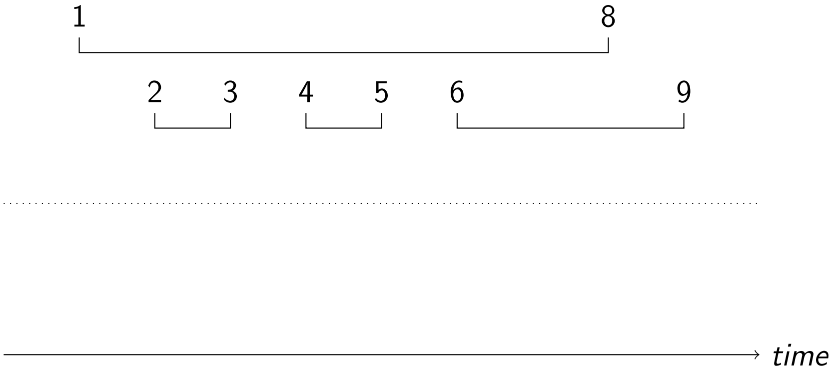
- Initialize a set as empty

$$S = \emptyset$$

- Select the intervals one by one **according to a specific rule**, and put it into S.
- Stop when no interval can be added.

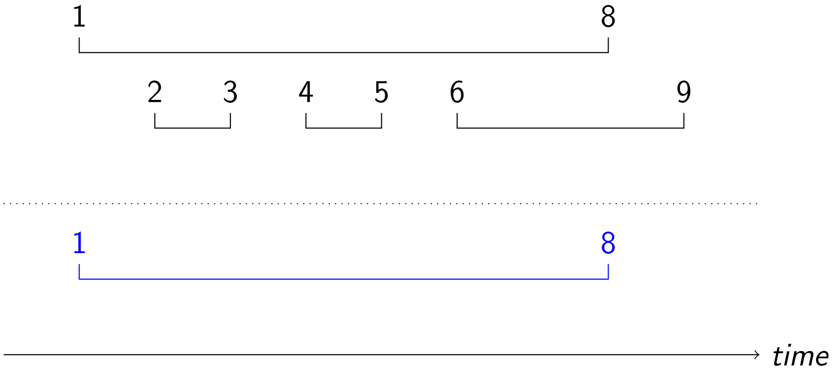
The First Try

Rule: select the interval (among the remaining ones) that starts earliest, but not overlapping the already selected ones.



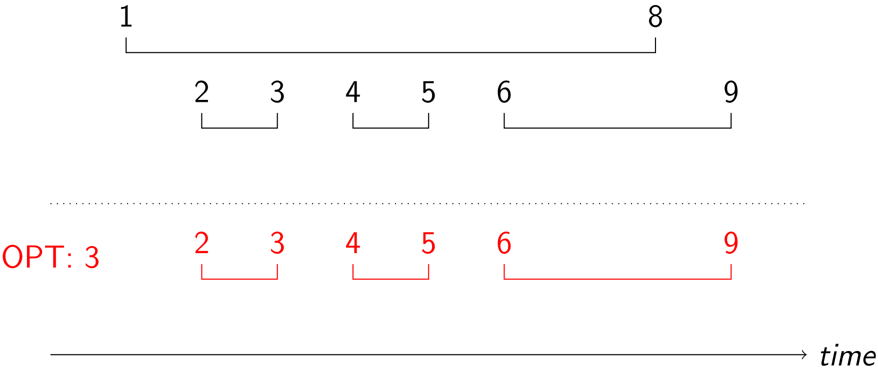
The First Try

Rule: select the interval (among the remaining ones) that starts earliest, but not overlapping the already selected ones.



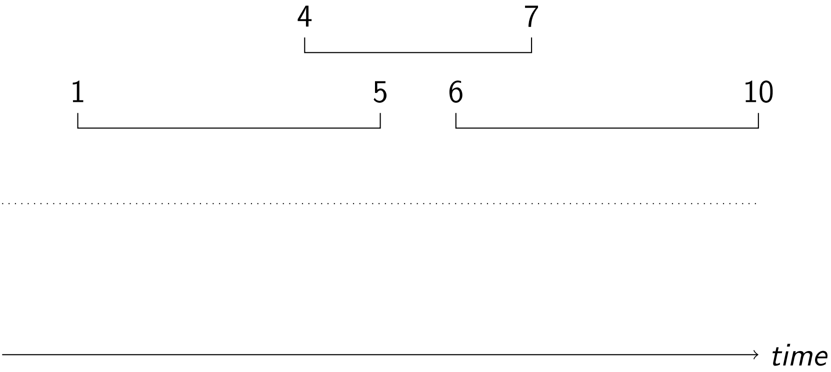
The First Try

Rule: select the interval (among the remaining ones) that starts earliest, but not overlapping the already selected ones.



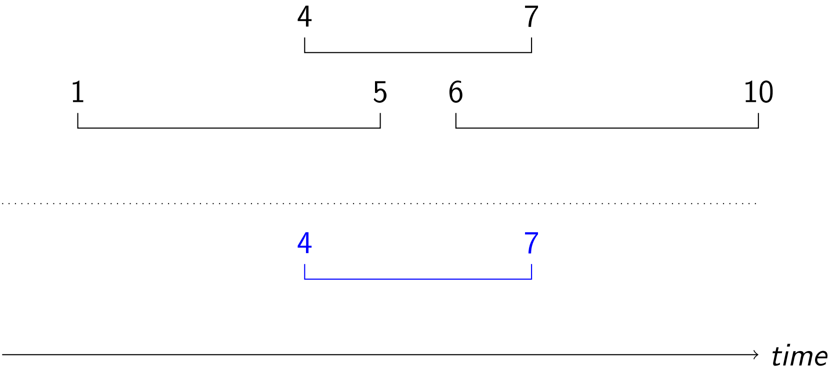
The Second Try

Rule: select the interval (among the remaining ones) that is the shortest, but not overlapping the already selected ones.



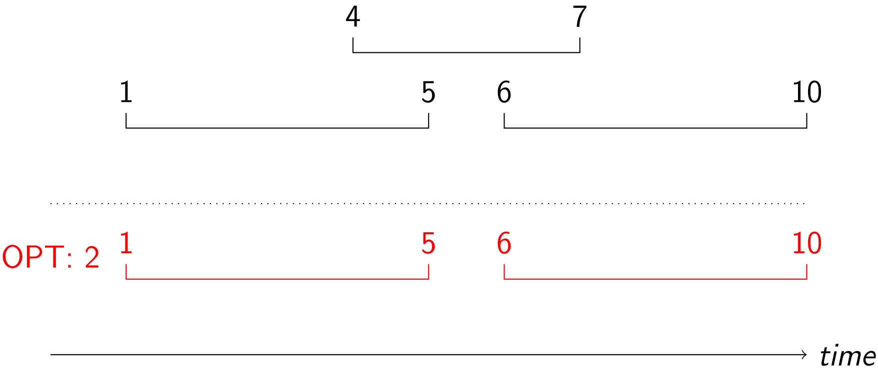
The Second Try

Rule: select the interval (among the remaining ones) that is the shortest, but not overlapping the already selected ones.



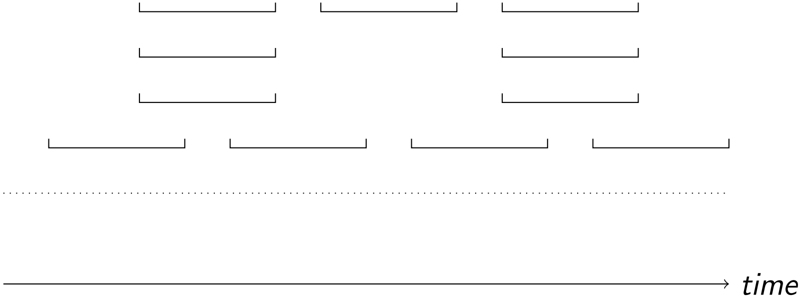
The Second Try

Rule: select the interval (among the remaining ones) that is the shortest, but not overlapping the already selected ones.



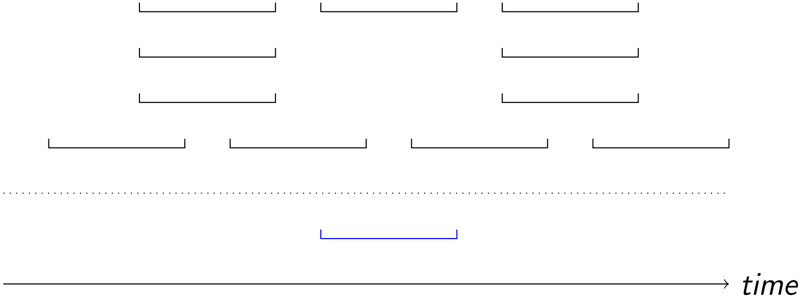
The Third Try

Rule: select the interval (among the remaining ones) that conflicts fewest, but not overlapping the already selected ones.



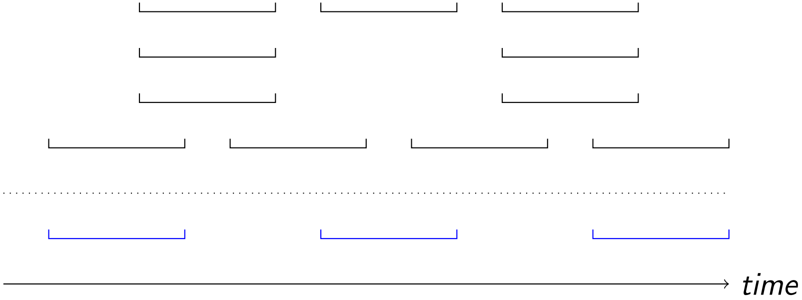
The Third Try

Rule: select the interval (among the remaining ones) that conflicts fewest, but not overlapping the already selected ones.



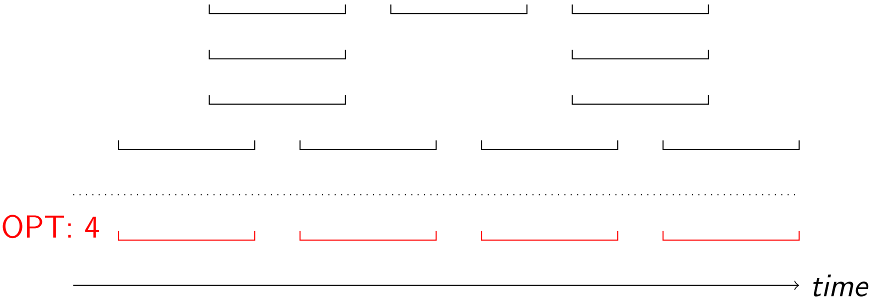
The Third Try

Rule: select the interval (among the remaining ones) that conflicts fewest, but not overlapping the already selected ones.



The Third Try

Rule: select the interval (among the remaining ones) that conflicts fewest, but not overlapping the already selected ones.



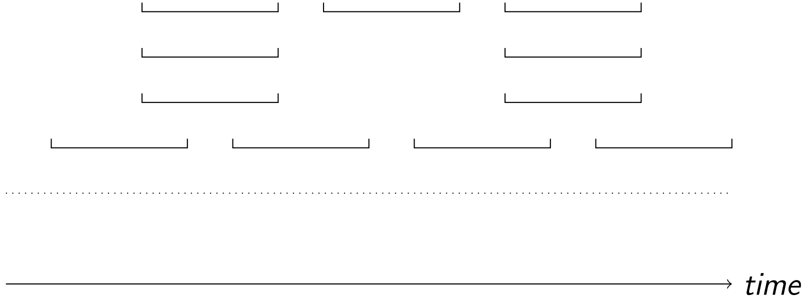
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.

Idea: after processing a job, we want as much remaining time as possible, and thus we have more chance to process the other jobs.

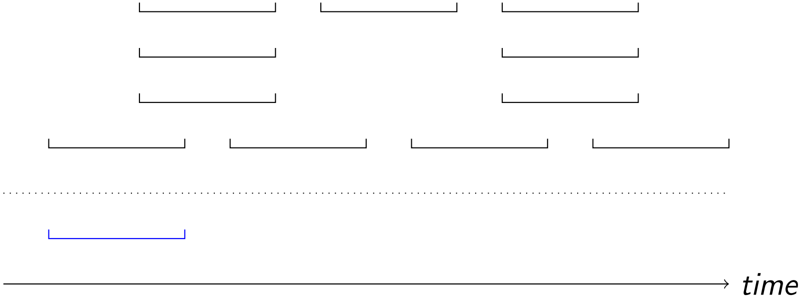
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



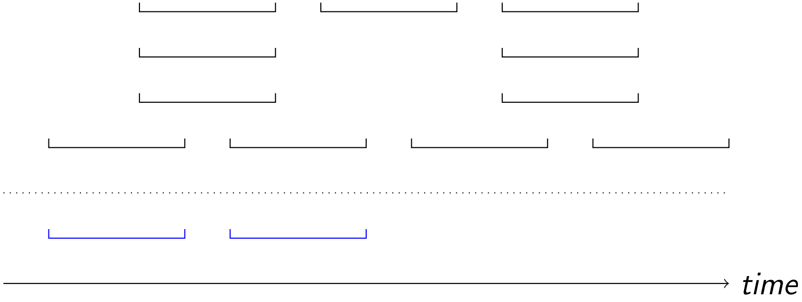
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



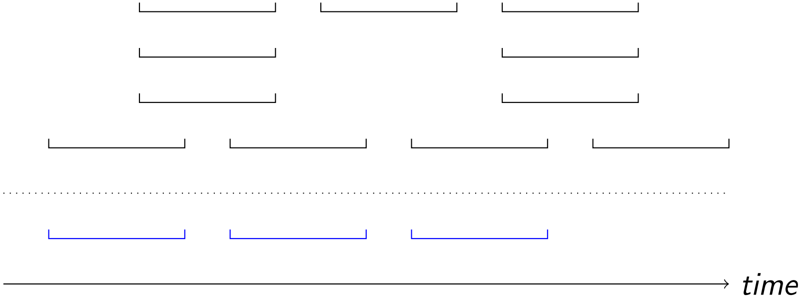
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



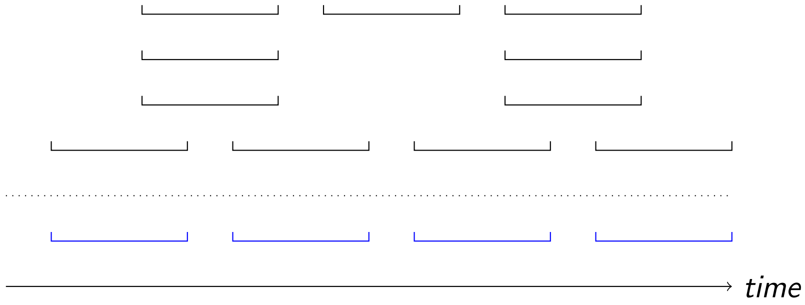
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



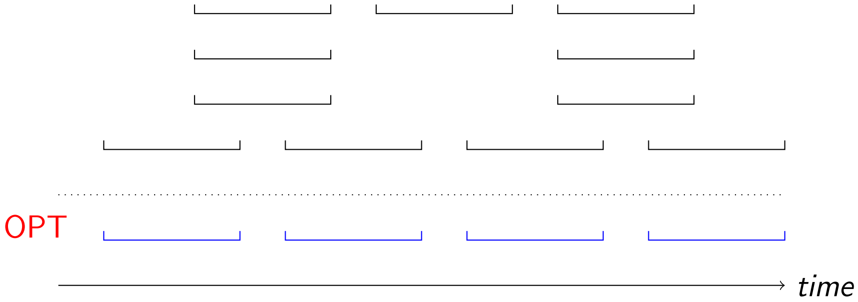
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



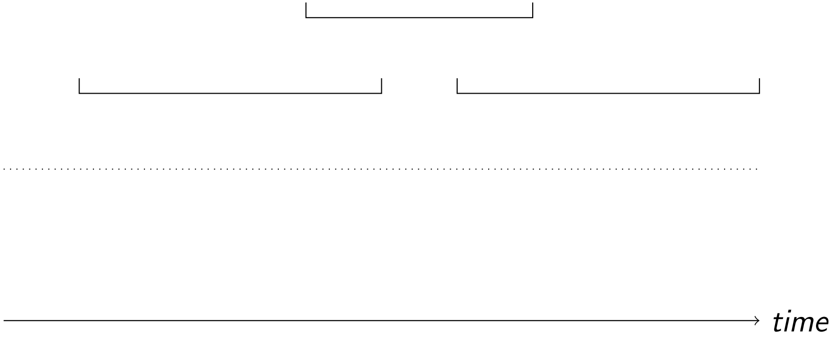
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



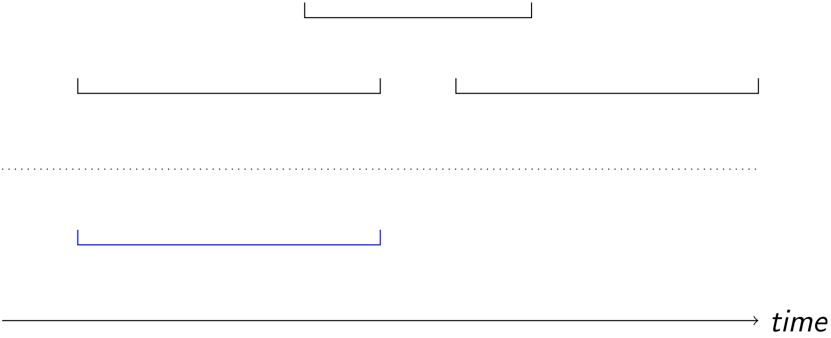
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



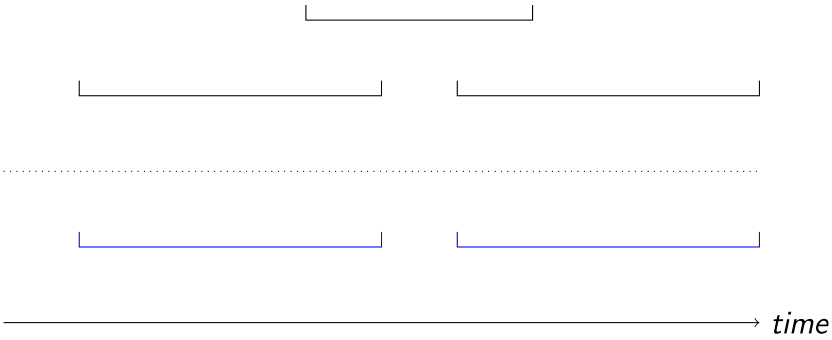
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



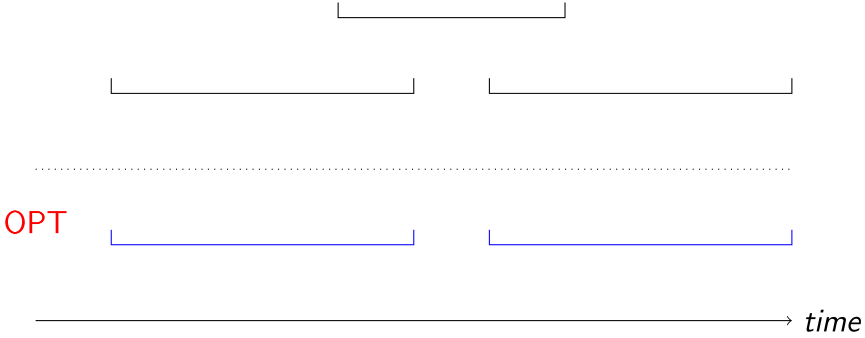
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



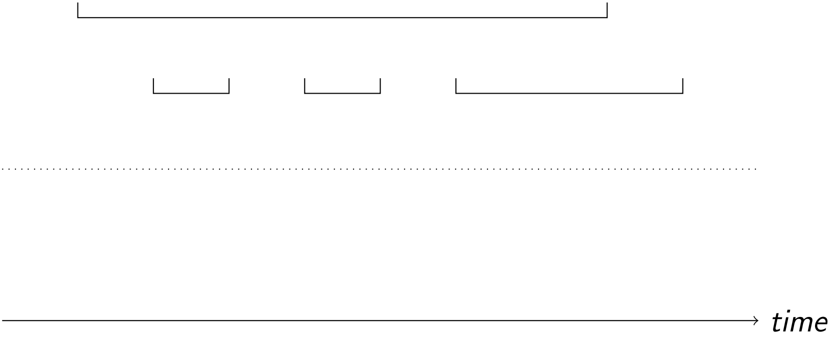
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



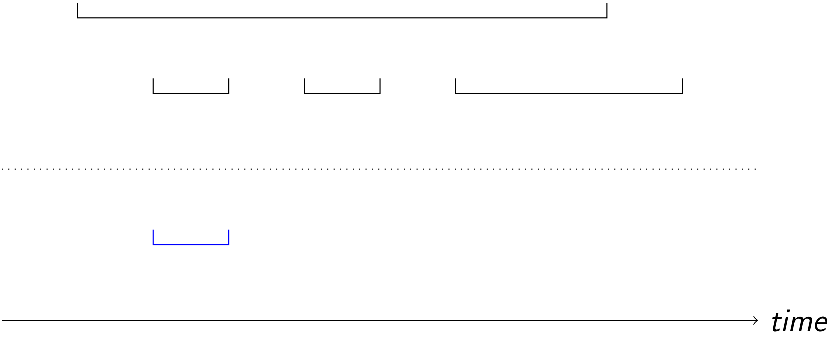
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



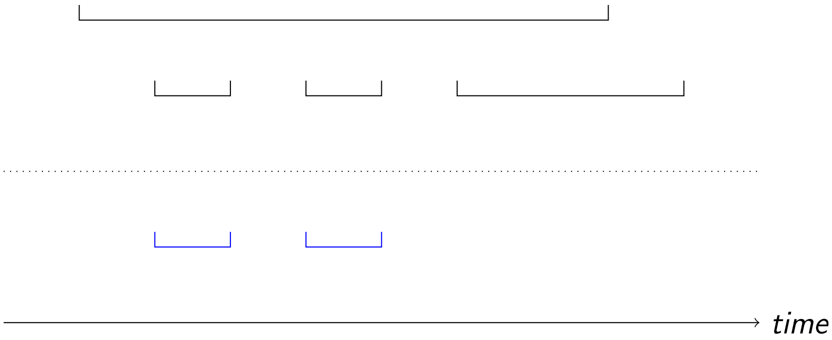
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



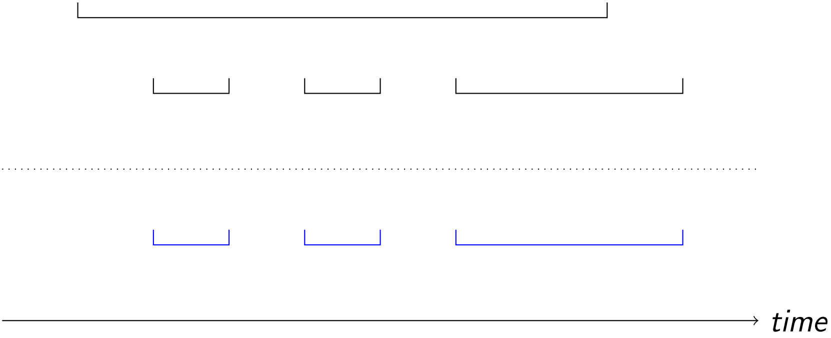
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



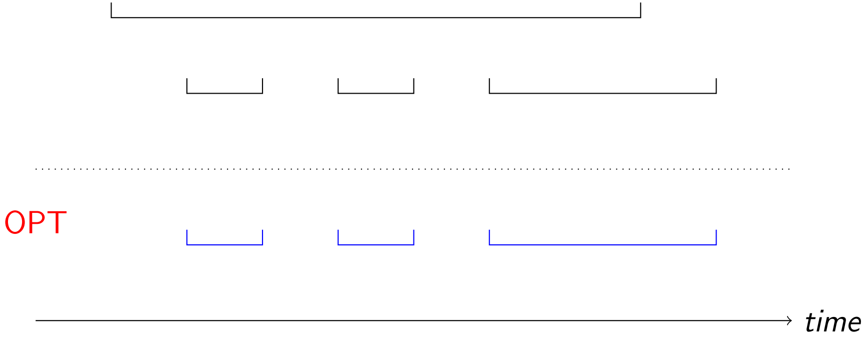
The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



The Fourth Try

Rule: select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.



The Fourth Try: Analysis

Followed the restriction: when a job is selected, it has been checked that the new added job overlaps with none of the previously added ones.

The Fourth Try: Analysis

Followed the restriction: when a job is selected, it has been checked that the new added job overlaps with none of the previously added ones.

- S: the set of intervals selected according to the rule
  - select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.

The Fourth Try: Analysis

Followed the restriction: when a job is selected, it has been checked that the new added job overlaps with none of the previously added ones.

- S: the set of intervals selected according to the rule
  - select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.
- P: the optimal solution
  - one of the largest set of non-overlapping intervals.

The Fourth Try: Analysis

Followed the restriction: when a job is selected, it has been checked that the new added job overlaps with none of the previously added ones.

- S: the set of intervals selected according to the rule
  - select the interval (among the remaining ones) that ends first, but not overlapping the already selected ones.
- P: the optimal solution
  - one of the largest set of non-overlapping intervals.
- we are going to show: $|S| = |P|$.

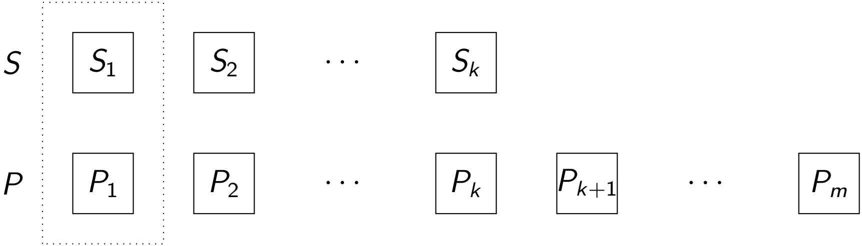
The Fourth Try: $|S| = |P|$, i.e. $k = m$

S $\boxed{S_1}$ $\boxed{S_2}$ $\cdots$ $\boxed{S_k}$

P $\boxed{P_1}$ $\boxed{P_2}$ $\cdots$ $\boxed{P_k}$ $\boxed{P_{k+1}}$ $\cdots$ $\boxed{P_m}$

Intervals in S and P are sorted according to the ending time.

The Fourth Try: $|S| = |P|$, i.e. $k = m$



S_1 ends before or at the same time with P_1

The Fourth Try: $|S| = |P|$, i.e. $k = m$

$$\begin{array}{cccccccc} S & \boxed{S_1} & \boxed{S_2} & \cdots & \boxed{S_k} & & & \\ P & \boxed{S_1} & \boxed{P_2} & \cdots & \boxed{P_k} & \boxed{P_{k+1}} & \cdots & \boxed{P_m} \end{array}$$

S_1 ends before or at the same time with P_1

The Fourth Try: $|S| = |P|$, i.e. $k = m$

S	S_1	S_2	$\cdots$	S_k			
P	S_1	P_2	$\cdots$	P_k	P_{k+1}	$\cdots$	P_m

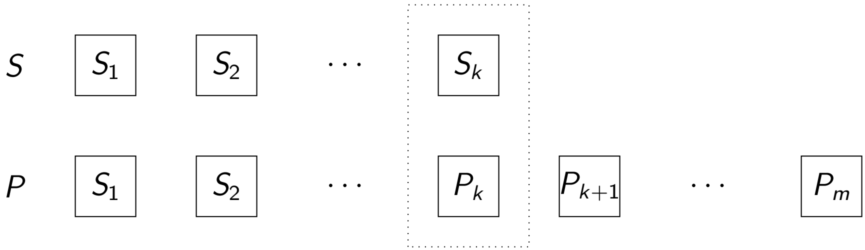
- P_2 starts after S_1 ends
- S_2 ends before or at the same time with P_2

The Fourth Try: $|S| = |P|$, i.e. $k = m$

S	S_1	S_2	$\cdots$	S_k			
P	S_1	S_2	$\cdots$	P_k	P_{k+1}	$\cdots$	P_m

- P_2 starts after S_1 ends
- S_2 ends before or at the same time with P_2

The Fourth Try: $|S| = |P|$, i.e. $k = m$



- P_k starts after S_{k-1} ends
- S_k ends before or at the same time with P_k

The Fourth Try: $|S| = |P|$, i.e. $k = m$

S	$\boxed{S_1}$	$\boxed{S_2}$	$\cdots$	$\boxed{S_k}$			
P	$\boxed{S_1}$	$\boxed{S_2}$	$\cdots$	$\boxed{S_k}$	$\boxed{P_{k+1}}$	$\cdots$	$\boxed{P_m}$

- P_k starts after S_{k-1} ends
- S_k ends before or at the same time with P_k

The Fourth Try: $|S| = |P|$, i.e. $k = m$

S	$\boxed{S_1}$	$\boxed{S_2}$	$\cdots$	$\boxed{S_k}$			
P	$\boxed{S_1}$	$\boxed{S_2}$	$\cdots$	$\boxed{S_k}$	$\boxed{P_{k+1}}$	$\cdots$	$\boxed{P_m}$

- P_{k+1} starts after S_k ends
- $k = m$; otherwise, S can be extended.

The Fourth Try: $|S| = |P|$, i.e. $k = m$

$$\begin{array}{ccccc} S & \boxed{S_1} & \boxed{S_2} & \cdots & \boxed{S_k} \\ & & & & \\ P & \boxed{S_1} & \boxed{S_2} & \cdots & \boxed{S_k} \end{array}$$

- P_{k+1} starts after S_k ends
- $k = m$; otherwise, S can be extended.

The Fourth Try: Optimal

Lemma: For $1 \leq i \leq k+1$, P_i starts after the ending time of S_{i-1}.

Induction on i

- **Base case**: P_2 starts after the ending time of S_1.
  - P_2 starts after the ending time of P_1
  - P_1 ends after the ending time of S_1
- **Assumption**: for $i \geq 1$, P_i starts after the ending time of S_{i-1}.
- P_{i+1} starts after the ending time of S_i.

The Fourth Try: Optimal

Lemma: For $1 \le i \le k+1$, P_i starts after the ending time of S_{i-1}.

Induction on i

- **Base case**: P_2 starts after the ending time of S_1.
  - P_2 starts after the ending time of P_1
  - P_1 ends after the ending time of S_1
- **Assumption**: for $i \ge 1$, P_i starts after the ending time of S_{i-1}.
- P_{i+1} starts after the ending time of S_i.
  - P_{i+1} starts after the ending time of P_i.

The Fourth Try: Optimal

Lemma: For $1 \leq i \leq k+1$, P_i starts after the ending time of S_{i-1}.

Induction on i

- **Base case**: P_2 starts after the ending time of S_1.
  - P_2 starts after the ending time of P_1
  - P_1 ends after the ending time of S_1
- **Assumption**: for $i \geq 1$, P_i starts after the ending time of S_{i-1}.
- P_{i+1} starts after the ending time of S_i.
  - P_{i+1} starts after the ending time of P_i.
  - P_i starts after the ending time of S_{i-1}.

The Fourth Try: Optimal

Lemma: For $1 \leq i \leq k+1$, P_i starts after the ending time of S_{i-1}.

Induction on i

- **Base case**: P_2 starts after the ending time of S_1.
  - P_2 starts after the ending time of P_1
  - P_1 ends after the ending time of S_1
- **Assumption**: for $i \geq 1$, P_i starts after the ending time of S_{i-1}.
- P_{i+1} starts after the ending time of S_i.
  - P_{i+1} starts after the ending time of P_i.
  - P_i starts after the ending time of S_{i-1}.
  - P_i must ends after or at the same time with S_i; otherwise, P_i is added to S, following S_{i-1}.

The Fourth Try: Optimal

Lemma: For $1 \leq i \leq k+1$, P_i starts after the ending time of S_{i-1}.

Induction on i

- **Base case**: P_2 starts after the ending time of S_1.
  - P_2 starts after the ending time of P_1
  - P_1 ends after the ending time of S_1
- **Assumption**: for $i \geq 1$, P_i starts after the ending time of S_{i-1}.
- P_{i+1} starts after the ending time of S_i.
  - P_{i+1} starts after the ending time of P_i.
  - P_i starts after the ending time of S_{i-1}.
  - P_i must ends after or at the same time with S_i; otherwise, P_i is added to S, following S_{i-1}.
  - P_{i+1} starts after the ending time of S_i.

The Fourth Try: Optimal

Lemma: For $1 \leq i \leq k+1$, P_i starts after the ending time of S_{i-1}.

Theorem: $k = m$.

If $k < m$, then P_{k+1} starts after the ending time of S_k. Thus P_{k+1} can be added to S. **Contradiction!**

The Fourth Try: $O(n \log n)$

Algorithm: IntervalSchedule(I)

$S = \emptyset$;
$e = -1$;
Sort intervals in I in the ascending order of the ending time;
for $i = 0$ *to* $|I| - 1$ **do**
    if $I[i]$ *starts after* e **then**
        add $I[i]$ to S;
        $e =$ ending time of $I[i]$;
    end
end
Return S;

Complexity: $O(n \log n) + O(n) = O(n \log n)$

Matroid Optimization

Matroid

Matroid: $M(S, \mathcal{I})$, where

- S is a finite and nonempty set (of elements)
- $\mathcal{I} \subseteq 2^S$ is **hereditary**:

$$B \in \mathcal{I}, \; and \; A \subseteq B \Rightarrow A \in \mathcal{I}$$

- M satisfies the **exchange property**:

$$A, B \in \mathcal{I}, \; and \; |A| < |B| \Rightarrow \exists x \in B \setminus A, A \cup \{x\} \in \mathcal{I}$$

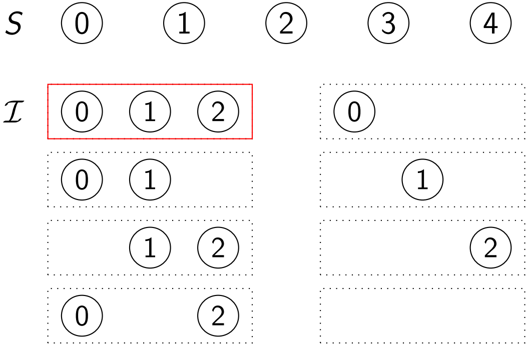
Matroid

S 0 1 2 3 4

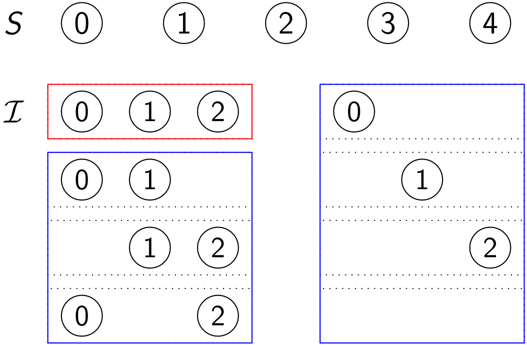
$\mathcal{I}$

0 1 2	0
0 1	1
1 2	2
0 2	

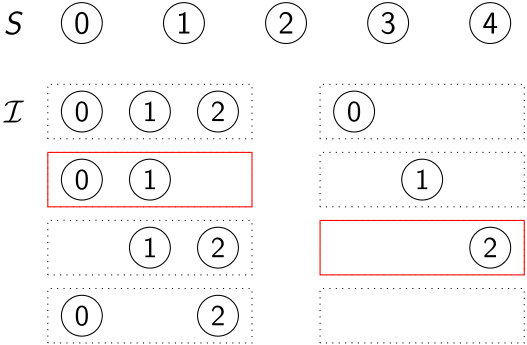
Matroid



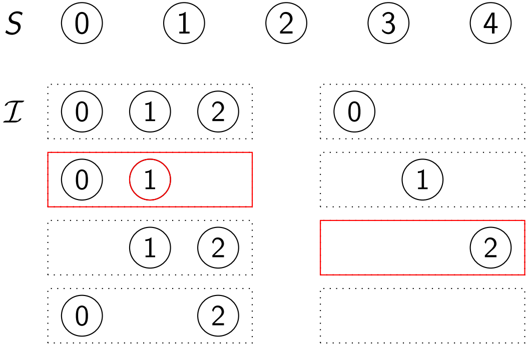
Matroid



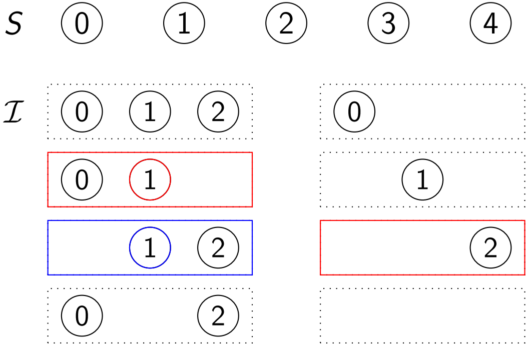
Matroid



Matroid



Matroid



Maximal Independent Subsets

Theorem: For $A, B \in \mathcal{I}$, if A and B are maximal (within $\mathcal{I}$), then $|A| = |B|$.

Maximal Independent Subsets

Theorem: For $A, B \in \mathcal{I}$, if A and B are maximal (within $\mathcal{I}$), then $|A| = |B|$.

A subset $A \subseteq 2^S$ is maximal in $\mathcal{I} \Leftrightarrow$

$$\forall B \in \mathcal{I} \setminus \{A\},\ s.t.\ A \setminus B \neq \emptyset$$

$\mathcal{I} = \{[0,1,2],[0,1],[1,2],[0,2],[0],[1],[2],\emptyset\}$

- $[0]$ is not maximal: $[0] \setminus [0,1] = \emptyset$
- $[1,2]$ is not maximal: $[1,2] \setminus [0,1,2] = \emptyset$
- $[0,1,2]$ is maximal

Maximal Independent Subsets

Theorem: Given $M(S, \mathcal{I})$, for $A, B \in \mathcal{I}$, if A and B are maximal (within $\mathcal{I}$), then $|A| = |B|$.

Proof: Suppose to the contrary that there are

- A is maximal
- B is maximal
- $|A| < |B|$

Maximal Independent Subsets

Theorem: Given $M(S, \mathcal{I})$, for $A, B \in \mathcal{I}$, if A and B are maximal (within $\mathcal{I}$), then $|A| = |B|$.

Proof: Suppose to the contrary that there are

- A is maximal
- B is maximal
- $|A| < |B|$
- M satisfies the exchange property $\Rightarrow$

$$\exists x \in B \setminus A,\ s.t.\ A \cup \{x\} \in \mathcal{I}$$

Maximal Independent Subsets

Theorem: Given $M(S, \mathcal{I})$, for $A, B \in \mathcal{I}$, if A and B are maximal (within $\mathcal{I}$), then $|A| = |B|$.

Proof: Suppose to the contrary that there are

- A is maximal
- B is maximal
- $|A| < |B|$
- M satisfies the exchange property $\Rightarrow$

$$\exists x \in B \setminus A, \ s.t. \ A \cup \{x\} \in \mathcal{I}$$

- A is not maximal
- **Controdiction!**

Weighted Matroid

Weighted Matroid: $M(S, \mathcal{I})$ is associated with a weighted function w

- $w(x) > 0, \ \forall x \in S$
- $w(A) = \sum_{x \in A} w(x), \ \forall A \subseteq S$

Maximum-Weight Subset

Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.

Maximum-Weight Subset

Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.

S	$\boxed{0}$	$\boxed{1}$	$\boxed{2}$
w	2	1	3

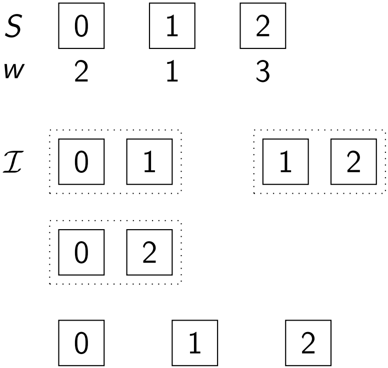
$\mathcal{I}$ $\quad \boxed{0}\ \boxed{1} \qquad \boxed{1}\ \boxed{2}$

$\boxed{0}\ \boxed{2}$

$\boxed{0} \qquad \boxed{1} \qquad \boxed{2}$

Maximum-Weight Subset

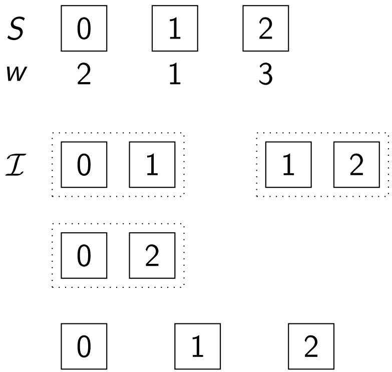
Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.



- $w(0) = 2$
- $w(1) = 1$
- $w(2) = 3$

Maximum-Weight Subset

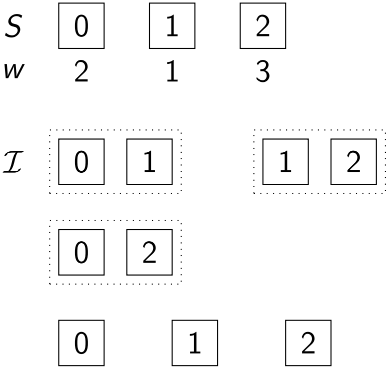
Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.



- $w(0) = 2$
- $w(1) = 1$
- $w(2) = 3$
- $w([0, 1]) = 3$
- $w([1, 2]) = 4$
- $w([0, 2]) = 5$

Maximum-Weight Subset

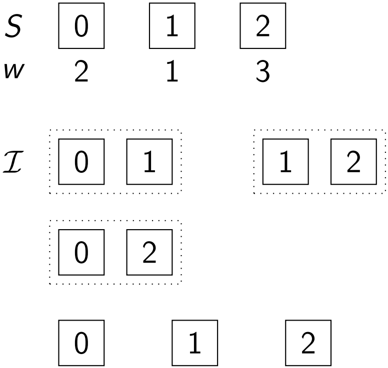
Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.



- $w(0) = 2$
- $w(1) = 1$
- $w(2) = 3$
- $w([0, 1]) = 3$
- $w([1, 2]) = 4$
- $w([0, 2]) = 5$
- $A = [0, 2]$

Maximum-Weight Subset

Problem: Given a weighted matroid $M(S, \mathcal{I})$, with weight function w, find the subset $A \in \mathcal{I}$ such that $w(A)$ is maximized.



- $w(0) = 2$
- $w(1) = 1$
- $w(2) = 3$
- $w([0, 1]) = 3$
- $w([1, 2]) = 4$
- $w([0, 2]) = 5$
- $A = [0, 2]$
- A is maximal.

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
    if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

Maximum-Weight Subset: Greedy

Lemma: Given $M(S, \mathcal{I})$ and w, sort S in the decreasing order of $w[s]$. Let x be the first element in S such that $\{x\} \in \mathcal{I}$, then $\exists A \in \mathcal{I}$, s.t.

- $x \in A$, and
- $w(A)$ is maximized

Maximum-Weight Subset: Greedy

Lemma: Given $M(S, \mathcal{I})$ and w, sort S in the decreasing order of $w[s]$. Let x be the first element in S such that $\{x\} \in \mathcal{I}$, then $\exists A \in \mathcal{I}$, s.t.

- $x \in A$, and
- $w(A)$ is maximized

Assume $x = S[i]$. Thus

- $\{S[i]\} \in \mathcal{I}$
- $\{S[j]\} \notin \mathcal{I}, \forall j < i$

Maximum-Weight Subset: Greedy

Lemma: Given $M(S, \mathcal{I})$ and w, sort S in the decreasing order of $w[s]$. Let x be the first element in S such that $\{x\} \in \mathcal{I}$, then $\exists A \in \mathcal{I}$, s.t.

- $x \in A$, and
- $w(A)$ is maximized

Proof: If $\{S[i]\} \notin \mathcal{I},\ \forall i$, then $\mathcal{I} = \{\emptyset\}$. Thus $A = \emptyset$.

When $\exists i, \{x\} = \{S[i]\} \in \mathcal{I}$ and $\{S[j]\} \notin \mathcal{I}, \forall j < i$, assume there is nonempty $B \in \mathcal{I}$, such that

- $w(B)$ is maximized
- $x \notin B$

Maximum-Weight Subset: Greedy

Then $w(y) \leq w(x), \forall y \in B$. Otherwise

- $y = S[j], j < i$
- $\{y\} = \{S[j]\} \in \mathcal{I}$
- **Contradiction**.

Maximum-Weight Subset: Greedy

Then $w(y) \leq w(x), \forall y \in B$. Otherwise

- $y = S[j], j < i$
- $\{y\} = \{S[j]\} \in \mathcal{I}$
- **Contradiction**.

Now, we try to construct A, such that $x \in A$ and $w(A) = w(B)$. As M satisfies the exchange property, we can do

- $A = \{x\}$
- **while** $|A| < |B|$
  - let $y \in B \setminus A$
  - $A = A \cup \{y\}$
- let z be the last element in $B \setminus A$

Maximum-Weight Subset: Greedy

Then $w(y) \le w(x), \forall y \in B$. Otherwise

- $y = S[j], j < i$
- $\{y\} = \{S[j]\} \in \mathcal{I}$
- **Contradiction**.

Now, we try to construct A, such that $x \in A$ and $w(A) = w(B)$. As M satisfies the exchange property, we can do

- $A = \{x\}$
- **while** $|A| < |B|$
  - let $y \in B \setminus A$
  - $A = A \cup \{y\}$
- let z be the last element in $B \setminus A$
- $w(A) = w(B) - w(z) + w(x) \ge w(B)$

Maximum-Weight Subset: Greedy

Lemma: Given $M(S, \mathcal{I})$, if $x \in S$, $A \in \mathcal{I}$ and $A \cup \{x\} \in \mathcal{I}$, then $\{x\} \in \mathcal{I}$.

Corollary: If $\{x\} \notin \mathcal{I}$, then

$$x \notin A, \forall A \in \mathcal{I}$$

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
    if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

$S[0]$	$S[1]$	$S[2]$	$\cdots$	$S[k]$	$\cdots$

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
    if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

$\boxed{S[0]}$ $\boxed{S[1]}$ $\boxed{S[2]}$ $\cdots$ $\boxed{S[k]}$ $\cdots$

$\{S[0]\} \notin \mathcal{I}$

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
  if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

$S[0]$	$S[1]$	$S[2]$	$\cdots$	$S[k]$	$\cdots$

$\{S[1]\} \notin \mathcal{I}$

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
  if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

$\boxed{S[0]}$ $\boxed{S[1]}$ $\boxed{S[2]}$ $\cdots$ $\boxed{S[k]}$ $\cdots$

$\{S[2]\} \in \mathcal{I}$

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
  if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
Return A;

$S[0]$ $S[1]$ $S[2]$ $\cdots$ $S[k]$ $\cdots$

$\{S[2]\} \in \mathcal{I}$

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$
- $\mathcal{I}' = \{B \subseteq S \setminus \{x\} : B \cup \{x\} \in \mathcal{I}\}$

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$
- $\mathcal{I}' = \{B \subseteq S \setminus \{x\} : B \cup \{x\} \in \mathcal{I}\}$
- $A' = \text{Greedy}(M'(S', \mathcal{I}'), w)$

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$
- $\mathcal{I}' = \{B \subseteq S \setminus \{x\} : B \cup \{x\} \in \mathcal{I}\}$
- $A' = \text{Greedy}(M'(S', \mathcal{I}'), w)$
- $A = \{x\} \cup A'$

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$
- $\mathcal{I}' = \{B \subseteq S \setminus \{x\} : B \cup \{x\} \in \mathcal{I}\}$
- $A' = \text{Greedy}(M'(S', \mathcal{I}'), w)$
- $A = \{x\} \cup A'$
- A' is optimal to M' $\Rightarrow$ A is optimal to M

Maximum-Weight Subset: Greedy

x is the first element, such that $\{x\} = \{S[i]\} \in \mathcal{I}$.

- $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$
- $\mathcal{I}' = \{B \subseteq S \setminus \{x\} : B \cup \{x\} \in \mathcal{I}\}$
- $A' = \text{Greedy}(M'(S', \mathcal{I}'), w)$
- $A = \{x\} \cup A'$
- A' is optimal to M' $\Rightarrow$ A is optimal to M
- Thus, we can reduce M to smaller M', step by step, until $\mathcal{I}' = \emptyset$, which implies $A' = \emptyset$.

Maximum-Weight Subset: Greedy

To prove: A' is optimal to $M' \Rightarrow A$ is optimal to M

Maximum-Weight Subset: Greedy

To prove: A' is optimal to $M' \Rightarrow A$ is optimal to M

- If A is not optimal, then there is $B \in \mathcal{I}$, such that
  - $w(B) > w(A)$
  - $x \in B$
  - otherwise construct B^*, such that $w(B^*) = w(B)$ and $x \in B^*$

Maximum-Weight Subset: Greedy

To prove: A' is optimal to M' $\Rightarrow$ A is optimal to M

- If A is not optimal, then there is $B \in \mathcal{I}$, such that
  - $w(B) > w(A)$
  - $x \in B$
  - otherwise construct B^*, such that $w(B^*) = w(B)$ and $x \in B^*$
- $S[j] \notin B, \forall j < i$
  - $\{x\} = \{S[i]\} \in \mathcal{I}$.
  - $\{S[j]\} \notin \mathcal{I} \Rightarrow x \notin C, \forall C \in \mathcal{I}$

Maximum-Weight Subset: Greedy

To prove: A' is optimal to M' $\Rightarrow$ A is optimal to M

- If A is not optimal, then there is $B \in \mathcal{I}$, such that
  - $w(B) > w(A)$
  - $x \in B$
  - otherwise construct B^*, such that $w(B^*) = w(B)$ and $x \in B^*$
- $S[j] \notin B, \forall j < i$
  - $\{x\} = \{S[i]\} \in \mathcal{I}$.
  - $\{S[j]\} \notin \mathcal{I} \Rightarrow x \notin C, \forall C \in \mathcal{I}$
- Let $B' = B \setminus \{x\}$. Then
  - $B' \in \mathcal{I}'$
  - $w(B') = w(B) - w(x) > w(A) - w(x) = w(A')$
  - **Contradiction**.

Maximum-Weight Subset: Greedy

Algorithm: Greedy(M, w)

$A = \emptyset$;
Sort S in the decreasing order of $w[s]$;
for $i = 0$ *to* $|S| - 1$ **do**
    if $A \cup \{S[i]\} \in \mathcal{I}$ **then** add $S[i]$ to A;
end
return A;

Theorem: Given matroid $M(S, \mathcal{I})$ and weight function w, then Greedy(M, w) returns an optimal subset $A \in \mathcal{I}$.

The Knapsack Problem: Greedy

Given

- a container $\mathcal{K}$ of capacity $W \in \mathbb{Z}^+$
- n items $\{x_0, x_1, \ldots, x_{n-1}\}$
  - integral weight $w_i \in \mathbb{Z}^+$
  - value $v_i > 0$

Fill the knapsack so as to maximize the total value.

The Knapsack Problem: Greedy

Given

- a container $\mathcal{K}$ of capacity $W \in \mathbb{Z}^+$
- n items $\{x_0, x_1, \ldots, x_{n-1}\}$
  - integral weight $w_i \in \mathbb{Z}^+$
  - value $v_i > 0$

Fill the knapsack so as to maximize the total value.

- Divide item x_i into w_i pieces. Each piece is of weight 1 and value v_i/w_i.

The Knapsack Problem: Greedy

Given

- a container $\mathcal{K}$ of capacity $W \in \mathbb{Z}^+$
- n items $\{x_0, x_1, \ldots, x_{n-1}\}$
  - integral weight $w_i \in \mathbb{Z}^+$
  - value $v_i > 0$

Fill the knapsack so as to maximize the total value.

- Divide item x_i into w_i pieces. Each piece is of weight 1 and value v_i/w_i.
- Define
  - S: the set of all the pieces
  - $\mathcal{I}$: all subsets of size $\leq W$
  - for a piece x from item x_i, $w(x) = v_i/w_i$

The Knapsack Problem: Greedy

- S: the set of all the pieces
- $\mathcal{I}$: all subsets of size $\leq W$
- for a piece x from item x_i, $w(x) = v_i / w_i$

Matroid $M(S, \mathcal{I})$ satisfies

- **hereditary**: For $A \in \mathcal{I}$,

$$A' \subseteq A \Rightarrow |A'| \leq |A| \leq W$$

- **exchange property**: For $A, B \in \mathcal{I}$, and $|A| < |B| \leq W$. Let x be any element in $B \setminus A$. Then

$$|A \cup \{x\}| = |A| + 1 \leq W$$

The Knapsack Problem: Greedy

- S: the set of all the pieces
- $\mathcal{I}$: all subsets of size $\leq W$
- for a piece x from item x_i, $w(x) = v_i/w_i$

Matroid $M(S, \mathcal{I})$ satisfies

- **hereditary**: For $A \in \mathcal{I}$,

$$A' \subseteq A \Rightarrow |A'| \leq |A| \leq W$$

- **exchange property**: For $A, B \in \mathcal{I}$, and $|A| < |B| \leq W$. Let x be any element in $B \setminus A$. Then

$$|A \cup \{x\}| = |A| + 1 \leq W$$

Thus, with the optimal subset for matroid $M(S, \mathcal{I})$ we know how to fill the knapsack to maximize the total value.

Set Cover

Set and Cover

Problem: Given

- B: a set of elements
- k sets
  $S_0, S_1, \ldots, S_{k-1} \in 2^B$

find a cover

- a selection
  $S_{i_0}, S_{i_1}, \ldots, S_{i_{h-1}}$
- $\cup_{i_j} S_{i_j} = B$

such that h is minimized.

Set and Cover

Problem: Given

- B: a set of elements
- k sets $S_0, S_1, \ldots, S_{k-1} \in 2^B$

find a cover

- a selection $S_{i_0}, S_{i_1}, \ldots, S_{i_{h-1}}$
- $\cup_{i_j} S_{i_j} = B$

such that h is minimized.

- $B = [0, 1, 2, 3, 4]$

Set and Cover

Problem: Given

- B: a set of elements
- k sets $S_0, S_1, \ldots, S_{k-1} \in 2^B$

find a cover

- a selection $S_{i_0}, S_{i_1}, \ldots, S_{i_{h-1}}$
- $\cup_{i_j} S_{i_j} = B$

such that h is minimized.

- $B = [0, 1, 2, 3, 4]$
- 4 sets
  - $[0, 1]$
  - $[2, 3]$
  - $[3, 4]$
  - $[0, 1, 4]$

Set and Cover

Problem: Given

- B: a set of elements
- k sets $S_0, S_1, \ldots, S_{k-1} \in 2^B$

find a cover

- a selection $S_{i_0}, S_{i_1}, \ldots, S_{i_{h-1}}$
- $\cup_{i_j} S_{i_j} = B$

such that h is minimized.

- $B = [0, 1, 2, 3, 4]$
- 4 sets
  - $[0, 1]$
  - $[2, 3]$
  - $[3, 4]$
  - $[0, 1, 4]$
- cover
  - $[0, 1] \cup [2, 3] \cup [3, 4] = B$
  - $[0, 1, 4] \cup [2, 3] = B$

Set and Cover: Greedy

Selection: choose the set (in the remaining ones) that covers the largest number of uncovered elements.

Set and Cover: Greedy

Selection: choose the set (in the remaining ones) that covers the largest number of uncovered elements.

- Let S be the result of the greedy selection.

Set and Cover: Greedy

Selection: choose the set (in the remaining ones) that covers the largest number of uncovered elements.

- Let S be the result of the greedy selection.
- Let P be the optimal solution.

Set and Cover: Greedy

Selection: choose the set (in the remaining ones) that covers the largest number of uncovered elements.

- Let S be the result of the greedy selection.
- Let P be the optimal solution.
- $|S| \leq |P| \ln |B|$

Set and Cover: Greedy

$|S| \leq |P| \ln |B|$

Proof: Let n_t be the number of uncovered elements after step t.

- $n_0 = n$
- at step 1, select a set S_1
- $n_1 = n - |S_1|$

The remaining n_t elements are covered by optimal selection P. Thus, there exists a not-selected set S' with $|S'| \geq n_t/|P|$.

- $|S_{t+1}| \geq n_t/|P|$
- $n_{t+1} \leq n_t - n_t/|P| = n_t(1 - 1/|P|)$
- $n_t \leq n_0(1 - 1/|P|)^t$

Set and Cover: Greedy

With $n_t \leq n_0(1 - 1/|P|)^t$

- recall that $1 - x < e^{-x}$ for all $x \neq 0$
- $n_t \leq n_0(1 - 1/|P|)^t < n \cdot e^{-t/|P|}$
- let $t^* = |P| \ln |B|$
- $n_{t^*} < n \cdot e^{-\ln |B|} = 1$
- $|S| \leq t^* = |P| \ln |B|$

Submodularity-Based Optimization

Matroid and Subset-Weight

Matroid: $M(S, \ell)$, where

- S is a finite and nonempty set (of elements)
- $\mathcal{I} \subseteq 2^S$ is **hereditary**:

$$B \in \ell, \text{ and } A \subseteq B \Rightarrow A \in \mathcal{I}$$

- M satisfies the **exchange property**:

$$A, B \in \mathcal{I}, \text{ and } |A| < |B| \Rightarrow \exists x \in B \setminus A, A \cup \{x\} \in \mathcal{I}$$

Subset-Weight: $w : 2^S \to \mathcal{R}^+$

Monotone and Submodular

w is **monotone** if

$$A \subseteq B \Rightarrow w(A) \leq w(B)$$

w is **submodular** if

$$\forall A \subseteq B \textit{ and } x \in S, w(A \cup \{x\}) - w(A) \geq w(B \cup \{x\}) - w(B)$$

Monotone and Submodular

w is **monotone** if

$$A \subseteq B \Rightarrow w(A) \leq w(B)$$

w is **submodular** if

$$\forall A \subseteq B \text{ and } x \in S, w(A \cup \{x\}) - w(A) \geq w(B \cup \{x\}) - w(B)$$

or, equivalently

$$w(A) + w(B) \geq w(A \cup B) + w(A \cap B)$$

Notice that

$$A \cap B \subseteq B,\ x = A \setminus B$$

Thus

$$A \cap B \cup x = A,\ B \cup x = A \cup B$$

Climb The Hill: Submodularity

Algorithm: HillClimb(M, w)

$A = \emptyset$;

while true do

  let $X = \{x | x \in S \setminus A, A \cup \{x\} \in \mathcal{I}\}$

  if $|X| < 1$ **then break**;

  let $x^* = \arg\max_{x \in X} w(A \cup \{x\}) - w(A)$

  add x to A;

end

return A;

Climb The Hill: Submodularity

Algorithm: HillClimb(M, w)

$A = \emptyset$;

while true do

  let $X = \{x | x \in S \setminus A, A \cup \{x\} \in \mathcal{I}\}$

  if $|X| < 1$ **then break**;

  let $x^* = \arg\max_{x \in X} w(A \cup \{x\}) - w(A)$

  add x to A;

end

return A;

$$w(A) \geq w(P) \cdot (1 - 1/e)$$

THANK YOU

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES