



# Chapter 2

# Greedy Algorithms

Algorithm Theory  
WS 2015/16

Fabian Kuhn

# Greedy Algorithms

- No clear definition, but essentially:

**In each step make the choice that looks best at the moment!**

- Depending on problem, greedy algorithms can give
  - Optimal solutions
  - Close to optimal solutions
  - No (reasonable) solutions at all
- If it works, very interesting approach!
  - And we might even learn something about the structure of the problem

**Goal:** Improve understanding where it works (mostly by examples)

# Traveling Salesperson Problem (TSP)

## Input:

- Set  $V$  of  $n$  nodes (points, cities, locations, sites)
- Distance function  $d: V \times V \rightarrow \mathbb{R}$ , i.e.,  $d(u, v)$ : dist. from  $u$  to  $v$
- Distances usually symmetric, asymm. distances  $\rightarrow$  asymm. TSP

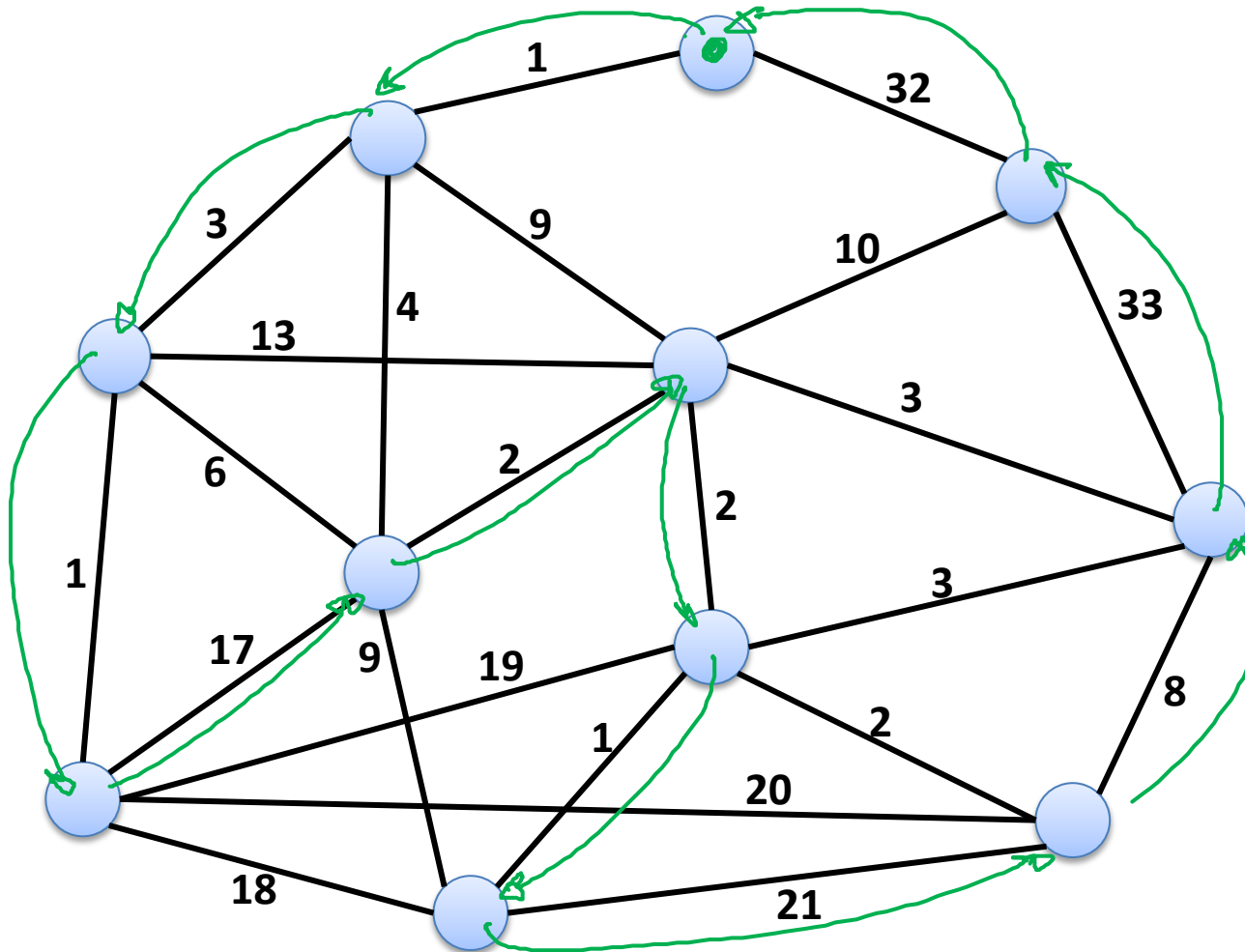
## Solution:

- Ordering/permutation  $v_1, v_2, \dots, v_n$  of nodes
- Length of TSP path:  $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$
- Length of TSP tour:  $d(v_n, v_1)$  +  $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$

## Goal:

- Minimize length of TSP path or TSP tour

# Example



Optimal Tour:

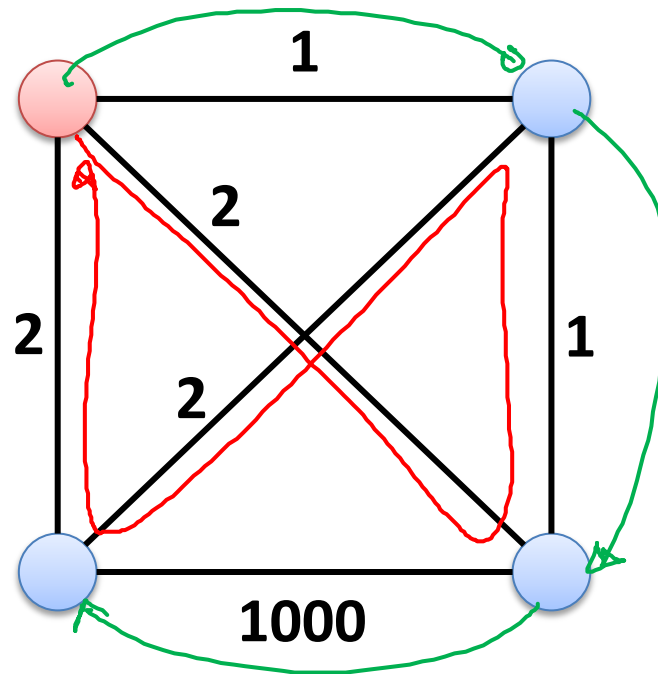
Length: 86

Greedy Algorithm?

Length: 121

# Nearest Neighbor (Greedy)

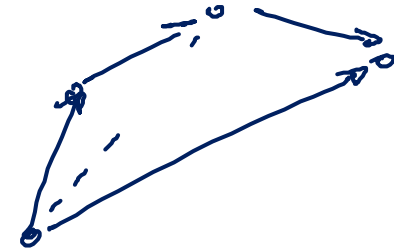
- Nearest neighbor can be arbitrarily bad, even for TSP paths



# TSP Variants

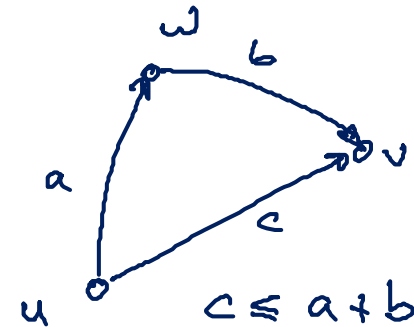
- ~~Symmetric~~ / Asymmetric TSP

- arbitrary non-negative distance/cost function
- most general, nearest neighbor arbitrarily bad
- NP-hard to get within any bound of optimum



- Symmetric TSP

- arbitrary non-negative distance/cost function
- nearest neighbor arbitrarily bad
- NP-hard to get within any bound of optimum



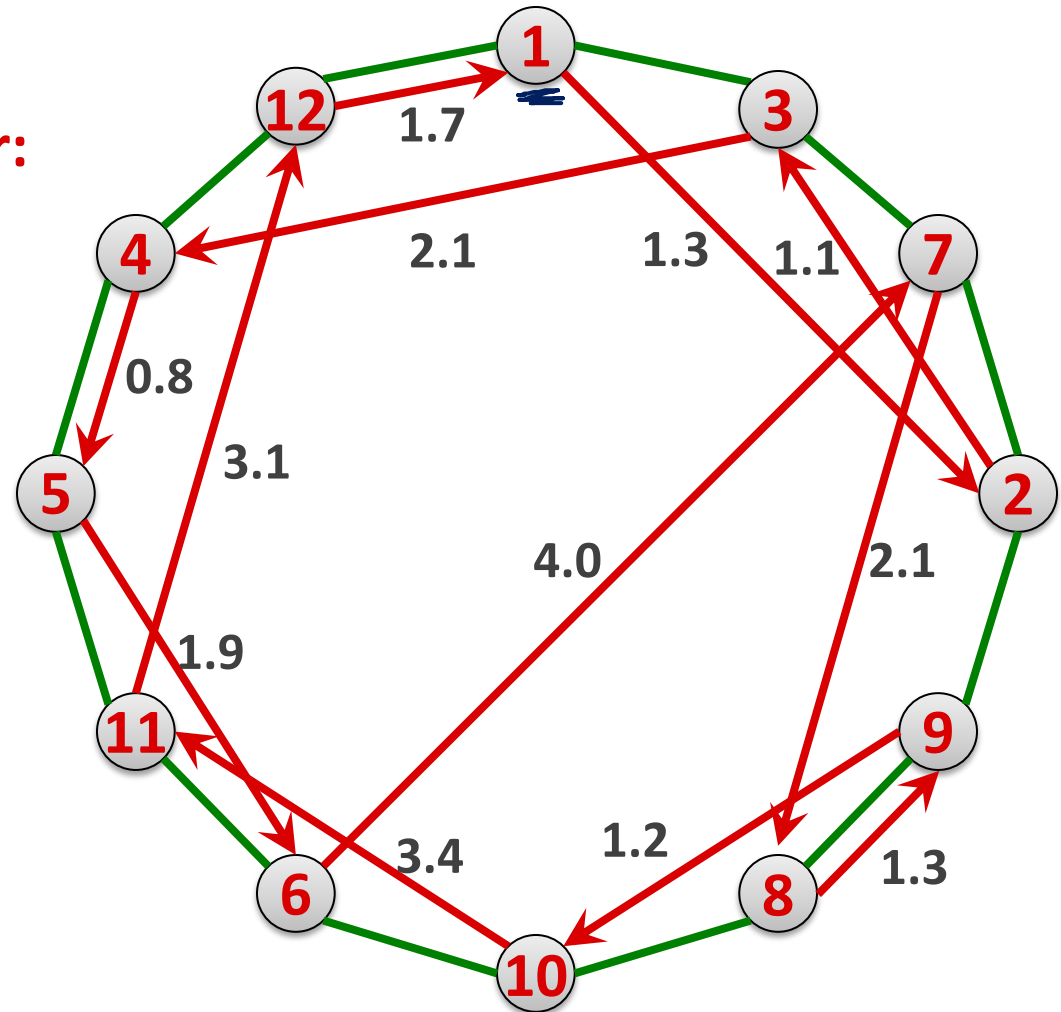
- Metric TSP

- distance function defines metric space: symmetric, non-negative, triangle inequality:  $d(u, v) \leq d(u, w) + d(w, v)$
- possible to get close to optimum (we will later see factor  $3/2$ )
- what about the nearest neighbor algorithm?

# Metric TSP, Nearest Neighbor

Optimal TSP tour:

Nearest-Neighbor TSP tour:



# Metric TSP, Nearest Neighbor

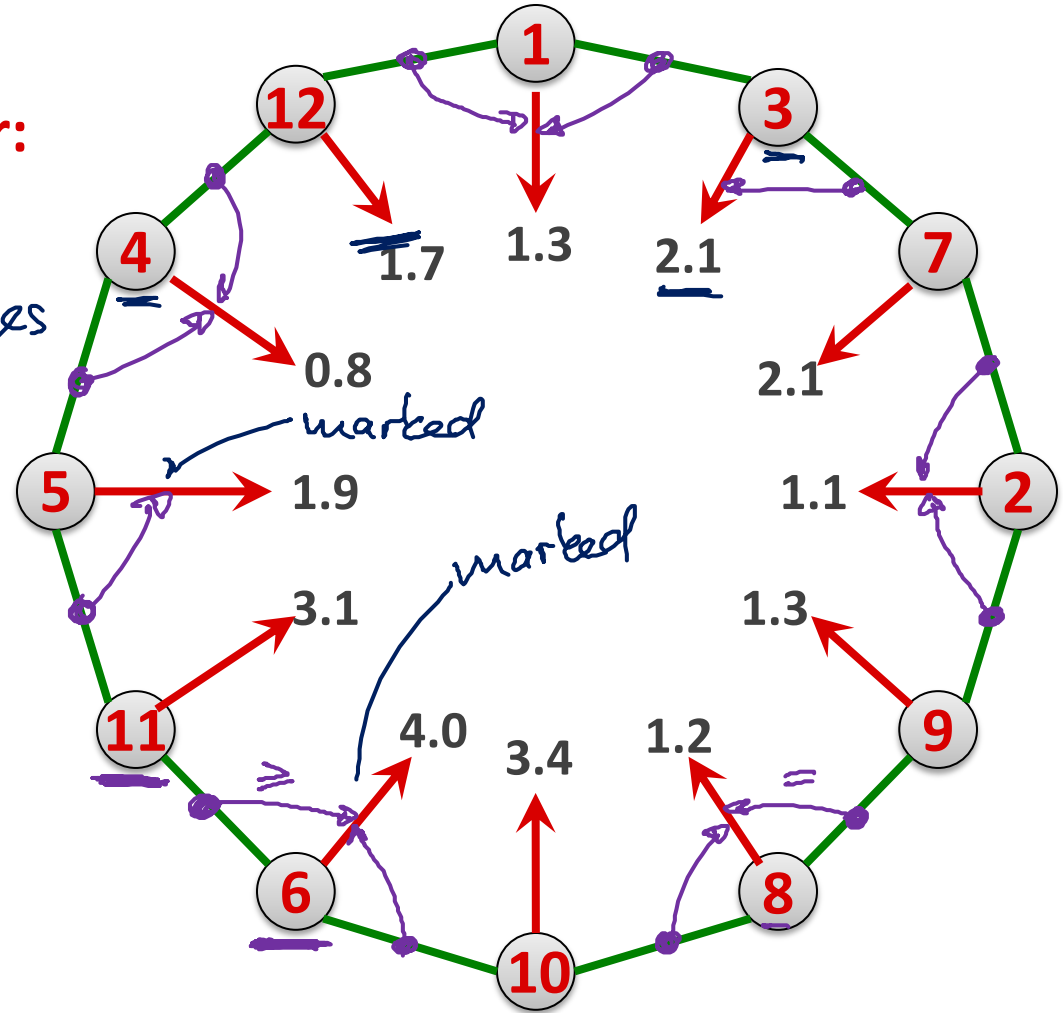
**Optimal TSP tour:**

**Nearest-Neighbor TSP tour:**

cost = 24

*green edges  $\geq$  marked red edges  
opt. tour*

*#marked red edges:  
at least half*





# Metric TSP, Nearest Neighbor

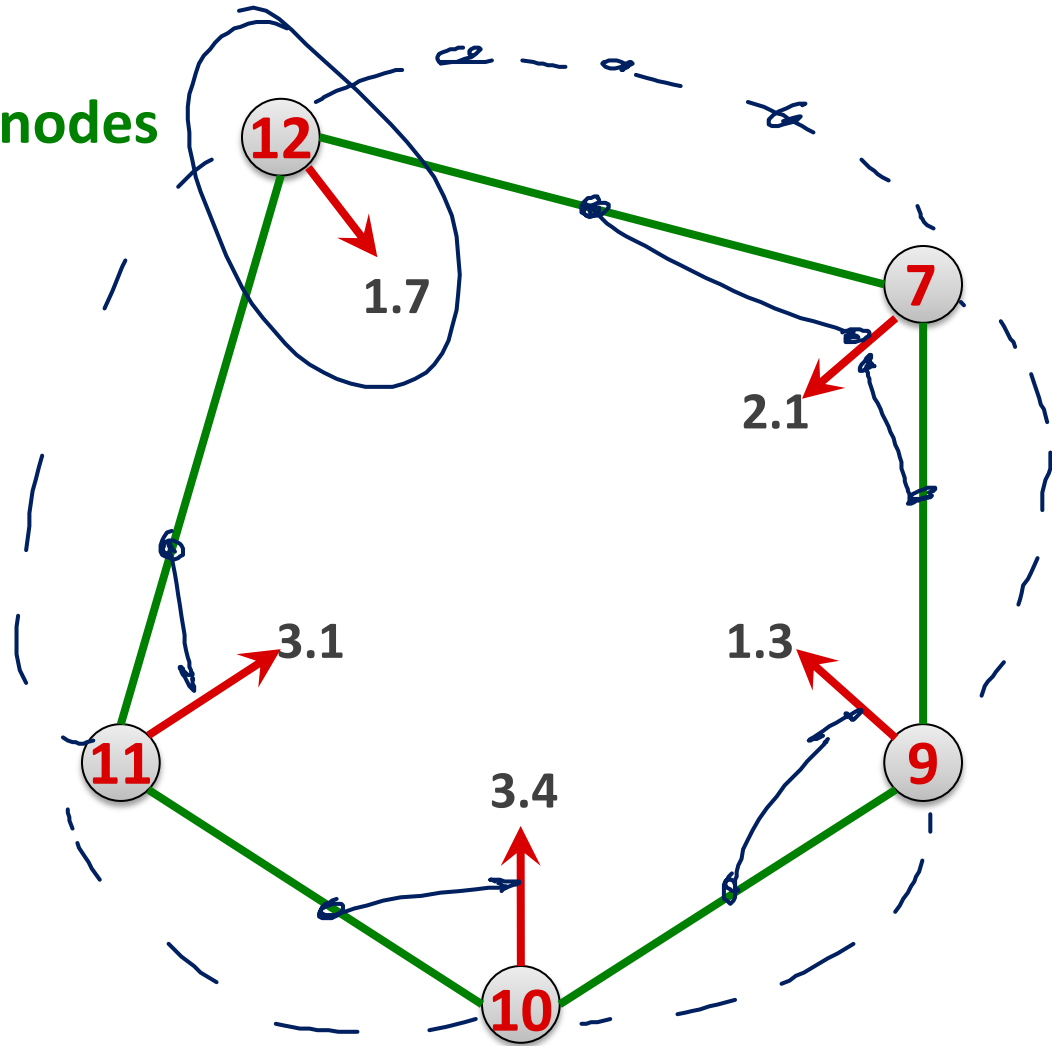
Triangle Inequality:

**optimal tour on remaining nodes**

$\leq$

**overall optimal tour**

*green  $\neq$  marked red*



# Metric TSP, Nearest Neighbor

Analysis works in **phases**:

- In each phase, assign each optimal edge to some greedy edge
  - Cost of greedy edge  $\leq$  cost of optimal edge
- Each greedy edge gets assigned  $\leq 2$  optimal edges
  - At least half of the greedy edges get assigned
- At end of phase:
  - Remove points for which greedy edge is assigned
  - Consider optimal solution for remaining points
- **Triangle inequality:** remaining opt. solution  $\leq$  overall opt. sol.
- Cost of greedy edges assigned in **each phase**  $\leq$  **opt. cost**
- **Number of phases**  $\leq$   **$\log_2 n$** 
  - +1 for last greedy edge in tour

# Metric TSP, Nearest Neighbor

- Assume:

NN: cost of greedy tour,    OPT: cost of optimal tour

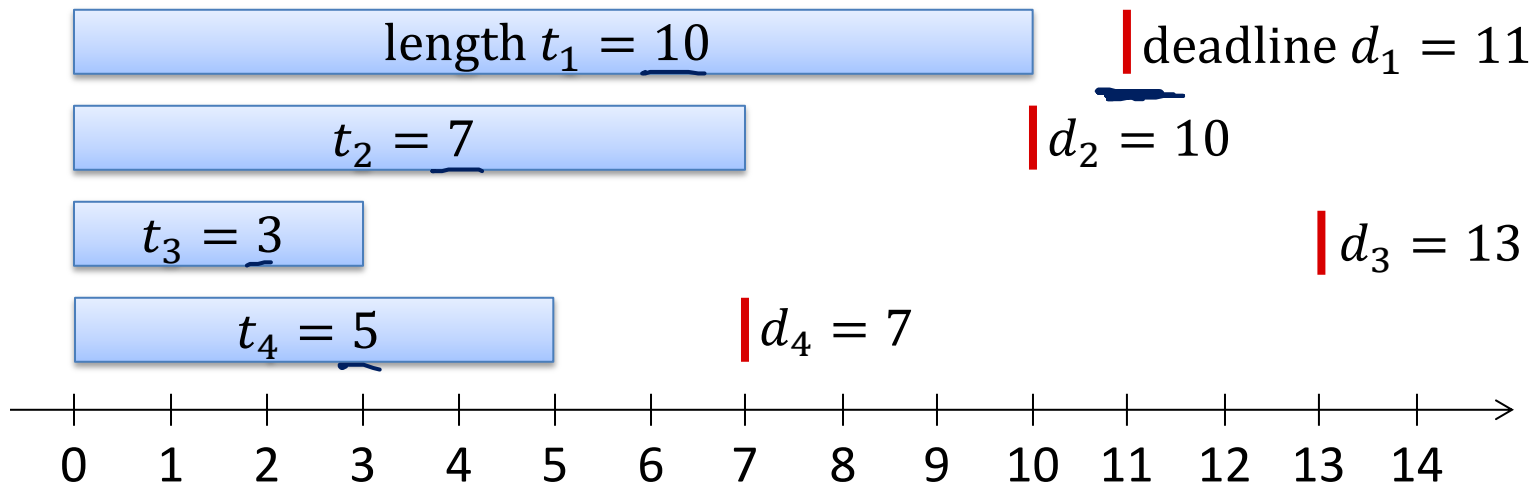
- We have shown:

$$\frac{NN}{OPT} \leq \underbrace{1 + \log_2 n}_{\text{approximation ratio}}$$

- Example of an **approximation algorithm**
- We will later see a  $3/2$ -approximation algorithm for metric TSP

# Back to Scheduling

- Given:  $n$  requests / jobs with deadlines:



- Goal: schedule all jobs with minimum lateness  $L$

– Schedule:  $s(i)$ ,  $f(i)$ : start and finishing times of request  $i$

Note:  $f(i) = s(i) + t_i$   $\max_i \{ \max\{0, f(i) - d_i\} \}$

- Lateness  $L := \max\{0, \max_i \{f(i) - d_i\}\}$

– largest amount of time by which some job finishes late

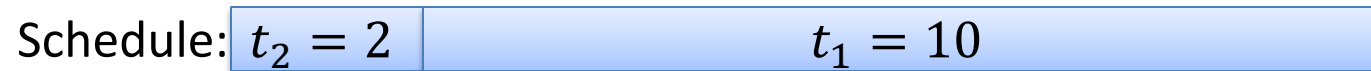
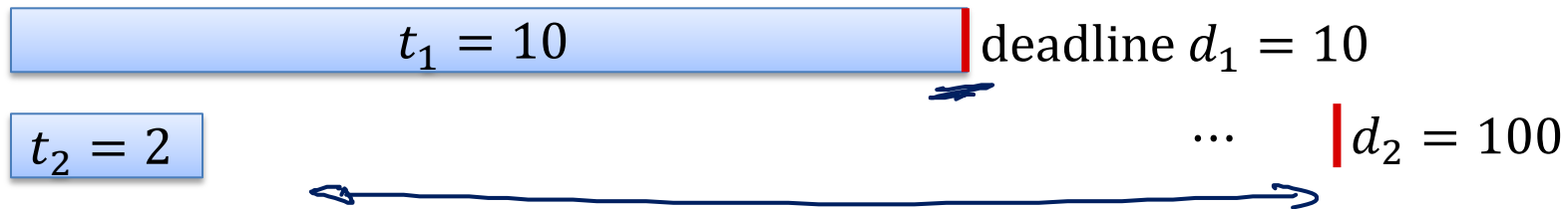
- Many other natural objective functions possible...

# Greedy Algorithm?

## Schedule jobs in order of increasing length?

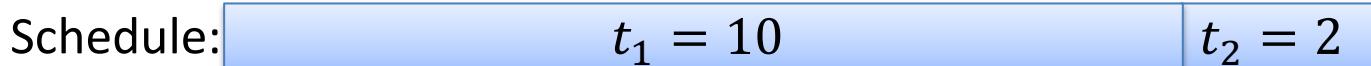
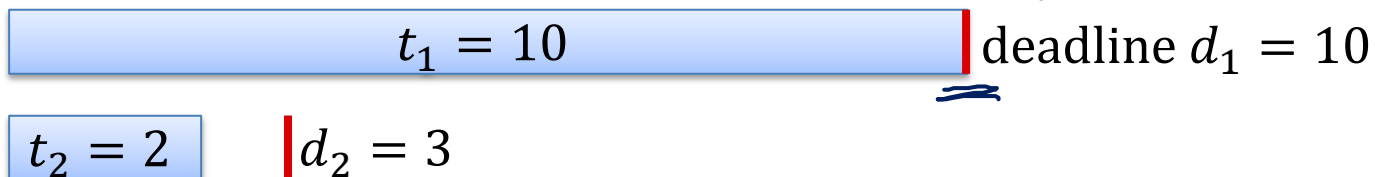
- Ignores deadlines: seems too simplistic...

- E.g.:



## Schedule by increasing slack time?

- Should be concerned about slack time:  $d_i - t_i$



# Greedy Algorithm

## Schedule by earliest deadline?

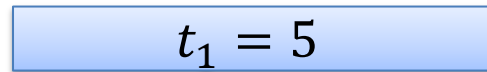
- Schedule in increasing order of  $d_i$
- Ignores lengths of jobs: too simplistic?
- Earliest deadline is optimal!

## Algorithm:

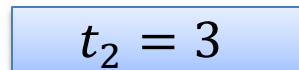
- Assume jobs are reordered such that  $\underline{d_1} \leq \underline{d_2} \leq \dots \leq \underline{d_n}$
  - Start/finishing times:
    - First job starts at time  $\underline{s(1) = 0}$
    - Duration of job  $i$  is  $t_i$ :  $\underline{f(i) = s(i) + t_i}$
    - No gaps between jobs:  $\underline{s(i + 1) = f(i)}$
- (idle time: gaps in a schedule  $\rightarrow$  alg. gives schedule with no idle time)

# Example

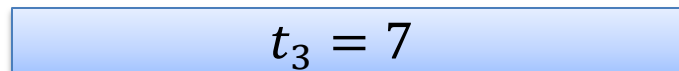
Jobs ordered by deadline:



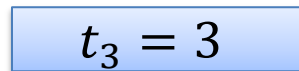
$d_4 = 7$



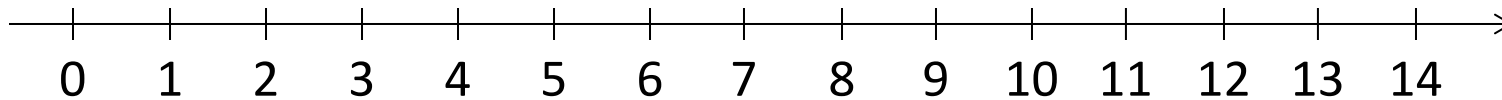
$d_2 = 10$



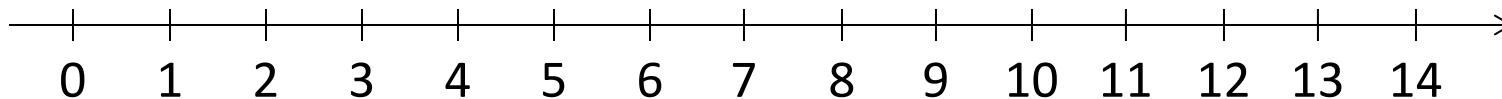
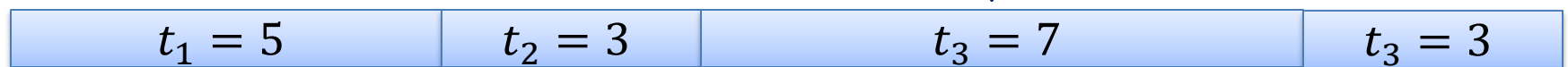
$d_1 = \underline{\underline{11}}$



$d_3 = 13$



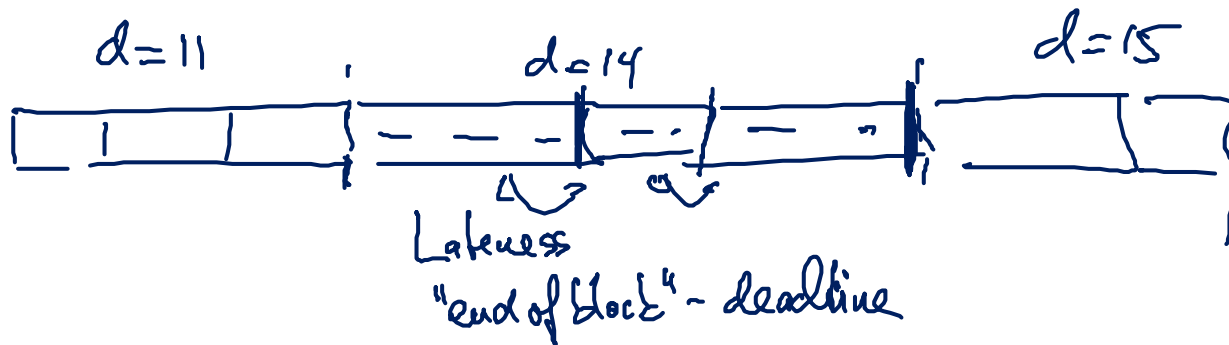
Schedule:



Lateness: job 1: 0, job 2: 0, job 3: 4, job 4: 5

# Basic Facts

1. There is an optimal schedule with no idle time
  - Can just schedule jobs earlier...
  
2. Inversion: Job  $i$  scheduled before job  $j$  if  $d_i > d_j$   
 Schedules with no inversions have the same maximum lateness





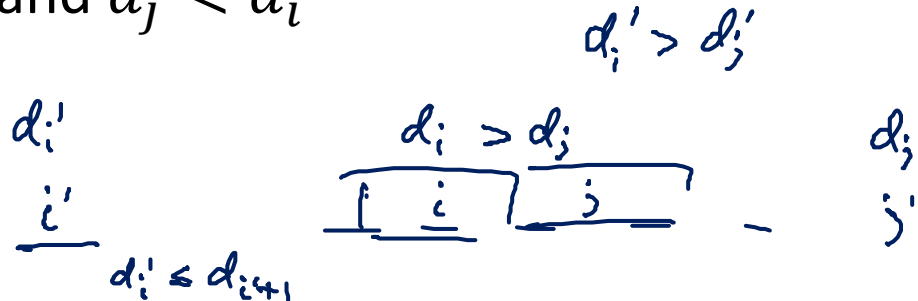
# Earliest Deadline is Optimal

## Theorem:

There is an optimal schedule  $\mathcal{O}$  with no inversions and no idle time.

**Proof:** if  $\mathcal{O}$  has inv.  $i'$  sched. before  $j'$   $d_{i'} > d_{j'}$

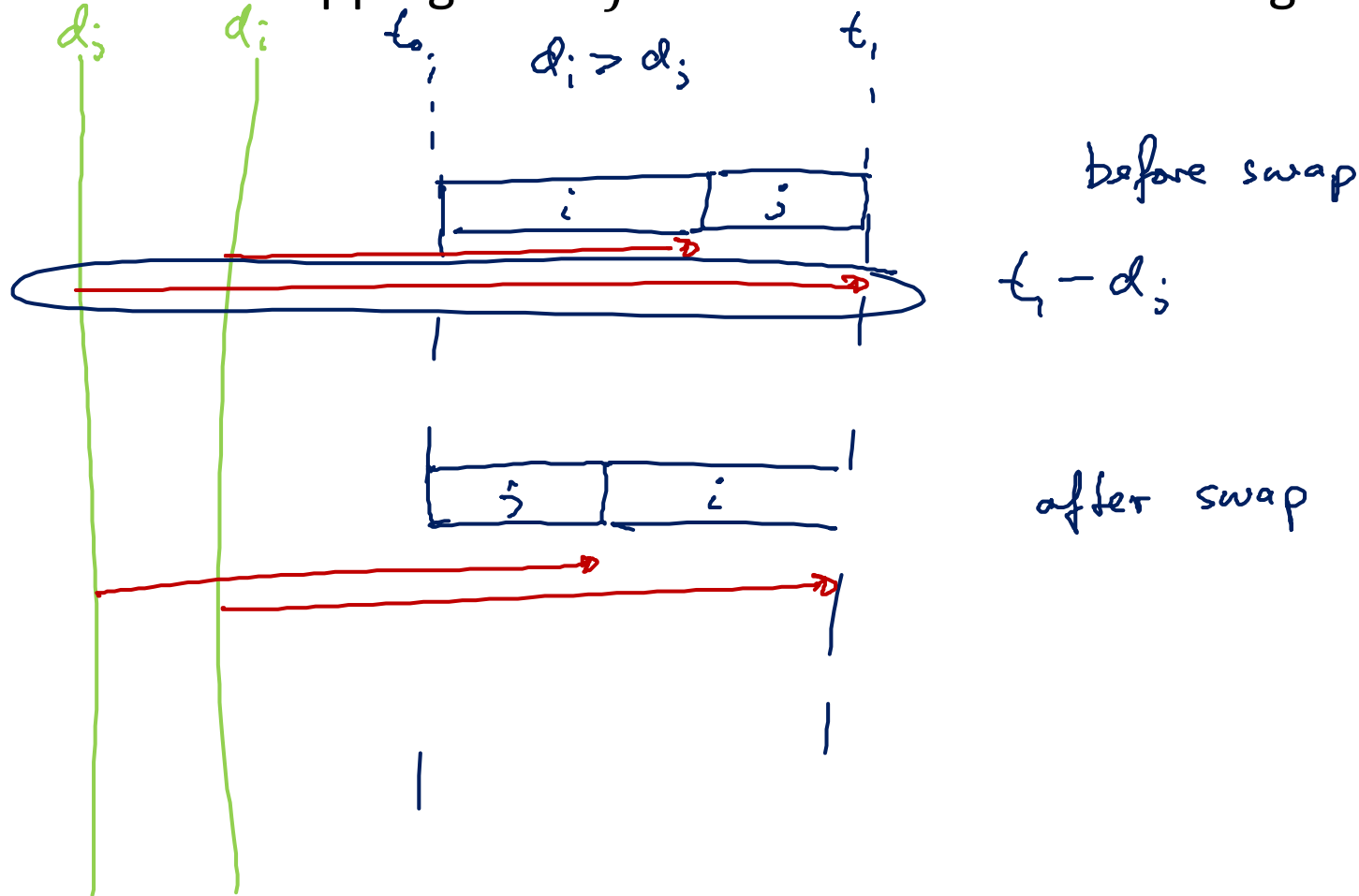
- Consider optimal schedule  $\mathcal{O}'$  with no idle time
- If  $\mathcal{O}'$  has inversions,  $\exists$  pair  $(i, j)$ , s.t.  $i$  is scheduled immediately before  $j$  and  $d_j < d_i$



- Claim: Swapping  $i$  and  $j$  gives schedule with
  1. Less inversions
  2. Maximum lateness no larger than in  $\mathcal{O}'$

# Earliest Deadline is Optimal

**Claim:** Swapping  $i$  and  $j$ : maximum lateness no larger than in  $\mathcal{O}'$



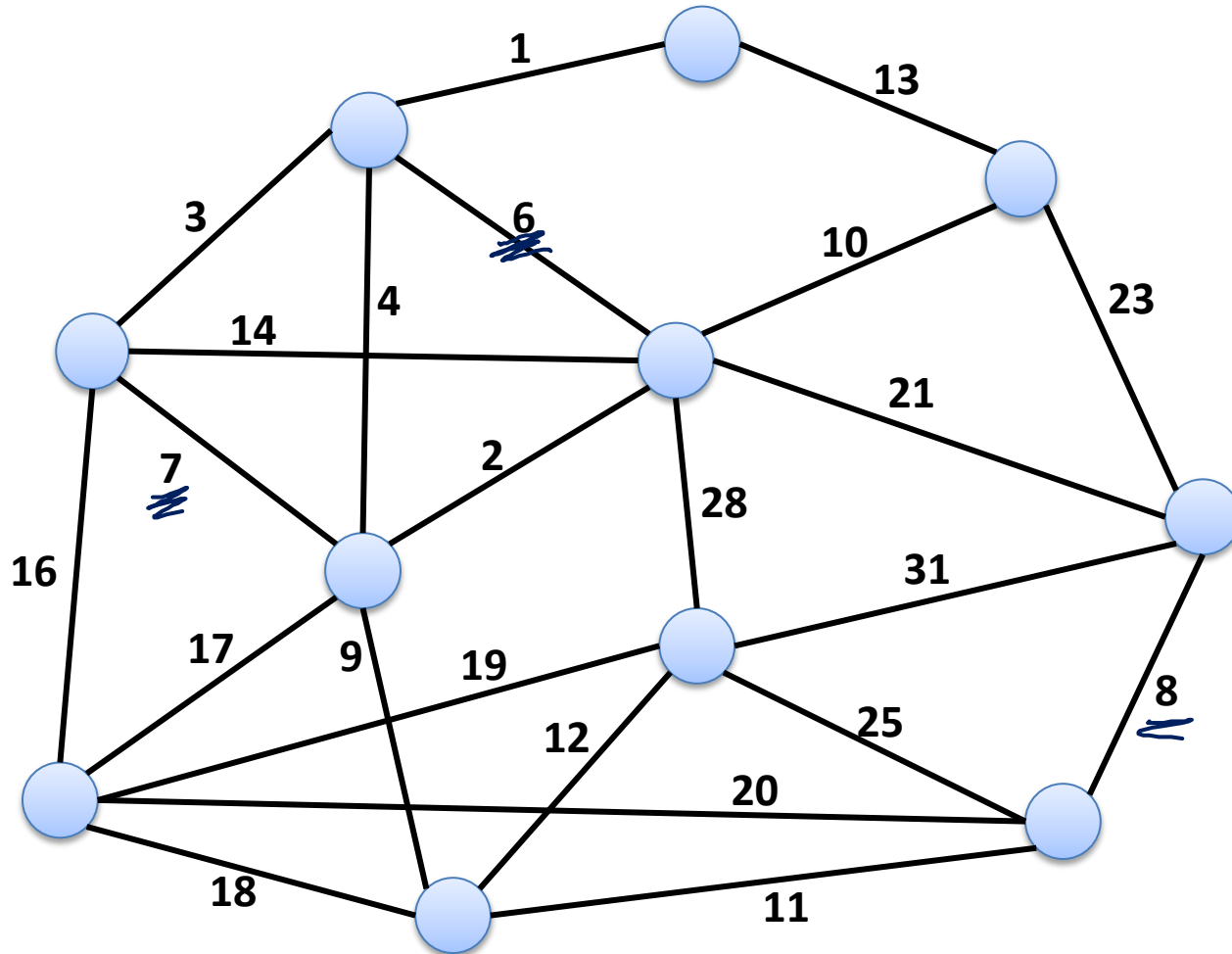
# Exchange Argument

- General approach that often works to analyze greedy algorithms
- Start with any solution
- Define basic exchange step that allows to transform solution into a new solution that is not worse
- Show that exchange step move solution closer to the solution produced by the greedy algorithm
- Number of exchange steps to reach greedy solution should be finite...

# Another Exchange Argument Example

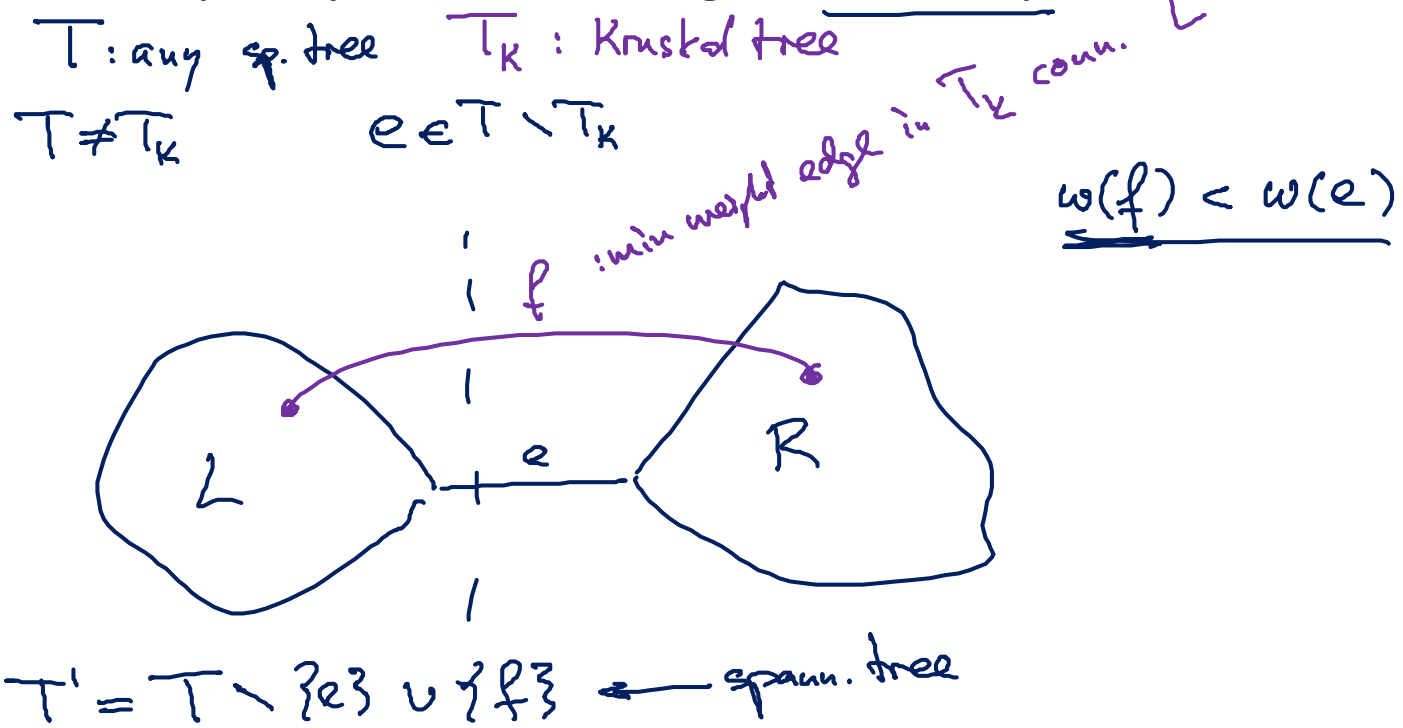
- **Minimum spanning tree (MST)** problem
  - Classic graph-theoretic optimization problem
- **Given:** weighted graph
- **Goal:** spanning tree with min. total weight
- Several greedy algorithms work
- Kruskal's algorithm:
  - Start with empty edge set
  - As long as we do not have a spanning tree:  
**add minimum weight edge that doesn't close a cycle**

# Kruskal Algorithm: Example



# Kruskal is Optimal

- Basic exchange step: swap <sup>two</sup> ~~to~~ edges to get from tree  $T$  to tree  $T'$ 
  - Swap out edge not in Kruskal tree, swap in edge in Kruskal tree
  - Swapping does not increase total weight
- For simplicity; assume, weights are unique:



# Matroids

$$E = \{1, 2, 3, 4\}$$

$$I = \{\emptyset, \{1, 3\}, \{2, 3\}, \{3\}, \{4\}, \{1, 2\}, \dots, \{1, 2, 3, 4\}\}$$

- Same, but more abstract...



## Matroid: pair $(E, I)$

- $E$ : set, called the **ground set** *set of elements*
- $I$ : finite family of finite subsets of  $E$  (i.e.,  $I \subseteq 2^E$ ), called independent sets

$$\forall A \in I \quad \forall A' \subseteq A: A' \in I$$

$(E, I)$  needs to satisfy 3 properties:

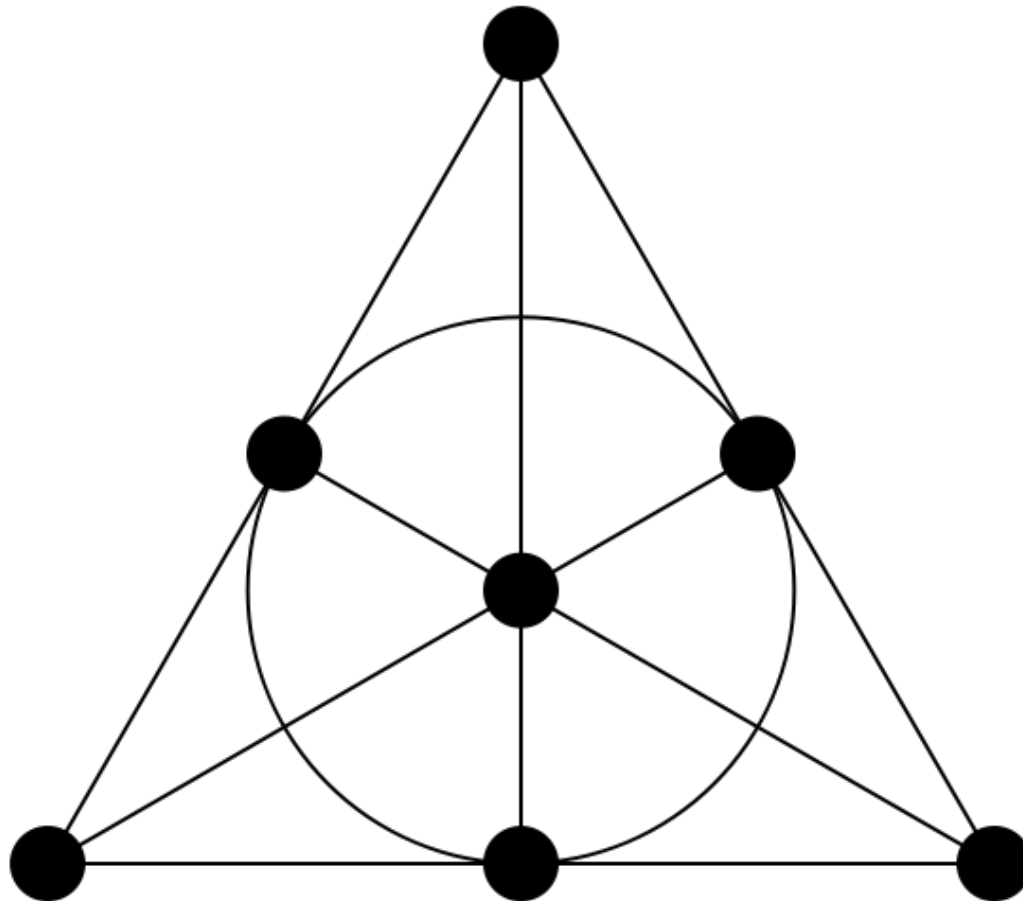
1. Empty set is independent, i.e.,  $\emptyset \in I$  (implies that  $I \neq \emptyset$ )
2. **Hereditary property:** For all  $A \subseteq E$  and all  $A' \subseteq A$ ,  
**if  $A \in I$ , then also  $A' \in I$**
3. **Augmentation / Independent set exchange property:**  
 If  $A, B \in I$  and  $|A| > |B|$ , there exists  $x \in A \setminus B$  such that

$$\underline{B'} := \underline{B} \cup \underline{\{x\}} \in \underline{I}$$



# Example

- Fano matroid:
  - Smallest finite projective plane of order 2...





# Matroids and Greedy Algorithms

**Weighted matroid:** each  $e \in E$  has a weight  $w(e) > 0$

**Goal:** find **maximum weight independent set**

**Greedy algorithm:**

1. Start with  $S = \emptyset$
2. Add max. weight  $e \in E \setminus S$  to  $S$  such that  $S \cup \{e\} \in I$

**Claim:** **greedy algorithm** computes **optimal** solution

# Greedy is Optimal

$(E, I)$

$s = |S|$

$a = |A|$



• S: greedy solution

$$S \subseteq E, S \in I$$

A: any other solution

$$A \subseteq E, A \in I$$

$$\underline{s \geq a}$$

$|S| \geq |A|$ :

for contr., assume  $|A| > |S|$

excl. prop:  $\exists x \in A \setminus S : S \cup \{x\} \in I$

For contradiction, assume

$$\underline{\omega(S) < \omega(A)}$$

$$\forall i: \underline{\omega(x_i)} \geq \omega(y_i)$$

$$S = \{ \underline{x_1}, \underline{x_2}, \dots, \underline{x_s} \}$$

$$\omega(x_1) \geq \omega(x_2) \geq \dots$$

$$A = \{ \underline{y_1}, \underline{y_2}, \dots, \underline{y_a} \}$$

$$\omega(y_1) \geq \omega(y_2) \geq \dots$$

there has to be a smallest  $k$  s.t.  $\omega(x_k) < \omega(y_k)$

$$S' = \{ \underline{x_1}, \dots, \underline{x_{k-1}} \}$$

excl. prop:

$$\exists y \in A' : S' \cup \{y\} \in I$$

$$\omega(y) \geq \omega(y_k) > \omega(x_k)$$

greedy should have added  $y$